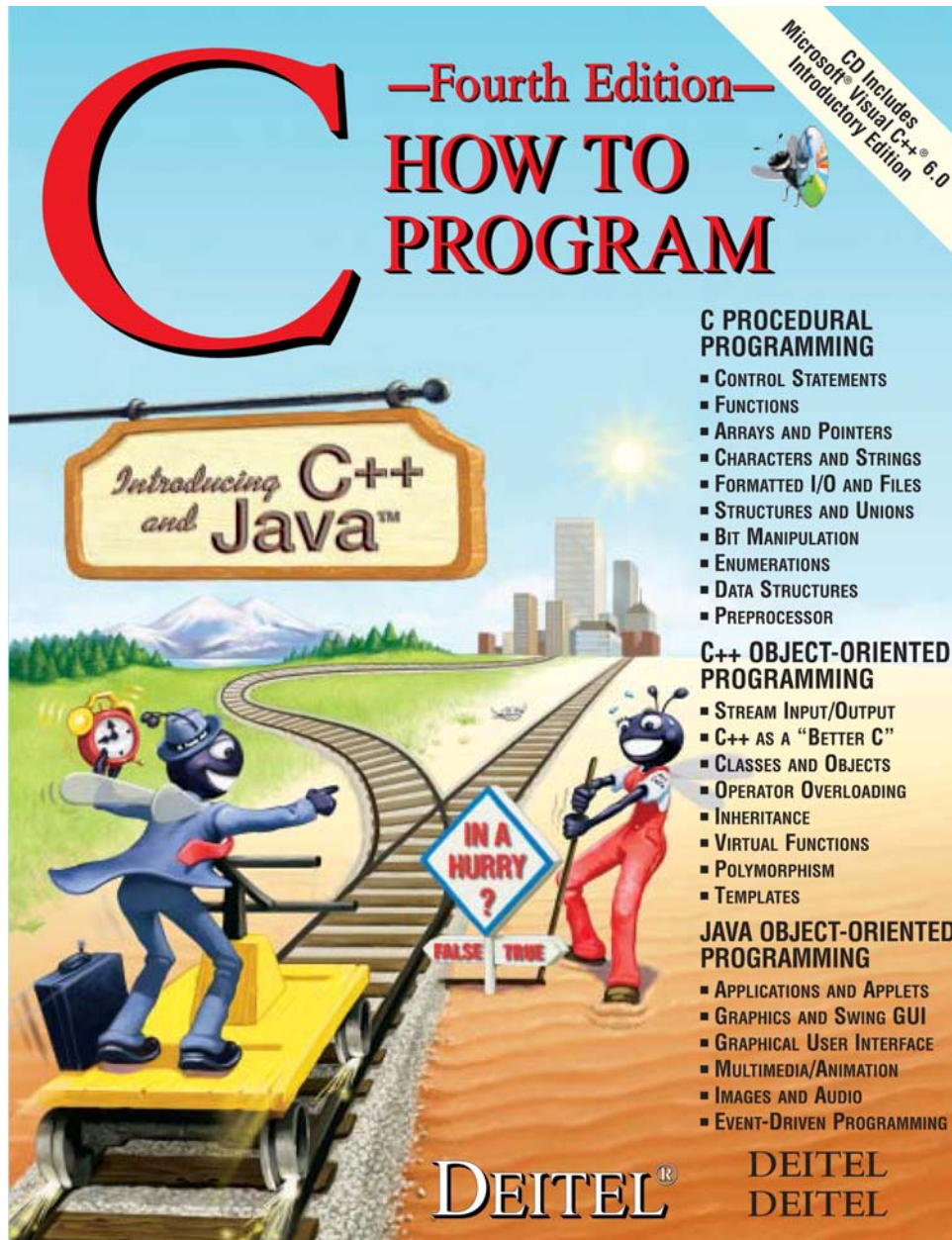


Instructor's Manual

for

C How to Program, 4/e



Deitel & Deitel

Contents

1	Introduction to Computers, the Internet and the World Wide Web	1
2	Introduction to C Programming	5
3	Structured Program Development in C	19
4	C Program Control	55
5	C Functions	97
6	C Arrays	169
7	Pointers	233
8	C Characters and Strings	283
9	C Formatted Input/Output	319
10	Structures, Unions, Bit Manipulations and Enumerations	333
11	C File Processing	353
12	Data Structures	375

ii

13	The Preprocessor	441
14	Other C Topics	447
15	C++ as a ‘Better C’	457
16	C++ Classes and Data Abstraction	463
17	C++ Classes: Part II	485
18	C++ Operator Overloading	493
19	C++ Inheritance	499
20	C++ Virtual Functions and Polymorphism	511
21	C++ Stream Input/Output	519
22	C++ Templates	537
23	C++ Exception Handling: Solution	543
24	Introduction to Java Applications and Applets	547
25	Beyond C & C++: Operators, Methods & Arrays in Java	557
26	Java Object-Based Programming	585
27	Java Object-Oriented Programming	603
28	Java Graphics and Java2D	617
29	Java Graphical User Interface Components	633
30	Java Multimedia: Images, Animation, and Audio	661

1

Introduction to Computers, the Internet and the World Wide Web: Solutions

SOLUTIONS

1.3 Categorize each of the following items as either hardware or software:

a) CPU

ANS: hardware.

b) C compiler

ANS: software.

c) ALU

ANS: hardware.

d) C preprocessor

ANS: software.

e) input unit

ANS: hardware.

f) a word processor program

ANS: software.

1.4 Why might you want to write a program in a machine-independent language instead of a machine-dependent language? Why might a machine-dependent language be more appropriate for writing certain types of programs?

ANS: Machine independent languages are useful for writing programs to be executed on multiple computer platforms. Machine dependent languages are appropriate for writing programs to be executed on a single platform. Machine dependent languages tend to exploit the efficiencies of a particular machine.

1.5 Translator programs such as assemblers and compilers convert programs from one language (referred to as the *source* language) to another language (referred to as the *object* language). Determine which of the following statements are true and which are false:

a) A compiler translates high-level language programs into object language.

ANS: True.

b) An assembler translates source language programs into machine language programs.

ANS: True.

c) A compiler converts source language programs into object language programs.

ANS: False.

d) High-level languages are generally machine-dependent.

ANS: False.

e) A machine language program requires translation before the program can be run on a computer.

ANS: False.

1.6 Fill in the blanks in each of the following statements:

a) Devices from which users access timesharing computer systems are usually called _____.

ANS: terminals.

b) A computer program that converts assembly language programs to machine language programs is called _____.

ANS: an assembler.

c) The logical unit of the computer that receives information from outside the computer for use by the computer is called _____.

ANS: The input unit.

d) The process of instructing the computer to solve specific problems is called _____.

ANS: computer programming.

e) What type of computer language uses English-like abbreviations for machine language instructions? _____.

ANS: a high-level language.

f) Which logical unit of the computer sends information that has already been processed by the computer to various devices so that the information may be used outside the computer? _____.

ANS: The output unit.

g) The general name for a program that converts programs written in a certain computer language into machine language is _____.

ANS: compiler.

h) Which logical unit of the computer retains information? _____.

ANS: memory unit and secondary storage unit.

i) Which logical unit of the computer performs calculations? _____.

ANS: arithmetic and logical unit.

j) Which logical unit of the computer makes logical decisions? _____.

ANS: arithmetic and logical unit

k) The commonly used abbreviation for the computer's control unit is _____.

ANS: CPU.

l) The level of computer language most convenient to the programmer for writing programs quickly and easily is _____.

ANS: high-level language.

m) The only language that a computer can directly understand is called that computer's _____.

ANS: machine language.

n) Which logical unit of the computer coordinates the activities of all the other logical units? _____.

ANS: central processing unit.

1.7 State whether each of the following is *true* or *false*. If *false*, explain your answer.

a) Machine languages are generally machine dependent.

ANS: True. Machine languages are closely related to the hardware of a particular machine.

b) Timesharing truly runs several users simultaneously on a computer.

ANS: False. Time sharing systems split CPU time amongst several users so that the users appear to be operating simultaneously

c) Like other high-level languages, C is generally considered to be machine independent.

ANS: True. C programs can be written on most machines, and with some care, C programs can be written on one machine and run on many machines with few changes or no changes.

1.8 Discuss the meaning of each of the following names:

a) `stdin`

ANS: This refers to the standard input device. The standard input device is normally connected to the keyboard

b) `stdout`

ANS: This refers to the standard output device. The standard output device is normally connected to the computer screen.

c) `stderr`

ANS: This refers to the standard error device. Error messages are normally sent to this device which is typically connected to the computer screen.

1.9 Why is so much attention today focused on object-oriented programming in general and C++ in particular?

ANS: Object-oriented programming enables the programmer to build reusable software components that model items in the real world. Building software quickly, correctly, and economically has been an elusive goal in the software industry. The modular, object-oriented design and implementation approach has been found to increase productivity 10 to 100 times over conventional programming languages while reducing development time, errors, and cost. C++ is used for object-oriented programming because it is a superset of the C programming language and C is widely used.

- 1.10** Which programming language is best described by each of the following?
- a) Developed by IBM for scientific and engineering applications.
ANS: FORTRAN
 - b) Developed specifically for business applications.
ANS: COBOL
 - c) Developed for teaching structured programming.
ANS: Pascal
 - d) Named after the world's first computer programmer.
ANS: Ada
 - e) Developed to familiarize novices with programming techniques.
ANS: BASIC
 - f) Specifically developed to help programmers migrate to .NET.
ANS: C#
 - g) Known as the development language of UNIX.
ANS: C
 - h) Formed primarily by adding object-oriented programming to C.
ANS: C++
 - i) Succeeded initially because of its ability to create Web pages with dynamic content.
ANS: Java

2

Introduction to C Programming: Solutions

SOLUTIONS:

2.7 Identify and correct the errors in each of the following statements (*Note: there may be more than one error per statement*):

a) `scanf("d", value);`

ANS: `scanf("%d", &value);`

b) `printf("The product of %d and %d is %d"\n, x, y);`

ANS: `printf("The product of %d and %d is %d\n", x, y, z);`

c) `firstNumber + secondNumber = sumOfNumbers`

ANS: `sumOfNumbers = firstNumber + secondNumber;`

d) `if (number => largest)`

`largest == number;`

ANS:

`if (number >= largest)`

`largest = number;`

e) `/* Program to determine the largest of three integers */`

ANS: `/* Program to determine the largest of three integers */`

f) `Scanf("%d", anInteger);`

ANS: `scanf("%d", &anInteger);`

g) `printf("Remainder of %d divided by %d is\n", x, y, x % y);`

ANS: `printf("Remainder of %f divided by %d is %d\n", x, y, x % y);`

h) `if (x = y);`

`printf(%d is equal to %d\n", x, y);`

ANS:

`if (x == y)`

`printf("%d is equal to %d\n", x, y);`

i) `print("The sum is %d\n," x + y);`

ANS: `printf("The sum is %d\n", x + y);`

j) `Printf("The value you entered is: %d\n, &value);`

ANS: `printf("The value you entered is: %d\n", value);`

2.8 Fill in the blanks in each of the following:

a) _____ are used to document a program and improve its readability.

ANS: comments.

b) The function used to display information on the screen is _____.

ANS: `printf`.

c) A C statement that makes a decision is _____.

ANS: `if`.

d) Calculations are normally performed by _____ statements.

ANS: assignment.

e) The _____ function inputs values from the keyboard.

ANS: `scanf`.

2.9 Write a single C statement or line that accomplishes each of the following:

a) Print the message "Enter two numbers."

ANS: `printf("Enter two numbers\n");`

b) Assign the product of variables b and c to variable a.

ANS: `a = b * c;`

c) State that a program performs a sample payroll calculation (i.e., use text that helps to document a program).

ANS: `/* Sample payroll calculation program */`

d) Input three integer values from the keyboard and place these values in integer variables a, b and c.

ANS: `scanf("%d%d%d", &a, &b, &c);`

2.10 State which of the following are *true* and which are *false*. If *false*, explain your answer.

a) C operators are evaluated from left to right.

ANS: False. Some operators are evaluated left to right and others are evaluated from right to left depending on their associativity (see Appendix C).

b) The following are all valid variable names: `_under_bar_`, `m928134`, `t5`, `j7`, `her_sales`, `his_account_total`, `a`, `b`, `c`, `z`, `z2`.

ANS: True.

c) The statement `printf("a = 5;");` is a typical example of an assignment statement.

ANS: False. The statement prints `a = 5;` on the screen.

d) A valid arithmetic expression containing no parentheses is evaluated from left to right.

ANS: False. Multiplication, division, and modulus are all evaluated first from left to right, then addition and subtraction are evaluated from left to right.

e) The following are all invalid variable names: `3g`, `87`, `67h2`, `h22`, `2h`.

ANS: False. Those beginning with a number are invalid.

2.11 Fill in the blanks in each of the following:

a) What arithmetic operations are on the same level of precedence as multiplication? _____.

ANS: division, modulus.

b) When parentheses are nested, which set of parentheses is evaluated first in an arithmetic expression? _____.

ANS: The innermost pair of parenthesis.

c) A location in the computer's memory that may contain different values at various times throughout the execution of a program is called a _____.

ANS: variable.

2.12 What, if anything, prints when each of the following C statements is performed? If nothing prints, then answer "nothing."

Assume `x = 2` and `y = 3`.

a) `printf("%d", x);`

ANS: 2

b) `printf("%d", x + x);`

ANS: 4

c) `printf("x=");`

ANS: `x=`

d) `printf("x=%d", x);`

ANS: `x=2`

e) `printf("%d = %d", x + y, y + x);`

ANS: `5 = 5`

f) `z = x + y;`

ANS: Nothing. Value of `x + y` is assigned to `z`.

g) `scanf("%d%d", &x, &y);`

ANS: Nothing. Two integer values are read into the location of `x` and the location of `y`.

h) `/* printf("x + y = %d", x + y); */`

ANS: Nothing. This is a comment.

i) `printf("\n");`

ANS: A newline character is printed, and the cursor is positioned at the beginning of the next line on the screen.

2.13 Which, if any, of the following C statements contain variables involved in destructive read-in?

- a) `scanf("%d%d%d%d", &b, &c, &d, &e, &f);`
- b) `p = i + j + k + 7;`
- c) `printf("Destructive read-in");`
- d) `printf("a = 5");`

ANS: (a).

2.14 Given the equation $y = ax^3 + 7$, which of the following, if any, are correct C statements for this equation?

- a) `y = a * x * x * x + 7;`
- b) `y = a * x * x * (x + 7);`
- c) `y = (a * x) * x * (x + 7);`
- d) `y = (a * x) * x * x + 7;`
- e) `y = a * (x * x * x) + 7;`
- f) `y = a * x * (x * x + 7);`

ANS: (a), (d), and (e).

2.15 State the order of evaluation of the operators in each of the following C statements and show the value of `x` after each statement is performed.

- a) `x = 7 + 3 * 6 / 2 - 1;`

ANS: `*` is first, `/` is second, `+` is third, and `-` is fourth. Value of `x` is 15.

- b) `x = 2 % 2 + 2 * 2 - 2 / 2;`

ANS: `%` is first, `*` is second, `/` is third, `+` is fourth, `-` is fifth. Value of `x` is 3.

- c) `x = (3 * 9 * (3 + (9 * 3 / (3))));`

ANS: 5 6 4 2 3 1. Value of `x` is 338.

2.16 Write a program that asks the user to enter two numbers, obtains the two numbers from the user and prints the sum, product, difference, quotient and remainder of the two numbers.

ANS:

```

1  /* Exercise 2.16 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int x; /* define first number */
7      int y; /* define second number */
8
9      printf( "Enter two numbers: " ); /* prompt user */
10     scanf( "%d%d", &x, &y ); /* read values from keyboard */
11
12     /* output results */
13     printf( "The sum is %d\n", x + y );
14     printf( "The product is %d\n", x * y );
15     printf( "The difference is %d\n", x - y );
16     printf( "The quotient is %d\n", x / y );
17     printf( "The modulus is %d\n", x % y );
18
19     return 0; /* indicate successful termination */
20
21 } /* end main */

```

```

Enter two numbers: 20 5
The sum is 25
The product is 100
The difference is 15
The quotient is 4
The modulus is 0

```

2.17 Write a program that prints the numbers 1 to 4 on the same line. Write the program using the following methods.

- Using one `printf` statement with no conversion specifiers.
- Using one `printf` statement with four conversion specifiers.
- Using four `printf` statements.

ANS:

```

1  /* Exercise 2.17 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      printf( "1 2 3 4\n\n" ); /* part a */
7
8      printf( "%d %d %d %d\n\n", 1, 2, 3, 4 ); /* part b */
9
10     printf( "1 " ); /* part c */
11     printf( "2 " );
12     printf( "3 " );
13     printf( "4\n" );
14
15     return 0; /* indicates successful termination */
16
17 } /* end main */

```

```

1 2 3 4
1 2 3 4
1 2 3 4

```

2.18 Write a program that asks the user to enter two integers, obtains the numbers from the user, then prints the larger number followed by the words “is larger.” If the numbers are equal, print the message “These numbers are equal.” Use only the single-selection form of the `if` statement you learned in this chapter.

ANS:

```

1  /* Exercise 2.18 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int x; /* define first number */
7      int y; /* define second number */
8
9      printf( "Enter two numbers: " ); /* prompt */
10     scanf( "%d%d", &x, &y ); /* read two integers */
11
12     /* compare the two numbers */
13     if ( x > y ) {
14         printf( "%d is larger\n", x );
15     } /* end if */
16
17     if ( x < y ) {
18         printf( "%d is larger\n", y );
19     } /* end if */
20
21     if ( x == y ) {
22         printf( "These numbers are equal\n" );
23     } /* end if */
24

```

```

25     return 0; /* indicate successful termination */
26
27 } /* end main */

```

```

Enter two numbers: 5 20
20 is larger

```

```

Enter two numbers: 239 92
239 is larger

```

```

Enter two numbers: 17 17
These numbers are equal

```

2.19 Write a program that inputs three different integers from the keyboard, then prints the sum, the average, the product, the smallest and the largest of these numbers. Use only the single-selection form of the `if` statement you learned in this chapter. The screen dialogue should appear as follows:

```

Input three different integers: 13 27 14
Sum is 54
Average is 18
Product is 4914
Smallest is 13
Largest is 27

```

ANS:

```

1  /* Exercise 2.19 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int a;          /* define first integer */
7      int b;          /* define second integer */
8      int c;          /* define third integer */
9      int smallest;  /* smallest integer */
10     int largest;   /* largest integer */
11
12     printf( "Input three different integers: " ); /* prompt user */
13     scanf( "%d%d%d", &a, &b, &c ); /* read three integers */
14
15     /* output sum, average and product of the three integers */
16     printf( "Sum is %d\n", a + b + c );
17     printf( "Average is %d\n", ( a + b + c ) / 3 );
18     printf( "Product is %d\n", a * b * c );
19
20     smallest = a; /* assume first number is the smallest */
21
22     if ( b < smallest ) { /* is b smaller? */
23         smallest = b;
24     } /* end if */
25
26     if ( c < smallest ) { /* is c smaller? */
27         smallest = c;
28     } /* end if */

```

```

29
30     printf( "Smallest is %d\n", smallest );
31
32     largest = a; /* assume first number is the largest */
33
34     if ( b > largest ) { /* is b larger? */
35         largest = b;
36     } /* end if */
37
38     if ( c > largest ) { /* is c larger? */
39         largest = c;
40     } /* end if */
41
42     printf( "Largest is %d\n", largest );
43
44     return 0; /* indicate successful termination */
45
46 } /* end main */

```

2.20 Write a program that reads in the radius of a circle and prints the circle's diameter, circumference and area. Use the constant value 3.14159 for π . Perform each of these calculations inside the `printf` statement(s) and use the conversion specifier `%f`. [*Note:* In this chapter, we have discussed only integer constants and variables. In Chapter 3 we will discuss floating-point numbers, i.e., values that can have decimal points.]

ANS:

```

1  /* Exercise 2.20 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int radius; /* circle radius */
7
8      printf( "Input the circle radius: " ); /* prompt user */
9      scanf( "%d", &radius ); /* read integer radius */
10
11     /* calculate and output diameter, circumference and area */
12     printf( "\nThe diameter is %d\n", 2 * radius );
13     printf( "The circumference is %f\n", 2 * 3.14159 * radius );
14     printf( "The area is %f\n", 3.14159 * radius * radius );
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */

```

```

Input the circle radius: 9
The diameter is 18
The circumference is 56.548620
The area is 254.468790

```

2.21 Write a program that prints a box, an oval, an arrow and a diamond as follows:

```

*****      ***      *      *
* * * * *   * * *   * * *   * *
* * * * *   * * *   * * *   * *
* * * * *   * * *   * * *   * *
* * * * *   * * *   * * *   * *
* * * * *   * * *   * * *   * *
* * * * *   * * *   * * *   * *
*****      ***      *      *

```

ANS:

```

1  /* Exercise 2.21 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      printf( "*****      ***      *      *\n" );
7      printf( "* * * * *   * * *   * * *   * *\n" );
8      printf( "* * * * *   * * *   * * *   * *\n" );
9      printf( "* * * * *   * * *   * * *   * *\n" );
10     printf( "* * * * *   * * *   * * *   * *\n" );
11     printf( "* * * * *   * * *   * * *   * *\n" );
12     printf( "* * * * *   * * *   * * *   * *\n" );
13     printf( "* * * * *   * * *   * * *   * *\n" );
14     printf( "*****      ***      *      *\n" );
15
16     return 0; /* indicates successful termination */
17
18 } /* end main */

```

2.22 What does the following code print?

```
printf( "\n*\n**\n***\n****\n*****\n" );
```

ANS:

```

*
**
***
****
*****

```

2.23 Write a program that reads in five integers and then determines and prints the largest and the smallest integers in the group. Use only the programming techniques you have learned in this chapter.

ANS:

```

1  /* Exercise 2.23 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int largest; /* largest integer */
7      int smallest; /* smallest integer */
8      int int1; /* define int1 for user input */
9      int int2; /* define int2 for user input */

```

```

10  int int3;    /* define int3 for user input */
11  int temp;   /* temporary integer for swapping */
12
13  printf( "Input 5 integers: " ); /* prompt user and read 5 ints */
14  scanf( "%d%d%d%d%d", &largest, &smallest, &int1, &int2, &int3 );
15
16  if ( smallest > largest ) { /* make comparisons */
17      temp = largest;
18      largest = smallest;
19      smallest = temp;
20  } /* end if */
21
22  if ( int1 > largest ) {
23      largest = int1;
24  } /* end if */
25
26  if ( int1 < smallest ) {
27      smallest = int1;
28  } /* end if */
29
30  if ( int2 > largest ) {
31      largest = int2;
32  } /* end if */
33
34  if ( int2 < smallest ) {
35      smallest = int2;
36  } /* end if */
37
38  if ( int3 > largest ) {
39      largest = int3;
40  } /* end if */
41
42  if ( int3 < smallest ) {
43      smallest = int3;
44  } /* end if */
45
46  printf( "The largest value is %d\n", largest );
47  printf( "The smallest value is %d\n", smallest );
48
49  return 0; /* indicate successful termination */
50
51 } /* end main */

```

```

Input 5 integers: 9 4 5 8 7
The largest value is 9
The smallest value is 4

```

2.24 Write a program that reads an integer and determines and prints whether it is odd or even. [*Hint*: Use the remainder operator. An even number is a multiple of two. Any multiple of two leaves a remainder of zero when divided by 2.]

ANS:

```

1  /* Exercise 2.24 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int integer; /* integer input by user */
7
8      printf( "Input an integer: " ); /* prompt */

```

```

9   scanf( "%d", &integer );      /* read integer */
10
11  /* test if integer is even */
12  if ( integer % 2 == 0 ) {
13      printf( "%d is an even integer\n", integer );
14  } /* end if */
15
16  /* test if integer is odd */
17  if ( integer % 2 != 0 ) {
18      printf( "%d is an odd integer\n", integer );
19  } /* end if */
20
21  return 0; /* indicate successful termination */
22
23 } /* end main */

```

```

Input an integer: 78
78 is an even integer

```

```

Input an integer: 79
79 is an odd integer

```

2.25 Print your initials in block letters down the page. Construct each block letter out of the letter it represents as shown below.

```

PPPPPPPP
 P   P
 P   P
 P   P
  P P

 JJ
 J
 J
 J
 JJJJJJ

 DDDDDDDD
 D   D
 D   D
 D   D
  D D
 DDDDD

```

ANS:

```

1  /* Exercise 2.25 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      printf( "PPPPPPPP\n" );
7      printf( "  P   P\n" );
8      printf( "   P  P\n" );
9      printf( "    P  P\n" );
10     printf( "     P P\n" );
11     printf( "\n" );
12     printf( "  JJ\n" );
13     printf( " J\n" );
14     printf( "J\n" );

```



```

15     printf( " J\n" );
16     printf( " JJJJJJJ\n" );
17     printf( "\n" );
18     printf( "DDDDDDDD\n" );
19     printf( "D      D\n" );
20     printf( "D      D\n" );
21     printf( " D      D\n" );
22     printf( " DDDDD\n" );
23
24     return 0; /* indicate successful termination */
25
26 } /* end main */

```

2.26 Write a program that reads in two integers and determines and prints if the first is a multiple of the second. [*Hint*: Use the remainder operator.]

ANS:

```

1  /* Exercise 2.26 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int integer1; /* first integer */
7      int integer2; /* second integer */
8
9      printf( "Input two integers: " ); /* prompt user */
10     scanf( "%d%d", &integer1, &integer2 ); /* read two integers */
11
12     /* use remainder operator */
13     if ( integer1 % integer2 == 0 ) {
14         printf( "%d is a multiple of %d ", integer1, integer2 );
15         printf( "by a factor of %d\n", integer1 / integer2 );
16     } /* end if */
17
18     if ( integer1 % integer2 != 0 ) {
19         printf( "%d is not a multiple of %d\n", integer1, integer2 );
20     } /* end if */
21
22     return 0; /* indicate successful termination */
23
24 } /* end main */

```

```

Input two integers: 88 11
88 is a multiple of 11 by a factor of 8

```

```

Input two integers: 777 5
777 is not a multiple of 5

```


2.28 Distinguish between the terms fatal error and non-fatal error. Why might you prefer to experience a fatal error rather than a non-fatal error?

ANS: A fatal error causes the program to terminate prematurely. A nonfatal error occurs when the logic of the program is incorrect, and the program does not work properly. A fatal error is preferred for debugging purposes. A fatal error immediately lets you know there is a problem with the program, whereas a nonfatal error can be subtle and possibly go undetected.

2.29 Here's a peek ahead. In this chapter you learned about integers and the type `int`. C can also represent uppercase letters, lowercase letters and a considerable variety of special symbols. C uses small integers internally to represent each different character. The set of characters a computer uses and the corresponding integer representations for those characters is called that computer's character set. You can print the integer equivalent of uppercase A for example, by executing the statement

```
printf( "%d", 'A' );
```

Write a C program that prints the integer equivalents of some uppercase letters, lowercase letters, digits and special symbols. As a minimum, determine the integer equivalents of the following: A B C a b c 0 1 2 \$ * + / and the blank character.

ANS:

```
1  /* Exercise 2.29 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      char intEquivalent; /* letter, digit or character */
7
8      printf( "Input a letter, digit, or character: " ); /* prompt */
9      scanf( "%c", &intEquivalent ); /* read user input */
10
11     printf( "%c's integer equivalent is %d\n", intEquivalent,
12            intEquivalent );
13
14     return 0; /* indicate successful termination */
15
16 } /* end main */
```

```
Input a letter, digit, or character: %
%'s integer equivalent is 37
```

```
Input a letter, digit, or character: y
y's integer equivalent is 121
```

```
Input a letter, digit, or character: 0
0's integer equivalent is 48
```

2.30 Write a program that inputs one five-digit number, separates the number into its individual digits and prints the digits separated from one another by three spaces each. [*Hint:* Use combinations of integer division and the remainder operation.] For example, if the user types in 42139, the program should print

```
4 2 1 3 9
```

ANS:

```

1  /* Exercise 2.30 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int number; /* number input by user */
7      int temp1; /* first temporary integer */
8      int temp2; /* second temporary integer */
9
10     printf( "Enter a five-digit number: " ); /* prompt user */
11     scanf( "%d", &number ); /* read integer */
12
13     printf( "%d ", number / 10000 ); /* print left-most digit */
14     temp2 = number % 10000;
15
16     printf( " %d ", temp2 / 1000 );
17     temp1 = temp2 % 1000;
18
19     printf( " %d ", temp1 / 100 );
20     temp2 = temp1 % 100;
21
22     printf( " %d ", temp2 / 10 );
23     temp1 = temp2 % 10;
24
25     printf( " %d\n", temp1 ); /* print right-most digit */
26
27     return 0; /* indicate successful termination */
28
29 } /* end main */

```

```

Enter a five-digit number: 23456
2 3 4 5 6

```

2.31 Using only the techniques you learned in this chapter, write a program that calculates the squares and cubes of the numbers from 0 to 10 and uses tabs to print the following table of values:

number	square	cube
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

ANS:

```

1  /* Exercise 2.31 Solution */
2  #include <stdio.h>
3
4  int main()
5  {

```

```
6     int count = 0; /* initialize count to zero */
7
8     /* calculate the square and cube for the numbers 0 to 10 */
9     printf( "\nnumber\tsquare\tcube\n" );
10    printf( "%d\t%d\t%d\n", count, count * count,
11           count * count * count );
12
13    count = count + 1; /* increment count by 1 */
14    printf( "%d\t%d\t%d\n", count, count * count,
15           count * count * count );
16
17    count = count + 1;
18    printf( "%d\t%d\t%d\n", count, count * count,
19           count * count * count );
20
21    count = count + 1;
22    printf( "%d\t%d\t%d\n", count, count * count,
23           count * count * count );
24
25    count = count + 1;
26    printf( "%d\t%d\t%d\n", count, count * count,
27           count * count * count );
28
29    count = count + 1;
30    printf( "%d\t%d\t%d\n", count, count * count,
31           count * count * count );
32
33    count = count + 1;
34    printf( "%d\t%d\t%d\n", count, count * count,
35           count * count * count );
36
37    count = count + 1;
38    printf( "%d\t%d\t%d\n", count, count * count,
39           count * count * count );
40
41    count = count + 1;
42    printf( "%d\t%d\t%d\n", count, count * count,
43           count * count * count );
44
45    count = count + 1;
46    printf( "%d\t%d\t%d\n", count, count * count,
47           count * count * count );
48
49    count = count + 1;
50    printf( "%d\t%d\t%d\n", count, count * count,
51           count * count * count );
52
53    return 0; /* indicate successful termination */
54
55 } /* end main */
```

3

Structured Program Development in C: Solutions

SOLUTIONS

3.11 Identify and correct the errors in each of the following [Note: There may be more than one error in each piece of code]:

```
a) if ( age >= 65 );
    printf( "Age is greater than or equal to 65\n" );
    else
    printf( "Age is less than 65\n" );
```

ANS:

```
if ( age >= 65 ) /* ; removed */
    printf( "Age is greater than or equal to 65\n" );
else
    printf( "Age is less than 65\n" );
```

```
b) int x = 1, total;
```

```
    while ( x <= 10 ) {
        total += x;
        ++x;
    }
```

ANS:

```
int x = 1, total = 0;
while ( x <= 10 ) {
    total += x;
    ++x;
}
```

```
c) while ( x <= 100 )
    total += x;
    ++x;
```

ANS:

```
while ( x <= 100 ) {
    total += x;
    ++x;
}
```

```
d) while ( y > 0 ) {
    printf( "%d\n", y );
    ++y;
}
```

ANS:

```
while ( y > 0 ) {
    printf( "%d\n", y );
    --y;
}
```

3.12 Fill in the blanks in each of the following:

a) The solution to any problem involves performing a series of actions in a specific _____.

ANS: order.

b) A synonym for procedure is _____.

ANS: algorithm

c) A variable that accumulates the sum of several numbers is a _____.

ANS: total.

d) The process of setting certain variables to specific values at the beginning of a program is called _____.

ANS: initialization.

e) A special value used to indicate “end of data entry” is called a _____, a _____, a _____ or a _____ value.

ANS: sentinel value, dummy value, signal value, flag value.

f) A _____ is a graphical representation of an algorithm.

ANS: flowchart.

g) In a flowchart, the order in which the steps should be performed is indicated by _____ symbols.

ANS: arrow (flowline).

h) The termination symbol indicates the _____ and _____ of every algorithm.

ANS: beginning, end.

i) Rectangle symbols correspond to calculations that are normally performed by _____ statements and input/output operations that are normally performed by calls to the _____ and _____ standard library functions.

ANS: assignment, `printf`, `scanf`.

j) The item written inside a decision symbol is called a _____.

ANS: condition.

3.13 What does the following program print?

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 1, total = 0, y;
6
7     while ( x <= 10 ) {
8         y = x * x;
9         printf( "%d\n", y );
10        total += y;
11        ++x;
12    }
13
14    printf("Total is %d\n", total);
15
16    return 0;
17 }
```

```

1
4
9
16
25
36
49
64
81
100
Total is 385

```

- 3.14** Write a single pseudocode statement that indicates each of the following:
- Display the message "Enter two numbers".
ANS: *print "enter two numbers"*
 - Assign the sum of variables *x*, *y*, and *z* to variable *p*.
ANS: $p = x + y + z$
 - The following condition is to be tested in an *if...else* selection statement: The current value of variable *m* is greater than twice the current value of variable *v*.
ANS: *if m is greater than twice v*
 do this ...
 else
 do this ...
 - Obtain values for variables *s*, *r*, and *t* from the keyboard.
ANS: *input s, input r, input t*
- 3.15** Formulate a pseudocode algorithm for each of the following:
- Obtain two numbers from the keyboard, compute the sum of the numbers and display the result.
ANS:
 Input the first number
 Input the second number
 Add the two numbers
 Output the sum
 - Obtain two numbers from the keyboard, and determine and display which (if either) is the larger of the two numbers.
ANS:
 Input the first number from the keyboard
 Input the second number from the keyboard
 If the first number is greater than the second number
 print it
 Else if the second number is greater than the first number
 print it
 Else
 print a message stating that the numbers are equal
 - Obtain a series of positive numbers from the keyboard, and determine and display the sum of the numbers. Assume that the user types the sentinel value -1 to indicate "end of data entry."
ANS:
 Input a value from the keyboard
 While the input value is not equal to -1
 add the number to the running total
 input the next number
 Print the sum
- 3.16** State which of the following are *true* and which are *false*. If a statement is *false*, explain why.
- Experience has shown that the most difficult part of solving a problem on a computer is producing a working C program.

ANS: False. The algorithm is the hardest of solving a problem.

b) A sentinel value must be a value that cannot be confused with a legitimate data value.

ANS: True.

c) Flowlines indicate the actions to be performed.

ANS: False. Flowlines indicate the order in which steps are performed.

d) Conditions written inside decision symbols always contain arithmetic operators (i.e., +, -, *, /, and %).

ANS: False. They normally contain conditional operators.

e) In top-down, stepwise refinement, each refinement is a complete representation of the algorithm.

ANS: True.

For Exercises 3.17 to 3.21, perform each of these steps:

1. Read the problem statement.
2. Formulate the algorithm using pseudocode and top-down, stepwise refinement.
3. Write a C program.
4. Test, debug, and execute the C program.

3.17 Drivers are concerned with the mileage obtained by their automobiles. One driver has kept track of several tankfuls of gasoline by recording miles driven and gallons used for each tankful. Develop a program that will input the miles driven and gallons used for each tankful. The program should calculate and display the miles per gallon obtained for each tankful. After processing all input information, the program should calculate and print the combined miles per gallon obtained for all tankfuls. Here is a sample input/output dialog:.

```
Enter the gallons used (-1 to end): 12.8
Enter the miles driven: 287
The miles / gallon for this tank was 22.421875

Enter the gallons used (-1 to end): 10.3
Enter the miles driven: 200
The miles / gallon for this tank was 19.417475

Enter the gallons used (-1 to end): 5
Enter the miles driven: 120
The miles / gallon for this tank was 24.000000

Enter the gallons used (-1 to end): -1

The overall average miles/gallon was 21.601423
```

ANS:

2)

Top:

Determine the average miles/gallon for each tank of gas, and the overall miles/gallon for an arbitrary number of tanks of gas

First refinement:

Initialize variables

Input the gallons used and the miles driven, and calculate and print the miles/gallon for each tank of gas. Keep track of the total miles and the total gallons.

Calculate and print the overall average miles/gallon.

Second refinement:

Initialize totalGallons to zero.

Initialize totalMiles to zero.

Input the gallons used for the first tank.

*While the sentinel value (-1) has not been entered for the gallons
 Add gallons to the running total in totalGallons
 Input the miles driven for the current tank
 Add miles to the running total in totalMiles
 Calculate and print the miles/gallon
 Input the gallons used for the next tank*

*Set totalAverage to totalMiles divided by totalGallons.
 print the average miles/gallon*

3)

```

1  /* Exercise 3.17 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      double gallons;          /* gallons used for current tank*/
7      double miles;           /* miles driven for current tank*/
8      double totalGallons = 0.0; /* total gallons used */
9      double totalMiles = 0.0; /* total miles driven */
10     double totalAverage;     /* average miles/gallon */
11
12     /* get gallons used for first tank */
13     printf( "Enter the gallons used ( -1 to end): " );
14     scanf( "%lf", &gallons );
15
16     /* loop until sentinel value read from user */
17     while ( gallons != -1.0 ) {
18         totalGallons += gallons; /* add current tank gallons to total */
19
20         printf( "Enter the miles driven: " ); /* get miles driven */
21         scanf( "%lf", &miles );
22         totalMiles += miles; /* add current tank miles to total */
23
24         /* display miles per gallon for current tank */
25         printf( "The Miles / Gallon for this tank was %f\n\n",
26             miles / gallons );
27
28         /* get next tank's gallons */
29         printf( "Enter the gallons used ( -1 to end): " );
30         scanf( "%lf", &gallons );
31     } /* end while */
32
33     /* calculate average miles per gallon over all tanks */
34     totalAverage = totalMiles / totalGallons;
35     printf( "\nThe overall average Miles/Gallon was %f\n", totalAverage );
36
37     return 0; /* indicate successful termination */
38
39 } /* end main */

```

3.18 Develop a C program that will determine if a department store customer has exceeded the credit limit on a charge account. For each customer, the following facts are available:

1. Account number
2. Balance at the beginning of the month
3. Total of all items charged by this customer this month
4. Total of all credits applied to this customer's account this month
5. Allowed credit limit

The program should input each of these facts, calculate the new balance (= *beginning balance* + *charges* – *credits*), and determine if the new balance exceeds the customer's credit limit. For those customers whose credit limit is exceeded, the program should display the customer's account number, credit limit, new balance and the message “Credit limit exceeded.” Here is a sample input/output dialog:

```

Enter account number ( -1 to end): 100
Enter beginning balance: 5394.78
Enter total charges: 1000.00
Enter total credits: 500.00
Enter credit limit: 5500.00
Account:      100
Credit limit: 5500.00
Balance:      5894.78
Credit Limit Exceeded.

Enter account number ( -1 to end ): 200
Enter beginning balance: 1000.00
Enter total charges: 123.45
Enter total credits: 321.00
Enter credit limit: 1500.00

Enter account number ( -1 to end ): 300
Enter beginning balance: 500.00
Enter total charges: 274.73
Enter total credits: 100.00
Enter credit limit: 800.00

Enter account number ( -1 to end ): -1

```

ANS:

2)

Top:

Determine if each of an arbitrary number of department store customers has exceeded the credit limit on a charge account.

First refinement:

Input the account number, beginning balance, total charges, total credits, and credit limit for a customer, calculate the customer's new balance and determine if the balance exceeds the credit limit. Then process the next customer.

Second refinement:

Input the first customer's account number.

While the sentinel value (-1) has not been entered for the account number

Input the customer's beginning balance

Input the customer's total charges

Input the customer's total credits

Input the customer's credit limit

Calculate the customer's new balance

If the balance exceeds the credit limit

Print the account number

Print the credit limit

Print the balance

Print "Credit Limit Exceeded"

Input the next customer's account number.

3)

```
1  /* Exercise 3.18 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int accountNumber; /* current account's number */
7      double balance;    /* current account's starting balance */
8      double charges;    /* current account's total charges */
9      double credits;    /* current account's total credits */
10     double limit;      /* current account's credit limit */
11
12     /* get account number */
13     printf( "\nEnter account number ( -1 to end): " );
14     scanf( "%d", &accountNumber );
15
16     /* loop until sentinel value read from user */
17     while ( accountNumber != -1 ) {
18         printf( "Enter beginning balance: " );
19         scanf( "%lf", &balance );
20
21         printf( "Enter total charges: " );
22         scanf( "%lf", &charges );
23
24         printf( "Enter total credits: " );
25         scanf( "%lf", &credits );
26
27         printf( "Enter credit limit: " );
28         scanf( "%lf", &limit );
29
30         balance += charges - credits; /* calculate balance */
31
32         /* if balance is over limit, display account number
33            with credit limit and balance to two digits of precision */
34         if ( balance > limit ) {
35             printf( "%s%d\n%s%.2f\n%s%.2f\n%s\n",
36                 "Account:      ", accountNumber, "Credit limit: ", limit,
37                 "Balance:      ", balance, "Credit Limit Exceeded." );
38         } /* end if */
39
40         /* prompt for next account */
41         printf( "\nEnter account number ( -1 to end): " );
42         scanf( "%d", &accountNumber );
43     } /* end while */
44
45     return 0; /* indicate successful termination */
46
47 } /* end main */
```

3.19 One large chemical company pays its salespeople on a commission basis. The salespeople receive \$200 per week plus 9% of their gross sales for that week. For example, a salesperson who sells \$5000 worth of chemicals in a week receives \$200 plus 9% of \$5000, or a total of \$650. Develop a program that will input each salesperson's gross sales for last week and will calculate and display that salesperson's earnings. Process one salesperson's figures at a time. Here is a sample input/output dialog:

```
Enter sales in dollars ( -1 to end): 5000.00
Salary is: $650.00

Enter sales in dollars ( -1 to end ): 1234.56
Salary is: $311.11

Enter sales in dollars ( -1 to end ): 1088.89
Salary is: $298.00

Enter sales in dollars ( -1 to end ): -1
```

ANS:

2) **Top:**

For an arbitrary number of salespeople, determine each salesperson's earnings for the last week.

First refinement:

Input the salesperson's sales for the week, calculate and print the salesperson's wages for the week, then process the next salesperson.

Second refinement:

Input the first salesperson's sales in dollars.

While the sentinel value (-1) has not been entered for the sales

Calculate the salesperson's wages for the week

Print the salesperson's wages for the week

Input the next salesperson's sales in dollars

3)

```
1  /* Exercise 3.19 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      double sales; /* gross weekly sales */
7      double wage; /* commissioned earnings */
8
9      /* get first sales */
10     printf( "Enter sales in dollars ( -1 to end): " );
11     scanf( "%lf", &sales );
12
13     /* loop until sentinel value read from user */
14     while ( sales != -1.0 ) {
15         wage = 200.0 + 0.09 * sales; /* calculate wage */
16
17         /* display salary */
18         printf( "Salary is: $%.2f\n\n", wage );
19
20         /* prompt for next sales */
21         printf( "Enter sales in dollars ( -1 to end): " );
22         scanf( "%lf", &sales );
23     } /* end while */
24
25     return 0; /* indicate successful termination */
26
27 }
```

3.20 The simple interest on a loan is calculated by the formula

$$\text{interest} = \text{principal} * \text{rate} * \text{days} / 365;$$

The preceding formula assumes that `rate` is the annual interest rate, and therefore includes the division by 365 (days). Develop a program that will input `principal`, `rate` and `days` for several loans, and will calculate and display the simple interest for each loan, using the preceding formula. Here is a sample input/output dialog:

```
Enter loan principal ( -1 to end): 1000.00
Enter interest rate: .1
Enter term of the loan in days: 365
The interest charge is $100.00

Enter loan principal ( -1 to end ): 1000.00
Enter interest rate: .08375
Enter term of the loan in days: 224
The interest charge is $51.40

Enter loan principal ( -1 to end ): 10000.00
Enter interest rate: .09
Enter term of the loan in days: 1460
The interest charge is $3600.00

Enter loan principal ( -1 to end ): -1
```

ANS:

2) **Top:**
For an arbitrary number of loans determine the simple interest for each loan.

First refinement:
Input the principal of the loan, the interest rate, and the term of the loan, calculate and print the simple interest for the loan, and process the next loan.

Second refinement:
input the first loan principal in dollars.
While the sentinel value (-1) has not been entered for the loan principal
Input the interest rate
Input the term of the loan in days
Calculate the simple interest for the loan
Print the simple interest for the loan
Input the loan principal for the next loan

3)

```
1  /* Exercise 3.20 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      double principal; /* loan principal */
7      double rate;      /* interest rate */
8      double interest; /* interest charge */
9      int term;         /* length of loan in days */
10
11     /* get loan principal */
12     printf( "Enter loan principal ( -1 to end): " );
13     scanf( "%lf", &principal );
```

```

14
15  /* loop until sentinel value is read from user */
16  while ( principal != -1.0 ) {
17      printf( "Enter interest rate: " ); /* get rate */
18      scanf( "%lf", &rate );
19
20      printf( "Enter term of the loan in days: " ); /* get term */
21      scanf( "%d", &term );
22
23      /* calculate interest charge */
24      interest = principal * rate * term / 365.0;
25      printf( "The interest charge is $%.2f\n\n", interest );
26
27      /* get next loan principal */
28      printf( "Enter loan principal ( -1 to end ): " );
29      scanf( "%lf", &principal );
30  } /* end while */
31
32  return 0; /* indicate successful termination */
33
34  } /* end main */

```

3.21 Develop a program that will determine the gross pay for each of several employees. The company pays “straight-time” for the first 40 hours worked by each employee and pays “time-and-a-half” for all hours worked in excess of 40 hours. You are given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your program should input this information for each employee, and should determine and display the employee's gross pay. Here is a sample input/output dialog:

```

Enter number of hours worked ( -1 to end ): 39
Enter hourly rate of the worker ( $00.00 ): 10.00
Salary is $390.00

Enter number of hours worked ( -1 to end ): 40
Enter hourly rate of the worker ( $00.00 ): 10.00
Salary is $400.00

Enter number of hours worked ( -1 to end ): 41
Enter hourly rate of the worker ( $00.00 ): 10.00
Salary is $415.00

Enter number of hours worked ( -1 to end ): -1

```

ANS:

2) **Top:**
For an arbitrary number of employees, determine the gross pay for each employee.

First refinement:

Input the number of hours worked for the employee, enter the employee's hourly wage, calculate and print the employee's gross pay, and process the next employee.

Second refinement:

*Input the first employee's number of hours worked.
While the sentinel value (-1) has not been entered for the hours worked
Input the employee's hourly wage
Calculate the employee's gross pay with overtime for hours over 40
Print the employee's gross pay
Input the number of hours worked for the next computer*

3)

```

1  /* Exercise 3.21 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      double hours; /* total hours worked */
7      double rate; /* hourly pay rate */
8      double salary; /* gross pay */
9
10     /* get first employee's hours worked */
11     printf( "Enter number of hours worked ( -1 to end ): " );
12     scanf( "%lf", &hours );
13
14     /* loop until sentinel value read from user */
15     while ( hours != -1.0 ) {
16
17         /* get hourly rate */
18         printf( "Enter hourly rate of the worker ( $00.00 ): " );
19         scanf( "%lf", &rate );
20
21         /* if employee worked less than 40 hours */
22         if ( hours <= 40 ) {
23             salary = hours * rate;
24         } /* end if */
25         else { /* compute "time-and-a-half" pay */
26             salary = 40.0 * rate + ( hours - 40.0 ) * rate * 1.5;
27         } /* end else */
28
29         /* display gross pay */
30         printf( "Salary is $%.2lf\n\n", salary );
31
32         /* prompt for next employee's data */
33         printf( "Enter number of hours worked ( -1 to end ): " );
34         scanf( "%lf", &hours );
35     } /* end while */
36
37     return 0; /* indicate successful termination */
38
39 } /* end main */

```

3.22 Write a program that demonstrates the difference between predecrementing and postdecrementing using the decrement operator --.

ANS:

```

1  /* Exercise 3.22 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int c; /* define c to use decrement operator */
7
8      c = 5;
9      printf( "%d\n", c );
10     printf( "%d\n", --c ); /* predecrement */
11     printf( "%d\n\n", c );
12

```



```

13     c = 5;
14     printf( "%d\n", c );
15     printf( "%d\n", c-- ); /* postdecrement */
16     printf( "%d\n\n", c );
17
18     return 0; /* indicate successful termination */
19
20 } /* end main */

```

```

5
4
4

5
5
4

```

3.23 Write a program that utilizes looping to print the numbers from 1 to 10 side-by-side on the same line with 3 spaces between each number.

ANS:

```

1  /* Exercise 3.23 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int i = 0; /* initialize i */
7
8      /* loop while i is less than 11 */
9      while ( ++i < 11 ) {
10         printf( "%d  ", i );
11     } /* end while */
12
13     return 0; /* indicate successful termination */
14
15 } /* end main */

```

```

1 2 3 4 5 6 7 8 9 10

```

3.24 The process of finding the largest number (i.e., the maximum of a group of numbers) is used frequently in computer applications. For example, a program that determines the winner of a sales contest would input the number of units sold by each salesperson. The salesperson who sells the most units wins the contest. Write a pseudocode program and then a program that inputs a series of 10 numbers, and determines and prints the largest of the numbers. [*Hint*: Your program should use three variables as follows]:

counter:	A counter to count to 10 (i.e., to keep track of how many numbers have been input and to determine when all 10 numbers have been processed)
number:	The current number input to the program
largest:	The largest number found so far

ANS:

*Input the first number directly into the variable largest
Increment counter to 2*

*While counter is less than or equal to 10
 input a new variable into the variable number
 If number is greater than largest
 replace largest with number
 Increment counter
 Print the value of the largest (while condition false when counter is 11)*

```

1  /* Exercise 3.24 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int counter; /* counter for 10 repetitions */
7      int number; /* current number input */
8      int largest; /* largest number found so far */
9
10     /* get first number */
11     printf( "Enter the first number: " );
12     scanf( "%d", &largest );
13     counter = 2;
14
15     /* loop 9 more times */
16     while ( counter <= 10 ) {
17         printf( "Enter next number: " ); /* get next number */
18         scanf( "%d", &number );
19
20         /* if current number input is greater than largest number,
21            update largest */
22         if ( number > largest ) {
23             largest = number;
24         } /* end if */
25
26         counter++;
27     } /* end while */
28
29     printf( "Largest is %d\n", largest ); /* display largest number */
30
31     return 0; /* indicate successful termination */
32
33 } /* end main */

```

```

Enter the first number: 7
Enter next number: 37
Enter next number: 78
Enter next number: 2
Enter next number: 437
Enter next number: 72
Enter next number: 1
Enter next number: 4
Enter next number: 36
Enter next number: 100
Largest is 437

```

3.25 Write a program that utilizes looping to print the following table of values:

N	10 * N	100 * N	1000 * N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000
6	60	600	6000
7	70	700	7000
8	80	800	8000
9	90	900	9000
10	100	1000	10000

The tab character, `\t`, may be used in the `printf` statement to separate the columns with tabs.

ANS:

```

1  /* Exercise 3.25 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int n = 0; /* counter */
7
8      /* display table headers */
9      printf( "\tN\t\t10 * N\t\t100 * N\t\t1000 * N\n\n" );
10
11     /* loop 10 times */
12     while ( ++n <= 10 ) {
13
14         /* calculate and display table values */
15         printf( "\t%-2d\t\t%-5d\t\t%-5d\t\t%-6d\n",
16             n, 10 * n, 100 * n, 1000 * n );
17     } /* end while */
18
19     return 0; /* indicate successful termination */
20
21 } /* end main */

```

3.26 Write a program that utilizes looping to produce the following table of values:

A	A+2	A+4	A+6
3	5	7	9
6	8	10	12
9	11	13	15
12	14	16	18
15	17	19	21

ANS:

```

1  /* Exercise 3.26 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int a = 3; /* counter */
7
8      /* display table headers */
9      printf( "A\tA+2\tA+4\tA+6\n\n" );
10
11     /* loop 5 times */
12     while ( a <= 15 ) {
13
14         /* calculate and display table values */
15         printf( "%d\t%d\t%d\t%d\n", a, a + 2, a + 4, a + 6 );
16         a += 3;
17     } /* end while */
18
19     return 0; /* indicate successful termination */
20
21 } /* end main */

```

3.27 Using an approach similar to Exercise 3.24, find the *two* largest values of the 10 numbers. [Note: You may input each number only once.]

ANS:

```

1  /* Exercise 3.27 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6
7      int counter;          /* counter for 10 repetitions */
8      int number;          /* current number input */
9      int largest;         /* largest number found */
10     int secondLargest = 0; /* second largest number found */
11
12     printf( "Enter the first number: " ); /* get first number */
13     scanf( "%d", &largest );
14     counter = 2;
15
16     /* loop 9 more times */
17     while ( counter <= 10 ) {
18         printf( "Enter next number: " ); /* prompt for next number */
19         scanf( "%d", &number );
20
21         /* if current number is greater than largest */
22         if ( number > largest ) {
23
24             /* update second largest with previous largest */
25             secondLargest = largest;
26
27             /* update largest with current number */
28             largest = number;
29         } /* end if */
30     } else {
31

```

```

32     /* if number is between secondLargest and largest */
33     if ( number > secondLargest ) {
34         secondLargest = number;
35     } /* end if */
36
37     } /* end else */
38
39     ++counter;
40 } /* end while */
41
42 /* display largest two numbers */
43 printf( "Largest is %d\n", largest );
44 printf( "Second largest is %d\n", secondLargest );
45
46 return 0; /* indicate successful termination */
47
48 } /* end main */

```

```

Enter the first number: 100
Enter next number: 102
Enter next number: 83
Enter next number: 3883
Enter next number: 328
Enter next number: 28
Enter next number: 839
Enter next number: 2398
Enter next number: 182
Enter next number: 0
Largest is 3883
Second largest is 2398

```

3.28 Modify the program in Figure 3.10 to validate its inputs. On any input, if the value entered is other than 1 or 2, keep looping until the user enters a correct value.

ANS:

```

1  /* Exercise 3.28 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int passes = 0; /* number of passes */
7      int failures = 0; /* number of failures */
8      int student = 1; /* student counter */
9      int result; /* one exam result */
10
11     /* process 10 students using counter-controlled loop */
12     while ( student <= 10 ) {
13
14         /* prompt user for input and obtain value from user */
15         printf( "Enter result ( 1=pass, 2=fail ): " );
16         scanf( "%d", &result );
17
18         /* loop until valid input */
19         while ( result != 1 && result != 2 ) {
20             printf( "Invalid result\nEnter result ( 1=pass, 2=fail ): " );
21             scanf( "%d", &result );
22         } /* end inner while */
23

```

```

24     /* if result 1, increment passes */
25     if ( result == 1 ) {
26         ++passes;
27     } /* end if */
28     else { /* if result is not 1, increment failures */
29         ++failures;
30     } /* end else */
31
32     ++student;
33 } /* end while */
34
35 printf( "Passed %d\nFailed %d\n", passes, failures );
36
37 /* if more than eight students passed, print "raise tuition" */
38 if ( passes >= 8 ) {
39     printf( "Raise tuition\n" );
40 } /* end if */
41
42 return 0; /* indicate successful termination */
43
44 } /* end main */

```

```

Enter result ( 1=pass, 2=fail ): 1
Enter result ( 1=pass, 2=fail ): 2
Enter result ( 1=pass, 2=fail ): 3
Invalid result
Enter result ( 1=pass, 2=fail ): 4
Invalid result
Enter result ( 1=pass, 2=fail ): 2
Enter result ( 1=pass, 2=fail ): 2
Enter result ( 1=pass, 2=fail ): 2
Enter result ( 1=pass, 2=fail ): 1
Enter result ( 1=pass, 2=fail ): 1
Enter result ( 1=pass, 2=fail ): 1
Enter result ( 1=pass, 2=fail ): 1
Enter result ( 1=pass, 2=fail ): 1
Enter result ( 1=pass, 2=fail ): 1
Passed 6
Failed 4

```

3.29 What does the following program print?

```

1  #include <stdio.h>
2
3  /* function main begins program execution */
4  int main()
5  {
6      int count = 1; /* initialize count */
7
8      while ( count <= 10 ) { /* loop 10 times */
9
10         /* output line of text */
11         printf( "%s\n", count % 2 ? "*****" : "+++++++" );
12         count++; /* increment count */
13     } /* end while */
14
15     return 0; /* indicate program ended successfully */
16
17 } /* end function main */

```

ANS:

```

****
+++++++
****
+++++++
****
+++++++
****
+++++++
****
+++++++

```

3.30 What does the following program print?

```

1  #include <stdio.h>
2
3  /* function main begins program execution */
4  int main()
5  {
6      int row = 10; /* initialize row */
7      int column; /* define column */
8
9      while ( row >= 1 ) { /* loop until row < 1 */
10         column = 1; /* set column to 1 as iteration begins */
11
12         while ( column <= 10 ) { /* loop 10 times */
13             printf( "%s", row % 2 ? "<": ">" ); /* output */
14             column++; /* increment column */
15         } /* end inner while */
16
17         row--; /* decrement row */
18         printf( "\n" ); /* begin new output line */
19     } /* end outer while */
20
21     return 0; /* indicate program ended successfully */
22
23 } /* end function main */

```

ANS:

```

>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<

```

3.31 (*Dangling Else Problem*) Determine the output for each of the following when x is 9 and y is 11 and when x is 11 and y is 9. Note that the compiler ignores the indentation in a C program. Also, the compiler always associates an `else` with the previous `if` unless told to do otherwise by the placement of braces `{}`. Because, on first glance, the programmer may not be sure which `if` an `else` matches, this is referred to as the “dangling else” problem. We have eliminated the indentation from the following code to make the problem more challenging. [*Hint*: Apply indentation conventions you have learned.]

```
a) if ( x < 10 )
    if ( y > 10 )
        printf( "*****\n" );
    else
        printf( "####\n" );
    printf( "$$$$$\n" );
```

ANS:

$x = 9, y = 11$

```
*****
$$$$$
```

$x = 11, y = 9$

```
$$$$$
```

```
b) if ( x < 10 ) {
    if ( y > 10 )
        printf( "*****\n" );
    }
    else {
        printf( "####\n" );
        printf( "$$$$$\n" );
    }
```

ANS:

$x = 9, y = 11$

```
*****
```

$x = 11, y = 9$

```
####
$$$$$
```

3.32 (*Another Dangling Else Problem*) Modify the following code to produce the output shown. Use proper indentation techniques. You might not make any changes other than inserting braces. The compiler ignores the indentation in a program. We have eliminated the indentation from the following code to make the problem more challenging. [*Note*: It is possible that no modification is necessary.]

```
if ( y == 8 )
if ( x == 5 )
printf( "@@@@" );
else
printf( "####\n" );
printf( "$$$$$\n" );
printf( "&&&&\n" );
```


- a) Assuming $x = 5$ and $y = 8$, the following output is produced.

```
@@@@
$$$$
&&&&&
```

ANS:

```
if ( y == 8 ) {
    if ( x == 5 )
        printf( "@@@@\n" );
    else
        printf( "####\n" );
    printf( "$$$$\\n" );
    printf( "&&&&\\n" );
}
```

- b) Assuming $x = 5$ and $y = 8$, the following output is produced.

```
@@@@
```

ANS:

```
if ( y == 8 )
    if ( x == 5 )
        printf( "@@@@\n" );
    else {
        printf( "####\n" );
        printf( "$$$$\\n" );
        printf( "&&&&\\n" );
    }
```

- c) Assuming $x = 5$ and $y = 8$, the following output is produced.

```
@@@@
&&&&&
```

ANS:

```
if ( y == 8 )
    if ( x == 5 )
        printf( "@@@@\n" );
    else {
        printf( "####\n" );
        printf( "$$$$\\n" );
    }
    printf( "&&&&\\n" );
```

- d) Assuming $x = 5$ and $y = 7$, the following output is produced. [Note: The last three `printf` statements are all part of a compound statement.

```
####
$$$$
&&&&&
```

ANS:

```

if ( y == 8 ) {
    if ( x == 5 )
        printf( "####\n" );
}
else {
    printf( "####\n" );
    printf( "$$$$\\n" );
    printf( "&&&&\\n" );
}

```

3.33 Write a program that reads in the side of a square and then prints that square out of asterisks. Your program should work for squares of all side sizes between 1 and 20. For example, if your program reads a size of 4, it should print

```

****
****
****
****

```

ANS:

```

1  /* Exercise 3.33 Solution */
2  #include<stdio.h>
3
4  int main()
5  {
6      int side;      /* side counter */
7      int temp;     /* temporary integer */
8      int asterisk; /* asterisk counter */
9
10     printf( "Enter the square side: " ); /* get size of square */
11     scanf( "%d", &side );
12
13     temp = side;
14
15     /* loop through rows of square */
16     while ( side-- > 0 ) {
17         asterisk = temp;
18
19         /* loop through columns of square */
20         while ( asterisk-- > 0 ) {
21             printf( "*" );
22         } /* end inner while */
23
24         putchar( '\\n' );
25     } /* end outer while */
26
27     return 0; /* indicate successful termination */
28
29 } /* end main */

```

3.34 Modify the program you wrote in Exercise 3.33 so that it prints a hollow square. For example, if your program reads a size of 5, it should print

```
*****
*   *
*   *
*   *
*****
```

ANS:

```
1  /* Exercise 3.34 Solution */
2  #include<stdio.h>
3
4  int main()
5  {
6      int side;          /* side counter */
7      int rowPosition; /* row counter */
8      int size;         /* length of side */
9
10     printf( "Enter the square side: " ); /* prompt for side length */
11     scanf( "%d", &side );
12
13     size = side; /* set size counter to length of side */
14
15     /* loop side number of times */
16     while ( side > 0 ) {
17         rowPosition = size; /* set row counter to length of size */
18
19         /* loop rowPosition number of times */
20         while ( rowPosition > 0 ) {
21
22             /* if side or row counter is 1 or size print an '*' */
23             if ( size == side ) {
24                 printf( "*" );
25             } /* end if */
26             else if ( side == 1 ) {
27                 printf( "*" );
28             } /* end else if */
29             else if ( rowPosition == 1 ) {
30                 printf( "*" );
31             } /* end else if */
32             else if ( rowPosition == size ) {
33                 printf( "*" );
34             } /* end else if */
35             else { /* otherwise, print a space */
36                 printf( " " );
37             } /* end else */
38
39             --rowPosition; /* decrement row counter */
40         } /* end inner while */
41
42         printf( "\n" ); /* new line for next row */
43         --side; /* decrement side counter */
44     } /* end outer while */
45
46     return 0; /* indicate successful termination */
47
48 } /* end main */
```

3.35 A palindrome is a number or a text phrase that reads the same backwards as forwards. For example, each of the following five-digit integers are palindromes: 12321, 55555, 45554 and 11611. Write a program that reads in a five-digit integer and determines whether or not it is a palindrome. [*Hint*: Use the division and remainder operators to separate the number into its individual digits.]

ANS:

```
1  /* Exercise 3.35 Solution */
2  #include<stdio.h>
3
4  int main()
5  {
6      int number;      /* input number */
7      int temp1;      /* first temporary integer */
8      int temp2;      /* second temporary integer */
9      int firstDigit; /* first digit of input */
10     int secondDigit; /* second digit of input */
11     int fourthDigit; /* fourth digit of input */
12     int fifthDigit; /* fifth digit of input */
13
14     printf( "Enter a five-digit number: " ); /* get number */
15     scanf( "%d", &number );
16
17     temp1 = number;
18
19     /* determine first digit by integer division by 10000 */
20     firstDigit = temp1 / 10000;
21     temp2 = temp1 % 10000;
22
23     /* determine second digit by integer division by 1000 */
24     secondDigit = temp2 / 1000;
25     temp1 = temp2 % 1000;
26
27     temp2 = temp1 % 100;
28
29     /* determine fourth digit by integer division by 10 */
30     fourthDigit = temp2 / 10;
31     temp1 = temp2 % 10;
32
33     fifthDigit = temp1;
34
35     /* if first and fifth digits are equal */
36     if ( firstDigit == fifthDigit ) {
37
38         /* if second and fourth digits are equal */
39         if ( secondDigit == fourthDigit ) {
40
41             /* number is a palindrome */
42             printf( "%d is a palindrome\n", number );
43         } /* end if */
44         else { /* number is not a palindrome */
45             printf( "%d is not a palindrome\n", number );
46         } /* end else */
47
48     } /* end if */
49     else { /* number is not a palindrome */
50         printf( "%d is not a palindrome\n", number );
51     } /* end else */
52
53     return 0; /* indicate successful termination */
54
55 } /* end main */
```

```
Enter a five-digit number: 18181
18181 is a palindrome
```

```
Enter a five-digit number: 16738
16738 is not a palindrome
```

3.36 Input an integer containing only 0s and 1s (i.e., a “binary” integer) and print its decimal equivalent. [*Hint*: Use the remainder and division operators to pick off the “binary” number’s digits one at a time from right to left. Just as in the decimal number system in which the rightmost digit has a positional value of 1, and the next digit left has a positional value of 10, then 100, then 1000, etc., in the binary number system the rightmost digit has a positional value of 1, the next digit left has a positional value of 2, then 4, then 8, etc. Thus the decimal number 234 can be interpreted as $4 * 1 + 3 * 10 + 2 * 100$. The decimal equivalent of binary 1101 is $1 * 1 + 0 * 2 + 1 * 4 + 1 * 8$ or $1 + 0 + 4 + 8$ or 13.]

ANS:

```
1  /* Exercise 3.36 Solution */
2  #include<stdio.h>
3
4  int main()
5  {
6      int binary;          /* current value of binary number */
7      int number;         /* input binary number */
8      int decimal = 0;    /* current value of decimal number */
9      int highBit = 16;   /* value of highest bit */
10     int factor = 10000; /* factor of 10 to pick off digits */
11
12     /* prompt for binary input */
13     printf( "Enter a binary number ( 5 digits maximum ): " );
14     scanf( "%d", &binary );
15
16     number = binary; /* save in number for final display */
17
18     /* loop 5 times using powers of 2 */
19     while ( highBit >= 1 ) {
20
21         /* update decimal value with decimal value corresponding
22          to current highest binary bit */
23         decimal += binary / factor * highBit;
24
25         /* reduce high bit by factor of 2, i.e.,
26          move one bit to the right */
27         highBit /= 2;
28
29         /* reduce binary number to eliminate current highest bit */
30         binary %= factor;
31
32         /* reduce factor by power of 10, i.e.,
33          move one bit to the right */
34         factor /= 10;
35     } /* end while */
36
37     /* display decimal value */
38     printf( "The decimal equivalent of %d is %d\n", number, decimal );
39
40     return 0; /* indicate successful termination */
41
42 } /* end main */
```

```
Enter a binary number ( 5 digits maximum ): 10111
The decimal equivalent of 10111 is 23
```

```
Enter a binary number ( 5 digits maximum ): 1101
The decimal equivalent of 1101 is 13
```

3.37 How can you determine how fast your own computer really operates? Write a program with a `while` loop that counts from 1 to 300,000,000 by 1s. Every time the count reaches a multiple of 100,000,000 print that number on the screen. Use your watch to time how long each million repetitions of the loop takes.

ANS:

```
1  /* Exercise 3.37 Solution */
2  #include<stdio.h>
3
4  int main()
5  {
6      long int count = 1; /* counter */
7
8      /* loop to 300,000,000 */
9      while( count <= 300000000 ) {
10
11         if ( count % 100000000 == 0 ) {
12             printf( "Multiple is %d\n", count / 100000000 );
13         } /* end if */
14
15         ++count; /* increment count */
16     } /* end while */
17
18     return 0; /* indicate successful termination */
19
20 }
```

```
Multiple is 1
Multiple is 2
Multiple is 3
```

3.38 Write a program that prints 100 asterisks, one at a time. After every tenth asterisk, your program should print a newline character. (Hint: Count from 1 to 100. Use the remainder operator to recognize each time the counter reaches a multiple of 10.)

ANS:

```
1  /* Exercise 3.38 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int count = 0; /* counter */
7
8      /* loop to 100 */
9      while( ++count <= 100 )
10
11         /* print a new line after every 10th asterisk */
12         count % 10 == 0 ? printf( "\n" ) : printf( "*" );
13
14     return 0; /* indicate successful termination */
```

```

15
16 } /* end main */

```

```

*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

- 3.39** Write a program that reads an integer and determines and prints how many digits in the integer are 7s.
ANS:

```

1  /* Exercise 3.39 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int number;          /* user input */
7      int numCopy;        /* copy of number */
8      int factor = 10000; /* set factor to pick off digits */
9      int digit;          /* individual digit of number */
10     int sevens = 0;     /* sevens counter */
11
12     printf( "Enter a 5-digit number: " ); /* get number from user */
13     scanf( "%d", &number );
14
15     numCopy = number;
16
17     /* loop through each of the 5 digits */
18     while ( factor >= 1 ) {
19         digit = numCopy / factor; /* pick off next digit */
20
21         if ( digit == 7 ) { /* if digit equals 7, increment sevens */
22             ++sevens;
23         } /* end if */
24
25         numCopy %= factor;
26         factor /= 10;
27     } /* end while */
28
29     /* output number of sevens */
30     printf( "The number %ld has %d seven(s) in it\n", number, sevens );
31
32     return 0; /* indicate successful termination */
33
34 } /* end main */

```

```

Enter a 5-digit number: 17737
The number 17737 has 3 seven(s) in it

```

```
Enter a 5-digit number: 11727
The number 11727 has 2 seven(s) in it
```

3.40 Write a program that displays the following checkerboard pattern

```
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
```

Your program must use only three output statements, one of each of the following forms:

```
printf( "* " );
printf( " " );
printf( "\n" );
```

ANS:

```
1  /* Exercise 3.40 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int side = 8; /* side counter */
7      int row;     /* row counter */
8      int mod;    /* remainder */
9
10     /* loop 8 times */
11     while ( side >= 1 ) {
12         row = 8; /* reset row counter */
13         mod = side % 2;
14
15         /* loop 8 times */
16         while ( row >= 1 ) {
17
18             /* if odd row, begin with a space */
19             if ( mod != 0 ) {
20                 printf( " " );
21                 mod = 0;
22             } /* end if */
23
24             printf( "* " );
25             --row;
26         } /* end while */
27
28         printf( "\n" ); /* go to next line */
29         --side;
30     } /* end while */
31
32     return 0; /* indicate successful termination */
33
34 }
```


3.41 Write a program that keeps printing the multiples of the integer 2, namely 2, 4, 8, 16, 32, 64, etc. Your loop should not terminate (i.e., you should create an infinite loop). What happens when you run this program?

ANS: Program execution terminates when largest integer is exceeded (i.e., the loop continuation test fails when the maximum value for an integer is exceeded. On a 4-byte system, the largest integer value is 2147483647 and anything above that is represented by a negative number, which fails the loop continuation test).

```
1  /* Exercise 3.41 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int multiple = 1; /* counter */
7
8      /* infinite loop */
9      while ( multiple > 0 ) {
10
11         /* calculate the next power of two */
12         multiple *= 2;
13         printf( "%d\n", multiple );
14     } /* end while */
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */
```

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
```

3.42 Write a program that reads the radius of a circle (as a float value) and computes and prints the diameter, the circumference and the area. Use the value 3.14159 for π .

ANS:

```

1  /* Exercise 3.42 Solution */
2  #include<stdio.h>
3
4  int main()
5  {
6      float radius;      /* input radius */
7      float pi = 3.14159; /* value for pi */
8
9      printf( "Enter the radius: "); /* get radius value */
10     scanf( "%f", &radius );
11
12     /* compute and display diameter */
13     printf( "The diameter is %.2f\n", radius * 2 );
14
15     /* compute and display circumference */
16     printf( "The circumference is %.2f\n", 2 * pi * radius );
17
18     /* compute and display area */
19     printf( "The area is %.2f\n", pi * radius * radius );
20
21     return 0; /* indicate successful termination */
22
23 } /* end main */

```

```

Enter the radius: 4.7
The diameter is 9.40
The circumference is 29.53
The area is 69.40

```

3.43 What is wrong with the following statement? Rewrite the statement to accomplish what the programmer was probably trying to do.

```
printf( "%d", ++( x + y ) );
```

ANS: `printf("%d", 1 + x + y);`

3.44 Write a program that reads three nonzero float values and determines and prints if they could represent the sides of a triangle.

ANS:

```

1  /* Exercise 3.44 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      double a; /* first number */
7      double b; /* second number */
8      double c; /* third number */
9
10     /* input 3 numbers */
11     printf( "Enter three doubleing point numbers: " );
12     scanf( "%lf%lf%lf", &a, &b, &c);
13
14     /* use Pythagorean Theorem */
15     if ( c * c == a * a + b * b ) {
16         printf( "The three numbers could be sides of a triangle\n" );
17     } /* end if */

```

```

18     else {
19         printf( "The three numbers probably");
20         printf( " are not the sides of a triangle\n" );
21     } /* end if */
22
23     return 0; /* indicate successful termination */
24
25 } /* end main */

```

Enter three doubleing point numbers: 5.7 3.6 2.2
The three numbers probably are not the sides of a triangle

Enter three doubleing point numbers: 3.0 4.0 5.0
The three numbers could be sides of a triangle

- 3.45** Write a program that reads three nonzero integers and determines and prints if they could be the sides of a right triangle.
ANS:

```

1  /* Exercise 3.45 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int a; /* first number */
7      int b; /* second number */
8      int c; /* third number */
9
10     /* input three numbers */
11     printf( "Enter three integers: ");
12     scanf( "%d%d%d", &a, &b, &c );
13
14     /* use Pythagorean Theorem */
15     if ( c * c == a * a + b * b ) {
16         printf( "The three integers are the sides of");
17         printf( " a right triangle\n" );
18     } /* end if */
19     else {
20         printf( "The three integers are not the sides");
21         printf( " of a right triangle\n" );
22     } /* end else */
23
24     return 0; /* indicate successful termination */
25
26 } /* end main */

```

Enter three integers: 3 4 5
The three integers are the sides of a right triangle

Enter three integers: 9 4 1
The three integers are not the sides of a right triangle

3.46 A company wants to transmit data over the telephone, but they are concerned that their phones may be tapped. All of their data is transmitted as four-digit integers. They have asked you to write a program that will encrypt their data so that it may be transmitted more securely. Your program should read a four-digit integer and encrypt it as follows: Replace each digit by the remainder after (*the sum of that digit plus 7*) is divided by 10. Then, swap the first digit with the third, and swap the second digit with the fourth. Then print the encrypted integer. Write a separate program that inputs an encrypted four-digit integer and decrypts it to form the original number.

ANS:

```

1  /* Exercise 3.46 Part A solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int first; /* first digit replacement */
7      int second; /* second digit replacement */
8      int third; /* third digit replacement */
9      int fourth; /* fourth digit replacement */
10     int digit; /* input number */
11     int temp1; /* temporarily hold digit */
12     int temp2; /* temporarily hold digit */
13     int encryptedNumber; /* resulting encrypted number */
14
15     /* prompt for input */
16     printf( "Enter a four digit number to be encrypted: " );
17     scanf( "%d", &digit );
18
19     temp1 = digit;
20
21     /* retrieve each digit and replace with
22      (sum of digit and 7) mod 10 */
23     first = ( temp1 / 1000 + 7 ) % 10;
24     temp2 = temp1 % 1000;
25
26     second = ( temp2 / 100 + 7 ) % 10;
27     temp1 = temp2 % 100;
28
29     third = ( temp1 / 10 + 7 ) % 10;
30     temp2 = temp1 % 10;
31
32     fourth = ( temp2 + 7 ) % 10;
33
34     /* swap first and third */
35     temp1 = first;
36     first = third * 1000; /* multiply by 1000 for 1st digit component */
37     third = temp1 * 10; /* multiply by 10 for 3rd digit component */
38
39     /* swap second and fourth */
40     temp1 = second;
41     second = fourth * 100; /* multiply by 100 for 2nd digit component */
42     fourth = temp1 * 1;
43
44     /* add components to obtain encrypted number */
45     encryptedNumber = first + second + third + fourth;
46
47     /* display encrypted number */
48     printf( "Encrypted number is %d\n", encryptedNumber );
49
50     return 0; /* indicate successful termination */
51
52 } /* end main */

```

```
Enter a four digit number to be encrypted: 5678
Encrypted number is 4523
```

```

1  /* Exercise 3.46 Part B Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int first;      /* first decrypted digit */
7      int second;    /* second decrypted digit */
8      int third;     /* third decrypted digit */
9      int fourth;    /* fourth decrypted digit */
10     int decrypted; /* decrypted number */
11     int temp1;     /* temporarily hold digit */
12     int temp2;     /* temporarily hold digit */
13     int encryptedNumber; /* input number */
14
15     /* prompt for input */
16     printf( "Enter a four digit encrypted number: " );
17     scanf( "%d", &encryptedNumber );
18
19     temp1 = encryptedNumber;
20
21     /* retrieve each digit and decrypt by
22      (sum of digit and 3) mod 10 */
23     first = ( temp1 / 1000 );
24     temp2 = temp1 % 1000;
25
26     second = ( temp2 / 100 );
27     temp1 = temp2 % 100;
28
29     third = ( temp1 / 10 );
30     temp2 = temp1 % 10;
31
32     fourth = temp2;
33
34     temp1 = ( first + 3 ) % 10;
35     first = ( third + 3 ) % 10;
36     third = temp1;
37
38     temp1 = ( second + 3 ) % 10;
39     second = ( fourth + 3 ) % 10;
40     fourth = temp1;
41
42     /* add components to obtain decrypted number */
43     decrypted = ( first * 1000 ) + ( second * 100 ) +
44                ( third * 10 ) + fourth;
45
46     /* display decrypted number */
47     printf( "Decrypted number is %d\n", decrypted );
48
49     return 0; /* indicate successful termination */
50
51 } /* end main */

```

```
Enter a four digit encrypted number: 4523
Decrypted number is 5678
```

1.1 The factorial of a nonnegative integer n is written $n!$ (pronounced “ n factorial”) and is defined as follows:
 $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 1$ (for values of n greater than or equal to 1)

and

$n! = 1$ (for $n = 0$).

For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is 120.

a) Write a program that reads a nonnegative integer and computes and prints its factorial.

ANS:

```

1  /* Exercise 3.47 Part A Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int n;                /* current multiplication factor */
7      int number = -1;      /* input number */
8      unsigned factorial = 1; /* resulting factorial */
9
10     /* loop until valid input */
11     do {
12         printf( "Enter a positive integer: " );
13         scanf( "%d", &number );
14     } while ( number < 0 ); /* end do...while */
15
16     n = number;
17
18     /* compute factorial */
19     while( n >= 0 ) {
20
21         if ( n == 0 ) {
22             factorial *= 1;
23         } /* end if */
24         else {
25             factorial *= n;
26         } /* end else */
27
28         --n;
29     } /* end while */
30
31     /* display factorial */
32     printf( "%d! is %u\n", number, factorial );
33
34     return 0; /* indicate successful termination */
35
36 } /* end main */

```

```

Enter a positive integer: 5
5! is 120

```

```

Enter a positive integer: 9
9! is 362880

```

```

Enter a positive integer: -8
Enter a positive integer: 0
0! is 1

```

b) Write a program that estimates the value of the mathematical constant e by using the formula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

ANS:

```

1  /* Exercise 3.47 Part B Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int n = 0;          /* loop counter for accuracy */
7      int fact = 1;      /* current n factorial */
8      int accuracy = 10; /* degree of accuracy */
9      double e = 0;     /* current estimated value of e */
10
11     /* loop until degree of accuracy */
12     while( n <= accuracy ) {
13
14         if ( n == 0 ) {
15             fact *= 1;
16         } /* end if */
17         else {
18             fact *= n;
19         } /* end else */
20
21         e += 1.0 / fact;
22         ++n;
23     } /* end while */
24
25     printf( "e is %f\n", e ); /* display estimated value */
26
27     return 0; /* indicate successful termination */
28
29 } /* end main */

```

```
e is 2.718282
```

c) Write a program that computes the value of e^x by using the formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

ANS:

```

1  /* Exercise 3.47 Part C Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int n = 0;          /* counter */
7      int accuracy = 15; /* degree of accuracy */
8      int x = 3;         /* exponent */
9      int times = 0;     /* counter */
10     int count;         /* copy of n */
11     double e = 1.0;    /* e raised to the x power */
12     double exp = 0.0;  /* x raised to the n power */
13     double fact = 1.0; /* n factorial */
14

```

```
15  /* loop while less than degree of accuracy */
16  while( n <= accuracy ) {
17      count = n;
18
19      /* update n! */
20      if ( n == 0 ) {
21          fact *= 1.0;
22      } /* end if */
23      else {
24          fact *= n;
25      } /* end else */
26
27      while ( times < count ) {
28
29          /* calculate x raised to the n power */
30          if ( times == 0 ) {
31              exp = 1.0;
32              exp *= x;
33          } /* end if */
34          else {
35              exp *= x;
36          } /* end else */
37
38          ++times;
39      } /* end while */
40
41      e += exp / fact; /* update e raised to the x power */
42      ++n;
43  } /* end while */
44
45  /* display result */
46  printf( "e raised to the %d power is %f\n", x, e );
47
48  return 0; /* indicate successful termination */
49
50 } /* end main */
```

```
e raised to the 3 power is 20.085534
```


4

C Program Control: Solutions

SOLUTIONS

4.5 Find the error in each of the following (Note: there may be more than one error):

a) `For (x = 100, x >= 1, x++)
printf("%d\n", x);`

ANS: F in for should be lowercase. The infinite loop can be corrected by switching the 1 and the 100 and changing the relational operator to <=. Semicolons are needed between the for conditions, not comma operators.

`for (x = 1; x <= 100; x++)
printf("%d\n", x);`

b) The following code should print whether a given integer is odd or even:

```
switch ( value % 2 ) {  
    case 0:  
        printf( "Even integer\n" );  
    case 1:  
        printf( "Odd integer\n" );  
}
```

ANS: A break is needed for case 0, otherwise both statements will be printed out.

```
switch ( value % 2 ) {  
    case 0:  
        printf( "Even integer\n" );  
        break;  
    case 1:  
        printf( "Odd integer\n" );  
}
```

c) The following code should input an integer and a character and print them. Assume the user types as input 100 A.

```
scanf( "%d", &intVal );  
charVal = getchar();  
printf( "Integer: %d\nCharacter: %c\n", intVal, charVal );
```

ANS: charVal will read the return character when the user types in intVal and hits return. To correct this, scanf should be used to read in charVal.

```
scanf( "%d", &intVal );  
scanf( "\n%c", &charVal );  
printf( "Integer: %d\nCharacter: %c\n", intVal, charVal );
```

d)

```
for ( x = .000001; x <= .0001; x += .000001 )
    printf( "%.7f\n", x );
```

ANS: Floating point numbers should never be used in loops due to imprecision. This imprecision often causes infinite loops to occur. To correct this, an integer variable should be used in the for loop.

e) The following code should output the odd integers from 999 to 1:

```
for ( x = 999; x >= 1; x += 2 )
    printf( "%d\n", x );
```

ANS: loop should be decrementing not incrementing.

```
for ( x = 999; x >= 1; x -= 2 )
    printf( "%d\n", x );
```

f) The following code should output the even integers from 2 to 100:

```
counter = 2;

Do {
    if ( counter % 2 == 0 )
        printf( "%d\n", counter );

    counter += 2;
} While ( counter < 100 );
```

ANS: D in Do should be lowercase. W in While should be lowercase. The range of 2 to 100 needs to be printed, so the relational operator < should be changed to <=, to include 100. The if test is not necessary here, because counter is being incremented by 2, and will always be even within the body of the do...while.

g) The following code should sum the integers from 100 to 150 (assume total is initialized to 0):

```
for ( x = 100; x <= 150; x++ );
    total += x;
```

ANS: semicolon at the end of the for statement should be removed, such that total += x; is in the loop's body.

```
for ( x = 100; x <= 150; x++ ) /* ; removed */
    total += x;
```

4.6 State which values of the control variable x are printed by each of the following for statements:

a)

```
for ( x = 2; x <= 13; x += 2 )
    printf( "%d\n", x );
```

ANS: 2, 4, 6, 8, 10, 12

b)

```
for ( x = 5; x <= 22; x += 7 )
    printf( "%d\n", x );
```

ANS: 5, 12, 19

c)

```
for ( x = 3; x <= 15; x += 3 )
    printf( "%d\n", x );
```

ANS: 3, 6, 9, 12, 15

d)

```
for ( x = 1; x <= 5; x += 7 )
    printf( "%d\n", x );
```

ANS: 1

e)

```
for ( x = 12; x >= 2; x -= 3 )
    printf( "%d\n", x );
```

ANS: 12, 9, 6, 3

4.7 Write for statements that print the following sequences of values:

a) 1, 2, 3, 4, 5, 6, 7

ANS:

```
for ( i = 1; i <= 7; i++ )
    printf( "%d ", i );
```

b) 3, 8, 13, 18, 23

ANS:

```
/* increments of 5 */
```

```

for ( i = 3; i <= 23; i += 5 )
    printf( "%d ", i );
c) 20, 14, 8, 2, -4, -10
ANS:
/* decrements of 6 */
for ( i = 20; i >= -10; i -= 6 )
    printf( "%d ", i );
d) 19, 27, 35, 43, 51
ANS:
/* increments of 8 */
for ( i = 19; i <= 51; i += 8 )
    printf( "%d ", i );

```

4.8 What does the following program do?

```

1  #include <stdio.h>
2
3  /* function main begins program execution */
4  int main()
5  {
6      int x;
7      int y;
8      int i;
9      int j;
10
11     /* prompt user for input */
12     printf( "Enter two integers in the range 1-20: " );
13     scanf( "%d%d", &x, &y ); /* read values for x and y */
14
15     for ( i = 1; i <= y; i++ ) { /* count from 1 to y */
16
17         for ( j = 1; j <= x; j++ ) { /* count from 1 to x */
18             printf( "@" ); /* output @ */
19         } /* end inner for */
20
21         printf( "\n" ); /* begin new line */
22     } /* end outer for */
23
24     return 0; /* indicate program ended successfully */
25
26 } /* end function main */

```

ANS:

```

Enter integers in the range 1-20: 3 4
@@@
@@@
@@@
@@@

```

4.9 Write a program that sums a sequence of integers. Assume that the first integer read with `scanf` specifies the number of values remaining to be entered. Your program should read only one value each time `scanf` is executed. A typical input sequence might be

```
5 100 200 300 400 500
```

where the 5 indicates that the subsequent five values are to be summed.

ANS:

```

1  /* Exercise 4.9 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int sum = 0; /* current sum */
7      int number; /* number of values */
8      int value; /* current value */
9      int i; /* counter */
10
11     /* display prompt */
12     printf( "Enter the number of values"
13            " to be processed: " );
14     scanf( "%d", &number ); /* input number of values */
15
16     /* loop number times */
17     for ( i = 1; i <= number; i++ ) {
18         printf( "Enter a value: " );
19         scanf( "%d", &value );
20         sum += value; /* add to sum */
21     } /* end for */
22
23     /* display sum */
24     printf( "Sum of the %d values is %d\n", number, sum );
25
26     return 0; /* indicate successful termination */
27
28 } /* end main */

```

```

Enter the number of values to be processed: 5
Enter a value: 10
Enter a value: 15
Enter a value: 20
Enter a value: 25
Enter a value: 30
Sum of the 5 values is 100

```

4.10 Write a program that calculates and prints the average of several integers. Assume the last value read with `scanf` is the sentinel 9999. A typical input sequence might be

10 8 11 7 9 9999

indicating that the average of all the values preceding 9999 is to be calculated.

ANS:

```

1  /* Exercise 4.10 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int value; /* current value */
7      int count = 0; /* number of values */
8      int total = 0; /* sum of integers */
9
10     /* display prompt */
11     printf( "Enter an integer ( 9999 to end ): " );
12     scanf( "%d", &value );
13

```

```

14  /* loop while sentinel value not read from user */
15  while ( value != 9999 ) {
16      total += value; /* update total */
17      ++count;
18
19      /* get next value */
20      printf( "Enter next integer ( 9999 to end ): " );
21      scanf( "%d", &value );
22  } /* end while */
23
24  /* show average if more than 0 values entered */
25  if ( count != 0 ) {
26      printf( "\nThe average is: %.2f\n", ( double ) total / count );
27  } /* end if */
28  else {
29      printf( "\nNo values were entered.\n" );
30  } /* end else */
31
32  return 0; /* indicate successful termination */
33
34  } /* end main */

```

```

Enter an integer ( 9999 to end ): 1
Enter next integer ( 9999 to end ): 2
Enter next integer ( 9999 to end ): 3
Enter next integer ( 9999 to end ): 4
Enter next integer ( 9999 to end ): 5
Enter next integer ( 9999 to end ): 6
Enter next integer ( 9999 to end ): 9999

```

```
The average is: 3.50
```

4.11 Write a program that finds the smallest of several integers. Assume that the first value read specifies the number of values remaining.

ANS:

```

1  /* Exercise 4.11 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int number; /* number of integers */
7      int value; /* value input by user */
8      int smallest; /* smallest number */
9      int i; /* counter */
10
11     /* prompt user for number of integers */
12     printf( "Enter the number of integers to be processed: " );
13     scanf( "%d", &number );
14
15     /* prompt user for an integer */
16     printf( "Enter an integer: " );
17     scanf( "%d", &smallest );
18
19     /* loop until user has entered all integers */
20     for ( i = 2; i <= number; i++ ) {
21         printf( "Enter next integer: " ); /* get next integer */
22         scanf( "%d", &value );
23

```

```

24     /* if value is smaller than smallest */
25     if ( value < smallest ) {
26         smallest = value;
27     } /* end if */
28
29 } /* end for */
30
31 printf( "\nThe smallest integer is: %d\n", smallest );
32
33 return 0; /* indicate successful termination */
34
35 } /* end main */

```

```

Enter the number of integers to be processed: 5
Enter an integer: 372
Enter next integer: 920
Enter next integer: 73
Enter next integer: 8
Enter next integer: 3433

The smallest integer is: 8

```

- 4.12 Write a program that calculates and prints the sum of the even integers from 2 to 30.
ANS:

```

1  /* Exercise 4.12 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int i;          /* counter */
7      int sum = 0;    /* current sum of integers */
8
9      /* loop through even integers up to 30 */
10     for ( i = 2; i <= 30; i += 2 ) {
11         sum += i; /* add i to sum */
12     } /* end for */
13
14     printf( "Sum of the even integers from 2 to 30 is: %d\n", sum );
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */

```

```

Sum of the even integers from 2 to 30 is: 240

```

- 4.13 Write a program that calculates and prints the product of the odd integers from 1 to 15.
ANS:

```

1  /* Exercise 4.13 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      long i;          /* counter */
7      long product = 1; /* current product */
8

```

```

9    /* loop through odd integers up to 15 */
10   for ( i = 3; i <= 15; i += 2 ) {
11       product *= i; /* update product */
12   } /* end for */
13
14   printf( "Product of the odd integers from 1 to 15 is: %ld\n", product );
15
16   return 0; /* indicate successful termination */
17
18 } /* end main */

```

```
Product of the odd integers from 1 to 15 is: 2027025
```

4.14 The *factorial* function is used frequently in probability problems. The factorial of a positive integer n (written $n!$ and pronounced “ n factorial”) is equal to the product of the positive integers from 1 to n . Write a program that evaluates the factorials of the integers from 1 to 5. Print the results in tabular format. What difficulty might prevent you from calculating the factorial of 20?

ANS:

```

1    /* Exercise 4.14 Solution */
2    #include <stdio.h>
3
4    int main( void )
5    {
6        int i;          /* outer counter */
7        int j;          /* inner counter */
8        int factorial; /* current factorial value */
9
10       printf( "X\tFactorial of X\n" ); /* display table headers */
11
12       /* compute the factorial for 1 to 5 */
13       for ( i = 1; i <= 5; i++ ) {
14           factorial = 1;
15
16           /* calculate factorial of current number */
17           for ( j = 1; j <= i; j++ ) {
18               factorial *= j;
19           } /* end inner for */
20
21           printf( "%d\t%d\n", i, factorial );
22       } /* end outer for */
23
24       return 0; /* indicate successful termination */
25
26 } /* end main */

```

X	Factorial of X
1	1
2	2
3	6
4	24
5	120

4.15 Modify the compound interest program of Section 4.6 to repeat its steps for interest rates of 5 percent, 6 percent, 7 percent, 8 percent, 9 percent, and 10 percent. Use a `for` loop to vary the interest rate.

ANS:

```
1  /* Exercise 4.15 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  int main( void )
6  {
7      int year;           /* year counter */
8      int rate;          /* interest rate */
9      double amount;     /* amount on deposit */
10     double principal = 1000.0; /* starting principal */
11
12     /* loop through interest rates 5% to 10% */
13     for ( rate = 5; rate <= 10; rate++ ) {
14
15         /* display table headers */
16         printf( "Interest Rate: %f\n", rate / 100.0 );
17         printf( "%s%21s\n", "Year", "Amount on deposit" );
18
19         /* calculate amount on deposit for each of ten years */
20         for ( year = 1; year <= 10; year++ ) {
21
22             /* calculate new amount for specified year */
23             amount = principal * pow( 1 + ( rate / 100.0 ), year );
24
25             /* output one table row */
26             printf( "%4d%21.2f\n", year, amount );
27         } /* end for */
28
29         printf( "\n" );
30     } /* end for */
31
32     return 0; /* indicate successful termination */
33
34 } /* end main */
```

```

Interest Rate: 0.050000
Year   Amount on deposit
1      1050.00
2      1102.50
3      1157.63
4      1215.51
5      1276.28
6      1340.10
7      1407.10
8      1477.46
9      1551.33
10     1628.89

```

...

```

Interest Rate: 0.080000
Year   Amount on deposit
1      1080.00
2      1166.40
3      1259.71
4      1360.49
5      1469.33
6      1586.87
7      1713.82
8      1850.93
9      1999.00
10     2158.92

```

...

```

Interest Rate: 0.100000
Year   Amount on deposit
1      1100.00
2      1210.00
3      1331.00
4      1464.10
5      1610.51
6      1771.56
7      1948.72
8      2143.59
9      2357.95
10     2593.74

```

4.16 Write a program that prints the following patterns separately one below the other. Use for loops to generate the patterns. All asterisks (*) should be printed by a single `printf` statement of the form `printf("*");` (this causes the asterisks to print side by side). Hint: The last two patterns require that each line begin with an appropriate number of blanks.]

```

(A)          (B)          (C)          (D)
*            *****    *****    *
**           *****    *****    **
***          *****    *****    ***
****         *****    *****    ****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****
*****       *****    *****    *****

```

ANS:

```

1  /* Exercise 4.16 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int row; /* row counter */

```

```
7   int col; /* column counter */
8   int space; /* spaces counter */
9
10  /* Pattern A, loop 10 times for rows */
11  for ( row = 1; row <= 10; row++ ) {
12
13      /* print row asterisks */
14      for ( col = 1; col <= row; col++ ) {
15          printf( "*" );
16      } /* end for */
17
18      printf( "\n" );
19  } /* end for */
20
21  printf( "\n" );
22
23  /* Pattern B, loop 10 times for rows
24     row counts down to correspond to number of asterisks */
25  for ( row = 10; row >= 1; row-- ) {
26
27      /* print row asterisks */
28      for ( col = 1; col <= row; col++ ) {
29          printf( "*" );
30      } /* end for */
31
32      printf( "\n" );
33  } /* end for */
34
35  printf( "\n" );
36
37  /* Pattern C, loop 10 times for rows
38     row counts down to correspond to number of asterisks */
39  for ( row = 10; row >= 1; row-- ) {
40
41      /* print (10 - row) number of preceding spaces */
42      for ( space = 1; space <= 10 - row; space++ ) {
43          printf( " " );
44      } /* end for */
45
46      /* print row asterisks */
47      for ( col = 1; col <= row; col++ ) {
48          printf( "*" );
49      } /* end for */
50
51      printf( "\n" );
52  } /* end for */
53
54  printf( "\n" );
55
56  /* Pattern D, loop 10 times for rows */
57  for ( row = 1; row <= 10; row++ ) {
58
59      /* print (10 - row) number of preceding spaces */
60      for ( space = 1; space <= 10 - row; space++ ) {
61          printf( " " );
62      } /* end for */
63
64      /* print row asterisks */
65      for ( col = 1; col <= row; col++ ) {
66          printf( "*" );
67      } /* end for */
```

```

68
69     printf( "\n" );
70 } /* end for */
71
72     printf( "\n" );
73
74     return 0; /* indicate successful termination */
75
76 } /* end main */

```

4.17 Collecting money becomes increasingly difficult during periods of recession, so companies may tighten their credit limits to prevent their accounts receivable (money owed to them) from becoming too large. In response to a prolonged recession, one company has cut its customer's credit limits in half. Thus, if a particular customer had a credit limit of \$2000, this customer's credit limit is now \$1000. If a customer had a credit limit of \$5000, this customer's credit limit is now \$2500. Write a program that analyzes the credit status of three customers of this company. For each customer you are given:

- a) The customer's account number
- b) The customer's credit limit before the recession
- c) The customer's current balance (i.e., the amount the customer owes the company).

Your program should calculate and print the new credit limit for each customer and should determine (and print) which customers have current balances that exceed their new credit limits.

ANS:

```

1  /* Exercise 4.17 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int account;    /* current account number */
7      int i;         /* counter */
8      double limit;  /* current credit limit */
9      double balance; /* current balance */
10     double newLimit; /* new credit limit */
11
12     /* loop three times */
13     for ( i = 1; i <= 3; i++ ) {
14
15         /* get account number, credit limit and balance */
16         printf( "\nEnter account, limit, balance: " );
17         scanf( "%d%lf%lf", &account, &limit, &balance );
18
19         newLimit = limit / 2.0; /* calculate new limit */
20         printf( "New credit limit for account %d is %.2f\n", account, newLimit );
21
22         /* if balance is greater than new credit limit */
23         if ( balance > newLimit ) {
24             printf( "Limit exceeded for account %d\n", account );
25         } /* end if */
26
27     } /* end for */
28
29     return 0; /* indicate successful termination */
30
31 } /* end main */

```

```

Enter account, limit, balance: 100 4000.00 2136.87
New credit limit for account 100 is 2000.00
Limit exceeded for account 100

Enter account, limit, balance: 200 10500.00 4927.39
New credit limit for account 200 is 5250.00

Enter account, limit, balance: 300 1000.00 750.00
New credit limit for account 300 is 500.00
Limit exceeded for account 300

```

4.18 One interesting application of computers is drawing graphs and bar charts (sometimes called “histograms”). Write a program that reads five numbers (each between 1 and 30). For each number read, your program should print a line containing that number of adjacent asterisks. For example, if your program reads the number seven, it should print `*****`.

ANS:

```

1  /* Exercise 4.18 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int i;      /* outer counter */
7      int j;      /* inner counter */
8      int number; /* current number */
9
10     printf( "Enter 5 numbers between 1 and 30: " );
11
12     /* loop 5 times */
13     for ( i = 1; i <= 5; i++ ) {
14         scanf( "%d", &number );
15
16         /* print asterisks corresponding to current input */
17         for ( j = 1; j <= number; j++ ) {
18             printf( "*" );
19         } /* end for */
20
21         printf( "\n" );
22     } /* end for */
23
24     return 0; /* indicate successful termination */
25
26 } /* end main */

```

```

Enter 5 numbers between 1 and 30: 28 5 13 24 7
*****
*****
*****
*****
*****

```

4.19 A mail order house sells five different products whose retail prices are shown in the following table:

Product number	Retail price
1	\$ 2.98
2	\$ 4.50
3	\$ 9.98
4	\$ 4.49
5	\$ 6.87

Write a program that reads a series of pairs of numbers as follows:

- a) Product number
- b) Quantity sold for one day

Your program should use a `switch` statement to help determine the retail price for each product. Your program should calculate and display the total retail value of all products sold last week.

ANS:

```

1  /* Exercise 4.19 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int product;          /* current product number */
7      int quantity;        /* quantity of current product sold */
8      double total = 0.0; /* current total retail value */
9
10     /* prompt for input */
11     printf( "Enter pairs of item numbers and quantities.\n" );
12     printf( "Enter -1 for the item number to end input.\n" );
13     scanf( "%d", &product );
14
15     /* loop while sentinel value not read from user */
16     while ( product != -1 ) {
17         scanf( "%d", &quantity );
18
19         /* determine product number and corresponding retail price */
20         switch ( product ) {
21
22             case 1:
23                 total += quantity * 2.98; /* update total */
24                 break;
25
26             case 2:
27                 total += quantity * 4.50; /* update total */
28                 break;
29
30             case 3:
31                 total += quantity * 9.98; /* update total */
32                 break;
33
34             case 4:
35                 total += quantity * 4.49; /* update total */
36                 break;
37
38             case 5:
39                 total += quantity * 6.87; /* update total */
40                 break;

```

```

41
42     default:
43         printf( "Invalid product code: %d\n", product );
44         printf( "                Quantity: %d\n", quantity );
45     } /* end switch */
46
47     scanf( "%d", &product ); /* get next input */
48 } /* end while */
49
50 /* display total retail value */
51 printf( "The total retail value was: %.2f\n", total );
52
53 return 0; /* indicate successful termination */
54
55 } /* end main */

```

```

Enter pairs of item numbers and quantities.
Enter -1 for the item number to end input.
1 1
2 1
3 1
4 1
5 1
6 1
Invalid product code: 6
                Quantity: 1
1 1
-1
The total retail value was: 31.80

```

- 4.20 Complete the following truth tables by filling in each blank with 0 or 1
ANS: .

Condition1	Condition2	Condition1 && Condition2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Condition1	Condition2	Condition1 Condition2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Condition1	!Condition1
0	1
nonzero	0

4.21 Rewrite the program of Fig. 4.2 so that the initialization of the variable `counter` is done in the definition instead of the `for` statement.

ANS:

```

1  /* Exercise 4.21 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int counter = 1; /* initialize counter */
7
8      /* leave first statement empty */
9      for ( ; counter <= 10; counter++ ) {
10         printf( "%d\n", counter );
11     } /* end for */
12
13     return 0; /* indicate successful termination */
14
15 } /* end main */

```

```

1
2
3
4
5
6
7
8
9
10

```

4.22 Modify the program of Fig. 4.7 so that it calculates the average grade for the class.

ANS:

```

1  /* Exercise 4.22 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int grade;          /* current grade */
7      int aCount = 0;     /* total a grades */
8      int bCount = 0;     /* total b grades */
9      int cCount = 0;     /* total c grades */
10     int dCount = 0;     /* total d grades */
11     int fCount = 0;     /* total f grades */
12     double averageGrade; /* average grade for class */
13
14     /* prompt user for grades */
15     printf( "Enter the letter grades.\n" );
16     printf( "Enter the EOF character to end input.\n" );
17
18     /* loop while not end of file */
19     while ( ( grade = getchar() ) != EOF ) {
20
21         /* determine which grade was input */
22         switch ( grade ) {
23
24             case 'A': /* grade was uppercase A */
25             case 'a': /* grade was lowercase a */
26                 ++aCount; /* update grade A counter */
27                 break; /* exit switch */

```



```

28
29     case 'B': /* grade was uppercase B */
30     case 'b': /* grade was lowercase b */
31         ++bCount; /* update grade B counter */
32         break; /* exit switch */
33
34     case 'C': /* grade was uppercase C */
35     case 'c': /* grade was lowercase c */
36         ++cCount; /* update grade C counter */
37         break; /* exit switch */
38
39     case 'D': /* grade was uppercase C */
40     case 'd': /* grade was lowercase c */
41         ++dCount; /* update grade C counter */
42         break; /* exit switch */
43
44     case 'F': /* grade was uppercase C */
45     case 'f': /* grade was lowercase c */
46         ++fCount; /* update grade C counter */
47         break; /* exit switch */
48
49     case '\n': /* ignore newlines, */
50     case '\t': /* tabs, */
51     case ' ': /* and spaces in input */
52         break; /* exit switch */
53
54     default: /* catch all other characters */
55         printf( "Incorrect letter grade entered." );
56         printf( " Enter a new grade.\n" );
57         break; /* optional, will exit switch anyway */
58 } /* end switch */
59
60 } /* end while */
61
62 /* output totals for each grade */
63 printf( "\nThe totals for each letter grade are:\n" );
64 printf( "A: %d\n", aCount );
65 printf( "B: %d\n", bCount );
66 printf( "C: %d\n", cCount );
67 printf( "D: %d\n", dCount );
68 printf( "F: %d\n", fCount );
69
70 /* calculate average grade */
71 averageGrade =
72     ( 4 * aCount + 3 * bCount + 2 * cCount + dCount ) /
73     ( aCount + bCount + cCount + dCount + fCount );
74
75 /* output appropriate message for average grade */
76 if ( averageGrade > 3.4 ) {
77     printf( "Average grade is A\n" );
78 } /* end if */
79 else if ( averageGrade > 2.4 ) {
80     printf( "Average grade is B\n" );
81 } /* end else if */
82 else if ( averageGrade > 1.4 ) {
83     printf( "Average grade is C\n" );
84 } /* end else if */
85 else if ( averageGrade > 0.4 ) {
86     printf( "Average grade is D\n" );
87 } /* end else if */

```

```

88     else {
89         printf( "Average grade is F\n" );
90     } /* end else */
91
92     return 0; /* indicate successful termination */
93
94 } /* end main */

```

```

Enter the letter grades.
Enter the EOF character to end input.
A B B B C D F
C C C D D D C B
B A A B C C C
^Z

```

```

The totals for each letter grade are:
A: 3
B: 6
C: 8
D: 4
F: 1
Average grade is C

```

4.23 Modify the program of Fig. 4.6 so that it uses only integers to calculate the compound interest. [*Hint*: Treat all monetary amounts as integral numbers of pennies. Then “break” the result into its dollar portion and cents portion by using the division and remainder operations, respectively. Insert a period.]

ANS:

```

1  /* Exercise 4.23 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  int main( void )
6  {
7      int year;           /* year counter */
8      int amount;        /* amount on deposit, in pennies */
9      int dollars;       /* dollar portion of amount */
10     int cents;         /* cents portion of amount */
11     int principal = 100000; /* starting principal, in pennies ($1000) */
12     double rate = .05; /* interest rate */
13
14     /* display headers for table */
15     printf( "%s%21s\n", "Year", "Amount on deposit" );
16
17     /* loop 10 times */
18     for ( year = 1; year <= 10; year++ ) {
19
20         /* determine new amount (in pennies) */
21         amount = principal * pow( 1.0 + rate, year );
22
23         /* determine cents portion of amount (last two digits) */
24         cents = amount % 100;
25
26         /* determine dollars portion of amount */
27         /* integer division truncates decimal places */
28         dollars = amount / 100;
29
30         /* display year, dollar portion followed by period */
31         printf( "%4d%18d.", year, dollars );
32

```

```

33     /* display cents portion */
34     /* if cents portion only 1 digit, insert 0 */
35     if ( cents < 10 ) {
36         printf("0%d\n", cents);
37     } /* end if */
38     else {
39         printf("%d\n", cents);
40     } /* end else */
41
42 } /* end for */
43
44 return 0; /* indicate successful termination */
45
46 } /* end main */

```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.62
4	1215.50
5	1276.28
6	1340.09
7	1407.10
8	1477.45
9	1551.32
10	1628.89

4.24 Assume $i = 1$, $j = 2$, $k = 3$ and $m = 2$. What does each of the following statements print?

- a) `printf("%d", i == 1);`
ANS: 1
- b) `printf("%d", j == 3);`
ANS: 0
- c) `printf("%d", i >= 1 && j < 4);`
ANS: 1
- d) `printf("%d", m <= 99 && k < m);`
ANS: 0
- e) `printf("%d", j >= i || k == m);`
ANS: 1
- f) `printf("%d", k + m < j || 3 - j >= k);`
ANS: 0
- g) `printf("%d", !m);`
ANS: 0
- h) `printf("%d", !(j - m));`
ANS: 1
- i) `printf("%d", !(k > m));`
ANS: 0
- j) `printf("%d", !(j > k));`
ANS: 1

4.25 Print a table of decimal, binary, octal, and hexadecimal equivalents. If you are not familiar with these number systems, read Appendix E first if you would like to attempt this exercise.

ANS: see Exercise 4.34 Solution

4.26 Calculate the value of π from the infinite series

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Print a table that shows the value of π approximated by one term of this series, by two terms, by three terms, etc. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?

ANS: 3.14 occurs at an accuracy of 627, 3.141 occurs at an accuracy of 2458, 3.1415 occurs at an accuracy around 147,000, and 3.14159 occurs at an accuracy around 319,000.

```
1  /* Exercise 4.26 Solution */
2  #include<stdio.h>
3
4  int main( void )
5  {
6      long double pi = 0.0;    /* approximated value for pi */
7      long double num = 4.0;   /* numerator */
8      long double denom = 1.0; /* denominator of current term */
9      long int loop;          /* loop counter */
10     long int accuracy;      /* number of terms */
11
12     accuracy = 400000; /* set decimal accuracy */
13
14     /* display table headers */
15     printf( "Accuracy set at: %ld\n", accuracy );
16     printf( "term\t\t pi\n" );
17
18     /* loop through each term */
19     for ( loop = 1; loop <= accuracy; loop++ ) {
20
21         /* if odd-numbered term, add current term */
22         if ( loop % 2 != 0 ) {
23             pi += num / denom;
24         } /* end if */
25         else { /* if even-numbered term, subtract current term */
26             pi -= num / denom;
27         } /* end else */
28
29         /* display number of terms and approximated
30          value for pi with 8 digits of precision */
31         printf( "%ld\t\t%Lf\n", loop, pi );
32
33         denom += 2.0; /* update denominator */
34
35     } /* end for */
36
37     return 0; /* indicate successful termination */
38
39 }
```

```

Accuracy set at: 400000
term          pi
1             4.000000
2             2.666667
3             3.466667
4             2.895238
5             3.339683
6             2.976046
...
995          3.142598
996          3.140589
997          3.142596
998          3.140591
999          3.142594
...
399998      3.141590
399999      3.141595
400000      3.141590

```

4.27 (Pythagorean Triples) A right triangle can have sides that are all integers. The set of three integer values for the sides of a right triangle is called a Pythagorean triple. These three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Find all Pythagorean triples for side1, side2, and the hypotenuse all no larger than 500. Use a triple-nested for loop that simply tries all possibilities. This is an example of “brute force” computing. It is not aesthetically pleasing to many people. But there are many reasons why these techniques are important. First, with computing power increasing at such a phenomenal pace, solutions that would have taken years or even centuries of computer time to produce with the technology of just a few years ago can now be produced in hours, minutes or even seconds. Recent microprocessor chips can process a billion instructions per second! Second, as you will learn in more advanced computer science courses, there are large numbers of interesting problems for which there is no known algorithmic approach other than sheer brute force. We investigate many kinds of problem-solving methodologies in this book. We will consider many brute force approaches to various interesting problems.

ANS:

```

1  /* Exercise 4.27 Solution */
2  #include<stdio.h>
3
4  int main( void )
5  {
6      int count = 0;          /* number of triples found */
7      long int side1;        /* side1 value counter */
8      long int side2;        /* side2 value counter */
9      long int hypotenuse;    /* hypotenuse value counter */
10     long int hyptSquared;   /* hypotenuse squared */
11     long int sidesSquared; /* sum of squares of sides */
12
13     /* side1 values range from 1 to 500 */
14     for ( side1 = 1; side1 <= 500; side1++ ) {
15
16         /* side2 values range from current side1 to 500 */
17         for ( side2 = 1; side2 <= 500; side2++ ) {
18
19             /* hypotenuse values range from current side2 to 500 */
20             for ( hypotenuse = 1; hypotenuse <= 500; hypotenuse++ ) {
21
22                 /* calculate square of hypotenuse value */
23                 hyptSquared = hypotenuse * hypotenuse;

```

```
24
25     /* calculate sum of squares of sides */
26     sidesSquared = side1 * side1 + side2 * side2;
27
28     /* if hypotenuse squared = side1 squared + side2 squared,
29     Pythagorean triple */
30     if ( hypotSquared == sidesSquared ) {
31
32         /* display triple */
33         printf( "%ld %ld %ld\n", side1, side2, hypotenuse );
34         ++count; /* update count */
35     } /* end if */
36
37     } /* end for */
38
39     } /* end for */
40
41     } /* end for */
42
43     /* display total number of triples found */
44     printf( "A total of %d triples were found.\n", count );
45
46     return 0; /* indicate successful termination */
47
48 } /* end main */
```

```
3 4 5
4 3 5
5 12 13
6 8 10
7 24 25
8 6 10
...
476 93 485
480 31 481
480 88 488
480 108 492
480 140 500
483 44 485
A total of 772 triples were found.
```

4.28 A company pays its employees as managers (who receive a fixed weekly salary), hourly workers (who receive a fixed hourly wage for up to the first 40 hours they work and “time-and-a-half”—i.e., 1.5 times their hourly wage—for overtime hours worked), commission workers (who receive a \$250 plus 5.7% of their gross weekly sales), or pieceworkers (who receive a fixed amount of money per item for each of the items they produce—each pieceworker in this company works on only one type of item). Write a program to compute the weekly pay for each employee. You do not know the number of employees in advance. Each type of employee has its own pay code: Managers have paycode 1, hourly workers have code 2, commission workers have code 3 and pieceworkers have code 4. Use a `switch` to compute each employee’s pay based on that employee’s paycode. Within the `switch`, prompt the user (i.e., the payroll clerk) to enter the appropriate facts your program needs to calculate each employee’s pay based on that employee’s paycode.

ANS:

```

1  /* Exercise 4.28 Solution */
2  #include<stdio.h>
3
4  int main( void )
5  {
6      int payCode;      /* current employee's pay code */
7      int managers = 0; /* total number of managers */
8      int hWorkers = 0; /* total number of hourly workers */
9      int cWorkers = 0; /* total number of commission workers */
10     int pWorkers = 0; /* total number of pieceworkers */
11     int pieces;      /* current pieceworker's number of pieces */
12     double mSalary; /* manager's salary */
13     double hSalary; /* hourly worker's salary */
14     double cSalary; /* commission worker's salary */
15     double pSalary; /* pieceworker's salary */
16     double hours;   /* total hours worked */
17     double otPay;   /* overtime pay */
18     double otHours; /* overtime hours */
19     double pay;     /* current employee's weekly pay */
20
21     /* prompt for first employee input */
22     printf( "Enter paycode ( -1 to end): " );
23     scanf( "%d", &payCode );
24
25     /* loop while sentinel value not read from user */
26     while ( payCode != -1 ) {
27
28         /* switch to appropriate computation according to pay code */
29         switch ( payCode ) {
30
31             /* pay code 1 corresponds to manager */
32             case 1:
33
34                 /*prompt for weekly salary */
35                 printf( "Manager selected.\n" );
36                 printf( "Enter weekly salary: " );
37                 scanf( "%lf", &mSalary );
38
39                 /* manager's pay is weekly salary */
40                 printf( "The manager's pay is $%.2f\n", mSalary );
41
42                 ++managers; /* update total number of managers */
43                 break; /* exit switch */
44
45             /* pay code 2 corresponds to hourly worker */
46             case 2:
47
48                 /* prompt for hourly salary */
49                 printf( "Hourly worker selected.\n" );

```

```

50     printf( "Enter the hourly salary: " );
51     scanf( "%lf", &hSalary );
52
53     /* prompt for number of hours worked */
54     printf( "Enter the total hours worked: " );
55     scanf( "%lf", &hours );
56
57     /* pay fixed for up to 40 hours, 1.5 for hours over 40 */
58     if ( hours > 40.0 ) {
59
60         /* calculate OT hours and total pay */
61         otHours = hours - 40.0;
62         otPay = hSalary * 1.5 * otHours + hSalary * 40.0;
63
64         printf( "Worker has worked %.1f overtime hours.\n", otHours );
65         printf( "Workers pay is $%.2f\n", otPay );
66     } /* end if */
67     else { /* no overtime */
68         pay = hSalary * hours;
69         printf( "Worker's pay is $%.2f\n", pay );
70     } /* end else */
71
72     ++hWorkers; /* update total number of hourly workers */
73     break; /* exit switch */
74
75     /* pay code 3 corresponds to commission worker */
76     case 3:
77
78         /* prompt for gross weekly sales */
79         printf( "Commission worker selected.\n" );
80         printf( "Enter gross weekly sales: " );
81         scanf( "%lf", &cSalary );
82
83         /* pay $250 plus 5.7% of gross weekly sales */
84         pay = 250.0 + 0.057 * cSalary;
85         printf( "Commission Worker's pay is $%.2f\n", pay );
86
87         ++cWorkers; /* update total number of commission workers */
88         break; /* exit switch */
89
90     /* pay code 4 corresponds to pieceworker */
91     case 4:
92
93         /* prompt for number of pieces */
94         printf( "Piece worker selected.\nEnter number of pieces: " );
95         scanf( "%d", &pieces );
96
97         /* prompt for wage per piece */
98         printf( "Enter wage per piece: " );
99         scanf( "%lf", &pSalary );
100
101         pay = pieces * pSalary; /* compute pay */
102         printf( "Piece Worker's pay is $%.2f\n", pay );
103
104         ++pWorkers; /* update total number of pieceworkers */
105         break; /* exit switch */
106
107     /* default case */
108     default :
109         printf( "Invalid pay code.\n" );
110         break;
111 } /* end switch */

```



```

112
113     /* prompt for next employee input */
114     printf( "\nEnter paycode ( -1 to end ): " );
115     scanf( "%d", &payCode );
116 } /* end while */
117
118 /* display total counts for each type of employee */
119 printf( "\n" );
120 printf( "Total number of managers paid      : %d\n", managers );
121 printf( "Total number of hourly workers paid : %d\n", hWorkers );
122 printf( "Total number of commission workers paid: %d\n", cWorkers );
123 printf( "Total number of piece workers paid   : %d\n", pWorkers );
124
125 return 0; /* indicate successful termination */
126
127 } /* end main */

```

```

Enter paycode ( -1 to end): 4
Piece worker selected.
Enter number of pieces: 200
Enter wage per piece: 20
Piece Worker's pay is $4000.00

```

```

Enter paycode ( -1 to end ): -1

```

```

Total number of managers paid      : 0
Total number of hourly workers paid : 0
Total number of commission workers paid: 0
Total number of piece workers paid   : 1

```

```

Enter paycode ( -1 to end): 1
Manager selected.
Enter weekly salary: 2500
The manager's pay is $2500.00

```

```

Enter paycode ( -1 to end ): 2
Hourly worker selected.
Enter the hourly salary: 10.50
Enter the total hours worked: 75
Worker has worked 35.0 overtime hours.
Workers pay is $971.25

```

```

Enter paycode ( -1 to end ): 3
Commission worker selected.
Enter gross weekly sales: 9000
Commission Worker's pay is $763.00

```

```

Enter paycode ( -1 to end ): 4
Piece worker selected.
Enter number of pieces: 200
Enter wage per piece: 20
Piece Worker's pay is $4000.00

```

```

Enter paycode ( -1 to end ): -1

```

```

Total number of managers paid      : 1
Total number of hourly workers paid : 1
Total number of commission workers paid: 1
Total number of piece workers paid   : 1

```

4.29 (*De Morgan's Laws*) In this chapter, we discussed the logical operators `&&`, `||`, and `!`. De Morgan's Laws can sometimes make it more convenient for us to express a logical expression. These laws state that the expression `!(condition1 && condition2)` is logically equivalent to the expression `!(condition1 || !condition2)`. Also, the expression `!(condition1 || condition2)` is logically equivalent to the expression `!(condition1 && !condition2)`. Use De Morgan's Laws to write equivalent expressions for each of the following, and then write a program to show that both the original expression and the new expression in each case are equivalent.

- a) `!(x < 5) && !(y >= 7)`
- b) `!(a == b) || !(g != 5)`
- c) `!((x <= 8) && (y > 4))`
- d) `!((i > 4) || (j <= 6))`

ANS:

```

1  /* Exercise 4.29 Solution */
2  #include<stdio.h>
3
4  int main( void )
5  {
6      int x = 10; /* define current variable value */
7      int y = 1;  /* define current variable value */
8      int a = 3;  /* define current variable value */
9      int b = 3;  /* define current variable value */
10     int g = 5;  /* define current variable value */
11     int Y = 1;  /* define current variable value */
12     int i = 2;  /* define current variable value */
13     int j = 9;  /* define current variable value */
14
15     /* display variable values */
16     printf( "current variable values are: \n" );
17     printf( "x = %d, y = %d, a = %d,", x, y, a );
18     printf( " b = %d\n", b );
19     printf( "g = %d, Y = %d, i = %d,", g, Y, i );
20     printf( " j = %d\n\n", j );
21
22     /* part a */
23     if ( ( !( x < 5 ) && !( y >= 7 ) ) == ( !( ( x < 5 ) || ( y >= 7 ) ) ) ) {
24         printf( "!( x < 5 ) && !( y >= 7 ) is equivalent to"
25                " !( ( x < 5 ) || ( y >= 7 ) )\n" );
26     } /* end if */
27     else {
28         printf( "!( x < 5 ) && !( y >= 7 ) is not equivalent to"
29                " !( ( x < 5 ) || ( y >= 7 ) )\n" );
30     } /* end else */
31
32     /* part b */
33     if ( ( !( a == b ) || !( g != 5 ) ) == ( !( ( a == b ) && ( g != 5 ) ) ) ) {
34         printf( "!( a == b ) || !( g != 5 ) is equivalent to"
35                " !( ( a == b ) && ( g != 5 ) )\n" );
36     } /* end if */
37     else {
38         printf( "!( a == b ) || !( g != 5 ) is not equivalent to"
39                " !( ( a == b ) && ( g != 5 ) )\n" );
40     } /* end else */
41
42     /* part c */
43     if ( !( ( x <= 8 ) && ( Y > 4 ) ) == ( !( x <= 8 ) || !( Y > 4 ) ) ) {
44         printf( "!( ( x <= 8 ) && ( Y > 4 ) ) is equivalent to"
45                " ( !( x <= 8 ) || !( Y > 4 ) )\n" );
46     } /* end if */
47
48     } /* end if */
49

```

```

50     else {
51         printf( "!( ( x <= 8 ) && ( Y > 4 ) ) is not equivalent to"
52              " ( !( x <= 8 ) || !( Y > 4 ) )\n" );
53     } /* end else */
54
55     /* part d */
56     if ( !( ( i > 4 ) || ( j <= 6 ) ) == ( !( i > 4 ) && !( j <= 6 )
57         ) ) {
58         printf( "!( ( i > 4 ) || ( j <= 6 ) ) is equivalent to"
59              " ( !( i > 4 ) && !( j <= 6 ) )\n" );
60     } /* end if */
61     else {
62         printf( "!( ( i > 4 ) || ( j <= 6 ) ) is not equivalent to"
63              " ( !( i > 4 ) && !( j <= 6 ) )\n" );
64     } /* end else */
65
66     return 0; /* indicate successful termination */
67
68 } /* end main */

```

current variable values are:

x = 10, y = 1, a = 3, b = 3
g = 5, Y = 1, i = 2, j = 9

```

!( x < 5 ) && !( y >= 7 ) is equivalent to !( ( x < 5 ) || ( y >= 7 ) )
!( a == b ) || !( g != 5 ) is equivalent to !( ( a == b ) && ( g != 5 ) )
!( ( x <= 8 ) && ( Y > 4 ) ) is equivalent to ( !( x <= 8 ) || !( Y > 4 ) )
!( ( i > 4 ) || ( j <= 6 ) ) is equivalent to ( !( i > 4 ) && !( j <= 6 ) )

```

4.30 Rewrite the program of Fig. 4.7 by replacing the switch statement with a nested if...else statement; be careful to deal with the default case properly. Then rewrite this new version by replacing the nested if...else statement with a series of if statements; here, too, be careful to deal with the default case properly (this is more difficult than in the nested if...else version). This exercise demonstrates that switch is a convenience and that any switch statement can be written with only single-selection statements.

```

1  /* Exercise 4.30 Part A Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int grade; /* current grade */
7      int aCount = 0; /* total A grades */
8      int bCount = 0; /* total B grades */
9      int cCount = 0; /* total C grades */
10     int dCount = 0; /* total D grades */
11     int fCount = 0; /* total F grades */
12
13     /* prompt user for grades */
14     printf( "Enter the letter grades." );
15     printf( " Enter the EOF character to end input:\n" );
16
17     /* while EOF not entered by user */
18     while ( ( grade = getchar() ) != EOF ) {
19
20         /* Update count for appropriate grade */
21         if ( grade == 'A' || grade == 'a' ) {
22             ++aCount;
23         } /* end if */

```

```

24     else if ( grade == 'B' || grade == 'b' ) {
25         ++bCount;
26     } /* end else if */
27     else if ( grade == 'C' || grade == 'c' ) {
28         ++cCount;
29     } /* end else if */
30     else if ( grade == 'D' || grade == 'd' ) {
31         ++dCount;
32     } /* end else if */
33     else if ( grade == 'F' || grade == 'f' ) {
34         ++fCount;
35     } /* end else if */
36     else if ( grade == '\n' || grade == ' ' ) {
37         ; /* empty body */
38     } /* end else if */
39     else {
40         printf( "Incorrect letter grade entered." );
41         printf( " Enter a new grade.\n" );
42     } /* end else */
43
44 } /* end while */
45
46 /* display totals for each grade */
47 printf( "\nTotals for each letter grade were:\n" );
48 printf( "A: %d\n", aCount );
49 printf( "B: %d\n", bCount );
50 printf( "C: %d\n", cCount );
51 printf( "D: %d\n", dCount );
52 printf( "F: %d\n", fCount );
53
54 return 0; /* indicate successful termination */
55
56 } /* end main */

```

Enter the letter grades. Enter the EOF character to end input:

```

A
c
b
d
e
Incorrect letter grade entered. Enter a new grade.
f
^Z

```

Totals for each letter grade were:

```

A: 1
B: 1
C: 1
D: 1
F: 1

```

```

1  /* Exercise 4.30 Part B Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int grade; /* current grade */
7      int aCount = 0; /* total A grades */
8      int bCount = 0; /* total B grades */
9      int cCount = 0; /* total C grades */
10     int dCount = 0; /* total D grades */
11     int fCount = 0; /* total F grades */

```

```
12
13  /* prompt user for grades */
14  printf( "Enter the letter grades." );
15  printf( " Enter the EOF character to end input:\n" );
16
17  /* while EOF not entered by user */
18  while ( ( grade = getchar() ) != EOF ) {
19
20      /* update count for appropriate grade */
21      if ( grade == 'A' || grade == 'a' ) {
22          ++aCount;
23      } /* end if */
24
25      if ( grade == 'B' || grade == 'b' ) {
26          ++bCount;
27      } /* end if */
28
29      if ( grade == 'C' || grade == 'c' ) {
30          ++cCount;
31      } /* end if */
32
33      if ( grade == 'D' || grade == 'd' ) {
34          ++dCount;
35      } /* end if */
36
37      if ( grade == 'F' || grade == 'f' ) {
38          ++fCount;
39      } /* end if */
40
41      if ( grade == '\n' || grade == ' ' ) {
42          ; /* empty body */
43      } /* end if */
44
45      /* default */
46      if ( grade != 'a' && grade != 'A' &&
47          grade != 'B' && grade != 'b' &&
48          grade != 'C' && grade != 'C' &&
49          grade != 'd' && grade != 'd' &&
50          grade != 'f' && grade != 'F' &&
51          grade != '\n' && grade != ' ' ) {
52
53          printf( "Incorrect letter grade entered." );
54          printf( " Enter a new grade.\n" );
55      } /* end if */
56
57  } /* end while */
58
59  /* display totals for each grade */
60  printf( "\nTotals for each letter grade were:\n" );
61  printf( "A: %d\n", aCount );
62  printf( "B: %d\n", bCount );
63  printf( "C: %d\n", cCount );
64  printf( "D: %d\n", dCount );
65  printf( "F: %d\n", fCount );
66
67  return 0; /* indicate successful termination */
68
69 } /* end main */
```

```

Enter the letter grades. Enter the EOF character to end input:
A
b
c
s
Incorrect letter grade entered. Enter a new grade.
d
f
^Z

Totals for each letter grade were:
A: 1
B: 1
C: 1
D: 1
F: 1

```

4.31 Write a program that prints the following diamond shape. You may use `printf` statements that print either a single asterisk (*) or a single blank. Maximize your use of repetition (with nested `for` statements) and minimize the number of `printf` statements.

```

  *
 ***
*****
*****
*****
 *****
  ***
   *

```

ANS:

```

1  /* Exercise 4.31 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int line;      /* line counter */
7      int space;    /* space counter */
8      int asterisk; /* asterisk counter */
9
10     /* top half */
11     for ( line = 1; line <= 9; line += 2 ) {
12
13         /* print preceding spaces */
14         for ( space = ( 9 - line ) / 2; space > 0; space-- ) {
15             printf( " " );
16         } /* end for */
17
18         /* print asterisks */
19         for ( asterisk = 1; asterisk <= line; asterisk++ ) {
20             printf( "*" );
21         } /* end for */
22
23         printf( "\n" );
24     } /* end for */
25
26     /* bottom half */
27     for ( line = 7; line >= 0; line -= 2 ) {
28
29         /* print preceding spaces */
30         for ( space = ( 9 - line ) / 2; space > 0; space-- ) {
31             printf( " " );
32         } /* end for */

```

```

33
34     /* print asterisks */
35     for ( asterisk = 1; asterisk <= line; asterisk++ ) {
36         printf( "*" );
37     } /* end for */
38
39     printf( "\n" );
40 } /* end for */
41
42 return 0; /* indicate successful termination */
43
44 } /* end main */

```

4.32 Modify the program you wrote in Exercise 4.31 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your program should then display a diamond of the appropriate size.

ANS:

```

1  /* Exercise 4.32 Solution */
2  #include<stdio.h>
3
4  int main( void )
5  {
6      int line;     /* line counter */
7      int space;   /* space counter */
8      int asterisk; /* asterisk counter */
9      int size;    /* number of rows in diamond */
10
11     /* prompt for diamond size */
12     printf( "Enter an odd number for the diamond size ( 1-19 ):\n" );
13     scanf( "%d", &size );
14
15     /* top half */
16     for ( line = 1; line <= size - 2; line += 2 ) {
17
18         /* print preceding spaces */
19         for ( space = ( size - line ) / 2; space > 0; space-- ) {
20             printf( " " );
21         } /* end for */
22
23         /* print asterisks */
24         for ( asterisk = 1; asterisk <= line; asterisk++ ) {
25             printf( "*" );
26         } /* end for */
27
28         printf( "\n" );
29     } /* end for */
30
31     /* bottom half */
32     for ( line = size; line >= 0; line -= 2 ) {
33
34         /* print preceding spaces */
35         for ( space = ( size - line ) / 2; space > 0; space-- ) {
36             printf( " " );
37         } /* end for */
38
39         /* print asterisks */
40         for ( asterisk = 1; asterisk <= line; asterisk++ ) {
41             printf( "*" );
42         } /* end for */

```

```

43
44     printf( "\n" );
45 } /* end for */
46
47     return 0; /* indicate successful termination */
48
49 } /* end main */

```

Enter an odd number for the diamond size (1-19):

```

13
   *
  ***
 *****
 *******
 *****
  ***
   *

```

- 4.33** Write a program that prints a table of all the Roman numeral equivalents of the decimal numbers in the range 1 to 100.
ANS:

```

1  /* Exercise 4.33 Solution */
2  #include<stdio.h>
3
4  int main( void )
5  {
6      int loop; /* loop counter */
7      int div; /* tens digit */
8      int mod; /* ones digit */
9
10     /* display table headers */
11     printf( " Roman\nNumeral\t\tDecimal\n" );
12
13     /* loop 100 times */
14     for ( loop = 1; loop <= 100; loop++ ) {
15         div = loop / 10; /* separate tens digit */
16         mod = loop % 10; /* separate ones digit */
17
18         /* switch structure for tens digit */
19         switch ( div ) {
20
21             /* print appropriate Roman numeral for tens digit */
22             case 0:
23                 break;
24
25             case 1:
26                 printf( "X" );
27                 break; /* exit switch */
28
29             case 2:
30                 printf( "XX" );
31                 break; /* exit switch */
32
33             case 3:
34                 printf( "XXX" );
35                 break; /* exit switch */

```



```
36
37     case 4:
38         printf( "XL" );
39         break; /* exit switch */
40
41     case 5:
42         printf( "L" );
43         break; /* exit switch */
44
45     case 6:
46         printf( "LX" );
47         break; /* exit switch */
48
49     case 7:
50         printf( "LXX" );
51         break; /* exit switch */
52
53     case 8:
54         printf( "LXXX" );
55         break; /* exit switch */
56
57     case 9:
58         printf( "XC" );
59         break; /* exit switch */
60
61     case 10:
62         printf( "C" );
63         break; /* exit switch */
64
65     default:
66         break; /* exit switch */
67 } /* end switch */
68
69 /* switch structure for ones digit */
70 switch( mod ) {
71
72     /* print appropriate Roman numeral for ones digit */
73     case 0:
74         printf( "\t\t%4d\n", div * 10 );
75         break; /* exit switch */
76
77     case 1:
78         printf( "I\t\t%4d\n", div * 10 + mod );
79         break; /* exit switch */
80
81     case 2:
82         printf( "II\t\t%4d\n", div * 10 + mod );
83         break; /* exit switch */
84
85     case 3:
86         printf( "III\t\t%4d\n", div * 10 + mod );
87         break; /* exit switch */
88
89     case 4:
90         printf( "IV\t\t%4d\n", div * 10 + mod );
91         break; /* exit switch */
92
93     case 5:
94         printf( "V\t\t%4d\n", div * 10 + mod );
95         break; /* exit switch */
96
```

```

97     case 6:
98         printf( "VI\t\t%4d\n", div * 10 + mod );
99         break; /* exit switch */
100
101     case 7:
102         printf( "VII\t\t%4d\n", div * 10 + mod );
103         break; /* exit switch */
104
105     case 8:
106         printf( "VIII\t\t%4d\n", div * 10 + mod );
107         break; /* exit switch */
108
109     case 9:
110         printf( "IX\t\t%4d\n", div * 10 + mod );
111         break; /* exit switch */
112
113     case 10:
114         printf( "X\t\t%4d\n", div * 10 + mod );
115         break; /* exit switch */
116
117     default:
118         break; /* exit switch */
119 } /* end switch */
120
121 } /* end for */
122
123 return 0; /* indicate successful termination */
124
125 } /* end main */

```

Roman Numeral	Decimal
I	1
II	2
III	3
IV	4
V	5
VI	6
VII	7
VIII	8
IX	9
X	10
...	
LXXXIX	89
XC	90
XCI	91
XCII	92
XCIII	93
XCIV	94
XCV	95
XCVI	96
XCVII	97
XCVIII	98
XCIX	99
C	100

4.34 Write a program that prints a table of the binary, octal and hexadecimal equivalents of the decimal numbers in the range 1 through 256. If you are not familiar with these number systems, read Appendix E before you attempt this exercise.

ANS:

```

1  /* Exercise 4.34 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int loop; /* loop counter */
7      int number; /* current number */
8      int temp1; /* temporary integer */
9
10     /* print table headers */
11     printf( "Decimal\t\tBinary\t\tOctal\t\tHexadecimal\n" );
12
13     /* loop through values 1 to 256 */
14     for ( loop = 1; loop <= 256; loop++ ) {
15         printf( "%d\t\t", loop );
16         number = loop;
17
18         /* binary numbers */
19         printf( "%c", number == 256 ? '1' : '0' );
20
21         printf( "%c", number < 256 && number >= 128 ? '1' : '0' );
22         number %= 128;
23
24         printf( "%c", number < 128 && number >= 64 ? '1' : '0' );
25         number %= 64;
26
27         printf( "%c", number < 64 && number >= 32 ? '1' : '0' );
28         number %= 32;
29
30         printf( "%c", number < 32 && number >= 16 ? '1' : '0' );
31         number %= 16;
32
33         printf( "%c", number < 16 && number >= 8 ? '1' : '0' );
34         number %= 8;
35
36         printf( "%c", number < 8 && number >= 4 ? '1' : '0' );
37         number %= 4;
38
39         printf( "%c", number < 4 && number >= 2 ? '1' : '0' );
40         number %= 2;
41
42         printf( "%c\t", number == 1 ? '1' : '0' );
43
44         /* octal numbers */
45         number = loop;
46
47         printf( "%d", number < 512 && number >= 64 ? number / 64 : 0 );
48         number %= 64;
49
50         printf( "%d", number < 64 && number >= 8 ? number / 8 : 0 );
51         number %= 8;
52
53         printf( "%d\t\t", number == 0 ? 0 : number );
54
55         /* hexadecimal numbers */
56         number = loop;
57         temp1 = 16;
58

```

```

59     if ( number < 4096 && number >= 256 ) {
60         printf( "%d", number / 256 );
61     } /* end if */
62
63     if ( number < 256 && number >= 16 ) {
64         temp1 = number / 16;
65         number %= 16;
66     } /* end if */
67     else {
68         printf( "0" );
69     } /* end else */
70
71     /* convert to letter if temp1 is above 9 */
72     if ( temp1 <= 9 ) {
73         printf( "%d", temp1 );
74     } /* end if */
75     else if ( temp1 >= 10 && temp1 <= 15 ) {
76         printf( "%c", 'A' + ( temp1 - 10 ) );
77     } /* end else if */
78
79     /* convert to letter if number is above 9 */
80     if ( number <= 9 ) {
81         printf( "%d", number );
82     } /* end if */
83     else if ( number >= 10 && number <= 15 ) {
84         printf( "%c", 'A' + ( number - 10 ) );
85     } /* end else if */
86
87     printf( "\n" );
88 } /* end for */
89
90 return 0; /* indicate successful termination */
91
92 } /* end main */

```

Decimal	Binary	Octal	Hexadecimal
1	00000001	001	01
2	00000010	002	02
3	00000011	003	03
4	00000100	004	04
5	00000101	005	05
6	00000110	006	06
7	00000111	007	07
8	00001000	010	08
9	00001001	011	09
10	00001010	012	0A
...			
250	011111010	372	FA
251	011111011	373	FB
252	011111100	374	FC
253	011111101	375	FD
254	011111110	376	FE
255	011111111	377	FF
256	100000000	400	10F

4.35 Describe the process you would use to replace a `do...while` loop with an equivalent `while` loop. What problem occurs when you try to replace a `while` loop with an equivalent `do...while` loop? Suppose you have been told that you must remove a `while` loop and replace it with a `do...while`. What additional control statement would you need to use and how would you use it to ensure that the resulting program behaves exactly as the original?

ANS: The body of a `do...while` loop becomes the body of a `while` loop, and the contents of the body are repeated before the `while` loop. In a `do...while` loop, the body is executed at least once, whereas execution of the body in a `while` loop depends on the continuation condition.

Replacing a `while` loop with a `do...while` loop requires an `if` selection statement. The `do...while` loop would be the body of the `if` statement and the condition would be the same as the loop continuation condition in the `do...while`.

4.36 Write a program that inputs the year in the range 1994 through 1999 and uses `for`-loop repetition to produce a condensed, neatly printed calendar. Watch out for leap years.

ANS:

```

1  /* Exercise 4.36 Solution */
2  /* This is a simple calender solution, that does */
3  /* not account for the shifting of dates from */
4  /* year to year. */
5
6  #include<stdio.h>
7
8  int main( void )
9  {
10     int year;          /* current year */
11     int leapYear;     /* leap year, 1 = yes, 0 = no */
12     int days;        /* total days in current month */
13     int month;       /* current month */
14     int space;       /* space counter */
15     int dayPosition; /* starting day position of year */
16     int dayNum;      /* counter for days of the month */
17
18     /* loop until input is valid */
19     do {
20         printf( "Enter a calendar year between 1994 and 1999: " );
21         scanf( "%d", &year );
22     } while ( year < 1994 || year > 1999 ); /* end do...while */
23
24     /* determine starting day position */
25     switch ( year ) {
26
27         case 1994:
28             dayPosition = 7;
29             break; /* exit switch */
30
31         case 1995:
32             dayPosition = 1;
33             break; /* exit switch */
34
35         case 1996:
36             dayPosition = 2;
37             break; /* exit switch */
38
39         case 1997:
40             dayPosition = 4;
41             break; /* exit switch */
42
43         case 1998:
44             dayPosition = 5;
45             break; /* exit switch */
46

```

```
47     case 1999:
48         dayPosition = 6;
49         break; /* exit switch */
50     } /* end switch */
51
52     /* check for leap years */
53     if ( year % 400 == 0 ) {
54         leapYear = 1;
55     } /* end if */
56     else if ( year % 4 == 0 && year % 100 != 0 ) {
57         leapYear = 1;
58     } /* end else if */
59     else {
60         leapYear = 0;
61     } /* end else */
62
63     /* loop through months and print calendar */
64     for ( month = 1; month <= 12; month++ ) {
65
66         /* begin with the month */
67         switch ( month ) {
68
69             case 1:
70                 printf( "\n\nJanuary %d\n", year );
71                 days = 31;
72                 break; /* exit switch */
73
74             case 2:
75                 printf( "\n\nFebruary %d\n", year );
76                 days = leapYear == 1 ? 29 : 28;
77                 break; /* exit switch */
78
79             case 3:
80                 printf( "\n\nMarch %d\n", year );
81                 days = 31;
82                 break; /* exit switch */
83
84             case 4:
85                 printf( "\n\nApril %d\n", year );
86                 days = 30;
87                 break; /* exit switch */
88
89             case 5:
90                 printf( "\n\nMay %d\n", year );
91                 days = 31;
92                 break; /* exit switch */
93
94             case 6:
95                 printf( "\n\nJune %d\n", year );
96                 days = 30;
97                 break; /* exit switch */
98
99             case 7:
100                printf( "\n\nJuly %d\n", year );
101                days = 31;
102                break; /* exit switch */
103
104             case 8:
105                printf( "\n\nAugust %d\n", year );
106                days = 31;
107                break; /* exit switch */
```

```
108
109     case 9:
110         printf( "\n\nSeptember %d\n", year );
111         days = 30;
112         break; /* exit switch */
113
114     case 10:
115         printf( "\n\nOctober %d\n", year );
116         days = 31;
117         break; /* exit switch */
118
119     case 11:
120         printf( "\n\nNovember %d\n", year );
121         days = 30;
122         break; /* exit switch */
123
124     case 12:
125         printf( "\n\nDecember %d\n", year );
126         days = 31;
127         break; /* exit switch */
128 } /* end switch */
129
130 printf( " S M T W R F S\n" ); /* print heads */
131
132 /* move to proper space to begin printing month */
133 for ( space = 1; space < dayPosition; space++ ) {
134     printf( " " );
135 } /* end for */
136
137 /* print days of the month */
138 for ( dayNum = 1; dayNum <= days; dayNum++ ) {
139     printf( "%2d ", dayNum );
140
141     /* if end of the week, start a new line */
142     if ( dayPosition % 7 == 0 ) {
143         printf( "\n" );
144         dayPosition = 1; /* reset dayPosition */
145     } /* end if */
146     else {
147         ++dayPosition;
148     } /* end else */
149
150 } /* end for */
151
152 } /* end for */
153
154 return 0; /* indicate successful termination */
155
156 } /* end main */
```

```
Enter a calendar year between 1994 and 1999: 1999
```

```
January 1999
S M T W R F S
      1 2
 3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

February 1999
S M T W R F S
 1 2 3 4 5 6
 7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28

March 1999
S M T W R F S
 1 2 3 4 5 6
 7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
.
.
.
```

4.37 A criticism of the `break` statement and the `continue` statement is that each is unstructured. Actually, `break` statements and `continue` statements can always be replaced by structured statements, although doing so can be awkward. Describe in general how you would remove any `break` statement from a loop in a program and replace that statement with some structured equivalent. [Hint: The `break` statement leaves a loop from within the body of the loop. The other way to leave is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates “early exit because of a ‘break’ condition.”] Use the technique you developed here to remove the `break` statement from the program of Fig. 4.11.

ANS:

```
1  /* Exercise 4.37 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int x;          /* loop counter */
7      int breakOut = 1; /* breakout condition */
8
9      /* test for breakout condition */
10     for ( x = 1; x <= 10 && breakOut == 1; x++ ) {
11
12         /* break out of loop after x = 4 */
13         if ( x == 4 ) {
14             breakOut = -1;
15         } /* end if */
16
17         printf( "%d ", x );
18     } /* end for */
19
20     printf( "\nBroke out of loop at x = %d\n", x );
21
22     return 0; /* indicate successful termination */
23
24 }
```



```
1 2 3 4
Broke out of loop at x = 5
```

4.38 What does the following program segment do?

```
1 for ( i = 1; i <= 5; i++ ) {
2     for ( j = 1; j <= 3; j++ ) {
3         for ( k = 1; k <= 4; k++ )
4             printf( "*" );
5         printf( "\n" );
6     }
7     printf( "\n" );
8 }
```

ANS:

```
****
****
****

****
****
****

****
****
****

****
****
****
```

4.39 Describe in general how you would remove any `continue` statement from a loop in a program and replace that statement with some structured equivalent. Use the technique you developed here to remove the `continue` statement from the program of Fig. 4.12.

ANS:

```
1  /* Exercise 4.39 Solution */
2  #include <stdio.h>
3
4  int main( void )
5  {
6      int x; /* loop counter */
7
8      /* loop 10 times */
9      for ( x = 1; x <= 10; x++ ) {
10
11         /* if x == 5, skip to next iteration */
12         if ( x == 5 ) {
13             ++x;
14         } /* end if */
15
16         printf( "%d ", x );
17     } /* end for */
```

```
18
19     printf( "\nUsed ++x to skip printing the value 5\n" );
20
21     return 0; /* indicate successful termination */
22
23 } /* end main */
```

```
1 2 3 4 6 7 8 9 10
Used ++x to skip printing the value 5
```


5

C Functions: Solutions

SOLUTIONS

5.8 Show the value of `x` after each of the following statements is performed:

a) `x = fabs(7.5);`

ANS: 7.5

b) `x = floor(7.5);`

ANS: 7.0

c) `x = fabs(0.0);`

ANS: 0.0

d) `x = ceil(0.0);`

ANS: 0.0

e) `x = fabs(-6.4);`

ANS: 6.4

f) `x = ceil(-6.4);`

ANS: -6.0

g) `x = ceil(-fabs(-8 + floor(-5.5)));`

ANS: -14.0

5.9 A parking garage charges a \$2.00 minimum fee to park for up to three hours. The garage charges an additional \$0.50 per hour for each hour *or part thereof* in excess of three hours. The maximum charge for any given 24-hour period is \$10.00. Assume that no car parks for longer than 24 hours at a time. Write a program that will calculate and print the parking charges for each of 3 customers who parked their cars in this garage yesterday. You should enter the hours parked for each customer. Your program should print the results in a neat tabular format, and should calculate and print the total of yesterday's receipts. The program should use the function `calculateCharges` to determine the charge for each customer. Your outputs should appear in the following format:

```
Enter the hours parked for 3 cars: 1.5 4.0 24.0
Car      Hours      Charge
  1       1.5       2.00
  2       4.0       2.50
  3      24.0      10.00
TOTAL    29.5      14.50
```

ANS:

```

1  /* Exercise 5.9 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  double calculateCharges( double hours ); /* function prototype */
6
7  int main()
8  {
9      double h; /* number of hours for current car */
10     double currentCharge; /* parking charge for current car */
11     double totalCharges = 0.0; /* total charges */
12     double totalHours = 0.0; /* total number of hours */
13     int i; /* loop counter */
14     int first = 1; /* flag for printing table headers */
15
16     printf( "Enter the hours parked for 3 cars: " );
17
18     /* loop 3 times for 3 cars */
19     for ( i = 1; i <= 3; i++ ) {
20         scanf( "%lf", &h );
21         totalHours += h; /* add current hours to total hours */
22
23         /* if first time through loop, display headers */
24         if ( first ) {
25             printf( "%5s%15s%15s\n", "Car", "Hours", "Charge" );
26
27             /* set flag to false to prevent from printing again */
28             first = 0;
29         } /* end if */
30
31         /* calculate current car's charge and update total */
32         totalCharges += ( currentCharge = calculateCharges( h ) );
33
34         /* display row data for current car */
35         printf( "%5d%15.1f%15.2f\n", i, h, currentCharge );
36     } /* end for */
37
38     /* display row data for totals */
39     printf( "%5s%15.1f%15.2f\n", "TOTAL", totalHours, totalCharges );
40
41     return 0; /* indicate successful termination */
42 }
43 } /* end main */
44
45 /* calculateCharges returns charge according to number of hours */
46 double calculateCharges( double hours )
47 {
48     double charge; /* calculated charge */
49
50     /* $2 for up to 3 hours */
51     if ( hours < 3.0 ) {
52         charge = 2.0;
53     } /* end if */
54
55     /* $.50 for each hour or part thereof in excess of 3 hours */
56     else if ( hours < 19.0 ) {
57         charge = 2.0 + .5 * ceil( hours - 3.0 );
58     } /* end else if */
59     else { /* maximum charge $10 */

```

```

60     charge = 10.0;
61     } /* end else */
62
63     return charge; /* return calculated charge */
64
65 } /* end function calculateCharges */

```

5.10 An application of function `floor` is rounding a value to the nearest integer. The statement

$$y = \text{floor}(x + .5);$$

will round the number `x` to the nearest integer, and assign the result to `y`. Write a program that reads several numbers and uses the preceding statement to round each of these numbers to the nearest integer. For each number processed, print both the original number and the rounded number.

ANS:

```

1  /* Exercise 5.10 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  void calculateFloor( void ); /* function prototype */
6
7  int main()
8  {
9      calculateFloor(); /* call function calculateFloor */
10
11     return 0; /* indicate successful termination */
12
13 } /* end main */
14
15 /* calculateFloor rounds 5 inputs */
16 void calculateFloor( void )
17 {
18     double x; /* current input */
19     double y; /* current input rounded */
20     int loop; /* loop counter */
21
22     /* loop for 5 inputs */
23     for ( loop = 1; loop <= 5; loop++ ) {
24         printf( "Enter a floating point value: " );
25         scanf( "%lf", &x );
26
27         /* y holds rounded input */
28         y = floor( x + .5 );
29         printf( "%f rounded is %.1f\n\n", x, y );
30     } /* end for */
31
32 } /* end function calculateFloor */

```

```

Enter a floating point value: 1.5
1.500000 rounded is 2.0

Enter a floating point value: 5.55
5.550000 rounded is 6.0

Enter a floating point value: 73.2341231432
73.234123 rounded is 73.0

Enter a floating point value: 9.0
9.000000 rounded is 9.0

Enter a floating point value: 4
4.000000 rounded is 4.0

```

5.11 Function `floor` may be used to round a number to a specific decimal place. The statement

$$y = \text{floor}(x * 10 + .5) / 10;$$

rounds `x` to the tenths position (the first position to the right of the decimal point). The statement

$$y = \text{floor}(x * 100 + .5) / 100;$$

rounds `x` to the hundredths position (i.e., the second position to the right of the decimal point). Write a program that defines four functions to round a number `x` in various ways

- a) `roundToInteger(number)`
- b) `roundToTenths(number)`
- c) `roundToHundredths(number)`
- d) `roundToThousandths(number)`

For each value read, your program should print the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth, and the number rounded to the nearest thousandth.

ANS:

```

1  /* Exercise 5.11 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  double roundToInteger( double n );    /* function prototype */
6  double roundToTenths( double n );    /* function prototype */
7  double roundToHundredths( double n ); /* function prototype */
8  double roundToThousandths( double n ); /* function prototype */
9
10 int main()
11 {
12     int i;          /* loop counter */
13     int count;     /* number of values to process */
14     double number; /* current input */
15
16     printf( "How many numbers do you want to process? " );
17     scanf( "%d", &count );
18
19     /* loop for inputs */
20     for ( i = 0; i < count; i++ ) {
21         printf( "Enter number: " );
22         scanf( "%lf", &number );
23
24         /* display number rounded to nearest integer */
25         printf( "%f rounded to an integer is %f\n",
26             number, roundToInteger( number ) );

```

```

27
28     /* display number rounded to nearest tenth */
29     printf( "%f rounded to the nearest tenth is %f\n",
30             number, roundToTenths( number ) );
31
32     /* display number rounded to nearest hundredth */
33     printf( "%f rounded to the nearest hundredth is %f\n",
34             number, roundToHundredths( number ) );
35
36     /* display number rounded to nearest thousandth */
37     printf( "%f rounded to the nearest thousandth is %f\n",
38             number, roundToThousandths( number ) );
39 } /* end for */
40
41 return 0; /* indicate successful termination */
42
43 } /* end main */
44
45 /* roundToInteger rounds n to nearest integer */
46 double roundToInteger( double n )
47 {
48     return floor( n + .5 );
49 }
50 /* end function roundToInteger */
51
52 /* roundToTenths rounds n to nearest tenth */
53 double roundToTenths( double n )
54 {
55     return floor( n * 10 + .5 ) / 10;
56 }
57 /* end function roundToTenths */
58
59 /* roundToHundredths rounds n to nearest hundredth */
60 double roundToHundredths( double n )
61 {
62     return floor( n * 100 + .5 ) / 100;
63 }
64 /* end function roundToHundredths */
65
66 /* roundToThousandths rounds n to nearest thousandth */
67 double roundToThousandths( double n )
68 {
69     return floor( n * 1000 + .5 ) / 1000;
70 }
71 /* end function roundToThousandths */

```

```

How many numbers do you want to process? 1
Enter number: 8.54739
8.547390 rounded to an integer is 9.000000
8.547390 rounded to the nearest tenth is 8.500000
8.547390 rounded to the nearest hundredth is 8.550000
8.547390 rounded to the nearest thousandth is 8.547000

```


5.12 Answer each of the following questions.

a) What does it mean to choose numbers “at random?”

ANS: Every number has an equal chance of being chosen at any time.

b) Why is the `rand` function useful for simulating games of chance?

ANS: Because it produces a sequence of pseudo random numbers that when scaled appear to be random.

c) Why would you randomize a program by using `srand`? Under what circumstances is it desirable not to randomize?

ANS: Using `srand` enables the sequence of pseudo random numbers produced by `rand` to change each time the program is executed. The program should not be randomized while in the debugging stages because repetition is helpful in debugging.

d) Why is it often necessary to scale and/or shift the values produced by `rand`?

ANS: To produce random values in a specific range.

e) Why is computerized simulation of real-world situations a useful technique?

ANS: It enables more accurate predictions of random events such as cars arriving at toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at specific times.

5.13 Write statements that assign random integers to the variable n in the following ranges:

a) $1 \leq n \leq 2$

ANS: `n = 1 + rand() % 2;`

b) $1 \leq n \leq 100$

ANS: `n = 1 + rand() % 100;`

c) $0 \leq n \leq 9$

ANS: `n = rand() % 10;`

d) $1000 \leq n \leq 1112$

ANS: `n = 1000 + rand() % 113;`

e) $-1 \leq n \leq 1$

ANS: `n = -1 + rand() % 3;`

f) $-3 \leq n \leq 11$

ANS: `n = -3 + rand() % 15;`

5.14 For each of the following sets of integers, write a single statement that will print a number at random from the set.

a) 2, 4, 6, 8, 10.

ANS: `printf("%d\n", 2 * (1 + rand() % 5));`

b) 3, 5, 7, 9, 11.

ANS: `printf("%d\n", 1 + 2 * (1 + rand() % 5));`

c) 6, 10, 14, 18, 22.

ANS: `printf("%d\n", 6 + 4 * (rand() % 5));`

5.15 Define a function called `hypotenuse` that calculates the length of the hypotenuse of a right triangle when the other two sides are given. Use this function in a program to determine the length of the hypotenuse for each of the following triangles. The function should take two arguments of type `double` and return the hypotenuse as a `double`. Test your program with the side values specified in Fig. 5.18.

Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

```

1  /* Exercise 5.15 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  double hypotenuse( double s1, double s2 ); /* function prototype */
6
7  int main()
8  {
9      int i;          /* loop counter */
10     double side1; /* value for first side */
11     double side2; /* value for second side */
12
13     /* loop 3 times */
14     for ( i = 1; i <= 3; i++ ) {
15         printf( "Enter the sides of the triangle: " );
16         scanf( "%lf%lf", &side1, &side2 );
17
18         /* calculate and display hypotenuse value */
19         printf( "Hypotenuse: %.1f\n\n", hypotenuse( side1, side2 ) );
20     } /* end for */
21
22     return 0; /* indicate successful termination */
23
24 } /* end main */
25
26 /* hypotenuse calculates value of hypotenuse of
27    a right triangle given two side values */
28 double hypotenuse( double s1, double s2 )
29 {
30     return sqrt( pow( s1, 2 ) + pow( s2, 2 ) );
31 } /* end function hypotenuse */

```

```

Enter the sides of the triangle: 3.0 4.0
Hypotenuse: 5.0

```

```

Enter the sides of the triangle: 5.0 12.0
Hypotenuse: 13.0

```

```

Enter the sides of the triangle: 8.0 15.0
Hypotenuse: 17.0

```

5.16 Write a function `integerPower(base, exponent)` that returns the value of

$$\text{base}^{\text{exponent}}$$

For example, `integerPower(3, 4) = 3 * 3 * 3 * 3`. Assume that `exponent` is a positive, nonzero integer, and `base` is an integer. Function `integerPower` should use `for` to control the calculation. Do not use any math library functions.

ANS:

```

1  /* Exercise 5.16 Solution */
2  #include <stdio.h>
3
4  int integerPower( int b, int e );
5
6  int main()
7  {
8      int exp; /* integer exponent */
9      int base; /* integer base */
10
11     printf( "Enter integer base and exponent: " );
12     scanf( "%d%d", &base, &exp );
13
14     printf( "%d to the power %d is: %d\n",
15            base, exp, integerPower( base, exp ) );
16
17     return 0; /* indicate successful termination */
18
19 } /* end main */
20
21 /* integerPower calculates and returns b raised to the e power */
22 int integerPower( int b, int e )
23 {
24     int product = 1; /* resulting product */
25     int i;          /* loop counter */
26
27     /* multiply product times b (e repetitions) */
28     for ( i = 1; i <= e; i++ ) {
29         product *= b;
30     } /* end for */
31
32     return product; /* return resulting product */
33
34 } /* end function integerPower */

```

```

Enter integer base and exponent: 5 3
5 to the power 3 is: 125

```

5.17 Write a function `multiple` that determines for a pair of integers whether the second integer is a multiple of the first. The function should take two integer arguments and return 1 (true) if the second is a multiple of the first, and 0 (false) otherwise. Use this function in a program that inputs a series of pairs of integers.

ANS:

```
1  /* Exercise 5.17 Solution */
2  #include <stdio.h>
3
4  int multiple( int a, int b ); /* function prototype */
5
6  int main()
7  {
8      int x; /* first integer */
9      int y; /* second integer */
10     int i; /* loop counter */
11
12     /* loop 3 times */
13     for ( i = 1; i <= 3; i++ ) {
14         printf( "Enter two integers: " );
15         scanf( "%d%d", &x, &y );
16
17         /* determine if second is multiple of first */
18         if ( multiple( x, y ) ) {
19             printf( "%d is a multiple of %d\n\n", y, x );
20         } /* end if */
21         else {
22             printf( "%d is not a multiple of %d\n\n", y, x );
23         } /* end else */
24
25     } /* end for */
26
27     return 0; /* indicate successful termination */
28
29 } /* end main */
30
31 /* multiple determines if b is multiple of a */
32 int multiple( int a, int b )
33 {
34     return !( b % a );
35
36 } /* end function multiple */
```

```
Enter two integers: 2 10
10 is a multiple of 2

Enter two integers: 5 17
17 is not a multiple of 5

Enter two integers: 3 696
696 is a multiple of 3
```

5.18 Write a program that inputs a series of integers and passes them one at a time to function `even` which uses the remainder operator to determine if an integer is even. The function should take an integer argument and return 1 if the integer is even and 0 otherwise.

ANS:

```
1  /* Exercise 5.18 Solution */
2  #include <stdio.h>
3
4  int even( int a ); /* function prototype */
5
6  int main()
7  {
8      int x; /* current input */
9      int i; /* loop counter */
10
11     /* loop for 3 inputs */
12     for ( i = 1; i <= 3; i++ ) {
13         printf( "Enter an integer: " );
14         scanf( "%d", &x );
15
16         /* determine if input is even */
17         if ( even( x ) ) {
18             printf( "%d is an even integer\n\n", x );
19         } /* end if */
20         else {
21             printf( "%d is not an even integer\n\n", x );
22         } /* end else */
23     } /* end for */
24
25     return 0; /* indicate successful termination */
26 } /* end main */
27
28 /* even returns true if a is even */
29 int even( int a )
30 {
31     return !( a % 2 );
32 } /* end function even */
```

```
Enter an integer: 7
7 is not an even integer
```

```
Enter an integer: 6
6 is an even integer
```

```
Enter an integer: 10000
10000 is an even integer
```

5.19 Write a function that displays at the left margin of the screen a solid square of asterisks whose side is specified in integer parameter `side`. For example, if `side` is 4, the function displays:

```
Enter side: 4
****
****
****
****
```

ANS:

```
1  /* Exercise 5.19 Solution */
2  #include <stdio.h>
3
4  void square( int s ); /* function prototype */
5
6  int main()
7  {
8      int side; /* input side length */
9
10     printf( "Enter side: " );
11     scanf( "%d", &side );
12
13     square( side ); /* display solid square of asterisks */
14
15     return 0; /* indicate successful termination */
16
17 } /* end main */
18
19 /* square displays solid square of asterisks with specified side */
20 void square( int s )
21 {
22     int i; /* outer loop counter */
23     int j; /* inner loop counter */
24
25     /* loop side times for number of rows */
26     for ( i = 1; i <= s; i++ ) {
27
28         /* loop side times for number of columns */
29         for ( j = 1; j <= s; j++ ) {
30             printf( "*" );
31         } /* end for */
32
33         printf( "\n" );
34     } /* end for */
35
36 } /* end function square */
```

5.20 Modify the function created in Exercise 5.19 to form the square out of whatever character is contained in character parameter `fillCharacter`. Thus if `side` is 5 and `fillCharacter` is “#” then this function should print:

```
Enter a character and the side length: # 5
#####
#####
#####
#####
#####
```

ANS:

```
1  /* Exercise 5.20 Solution */
2  #include <stdio.h>
3
4  void square( int side, char fillCharacter ); /* function prototype */
5
6  int main()
7  {
8      int s; /* side length */
9      char c; /* fill character */
10
11     printf( "Enter a character and the side length: " );
12     scanf( "%c%d", &c, &s );
13
14     square( s, c ); /* display solid square of input character */
15
16     return 0; /* indicate successful termination */
17 }
18 /* end main */
19
20 /* square displays solid square of fillCharacter with specified side */
21 void square( int side, char fillCharacter )
22 {
23     int loop; /* outer loop counter */
24     int loop2; /* inner loop counter */
25
26     /* loop side times for number of rows */
27     for ( loop = 1; loop <= side; loop++ ) {
28
29         /* loop side times for number of columns */
30         for ( loop2 = 1; loop2 <= side; loop2++ ) {
31             printf( "%c", fillCharacter );
32         } /* end for */
33
34         printf( "\n" );
35     } /* end for */
36 }
37 /* end function square */
```

5.21 Use techniques similar to those developed in Exercises 5.19 and 5.20 to produce a program that graphs a wide range of shapes.

- 5.22 Write program segments that accomplish each of the following:
- Calculate the integer part of the quotient when integer a is divided by integer b.
 - Calculate the integer remainder when integer a is divided by integer b.
 - Use the program pieces developed in a) and b) to write a function that inputs an integer between 1 and 32767 and prints it as a series of digits, each pair of which is separated by two spaces. For example, the integer 4562 should be printed as:

```
4 5 6 2
```

ANS:

```

1  /* Exercise 5.22 Solution */
2  #include <stdio.h>
3
4  int quotient( int a, int b ); /* function prototype */
5  int remainder( int a, int b ); /* function prototype */
6
7  int main()
8  {
9      int number;          /* input number */
10     int divisor = 10000; /* current divisor */
11
12     printf( "Enter an integer between 1 and 32767: " );
13     scanf( "%d", &number );
14
15     printf( "The digits in the number are:\n" );
16
17     /* determine and print each digit */
18     while ( number >= 10 ) {
19
20         /* if number is >= current divisor, determine digit */
21         if ( number >= divisor ) {
22
23             /* use quotient to determine current digit */
24             printf( "%d ", quotient( number, divisor ) );
25
26             /* update number to be remainder */
27             number = remainder( number, divisor );
28
29             /* update divisor for next digit */
30             divisor = quotient( divisor, 10 );
31         } /* end if */
32         else { /* if number < current divisor, no digit */
33             divisor = quotient( divisor, 10 );
34         } /* end else */
35
36     } /* end while */
37
38     printf( "%d\n", number );
39
40     return 0; /* indicate successful termination */
41
42 } /* end main */
43
44 /* Part A: determine quotient using integer division */
45 int quotient( int a, int b )
46 {
47     return a / b;
48
49 } /* end function quotient */

```

```
50
51 /* Part B: determine remainder using the remainder operator */
52 int remainder( int a, int b )
53 {
54     return a % b;
55 }
56 /* end function remainder */
```

5.23 Write a function that takes the time as three integer arguments (for hours, minutes, and seconds), and returns the number of seconds since the last time the clock “struck 12.” Use this function to calculate the amount of time in seconds between two times, both of which are within one 12-hour cycle of the clock.

ANS:

```

1  /* Exercise 5.23 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  /* function prototype */
6  unsigned seconds( unsigned h, unsigned m, unsigned s );
7
8  int main()
9  {
10     int hours;      /* current time's hours */
11     int minutes;    /* current time's minutes */
12     int secs;       /* current time's seconds */
13     int first;      /* first time, in seconds */
14     int second;     /* second time, in seconds */
15     int difference; /* difference between two times, in seconds */
16
17     printf( "Enter the first time as three integers: " );
18     scanf( "%d%d%d", &hours, &minutes, &secs );
19
20     /* calculate first time in seconds */
21     first = seconds( hours, minutes, secs );
22
23     printf( "Enter the second time as three integers: " );
24     scanf( "%d%d%d", &hours, &minutes, &secs );
25
26     /* calculate second time in seconds */
27     second = seconds( hours, minutes, secs );
28
29     /* calculate difference */
30     difference = fabs( first - second );
31
32     /* display difference */
33     printf( "The difference between the times is %d seconds\n",
34            difference );
35
36     return 0; /* indicate successful termination */
37
38 } /* end main */
39
40 /* seconds returns number of seconds since clock "struck 12"
41    given input time as hours h, minutes m, seconds s */
42 unsigned seconds( unsigned h, unsigned m, unsigned s )
43 {
44     return 3600 * h + 60 * m + s;
45
46 } /* end function seconds */

```

```

Enter the first time as three integers: 4 20 39
Enter the second time as three integers: 7 20 39
The difference between the times is 10800 seconds

```

5.24 Implement the following integer functions:

- Function `celsius` returns the Celsius equivalent of a Fahrenheit temperature.
- Function `fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature.
- Use these functions to write a program that prints charts showing the Fahrenheit equivalents of all Celsius temperatures from 0 to 100 degrees, and the Celsius equivalents of all Fahrenheit temperatures from 32 to 212 degrees. Print the outputs in a neat tabular format that minimizes the number of lines of output while remaining readable.

ANS:

```

1  /* Exercise 5.24 Solution */
2  #include <stdio.h>
3
4  int celsius( int fTemp );    /* function prototype */
5  int fahrenheit( int cTemp ); /* function prototype */
6
7  int main()
8  {
9      int i; /* loop counter */
10
11     /* display table of Fahrenheit equivalents of Celsius temperature */
12     printf( "Fahrenheit equivalents of Celcius temperatures:\n" );
13     printf( "Celcius\t\tFahrenheit\n" );
14
15     /* display Fahrenheit equivalents of Celsius 0 to 100 */
16     for ( i = 0; i <= 100; i++ ) {
17         printf( "%d\t\t%d\n", i, fahrenheit( i ) );
18     } /* end for */
19
20     /* display table of Celsius equivalents of Fahrenheit temperature */
21     printf( "\nCelcius equivalents of Fahrenheit temperatures:\n" );
22     printf( "Fahrenheit\tCelcius\n" );
23
24     /* display Celsius equivalents of Fahrenheit 32 to 212 */
25     for ( i = 32; i <= 212; i++ ) {
26         printf( "%d\t\t%d\n", i, celsius( i ) );
27     } /* end for */
28
29     return 0; /* indicate successful termination */
30
31 } /* end main */
32
33 /* celsius returns Celsius equivalent of fTemp,
34    given in Fahrenheit */
35 int celsius( int fTemp )
36 {
37     return ( int ) ( 5.0 / 9.0 * ( fTemp - 32 ) );
38
39 } /* end function celsius */
40
41 /* fahrenheit returns Fahrenheit equivalent of cTemp,
42    given in Celsius */
43 int fahrenheit( int cTemp )
44 {
45     return ( int ) ( 9.0 / 5.0 * cTemp + 32 );
46
47 } /* end function fahrenheit */

```

Fahrenheit equivalents of Celcius temperatures:

Celcius	Fahrenheit
---------	------------

0	32
---	----

1	33
---	----

2	35
---	----

3	37
---	----

4	39
---	----

5	41
---	----

6	42
---	----

7	44
---	----

8	46
---	----

9	48
---	----

.

.

.

Celcius equivalents of Fahrenheit temperatures:

Fahrenheit	Celcius
------------	---------

32	0
----	---

33	0
----	---

34	1
----	---

35	1
----	---

36	2
----	---

37	2
----	---

38	3
----	---

39	3
----	---

40	4
----	---

41	5
----	---

.

.

.

- 5.25 Write a function that returns the smallest of three floating point numbers.
ANS:

```

1  /* Exercise 5.25 Solution */
2  #include <stdio.h>
3
4  /* function prototype */
5  double smallest3( double a, double b, double c );
6
7  int main()
8  {
9      double x; /* first input */
10     double y; /* second input */
11     double z; /* third input */
12
13     printf( "Enter three doubleing point values: " );
14     scanf( "%lf%lf%lf", &x, &y, &z );
15
16     /* determine smallest value */
17     printf( "The smallest value is %f\n", smallest3( x, y, z ) );
18
19     return 0; /* indicate successful termination */
20
21 } /* end main */
22
23 /* smallest3 returns the smallest of a, b and c */
24 double smallest3( double a, double b, double c )
25 {
26     double smallest = a; /* assume a is the smallest */
27
28     if ( b < smallest ) { /* if b is smaller */
29         smallest = b;
30     } /* end if */
31
32     if ( c < smallest ) { /* if c is smaller */
33         smallest = c;
34     } /* end if */
35
36     return smallest; /* return smallest value */
37
38 } /* end function smallest3 */

```

```

Enter three doubleing point values: 3.3 4.4 5.5
The smallest value is 3.300000

```

```

Enter three doubleing point values: 4.4 5.5 3.3
The smallest value is 3.300000

```

```

Enter three doubleing point values: 4.4 3.3 5.5
The smallest value is 3.300000

```

5.26 An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number because $6 = 1 + 2 + 3$. Write a function `perfect` that determines if parameter `number` is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000. Print the factors of each perfect number to confirm that the number is indeed perfect. Challenge the power of your computer by testing numbers much larger than 1000.

ANS:

```

1  /* Exercise 5.26 Solution */
2  #include <stdio.h>
3
4  int perfect( int value ); /* function prototype */
5
6  int main()
7  {
8      int j; /* loop counter */
9
10     printf( "For the integers from 1 to 1000:\n" );
11
12     /* loop from 2 to 1000 */
13     for ( j = 2; j <= 1000; j++ ) {
14
15         /* if current integer is perfect */
16         if ( perfect( j ) ) {
17             printf( "%d is perfect\n", j );
18         } /* end if */
19
20     } /* end for */
21
22     return 0; /* indicate successful termination */
23
24 } /* end main */
25
26 /* perfect returns true if value is perfect integer,
27    i.e., if value is equal to sum of its factors */
28 int perfect( int value )
29 {
30     int factorSum = 1; /* current sum of factors */
31     int i;           /* loop counter */
32
33     /* loop through possible factor values */
34     for ( i = 2; i <= value / 2; i++ ) {
35
36         /* if i is factor */
37         if ( value % i == 0 ) {
38             factorSum += i; /* add to sum */
39         } /* end if */
40
41     } /* end for */
42
43     /* return true if value is equal to sum of factors */
44     if ( factorSum == value ) {
45         return 1;
46     } /* end if */
47     else {
48         return 0;
49     } /* end else */
50
51 } /* end function perfect */

```

```
For the integers from 1 to 1000:  
6 is perfect  
28 is perfect  
496 is perfect
```

5.27 An integer is said to be *prime* if it is divisible only by 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

- Write a function that determines if a number is prime.
- Use this function in a program that determines and prints all the prime numbers between 1 and 10,000. How many of these 10,000 numbers do you really have to test before being sure that you have found all the primes?
- Initially you might think that $n/2$ is the upper limit for which you must test to see if a number is prime, but you need only go as high as the square root of n . Why? Rewrite the program, and run it both ways. Estimate the performance improvement.

ANS:

```

1  /* Exercise 5.27 Solution Part B Solution */
2  #include <stdio.h>
3
4  int prime( int n );
5
6  int main()
7  {
8      int loop;      /* loop counter */
9      int count = 0; /* total number of primes found */
10
11     printf( "The prime numbers from 1 to 10000 are:\n" );
12
13     /* loop through 1 to 10000 */
14     for ( loop = 1; loop <= 10000; loop++ ) {
15
16         /* if current number is prime */
17         if ( prime( loop ) ) {
18             ++count;
19             printf( "%6d", loop );
20
21             /* new line after 10 values displayed */
22             if ( count % 10 == 0 ) {
23                 printf( "\n" );
24             } /* end if */
25
26         } /* end if */
27
28     } /* end for */
29
30     return 0; /* indicate successful termination */
31 }
32 /* end main */
33
34 /* prime returns 1 if n is prime */
35 int prime( int n )
36 {
37     int loop2; /* loop counter */
38
39     /* loop through possible factors */
40     for ( loop2 = 2; loop2 <= n / 2; loop2++ ) {
41
42         /* if factor found, not prime */
43         if ( n % loop2 == 0 ) {
44             return 0;
45         } /* end if */
46
47     } /* end for */
48
49     return 1; /* return 1 if prime */
50 }
51 /* end function prime */

```



```

The prime numbers from 1 to 10000 are:
  1   2   3   5   7   11  13   17   19   23
 29  31  37  41  43  47  53   59   61   67
 71  73  79  83  89  97  101  103  107  109
113 127 131 137 139 149 151 157 163 167
.
.
.
9733 9739 9743 9749 9767 9769 9781 9787 9791 9803
9811 9817 9829 9833 9839 9851 9857 9859 9871 9883
9887 9901 9907 9923 9929 9931 9941 9949 9967 9973

```

```

1  /* Exercise 5.27 Part C Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  int prime( int n ); /* function prototype */
6
7  int main()
8  {
9      int j;          /* loop counter */
10     int count = 0; /* total number of primes found */
11
12     printf( "The prime numbers from 1 to 10000 are:\n" );
13
14     /* loop through numbers 1 to 10000 */
15     for ( j = 1; j <= 10000; j++ ) {
16
17         /* if current number prime */
18         if ( prime( j ) ) {
19             ++count;
20             printf( "%5d", j );
21
22             /* new line after 10 values displayed */
23             if ( count % 10 == 0 ) {
24                 printf( "\n" );
25             } /* end if */
26
27         } /* end if */
28     } /* end for */
29
30     return 0; /* indicate successful termination */
31 } /* end main */
32
33 /* prime returns 1 if n is prime */
34 int prime( int n )
35 {
36     int i; /* loop counter */
37
38     /* loop through possible factors */
39     for ( i = 2; i <= ( int ) sqrt( n ); i++ ) {
40
41         /* if factor found, not prime */
42         if ( n % i == 0 ) {
43             return 0;
44         } /* end if */
45     } /* end for */
46 }

```

```
49  
50     return 1;  
51  
52 } /* end function prime */
```

5.28 Write a function that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the function should return 1367.

ANS:

```

1  /* Exercise 5.28 Solution */
2  #include <stdio.h>
3
4  int reverseDigits( int n );
5
6  int main()
7  {
8      int number; /* input number */
9
10     printf( "Enter a number between 1 and 9999: " );
11     scanf( "%d", &number );
12
13     /* find number with digits reversed */
14     printf( "The number with its digits reversed is: %d\n",
15           reverseDigits( number ) );
16
17     return 0; /* indicate successful termination */
18 } /* end main */
19
20
21 /* reverseDigits returns number obtained by
22 reversing digits of n */
23 int reverseDigits( int n )
24 {
25     int reverse = 0; /* reversed number */
26     int divisor = 1000; /* current divisor */
27     int multiplier = 1; /* current multiplier */
28
29     /* loop until single-digit number */
30     while ( n > 9 ) {
31
32         /* if n >= current divisor, determine digit */
33         if ( n >= divisor ) {
34
35             /* update reversed number with current digit */
36             reverse += n / divisor * multiplier;
37
38             n %= divisor; /* update n */
39             divisor /= 10; /* update divisor */
40             multiplier *= 10; /* update multiplier */
41         } /* end if */
42         else { /* else, no digit */
43             divisor /= 10; /* update divisor */
44         } /* end else */
45     } /* end while */
46
47     reverse += n * multiplier;
48
49     return reverse; /* return reversed number */
50 } /* end function reverseDigits */

```

```

Enter a number between 1 and 9999: 6
The number with its digits reversed is: 6

```

```
Enter a number between 1 and 9999: 9273  
The number with its digits reversed is: 3729
```

5.29 The *greatest common divisor (GCD)* of two integers is the largest integer that evenly divides each of the two numbers. Write function `gcd` that returns the greatest common divisor of two integers.

ANS:

```
1  /* Exercise 5.29 Solution */
2  #include <stdio.h>
3
4  int gcd( int x, int y ); /* function prototype */
5
6  int main()
7  {
8      int j; /* loop counter */
9      int a; /* first number */
10     int b; /* second number */
11
12     /* loop for 5 pairs of inputs */
13     for ( j = 1; j <= 5; j++ ) {
14         printf( "Enter two integers: " );
15         scanf( "%d%d", &a, &b );
16
17         /* find greatest common divisor of a and b */
18         printf( "The greatest common divisor "
19             "of %d and %d is %d\n\n", a, b, gcd( a, b ) );
20     } /* end for */
21
22     return 0; /* indicate successful termination */
23
24 } /* end main */
25
26 /* gcd find greatest common divisor of x and y */
27 int gcd( int x, int y )
28 {
29     int i;
30     int greatest = 1; /* current gcd, 1 is minimum */
31
32     /* loop from 2 to smaller of x and y */
33     for ( i = 2; i <= ( ( x < y ) ? x : y ); i++ ) {
34
35         /* if current i divides both x and y */
36         if ( x % i == 0 && y % i == 0 ) {
37             greatest = i; /* update greatest common divisor */
38         } /* end if */
39
40     } /* end for */
41
42     return greatest; /* return greatest common divisor found */
43
44 } /* end function gcd */
```

```
Enter two integers: 75 225  
The greatest common divisor of 75 and 225 is 75
```

```
Enter two integers: 99 30  
The greatest common divisor of 99 and 30 is 3
```

```
Enter two integers: 17 22  
The greatest common divisor of 17 and 22 is 1
```

```
Enter two integers: 100 92  
The greatest common divisor of 100 and 92 is 4
```

```
Enter two integers: 10005 15  
The greatest common divisor of 10005 and 15 is 15
```

5.30 Write a function `qualityPoints` that inputs a student's average and returns 4 if a student's average is 90-100, 3 if the average is 80-89, 2 if the average is 70-79, 1 if the average is 60-69, and 0 if the average is lower than 60.

ANS:

```
1  /* Exercise 5.30 Solution */
2  #include <stdio.h>
3
4  int qualityPoints( int average ); /* function prototype */
5
6  int main()
7  {
8      int average; /* current average */
9      int loop;    /* loop counter */
10
11     /* loop for 5 inputs */
12     for ( loop = 1; loop <= 5; loop++ ) {
13         printf( "\nEnter the student's average: " );
14         scanf( "%d", &average );
15
16         /* determine and display corresponding quality points */
17         printf( "%d on a 4 point scale is %d\n",
18             average, qualityPoints( average ) );
19     } /* end for */
20
21     return 0; /* indicate successful termination */
22
23 } /* end main */
24
25 /* qualityPoints takes average in range 0 to 100 and
26    returns corresponding quality points on 0 to 4 scale */
27 int qualityPoints( int average )
28 {
29
30     /* 90 <= average <= 100 */
31     if ( average >= 90 ) {
32         return 4;
33     } /* end if */
34     else if ( average >= 80 ) { /* 80 <= average <= 89 */
35         return 3;
36     } /* end else if */
37     else if ( average >= 70 ) { /* 70 <= average <= 79 */
38         return 2;
39     } /* end else if */
40     else if ( average >= 60 ) { /* 60 <= average <= 69 */
41         return 1;
42     } /* end else if */
43     else { /* 0 <= average < 60 */
44         return 0;
45     } /* end else */
46
47 } /* end function qualityPoints */
```

```
Enter the student's average: 92  
92 on a 4 point scale is 4
```

```
Enter the student's average: 87  
87 on a 4 point scale is 3
```

```
Enter the student's average: 75  
75 on a 4 point scale is 2
```

```
Enter the student's average: 63  
63 on a 4 point scale is 1
```

```
Enter the student's average: 22  
22 on a 4 point scale is 0
```


5.31 Write a program that simulates coin tossing. For each toss of the coin the program should print Heads or Tails. Let the program toss the coin 100 times, and count the number of times each side of the coin appears. Print the results. The program should call a separate function `flip` that takes no arguments and returns 0 for tails and 1 for heads. [Note: If the program realistically simulates the coin tossing, then each side of the coin should appear approximately half the time for a total of approximately 50 heads and 50 tails.]

ANS:

```

1  /* Exercise 5.31 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  int flip(); /* function prototype */
7
8  int main()
9  {
10     int loop;          /* loop counter */
11     int headCount = 0; /* total Heads count */
12     int tailCount = 0; /* total Tails count */
13
14     srand( time( NULL ) ); /* seed random number generator */
15
16     /* simulate coin toss 100 times */
17     for ( loop = 1; loop <= 100; loop++ ) {
18
19         /* simulate coin toss, 0 refers to tails */
20         if ( flip() == 0 ) {
21             tailCount++; /* update Tails count */
22         } /* end if */
23         else {
24             headCount++; /* update Heads count */
25         } /* end else */
26
27         if ( loop % 10 == 0 ) {
28             printf( "\n" );
29         } /* end if */
30
31     } /* end for */
32
33     /* display totals */
34     printf( "\nThe total number of Heads was %d\n", headCount );
35     printf( "The total number of Tails was %d\n", tailCount );
36
37     return 0; /* indicate successful termination */
38 } /* end main */
39
40 /* flip uses random number to simulate coin toss */
41 int flip() {
42     int HorT = rand() %2; /* scale by 2 for binary result */
43
44     /* display result of flip */
45     if ( HorT == 0 ) {
46         printf( "Tails " );
47     } /* end if */
48     else {
49         printf( "Heads " );
50     } /* end else */
51
52     return HorT; /* return result of coin toss */
53 } /* end function flip */

```

```
Tails Heads Tails Tails Tails Tails Heads Tails Tails Tails
Tails Tails Tails Heads Tails Heads Tails Tails Heads Tails
Tails Heads Heads Tails Tails Heads Tails Tails Tails Tails
Tails Tails Heads Heads Heads Heads Heads Heads Tails Tails
Heads Heads Heads Heads Heads Tails Tails Tails Tails Tails
Tails Tails Tails Heads Heads Tails Tails Tails Tails Heads
Tails Tails Tails Heads Heads Tails Tails Heads Tails Tails
Heads Tails Tails Heads Tails Tails Tails Tails Heads Tails
Tails Tails Tails Tails Heads Tails Heads Heads Tails Tails
Heads Tails Tails Heads Tails Tails Heads Tails Tails Tails
```

The total number of Heads was 34

The total number of Tails was 66

5.32 Computers are playing an increasing role in education. Write a program that will help an elementary school student learn multiplication. Use `rand` to produce two positive one-digit integers. It should then type a question such as:

How much is 6 times 7?

The student then types the answer. Your program checks the student's answer. If it is correct, print "Very good!" and then ask another multiplication question. If the answer is wrong, print "No. Please try again." and then let the student try the same question again repeatedly until the student finally gets it right.

ANS:

```

1  /* Exercise 5.32 solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  void multiplication( void ); /* function prototype */
7
8  int main( void )
9  {
10     srand( time( NULL ) ); /* seed random number generator */
11     multiplication(); /* begin multiplication practice */
12
13     return 0; /* indicate successful termination */
14 }
15 /* end main */
16
17 /* multiplication produces pairs of random numbers and
18    prompts user for product */
19 void multiplication( void )
20 {
21     int x;          /* first factor */
22     int y;          /* second factor */
23     int response = 0; /* user response for product */
24
25     /* use sentinel-controlled repetition */
26     printf( "Enter -1 to end.\n" );
27
28     /* loop while sentinel value not read from user */
29     while ( response != -1 ) {
30         x = rand() % 10; /* generate 1-digit random number */
31         y = rand() % 10; /* generate another 1-digit random number */
32
33         printf( "How much is %d times %d? ", x, y );
34         scanf( "%d", &response );
35
36         /* loop while not sentinel value or correct response */
37         while ( response != -1 && response != x * y ) {
38             printf( "No. Please try again.\n? " );
39             scanf( "%d", &response );
40         } /* end while */
41
42         /* correct response */
43         if ( response != -1 ) {
44             printf( "Very good!\n\n" );
45         } /* end if */
46     } /* end while */
47
48     printf( "That's all for now. Bye.\n" );
49 } /* end function multiplication */
50
51

```

```
Enter -1 to end.  
How much is 0 times 7? 0  
Very good!  
  
How much is 0 times 0? 0  
Very good!  
  
How much is 2 times 6? 18  
No. Please try again.  
? 12  
Very good!  
  
How much is 5 times 0? 0  
Very good!  
  
How much is 9 times 2? 18  
Very good!  
  
How much is 6 times 1? -1  
That's all for now. Bye.
```

5.33 The use of computers in education is referred to as *computer-assisted instruction* (CAI). One problem that develops in CAI environments is student fatigue. This can be eliminated by varying the computer's dialogue to hold the student's attention. Modify the program of Exercise 5.32 so the various comments are printed for each correct answer and each incorrect answer as follows:

Responses to a correct answer

```
Very good!
Excellent!
Nice work!
Keep up the good work!
```

Responses to an incorrect answer

```
No. Please try again.
Wrong. Try once more.
Don't give up!
No. Keep trying.
```

Use the random number generator to choose a number from 1 to 4 to select an appropriate response to each answer. Use a switch statement with printf statements to issue the responses.

ANS:

```
1  /* Exercise 5.33 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  void correctMessage( void ); /* function prototype */
7  void incorrectMessage( void ); /* function prototype */
8  void multiplication( void ); /* function prototype */
9
10 int main()
11 {
12     srand( time( NULL ) ); /* seed random number generator */
13     multiplication(); /* begin multiplication practice */
14
15     return 0; /* indicate successful termination */
16 } /* end main */
17
18
19 /* correctMessage randomly chooses response to correct answer */
20 void correctMessage( void )
21 {
22
23     /* generate random number between 0 and 3 */
24     switch ( rand() % 4 ) {
25
26         /* display a random response */
27         case 0:
28             printf( "Very good!\n\n" );
29             break; /* exit switch */
30
31         case 1:
32             printf( "Excellent!\n\n" );
33             break; /* exit switch */
34
35         case 2:
36             printf( "Nice work!\n\n" );
37             break; /* exit switch */
38
39         case 3:
40             printf( "Keep up the good work!\n\n" );
41             break; /* exit switch */
42     } /* end switch */
```

```

43
44 } /* end function correctMessage */
45
46 /* incorrectMessage randomly chooses response to incorrect answer */
47 void incorrectMessage( void )
48 {
49
50     /* generate random number between 0 and 3 */
51     switch ( rand() % 4 ) {
52
53         /* display random response */
54         case 0:
55             printf( "No. Please try again.\n? " );
56             break; /* exit switch */
57
58         case 1:
59             printf( "Wrong. Try once more.\n? " );
60             break; /* exit switch */
61
62         case 2:
63             printf( "Don't give up!\n? " );
64             break; /* exit switch */
65
66         case 3:
67             printf( "No. Keep trying.\n? " );
68             break; /* exit switch */
69     } /* end switch */
70
71 } /* end function incorrectMessage */
72
73 /* multiplication produces pairs of random numbers and
74    prompts user for product */
75 void multiplication( void )
76 {
77     int x;           /* first factor */
78     int y;           /* second factor */
79     int response = 0; /* user response for product */
80
81     /* use sentinel-controlled repetition */
82     printf( "Enter -1 to end.\n" );
83
84     /* loop while sentinel value not read from user */
85     while ( response != -1 ) {
86         x = rand() % 10; /* generate 1-digit random number */
87         y = rand() % 10; /* generate another 1-digit random number */
88
89         printf( "How much is %d times %d? ", x, y );
90         scanf( "%d", &response );
91
92         /* loop while not sentinel value or correct response */
93         while ( response != -1 && response != x * y ) {
94             incorrectMessage();
95             scanf( "%d", &response );
96         } /* end while */
97
98         /* correct response */
99         if ( response != -1 ) {
100             correctMessage();
101         } /* end if */
102
103     } /* end while */

```

```
104
105     printf( "That's all for now. Bye.\n" );
106 } /* end function multiplication */
```

```
Enter -1 to end.
How much is 7 times 6? 42
Very good!
```

```
How much is 8 times 5? 40
Excellent!
```

```
How much is 7 times 2? 15
No. Please try again.
? 14
Keep up the good work!
```

```
How much is 9 times 6? 54
Keep up the good work!
```

```
How much is 3 times 7? -1
That's all for now. Bye.
```

5.34 More sophisticated computer-aided instructions systems monitor the student's performance over a period of time. The decision to begin a new topic is often based on the student's success with previous topics. Modify the program of Exercise 5.33 to count the number of correct and incorrect responses typed by the student. After the student types 10 answers, your program should calculate the percentage of correct responses. If the percentage is lower than 75 percent, your program should print "Please ask your instructor for extra help" and then terminate.

ANS:

```

1  /* Exercise 5.34 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  void multiplication( void ); /* function prototype */
7  void correctMessage( void ); /* function prototype */
8  void incorrectMessage( void ); /* function prototype */
9
10 int main()
11 {
12     srand( time( NULL ) ); /* seed random number generator */
13     multiplication(); /* begin multiplication practice */
14
15     return 0; /* indicate successful termination */
16 }
17 /* end main */
18
19 /* multiplication produces pairs of random numbers and
20    prompts user for product */
21 void multiplication( void )
22 {
23     int i; /* loop counter */
24     int x; /* first factor */
25     int y; /* second factor */
26     int response; /* user response for product */
27     int right = 0; /* total number of right responses */
28     int wrong = 0; /* total number of wrong responses */
29
30     /* loop 10 times */
31     for ( i = 1; i <= 10; i++ ) {
32         x = rand() % 10; /* generate 1-digit random number */
33         y = rand() % 10; /* generate another 1-digit random number */
34
35         printf( "How much is %d times %d? ", x, y );
36         scanf( "%d", &response );
37
38         /* loop while not correct response */
39         while ( response != x * y ) {
40             wrong++; /* update total number of wrong responses */
41             incorrectMessage();
42             scanf( "%d", &response );
43         } /* end while */
44
45         right++; /* update total number of correct responses */
46         correctMessage();
47     } /* end for */
48
49     /* determine if help is needed */
50     if ( ( double ) right / ( right + wrong ) < .75 ) {
51         printf( "Please ask your instructor for extra help.\n" );
52     } /* end if */
53

```



```
54     printf( "That's all for now. Bye.\n" );
55 } /* end function multiplication */
56
57 /* correctMessage randomly chooses response to correct answer */
58 void correctMessage( void )
59 {
60
61     /* generate random number between 0 and 3 */
62     switch ( rand() % 4 ) {
63
64         /* display random response */
65         case 0:
66             printf( "Very good!\n\n" );
67             break; /* exit switch */
68
69         case 1:
70             printf( "Excellent!\n\n" );
71             break; /* exit switch */
72
73         case 2:
74             printf( "Nice work!\n\n" );
75             break; /* exit switch */
76
77         case 3:
78             printf( "Keep up the good work!\n\n" );
79             break; /* exit switch */
80     } /* end switch */
81
82 } /* end function correctMessage */
83
84 /* incorrectMessage randomly chooses response to incorrect answer */
85 void incorrectMessage( void )
86 {
87
88     /* generate random number between 0 and 3 */
89     switch ( rand() % 4 ) {
90
91         /* display random response */
92         case 0:
93             printf( "No. Please try again.\n? " );
94             break; /* exit switch */
95
96         case 1:
97             printf( "Wrong. Try once more.\n? " );
98             break; /* exit switch */
99
100        case 2:
101            printf( "Don't give up!\n? " );
102            break; /* exit switch */
103
104        case 3:
105            printf( "No. Keep trying.\n? " );
106            break; /* exit switch */
107    } /* end switch */
108
109 } /* end function incorrectMessage */
```

```
How much is 3 times 9? 27  
Excellent!
```

```
How much is 1 times 3? 3  
Very good!
```

```
How much is 8 times 1? 8  
Very good!
```

```
How much is 3 times 6? 24  
No. Please try again.  
? 18  
Excellent!
```

```
...
```

```
How much is 1 times 9? 9  
Very good!
```

```
How much is 4 times 0? 4  
Wrong. Try once more.  
? 0  
Excellent!
```

```
How much is 5 times 8? 40  
Nice work!
```

```
That's all for now. Bye.
```

5.35 Write a C program that plays the game of “guess the number” as follows: Your program chooses the number to be guessed by selecting an integer at random in the range 1 to 1000. The program then types:

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.

The player then types a first guess. The program responds with one of the following:

1. Excellent! You guessed the number!
Would you like to play again (y or n)?
2. Too low. Try again.
3. Too high. Try again.

If the player’s guess is incorrect, your program should loop until the player finally gets the number right. Your program should keep telling the player Too high or Too low to help the player “zero in” on the correct answer. [Note: The searching technique employed in this problem is called *binary search*. We will say more about this in the next problem.]

ANS:

```

1  /* Exercise 5.35 solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  void guessGame( void ); /* function prototype */
7
8  int main()
9  {
10     srand( time( NULL ) ); /* seed random number generator */
11     guessGame();
12
13     return 0; /* indicate successful termination */
14 }
15 } /* end main */
16
17 /* guessGame generates numbers between 1 and 1000
18    and checks user's guess */
19 void guessGame( void )
20 {
21     int x; /* randomly generated number */
22     int guess; /* user's guess */
23     int response; /* response to continue game, 1=yes, 2=no */
24
25     /* loop until user types 2 to quit game */
26     do {
27
28         /* generate random number between 1 and 1000
29            1 is shift, 1000 is scaling factor */
30         x = 1 + rand() % 1000;
31
32         /* prompt for guess */
33         printf( "\nI have a number between 1 and 1000.\n" );
34         printf( "Can you guess my number?\n" );
35         printf( "Please type your first guess.\n? " );
36         scanf( "%d", &guess );
37

```

```
38     /* loop until correct number */
39     while ( guess != x ) {
40
41         /* if guess is too low */
42         if ( guess < x ) {
43             printf( "Too low. Try again.\n? " );
44         } /* end if */
45         else { /* guess is too high */
46             printf( "Too high. Try again.\n? " );
47         } /* end else */
48
49         scanf( "%d", &guess );
50     } /* end while */
51
52     /* prompt for another game */
53     printf( "\nExcellent! You guessed the number!\n" );
54     printf( "Would you like to play again?\n" );
55     printf( "Please type ( 1=yes, 2=no )? " );
56     scanf( "%d", &response );
57 } while ( response == 1 ); /* end do...while */
58
59 } /* end function guessGame */
```

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too low. Try again.
? 750
Too high. Try again.
? 625
Too low. Try again.
? 687
Too high. Try again.
? 656
Too low. Try again.
? 671
Too low. Try again.
? 678
Too high. Try again.
? 675
Too high. Try again.
? 673
Too high. Try again.
? 672

Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )? 2
```

5.36 Modify the program of Exercise 5.35 to count the number of guesses the player makes. If the number is 10 or fewer, print Either you know the secret or you got lucky! If the player guesses the number in 10 tries, then print Ahah! You know the secret! If the player makes more than 10 guesses, then print You should be able to do better! Why should it take no more than 10 guesses? Well with each “good guess” the player should be able to eliminate half of the numbers. Now show why any number 1 to 1000 can be guessed in 10 or fewer tries.

ANS:

```

1  /* Exercise 5.36 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  void guessGame( void ); /* function prototype */
7
8  int main()
9  {
10     srand( time( NULL ) ); /* seed random number generator */
11     guessGame();
12
13     return 0; /* indicate successful termination */
14 }
15 } /* end main */
16
17 /* guessGame generates numbers between 1 and 1000
18    and checks user's guess */
19 void guessGame( void )
20 {
21     int x;          /* randomly generated number */
22     int guess;     /* user's guess */
23     int total = 1; /* number of guesses */
24     int response; /* response to continue game, 1=yes, 0=no */
25
26     /* loop until user enters 0 to quit game */
27     do {
28
29         /* generate random number between 1 and 1000
30            1 is shift, 1000 is scaling factor */
31         x = 1 + rand() % 1000;
32
33         /* prompt for guess */
34         printf( "\nI have a number between 1 and 1000.\n" );
35         printf( "Can you guess my number?\n" );
36         printf( "Please type your first guess.\n? " );
37         scanf( "%d", &guess );
38
39         /* loop while not correct answer */
40         while ( guess != x ) {
41
42             /* guess is incorrect; display hint */
43             if ( guess < x ) {
44                 printf( "Too low. Try again.\n? " );
45             } /* end if */
46             else {
47                 printf( "Too high. Try again.\n? " );
48             } /* end else */
49
50             scanf( "%d", &guess );
51             total++;
52         } /* end while */
53

```

```
54     printf( "\nExcellent! You guessed the number!\n" );
55
56     /* determine if user knows "secret" */
57     if ( total < 10 ) {
58         printf( "Either you know the secret or you got lucky!\n" );
59     } /* end if */
60     else if ( total == 10 ) {
61         printf( "Ahah! You know the secret!\n" );
62     } /* end else if */
63     else {
64         printf( "You should be able to do better!\n\n" );
65     } /* end else */
66
67     /* prompt for another game */
68     printf( "Would you like to play again?\n" );
69     printf( "Please type ( 1=yes, 2=no )? " );
70     scanf( "%d", &response );
71     } while ( response == 1 ); /* end do...while */
72
73 } /* end function guessGame */
```

I have a number between 1 and 1000.

Can you guess my number?

Please type your first guess.

? 500

Too high. Try again.

? 250

Too high. Try again.

? 125

Too high. Try again.

? 62

Too high. Try again.

? 31

Too low. Try again.

? 46

Too high. Try again.

? 39

Too low. Try again.

? 42

Excellent! You guessed the number!

Either you know the secret or you got lucky!

Would you like to play again?

Please type (1=yes, 2=no)? 2

5.37 Write a recursive function `power(base, exponent)` that when invoked returns

$$\text{base}^{\text{exponent}}$$

For example, `power(3, 4) = 3 * 3 * 3 * 3`. Assume that `exponent` is an integer greater than or equal to 1. *Hint:* The recursion step would use the relationship

$$\text{base}^{\text{exponent}} = \text{base} * \text{base}^{\text{exponent} - 1}$$

and the terminating condition occurs when `exponent` is equal to 1 because

$$\text{base}^1 = \text{base}$$

ANS:

```

1  /* Exercise 5.37 Solution */
2  #include <stdio.h>
3
4  long power( long base, long exponent ); /* function prototype */
5
6  int main()
7  {
8      long b; /* base */
9      long e; /* exponent */
10
11     printf( "Enter a base and an exponent: " );
12     scanf( "%ld%ld", &b, &e );
13
14     /* calculate and display b raised to the e power */
15     printf( "%ld raised to the %ld is %ld\n", b, e, power( b, e ) );
16
17     return 0; /* indicate successful termination */
18
19 } /* end main */
20
21 /* power recursively calculates base raised to the exponent
22    assume exponent >= 1 */
23 long power( long base, long exponent )
24 {
25
26     /* base case: exponent equals 1, return base */
27     if ( exponent == 1 ) {
28         return base;
29     } /* end if */
30     else { /* recursive step */
31         return base * power( base, exponent - 1 );
32     } /* end else */
33
34 } /* end function power */

```

```

Enter a base and an exponent: 5 10
5 raised to the 10 is 9765625

```

5.38 The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms. a) Write a *non-recursive* function `fibonacci(n)` that calculates the *n*th Fibonacci number. b) Determine the largest Fibonacci number that can be printed on your system. Modify the program of part a) to use `double` instead of `int` to calculate and return Fibonacci numbers. Let the program loop until it fails because of an excessively high value.

ANS:

```

1  /* Exercise 5.38 Part A Solution */
2  #include <stdio.h>
3  #define MAX 23 /* the maximum number for which the */
4                /* fibonacci value can be calculated */
5                /* on 2-byte integer systems */
6  int fibonacci( int n );
7
8  int main()
9  {
10     int loop; /* loop counter */
11
12     /* calculate and display Fibonacci value for 0 to MAX */
13     for ( loop = 0; loop <= MAX; loop++ ) {
14         printf( "fibonacci( %d ) = %d\n", loop, fibonacci( loop ) );
15     } /* end for */
16
17     return 0; /* indicate successful termination */
18
19 } /* end main */
20
21 /* fibonacci nonrecursively calculates nth Fibonacci number */
22 int fibonacci( int n )
23 {
24     int j; /* loop counter */
25     int fib[ MAX ]; /* define array of size MAX */
26
27     fib[ 0 ] = 0;
28     fib[ 1 ] = 1;
29
30     /* loop to find nth Fibonacci value */
31     for ( j = 2; j <= n; j++ ) {
32         fib[ j ] = fib[ j - 1 ] + fib[ j - 2 ];
33     } /* end for */
34
35     return fib[ n ]; /* return nth Fibonacci value */
36
37 } /* end function fibonacci */

```

```

fibonacci( 0 ) = 0
fibonacci( 1 ) = 1
fibonacci( 2 ) = 1
fibonacci( 3 ) = 2
fibonacci( 4 ) = 3
fibonacci( 5 ) = 5
fibonacci( 6 ) = 8
fibonacci( 7 ) = 13
fibonacci( 8 ) = 21
fibonacci( 9 ) = 34
fibonacci( 10 ) = 55
fibonacci( 11 ) = 89
fibonacci( 12 ) = 144
fibonacci( 13 ) = 233
fibonacci( 14 ) = 377
fibonacci( 15 ) = 610
fibonacci( 16 ) = 987
fibonacci( 17 ) = 1597
fibonacci( 18 ) = 2584
fibonacci( 19 ) = 4181
fibonacci( 20 ) = 6765
fibonacci( 21 ) = 10946
fibonacci( 22 ) = 17711
fibonacci( 23 ) = 28658

```



```

1  /* Exercise 5.38 Part B Solution */
2  #include <stdio.h>
3  #define SIZE 100
4
5  double fibonacci( int n ); /* function prototype */
6
7  int main()
8  {
9      int loop; /* loop counter */
10
11     /* loop SIZE times and calculate Fibonacci values */
12     for ( loop = 0; loop < SIZE; loop++ ) {
13         printf( "fibonacci( %d ) = %.1f\n", loop, fibonacci( loop ) );
14     } /* end for */
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */
19
20 /* fibonacci nonrecursively calculates nth Fibonacci number */
21 double fibonacci( int n )
22 {
23     int j; /* loop counter */
24     double fib[ SIZE ]; /* define double array of size SIZE */
25
26     fib[ 0 ] = 0.0;
27     fib[ 1 ] = 1.0;
28
29     /* loop to find nth Fibonacci value */
30     for ( j = 2; j <= n; j++ ) {
31         fib[ j ] = fib[ j - 1 ] + fib[ j - 2 ];
32     } /* end for */
33
34     return fib[ n ]; /* return nth Fibonacci value */
35
36 } /* end function fibonacci */

```

```

fibonacci( 0 ) = 0.0
fibonacci( 1 ) = 1.0
fibonacci( 2 ) = 1.0
fibonacci( 3 ) = 2.0
fibonacci( 4 ) = 3.0
fibonacci( 5 ) = 5.0
fibonacci( 6 ) = 8.0
fibonacci( 7 ) = 13.0
fibonacci( 8 ) = 21.0
fibonacci( 9 ) = 34.0
fibonacci( 10 ) = 55.0
fibonacci( 11 ) = 89.0
fibonacci( 12 ) = 144.0
.
.
.
fibonacci( 96 ) = 51680708854858326000.0
fibonacci( 97 ) = 83621143489848426000.0
fibonacci( 98 ) = 135301852344706760000.0
fibonacci( 99 ) = 218922995834555200000.0

```

5.39 (*Towers of Hanoi*) Every budding computer scientist must grapple with certain classic problems, and the Towers of Hanoi (see Fig. 5.19) is one of the most famous of these. Legend has it that in a temple in the Far East, priests are attempting to move a stack of disks from one peg to another. The initial stack had 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from this peg to a second peg under the constraints that exactly one disk is moved at a time, and at no time may a larger disk be placed above a smaller disk. A third peg is available for temporarily holding the disks. Supposedly the world will end when the priests complete their task, so there is little incentive for us to facilitate their efforts.

Let us assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that will print the precise sequence of disk-to-disk peg transfers.

If we were to approach this problem with conventional methods, we would rapidly find ourselves hopelessly knotted up in managing the disks. Instead, if we attack the problem with recursion in mind, it immediately becomes tractable. Moving n disks can be viewed in terms of moving only $n - 1$ disks (and hence the recursion) as follows:

- a) Move $n - 1$ disks from peg 1 to peg 2, using peg 3 as a temporary holding area.
- b) Move the last disk (the largest) from peg 1 to peg 3.
- c) Move the $n - 1$ disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

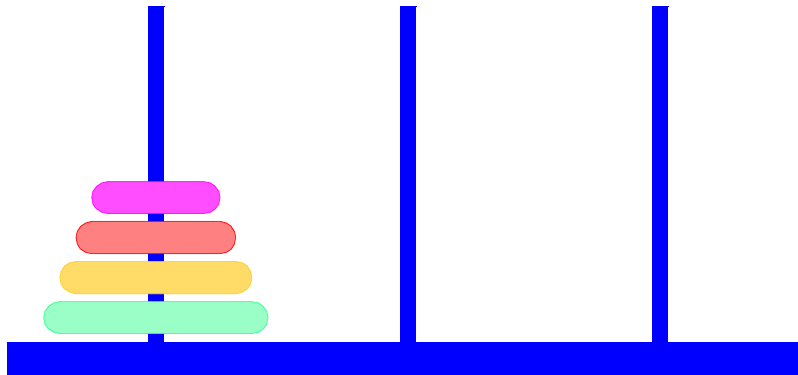


Fig. 5.18 The Towers of Hanoi for the case with four disks.

The process ends when the last task involves moving $n = 1$ disk, i.e., the base case. This is accomplished by trivially moving the disk without the need for a temporary holding area.

Write a program to solve the Towers of Hanoi problem. Use a recursive function with four parameters:

- a) The number of disks to be moved
- b) The peg on which these disks are initially threaded
- c) The peg to which this stack of disks is to be moved
- d) The peg to be used as a temporary holding area

Your program should print the precise instructions it will take to move the disks from the starting peg to the destination peg. For example, to move a stack of three disks from peg 1 to peg 3, your program should print the following series of moves:

```
1 → 3 (This means move one disk from peg 1 to peg 3.)
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3
```

ANS:

```

1  /* Exercise 5.39 solution */
2  #include <stdio.h>
3
4  /* function prototype */
5  void tower( int c, int start, int end, int temp );
6
7  int main()
8  {
9      int n; /* number of disks */
10
11     printf( "Enter the starting number of disks: " );
12     scanf( "%d", &n );
13
14     /* print instructions for moving disks from
15        peg 1 to peg 3 using peg 2 for temporary storage */
16     tower( n, 1, 3, 2 );
17
18     return 0; /* indicate successful termination */
19 }
20 } /* end main */
21
22 /* tower recursively prints instructions for moving disks
23    from start peg to end peg using temp peg for temporary storage */
24 void tower( int c, int start, int end, int temp )
25 {
26
27     /* base case */
28     if ( c == 1 ) {
29         printf( "%d --> %d\n", start, end );
30         return;
31     } /* end if */
32
33     /* move c - 1 disks from start to temp */
34     tower( c - 1, start, temp, end );
35
36     /* move last disk from start to end */
37     printf( "%d --> %d\n", start, end );
38
39     /* move c - 1 disks from temp to end */
40     tower( c - 1, temp, end, start );
41 } /* end function tower */

```

```

Enter the starting number of disks: 4
1 --> 2
1 --> 3
2 --> 3
1 --> 2
3 --> 1
3 --> 2
1 --> 2
1 --> 3
2 --> 3
2 --> 1
3 --> 1
2 --> 3
1 --> 2
1 --> 3
2 --> 3

```

5.40 Any program that can be implemented recursively can be implemented iteratively, although sometimes with considerably more difficulty and considerably less clarity. Try writing an iterative version of the Towers of Hanoi. If you succeed, compare your iterative version with the recursive version you developed in Exercise 5.39. Investigate issues of performance, clarity, and your ability to demonstrate the correctness of the programs.

5.41 (*Visualizing Recursion*) It is interesting to watch recursion “in action.” Modify the factorial function of Fig. 5.14 to print its local variable and recursive call parameter. For each recursive call, display the outputs on a separate line and add a level of indentation. Do your utmost to make the outputs clear, interesting, and meaningful. Your goal here is to design and implement an output format that helps a person understand recursion better. You may want to add such display capabilities to the many other recursion examples and exercises throughout the text.

ANS: *Note:* The `printf` in function `printRecursion` uses the conversion specification `%*d`. The `*` enables the programmer to specify the field width as a variable argument in the `printf`. In this case variable `n` is used as the field width, and its value is output.

```

1  /* Exercise 5.41 Solution */
2  #include <stdio.h>
3
4  long factorial( long number ); /* function prototype */
5  void printRecursion( int n ); /* function prototype */
6
7  int main()
8  {
9      int i; /* loop counter */
10
11     /* calculate factorial( i ) and display result */
12     for ( i = 0; i <= 10; i++ ) {
13         printf( "%2d! = %ld\n", i, factorial( i ) );
14     } /* end for */
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */
19
20 /* recursive definition of function factorial */
21 long factorial( long number )
22 {
23
24     /* base case */
25     if ( number <= 1 ) {
26         return 1;
27     } /* end if */
28     else { /* recursive step */
29         printRecursion( number ); /* add outputs and indentation */
30         return ( number * factorial( number - 1 ) );
31     } /* end else */
32
33 } /* end function factorial */
34
35 /* printRecursion adds outputs and indentation to help
36    visualize recursion */
37 void printRecursion( int n )
38 {
39     printf( "number = %*d\n", n, n );
40 } /* end function printRecursion */

```

```
0! = 1
1! = 1
number = 2
2! = 2
number = 3
number = 2
3! = 6
number = 4
number = 3
number = 2
4! = 24
number = 5
number = 4
number = 3
number = 2
5! = 120
.
.
.
number = 9
number = 8
number = 7
number = 6
number = 5
number = 4
number = 3
number = 2
9! = 362880
number = 10
number = 9
number = 8
number = 7
number = 6
number = 5
number = 4
number = 3
number = 2
10! = 3628800
```

5.42 The greatest common divisor of integers x and y is the largest integer that evenly divides both x and y . Write a recursive function `gcd` that returns the greatest common divisor of x and y . The `gcd` of x and y is defined recursively as follows: If y is equal to 0, then `gcd(x, y)` is x ; otherwise `gcd(x, y)` is `gcd(y, x % y)` where `%` is the remainder operator.

ANS:

```
1  /* Exercise 5.42 Solution */
2  #include <stdio.h>
3
4  /* function prototype */
5  unsigned int gcd( unsigned int xMatch, unsigned int yMatch );
6
7  int main()
8  {
9      unsigned int x;    /* first integer */
10     unsigned int y;    /* second integer */
11     unsigned int gcdDiv; /* greatest common divisor of x and y */
12
13     printf( "Enter two integers: " );
14     scanf( "%u%u", &x, &y );
15
16     gcdDiv = gcd( x, y );
17
18     printf( "Greatest common divisor of %u and %u is %u\n",
19           x, y, gcdDiv );
20
21     return 0; /* indicate successful termination */
22 } /* end main */
23
24
25 /* gcd recursively finds greatest common divisor
26    of xMatch and yMatch */
27 unsigned int gcd( unsigned int xMatch, unsigned int yMatch )
28 {
29
30     /* base case */
31     if ( yMatch == 0 ) {
32         return xMatch;
33     } /* end if */
34     else { /* recursive step */
35         return gcd( yMatch, xMatch % yMatch );
36     } /* end else */
37
38 } /* end function gcd */
```

```
Enter two integers: 10112 50500
Greatest common divisor of 10112 and 50500 is 4
```

5.43 Can `main` be called recursively? Write a program containing a function `main`. Include `static` local variable `count` initialized to 1. Postincrement and print the value of `count` each time `main` is called. Run your program. What happens?

ANS:

```
1  /* Exercise 5.43 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      static int count = 1; /* static local variable count */
7
8      printf( "%d\n", count );
9      count++;
10
11     main(); /* recursively call int main() */
12
13     return 0; /* indicate successful termination */
14
15 } /* end main */
```

```
1
2
3
4
5
6
7
8
9
10
...
```

5.44 Exercises 5.32 through 5.34 developed a computer-assisted instruction program to teach an elementary school student multiplication. This exercise suggests enhancements to that program.

- a) Modify the program to allow the user to enter a grade-level capability. A grade level of 1 means to use only single-digit numbers in the problems, a grade level of two means to use numbers as large as two-digits, etc.

ANS:

```

1  /* Exercise 5.44 Part A Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  int randValue( int level ); /* function prototype */
7  void multiplication( void ); /* function prototype */
8  void correctMessage( void ); /* function prototype */
9  void incorrectMessage( void ); /* function prototype */
10
11 int main()
12 {
13     srand( time( NULL ) ); /* seed random number generator */
14     multiplication(); /* being multiplication practice */
15
16     return 0; /* indicate successful termination */
17 } /* end main */
18
19 /* randValue generates random numbers based on grade level */
20 int randValue( int level )
21 {
22     /* level determines size of random number */
23     switch ( level ) {
24         case 1:
25             return rand() % 10;
26
27         case 2:
28             return rand() % 100;
29
30         case 3:
31             return rand() % 1000;
32
33         default:
34             return rand() % 10;
35     } /* end switch */
36 } /* end function randValue */
37
38 /* multiplication produces pairs of random numbers and
39 prompts user for product; level determines size of numbers */
40 void multiplication( void )
41 {
42     int i; /* loop counter */
43     int x; /* first factor */
44     int y; /* second factor */
45     int gradeLevel; /* grade-level capability */
46     int right = 0; /* total number of right responses */
47     int wrong = 0; /* total number of wrong responses */
48     unsigned int response; /* user response for product */
49
50     printf( "Enter the grade-level ( 1 to 3 ): " );
51     scanf( "%d", &gradeLevel );
52
53     /* loop 10 times */
54     for ( i = 1; i <= 10; i++ ) {
55
56         /* generate random numbers depending on level */
57         x = randValue( gradeLevel );

```



```
62     y = randValue( gradeLevel );
63
64     printf( "How much is %d times %d? ", x, y );
65     scanf( "%u", &response );
66
67     /* loop while response is incorrect */
68     while ( response != x * y ) {
69         ++wrong; /* update total number of wrong answers */
70         incorrectMessage();
71         scanf( "%u", &response );
72     } /* end while */
73
74     ++right; /* update total number of right answers */
75     correctMessage();
76 } /* end for */
77
78 /* if < 75% right */
79 if ( ( double ) right / ( right + wrong ) < .75 ) {
80     printf( "Please ask your instructor for extra help.\n" );
81 } /* end if */
82
83 printf( "That's all for now. Bye.\n" );
84 } /* end function multiplication */
85
86 /* correctMessage randomly chooses response to correct answer */
87 void correctMessage( void )
88 {
89
90     /* generate random number between 0 and 3 */
91     switch ( rand() % 4 ) {
92
93     case 0:
94         printf( "Very good!\n\n" );
95         break; /* exit switch */
96
97     case 1:
98         printf( "Excellent!\n\n" );
99         break; /* exit switch */
100
101     case 2:
102         printf( "Nice work!\n\n" );
103         break; /* exit switch */
104
105     case 3:
106         printf( "Keep up the good work!\n\n" );
107         break; /* exit switch */
108     } /* end switch */
109 } /* end function correctMessage */
110
111 /* incorrectMessage randomly chooses response to incorrect answer */
112 void incorrectMessage( void )
113 {
114
115     /* generate random number between 0 and 3 */
116     switch ( rand() % 4 ) {
117
118     case 0:
119         printf( "No. Please try again.\n? " );
120         break; /* exit switch */
121
122     case 1:
123         printf( "Wrong. Try once more.\n? " );
124         break; /* exit switch */
125
126     case 2:
127         printf( "Don't give up!\n? " );
128         break; /* exit switch */
129
```

```
130
131     case 3:
132         printf( "No. Keep trying.\n? " );
133         break; /* exit switch */
134     } /* end switch */
135
136 } /* end function incorrectMessage */
```

```
Enter the grade-level ( 1 to 3 ): 1
How much is 6 times 0? 0
Keep up the good work!
```

```
How much is 6 times 3? 18
Keep up the good work!
...
```

```
Enter the grade-level ( 1 to 3 ): 2
How much is 5 times 63? 315
Excellent!
```

```
How much is 29 times 13? 392
No. Please try again.
? 377
Excellent!
...
```

```
Enter the grade-level ( 1 to 3 ): 3
How much is 799 times 343? 274057
Keep up the good work!
```

```
How much is 201 times 349? 0
Don't give up!
...
```

- b) Modify the program to allow the user to pick the type of arithmetic problems he or she wishes to study. An option of 1 means addition problems only, 2 means subtraction problems only, 3 means multiplication problems only, 4 means division problems only, and 5 means to randomly intermix problems of all these types.

ANS:

```

1  /* Exercise 5.44 Part B Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  int menu( void );           /* function prototype */
7  void arithmetic( void );   /* function prototype */
8  void correctMessage( void ); /* function prototype */
9  void incorrectMessage( void ); /* function prototype */
10
11 int main()
12 {
13     srand( time( NULL ) ); /* seed random number generator */
14     arithmetic(); /* begin arithmetic process */
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */
19
20 /* menu displays user menu of choices */
21 int menu( void )
22 {
23     int choice; /* user's choice */
24
25     /* display menu and read user's choice */
26     do {
27         printf( "Choose type of problem to study.\n" );
28         printf( "Enter: 1 for addition, 2 for subtraction\n" );
29         printf( "Enter: 3 for multiplication, 4 for division\n" );
30         printf( "Enter: 5 for a combination of 1 through 4\n " );
31         printf( "? " );
32         scanf( "%d", &choice );
33     } while ( choice < 1 || choice > 5 ); /* end do...while */
34
35     return choice; /* return user's choice */
36
37 } /* end function menu */
38
39 /* incorrectMessage randomly chooses response to incorrect answer */
40 void incorrectMessage( void )
41 {
42
43     /* generate random number between 0 and 3 */
44     switch ( rand() % 4 ) {
45
46     case 0:
47         printf( "No. Please try again.\n? " );
48         break; /* exit switch */
49
50     case 1:
51         printf( "Wrong. Try once more.\n? " );
52         break; /* exit switch */
53
54     case 2:
55         printf( "Don't give up!\n? " );
56         break; /* exit switch */
57
58     case 3:
59         printf( "No. Keep trying.\n? " );
60         break; /* exit switch */
61     } /* end switch */
62
63 } /* end function incorrectMessage */

```

```
64
65 /* correctMessage randomly chooses response to correct answer */
66 void correctMessage( void )
67 {
68
69     /* generate random number between 0 and 3 */
70     switch ( rand() % 4 ) {
71
72         case 0:
73             printf( "Very good!\n\n" );
74             break; /* exit switch */
75
76         case 1:
77             printf( "Excellent!\n\n" );
78             break; /* exit switch */
79
80         case 2:
81             printf( "Nice work!\n\n" );
82             break; /* exit switch */
83
84         case 3:
85             printf( "Keep up the good work!\n\n" );
86             break; /* exit switch */
87     } /* end switch */
88
89 } /* end function correctMessage */
90
91 void arithmetic( void )
92 {
93     int i;           /* loop counter */
94     int x;           /* first number */
95     int y;           /* second number */
96     int response;    /* user response for product */
97     int answer;      /* correct answer */
98     int selection;   /* menu selection */
99     int right = 0;   /* total correct responses */
100    int wrong = 0;   /* total incorrect responses */
101    int type;         /* type of problems chosen */
102    int problemMix;   /* random choice of type of problem */
103    char operator;    /* arithmetic operator */
104
105    selection = menu();
106    type = selection;
107
108    /* loop 10 times */
109    for ( i = 1; i <= 10; i++ ) {
110        x = rand() % 10; /* generate first random number */
111        y = rand() % 10; /* generate second random number */
112
113        /* if option 5, randomly select type */
114        if ( selection == 5 ) {
115            problemMix = 1 + rand() % 4;
116            type = problemMix;
117        } /* end if */
118
119        /* generate answer and define operator depending on option */
120        switch ( type ) {
121
122            /* option 1: addition */
123            case 1:
124                operator = '+';
125                answer = x + y;
126                break; /* exit switch */
127
128            /* option 2: subtraction */
129            case 2:
130                operator = '-';
131
```

```
132     answer = x - y;
133     break; /* exit switch */
134
135     /* option 3: multiplication */
136     case 3:
137         operator = '*';
138         answer = x * y;
139         break; /* exit switch */
140
141     /* option 4: integer division */
142     case 4:
143         operator = '/';
144
145         /* eliminate divide by zero error */
146         if ( y == 0 ) {
147             y = 1;
148             answer = x / y;
149         } /* end if */
150         else {
151             x *= y; /* create "nice" division */
152             answer = x / y;
153         } /* end else */
154
155         break; /* exit switch */
156 } /* end switch */
157
158 printf( "How much is %d %c %d? ", x, operator, y );
159
160 scanf( "%d", &response );
161
162 /* while not correct answer */
163 while ( response != answer ) {
164     ++wrong;
165     incorrectMessage();
166     scanf( "%d", &response );
167 } /* end while */
168
169 ++right;
170 correctMessage();
171 } /* end for */
172
173 /* if < 75% right, suggest help */
174 if ( ( double ) right / ( right + wrong ) < .75 ) {
175     printf( "Please ask your instructor for extra help.\n" );
176 } /* end if */
177
178 printf( "That's all for now. Bye.\n" );
179 } /* end function arithmetic */
```

```
Choose type of problem to study.  
Enter: 1 for addition, 2 for subtraction  
Enter: 3 for multiplication, 4 for division  
Enter: 5 for a combination of 1 through 4  
? 5  
How much is 9 * 9? 81  
Nice work!  
  
How much is 3 - 1? 2  
Keep up the good work!  
  
How much is 1 * 3? 3  
Nice work!  
.  
.  
.  
How much is 1 - 9? -8  
Nice work!  
  
That's all for now. Bye.
```

5.45 Write function `distance` that calculates the distance between two points $(x1, y1)$ and $(x2, y2)$. All numbers and return values should be of type `double`.

ANS:

```

1  /* Exercise 5.45 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  /* function prototype */
6  double distance( double xOne, double yOne, double xTwo, double yTwo );
7
8  int main()
9  {
10     double x1; /* x coordinate of first point */
11     double y1; /* y coordinate of first point */
12     double x2; /* x coordinate of second point */
13     double y2; /* y coordinate of second point */
14     double dist; /* distance between two points */
15
16     /* prompt for first point coordinates */
17     printf( "Enter the first point: " );
18     scanf( "%lf%lf", &x1, &y1 );
19
20     /* prompt for second point coordinates */
21     printf( "Enter the second point: " );
22     scanf( "%lf%lf", &x2, &y2 );
23
24     dist = distance( x1, y1, x2, y2 ); /* calculate distance */
25
26     printf( "Distance between ( %.2f, %.2f )"
27            " and ( %.2f, %.2f ) is %.2f\n",
28            x1, y1, x2, y2, dist );
29
30     return 0; /* indicate successful termination */
31 }
32 } /* end main */
33
34 /* distance calculates distance between 2 points
35    given by (xOne, yOne) and (xTwo, yTwo) */
36 double distance( double xOne, double yOne, double xTwo, double yTwo )
37 {
38     double distance; /* distance between two points */
39
40     distance = sqrt( pow( xOne - xTwo, 2 ) + pow( yOne - yTwo, 2 ) );
41
42     return distance;
43 }
44 } /* end function distance */

```

```

Enter the first point: 3 4
Enter the second point: 0 0
Distance between ( 3.00, 4.00 ) and ( 0.00, 0.00 ) is 5.00

```

5.46 What does the following program do?

```
1  #include <stdio.h>
2
3  /* function main begins program execution */
4  int main()
5  {
6      int c; /* variable to hold character input by user */
7
8      if ( ( c = getchar() ) != EOF ) {
9          main();
10         printf( "%c", c );
11     } /* end if */
12
13     return 0; /* indicates successful termination */
14
15 }
```

ANS: Inputs a character and recursively calls `main()` until the EOF character is entered. Every character entered is then output in reverse order.

a b c

c b a

5.47 What does the following program do?

```
1 #include <stdio.h>
2
3 int mystery( int a, int b ); /* function prototype */
4
5 /* function main begins program execution */
6 int main()
7 {
8     int x; /* first integer */
9     int y; /* second integer */
10
11     printf( "Enter two integers: " );
12     scanf( "%d%d", &x, &y );
13
14     printf( "The result is %d\n", mystery( x, y ) );
15
16     return 0; /* indicates successful termination */
17
18 } /* end main */
19
20 /* Parameter b must be a positive integer
21    to prevent infinite recursion */
22 int mystery( int a, int b )
23 {
24     /* base case */
25     if ( b == 1 ) {
26         return a;
27     } /* end if */
28     else { /* recursive step */
29         return a + mystery( a, b - 1 );
30     } /* end else */
31
32 } /* end function mystery */
```

ANS: The problem mimics multiplication by adding up a, b times.

```
Enter two integers: 87 6
The result is 522
```

5.48 After you determine what the program of Exercise 5.47 does, modify the program to function properly after removing the restriction of the second argument being nonnegative.

ANS:

```

1  /* Exercise 5.48 Solution */
2  #include <stdio.h>
3
4  int mystery( int a, int b ); /* function prototype */
5
6  int main()
7  {
8      int x; /* first integer */
9      int y; /* second integer */
10
11     printf( "Enter two integers: " );
12     scanf( "%d%d", &x, &y );
13
14     printf( "The result is %d\n", mystery( x, y ) );
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */
19
20 /* mystery multiplies a * b using recursion */
21 int mystery( int a, int b )
22 {
23
24     /* if a and b or just b are negative */
25     if ( ( a < 0 && b < 0 ) || b < 0 ) {
26         a *= -1; /* multiply a and b by -1 to make positive */
27         b *= -1;
28     } /* end if */
29
30     /* base case */
31     if ( b == 1 ) {
32         return a;
33     } /* end if */
34     else { /* recursive step */
35         return a + mystery( a, b - 1 );
36     } /* end else */
37
38 } /* end function mystery */

```

```

Enter two integers: -97 6
The result is -582

```

```

Enter two integers: 97 -6
The result is -582

```

```

Enter two integers: -97 -6
The result is 582

```

5.49 Write a program that tests as many of the math library functions in Fig. 5.2 as you can. Exercise each of these functions by having your program print out tables of return values for a diversity of argument values.

ANS:

```

1  /* Exercise 5.49 Solution */
2  #include <stdio.h>
3  #include <math.h>
4
5  int main()
6  {
7      int loop;    /* integer loop counter */
8      int count;  /* loop counter */
9      double loop2; /* double loop counter */
10
11     /* loop and test each math function */
12     for ( count = 1; count < 14; count++) {
13
14         /* test math function based on count */
15         switch ( count ) {
16
17             /* print table headers */
18             case 1:
19                 printf( "func\t\t\t\t\t" );
20
21                 for ( loop = 1; loop < 6; loop++ ) {
22                     printf( "%10d\t", loop );
23                 } /* end for */
24
25                 break; /* exit switch */
26
27             /* display sqrt for range of values */
28             case 2:
29                 printf( "\nsqrt()\t\t\t\t\t" );
30
31                 for ( loop = 1; loop < 6; loop++ ) {
32                     printf( "%10.21f\t", sqrt( loop ) );
33                 } /* end for */
34
35                 break; /* exit switch */
36
37             /* display exp for range of values */
38             case 3:
39                 printf( "exp()\t\t\t\t\t" );
40
41                 for ( loop = 1; loop < 6; loop++ ) {
42                     printf( "%10.21f\t", exp( loop ) );
43                 } /* end for */
44
45                 break; /* exit switch */
46
47             /* display natural log for range of values */
48             case 4:
49                 printf( "log()\t\t\t\t\t" );
50
51                 for ( loop = 1; loop < 6; loop++ ) {
52                     printf( "%10.21f\t", log( loop ) );
53                 } /* end for */
54
55                 break; /* exit switch */
56
57             /* display log base 10 for range of values */
58             case 5:
59                 printf( "log10()\t\t\t\t\t" );
60
61                 for ( loop = 1; loop < 6; loop++ ) {
62                     printf( "%10.21f\t", log10( loop ) );
63                 } /* end for */

```

```
64
65         break; /* exit switch */
66
67     /* display pow function, test with 2 as base */
68     case 6:
69         printf( "pow( 2,x )" );
70
71         for ( loop = 1; loop < 6; loop++ ) {
72             printf( "%10.21f ", pow( 2, loop ) );
73         } /* end for */
74
75         break; /* exit switch */
76
77     /* display table headers */
78     case 7:
79         printf( "\n\nfunct      " );
80
81         for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
82             printf( "%10.21f ", loop2 );
83         } /* end for */
84
85         break; /* exit switch */
86
87     /* display fabs for range of values */
88     case 8:
89         printf( "\nfabs()      " );
90
91         for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
92             printf( "%10.21f ", fabs( loop2 ) );
93         } /* end for */
94
95         break; /* exit switch */
96
97     /* display ceil for range of values */
98     case 9:
99         printf( "ceil()      " );
100
101         for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
102             printf( "%10.21f ", ceil( loop2 ) );
103         } /* end for */
104
105         break; /* exit switch */
106
107     /* display floor for range of values */
108     case 10:
109         printf( "floor()     " );
110
111         for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
112             printf( "%10.21f ", floor( loop2 ) );
113         } /* end for */
114
115         break; /* exit switch */
116
117     /* display sin for range of values */
118     case 11:
119         printf( "sin()       " );
120
121         for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
122             printf( "%10.21f ", sin( loop2 ) );
123         } /* end for */
124
125         break; /* exit switch */
126
127     /* display cos for range of values */
128     case 12:
129         printf( "cos()       " );
130
```

```

131     for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
132         printf( "%10.21f ", cos( loop2 ) );
133     } /* end for */
134
135     break; /* exit switch */
136
137     /* display tan for range of values */
138     case 13:
139         printf( "tan()      " );
140
141         for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
142             printf( "%10.21f ", tan( loop2 ) );
143         } /* end for */
144
145         break; /* exit switch */
146     } /* end switch */
147
148     printf( "\n" );
149 } /* end for */
150
151 return 0; /* indicate successful termination */
152
153 } /* end main */

```

func	1	2	3	4	5
sqrt()	1.00	1.41	1.73	2.00	2.24
exp()	2.72	7.39	20.09	54.60	148.41
log()	0.00	0.69	1.10	1.39	1.61
log10()	0.00	0.30	0.48	0.60	0.70
pow(2,x)	2.00	4.00	8.00	16.00	32.00
func	-1.50	-0.40	0.70	1.80	2.90
fabs()	1.50	0.40	0.70	1.80	2.90
ceil()	-1.00	0.00	1.00	2.00	3.00
floor()	-2.00	-1.00	0.00	1.00	2.00
sin()	-1.00	-0.39	0.64	0.97	0.24
cos()	0.07	0.92	0.76	-0.23	-0.97
tan()	-14.10	-0.42	0.84	-4.29	-0.25

5.50 Find the error in each of the following program segments and explain how to correct it:

```

a) double cube( float ); /* function prototype */
...
cube( float number ) /* function definition */
{
    return number * number * number;
}

```

ANS: Function definition is missing return type.

```

double cube( float ); /* function prototype */
...
double cube( float number ) /* function definition */
{
    return number * number * number;
}

```

b) register auto int x = 7;

ANS: Too many storage class definitions. Auto class definition is not necessary.

```
register int x = 7; /* auto removed */
```

c) int randomNumber = srand();

ANS: srand() seeds the random number generator, and has a void return type. Function rand() produces random numbers.

```
int randomNumber = rand();
d) double y = 123.45678;
   int x;
   x = y;
   printf( "%f\n", (double) x );
```

ANS: Decimal value is lost when a double is assigned to an integer. Type-casting the int to double cannot bring back the original decimal value. Only 123.000000 can be printed.

```
double y = 123.45678;
double x;

x = y;
printf( "%f\n", x );
e) double square( double number )
   {
       double number;

       return number * number;
   }
```

ANS: number is defined twice.

```
double square( double number )
{
    return number * number;
}
f) int sum( int n )
   {
       if ( n == 0 )
           return 0;
       else
           return n + sum( n );
   }
```

ANS: Infinite recursion.

```
int sum( int n )
{
    if ( n == 0 )
        return 0;
    else
        return n + sum( n - 1 );
}
```

5.51 Modify the craps program of Fig. 5.10 to allow wagering. Package as a function the portion of the program that runs one game of craps. Initialize variable `bankBalance` to 1000 dollars. Prompt the player to enter a wager. Use a `while` loop to check that wager is less than or equal to `bankBalance` and if not prompt the user to reenter wager until a valid wager is entered. After a correct wager is entered, run one game of craps. If the player wins, increase `bankBalance` by wager and print the new `bankBalance`. If the player loses, decrease `bankBalance` by wager, print the new `bankBalance`, check if `bankBalance` has become zero, and if so print the message "Sorry. You busted!". As the game progresses, print various messages to create some "chatter" such as "Oh, you're going for broke, huh?", or "Aw cmon, take a chance!", or "You're up big. Now's the time to cash in your chips!".

ANS:

```

1  /* Exercise 5.51 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* enumeration constants represent game status */
7  enum Status { CONTINUE, WON, LOST };
8
9  int rollDice( void ); /* function prototype */
10 enum Status craps( void ); /* function prototype */
11 void chatter( void ); /* function prototype */
12
13 int main()
14 {
15     enum Status result; /* result of current game */
16     int wager = 0; /* wager for current game */
17     int bankBalance = 1000; /* current bank balance */
18
19     srand( time( NULL ) ); /* seed random number generator */
20
21     /* display current balance and prompt for wager */
22     printf( "You have $%d in the bank.\n", bankBalance );
23     printf( "Place your wager: " );
24     scanf( "%d", &wager );
25
26     /* loop while not valid wager */
27     while( wager <= 0 || wager > 1000 ) {
28         printf( "Please bet a valid amount.\n" );
29         scanf( "%d", &wager );
30     } /* end while */
31
32     result = craps(); /* play game of craps */
33
34     /* if player lost current game */
35     if ( result == LOST ) {
36
37         /* decrease balance by wager and display new balance */
38         bankBalance -= wager;
39         printf( "Your new bank balance is $%d\n", bankBalance );
40
41         /* if balance is 0 */
42         if ( bankBalance == 0 ) {
43             printf( "Sorry. You are Busted! Thank You For Playing.\n" );
44         } /* end if */
45     } /* end if */
46     else { /* player won game */
47
48         /* increase balance by wager and display new balance */
49         bankBalance += wager;
50         printf( "Your new bank balance is $%d\n", bankBalance );
51     } /* end else */
52
53     return 0; /* indicate successful termination */
54 } /* end main */
55
56

```

```

57
58 /* roll dice, calculate sum and display results */
59 int rollDice( void )
60 {
61     int die1; /* first die value */
62     int die2; /* second die value */
63     int workSum; /* sum of dice */
64
65     die1 = 1 + rand() % 6; /* pick random die1 value */
66     die2 = 1 + rand() % 6; /* pick random die2 value */
67     workSum = die1 + die2; /* sum die1 and die2 */
68
69     /* display results of this roll */
70     printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
71
72     return workSum; /* return sum of dice */
73
74 } /* end function rollDice */
75
76 /* craps plays one game of craps, returns result of game */
77 enum Status craps( void )
78 {
79     enum Status gameStatus; /* can contain CONTINUE, WON or LOST */
80     int sum; /* current roll of dice */
81     int myPoint; /* point value */
82
83     sum = rollDice(); /* first roll of dice */
84
85     /* determine game status and point based on sum of dice */
86     switch ( sum ) {
87
88         /* win on first roll */
89         case 7:
90         case 11:
91             gameStatus = WON;
92             chatter();
93             break; /* exit switch */
94
95         /* lose on first roll */
96         case 2:
97         case 3:
98         case 12:
99             gameStatus = LOST;
100            chatter();
101            break; /* exit switch */
102
103            /* remember point */
104            default:
105                gameStatus = CONTINUE;
106                myPoint = sum;
107                printf( "Point is %d\n", myPoint );
108                chatter();
109                break; /* exit switch */
110        } /* end switch */
111
112        /* while game not complete */
113        while ( gameStatus == CONTINUE ) {
114            chatter();
115            sum = rollDice(); /* roll dice again */
116
117            /* determine game status */
118            if ( sum == myPoint ) {
119                gameStatus = WON; /* win by making point */
120            } /* end if */
121            else {
122
123                if ( sum == 7 ) {
124                    gameStatus = LOST; /* lose by rolling 7 */
125                } /* end if */

```



```
126     } /* end else */
127
128
129 } /* end while */
130
131 /* display won or lost message and return status */
132 if ( gameStatus == WON ) {
133     printf( "Player wins\n" );
134     return WON;
135 } /* end if */
136 else {
137     printf( "Player loses\n" );
138     return LOST;
139 } /* end else */
140
141 } /* end function craps */
142
143 /* chatter displays messages at random to create "chatter" */
144 void chatter( void )
145 {
146     int select; /* random number */
147
148     select = 1 + rand() % 6;
149
150     /* choose message at random */
151     switch ( select ) {
152
153     case 1:
154         printf( "Oh, you're going for broke, huh?\n" );
155         break; /* exit switch */
156
157     case 2:
158         printf( "Aw cmon, take a chance!\n" );
159         break; /* exit switch */
160
161     case 3:
162         printf( "Hey, I think this guy is going to break the bank!!\n" );
163         break; /* exit switch */
164
165     case 4:
166         printf( "You're up big. Now's the time to cash in your chips!\n" );
167         break; /* exit switch */
168
169     case 5:
170         printf( "Way too lucky! Those dice have to be loaded!\n" );
171         break; /* exit switch */
172
173     case 6:
174         printf( "Bet it all! Bet it all!\n" );
175         break; /* exit switch */
176
177     default:
178         break; /* exit switch */
179     } /* end switch */
180
181 } /* end function chatter */
```

```
You have $1000 in the bank.  
Place your wager: 1000  
Player rolled 4 + 5 = 9  
Point is 9  
You're up big. Now's the time to cash in your chips!  
Oh, you're going for broke, huh?  
Player rolled 5 + 6 = 11  
Hey, I think this guy is going to break the bank!!  
Player rolled 3 + 1 = 4  
Bet it all! Bet it all!  
Player rolled 5 + 5 = 10  
Aw cmon, take a chance!  
Player rolled 6 + 6 = 12  
Bet it all! Bet it all!  
Player rolled 2 + 1 = 3  
Hey, I think this guy is going to break the bank!!  
Player rolled 5 + 6 = 11  
Hey, I think this guy is going to break the bank!!  
Player rolled 2 + 1 = 3  
Aw cmon, take a chance!  
Player rolled 2 + 4 = 6  
You're up big. Now's the time to cash in your chips!  
Player rolled 2 + 3 = 5  
Oh, you're going for broke, huh?  
Player rolled 6 + 3 = 9  
Player wins  
Your new bank balance is $2000
```


6

C Arrays: Solutions

EXERCISES

6.6 Fill in the blanks in each of the following:

a) C stores lists of values in _____.

ANS: arrays.

b) The elements of an array are related by the fact that they _____.

ANS: have the same name and type.

c) When referring to an array element, the position number contained within parentheses is called a(n) _____.

ANS: subscript.

d) The names of the five elements of array `p` are _____, _____, _____, _____ and _____.

ANS: `p[0]`, `p[1]`, `p[2]`, `p[3]`, `p[4]`.

e) The contents of a particular element of an array is called the _____ of that element.

ANS: value.

f) Naming an array, stating its type and specifying the number of elements in the array is called _____ the array.

ANS: defining.

g) The process of placing the elements of an array into either ascending or descending order is called _____.

ANS: sorting.

h) In a double-subscripted array, the first subscript (by convention) identifies the _____ of an element and the second subscript (by convention) identifies the _____ of an element.

ANS: row, column.

i) An m -by- n array contains _____ rows, _____ columns and _____ elements.

ANS: m , n , $m * n$.

j) The name of the element in row 3 and column 5 of array `d` is _____.

ANS: `d[3][5]`.

6.7 State which of the following are *true* and which are *false*. If *false*, explain why.

a) To refer to a particular location or element within an array, we specify the name of the array and the value of the particular element.

ANS: False. We specify the name and the subscript of the element.

b) An array definition reserves space for the array.

ANS: True.

c) To indicate that 100 locations should be reserved for integer array `p`, the programmer writes the definition

```
p[ 100 ];
```

ANS: True.

d) A C program that initializes the elements of a 15-element array to zero must contain one `for` statement.

ANS: False. The elements of an array can be initialized in the definition.

e) A C program that totals the elements of a double-subscripted array must contain nested `for` statements.

ANS: False. It is possible to total the elements of a double-subscripted array by enumerating all the elements in an assignment statement.

f) The mean, median and mode of the following set of values are 5, 6 and 7, respectively: 1, 2, 5, 6, 7, 7, 7.

ANS: True.

6.8 Write statements to accomplish each of the following:

a) Display the value of the seventh element of character array `f`.

ANS: `printf("%c\n", f[6]);`

b) Input a value into element 4 of single-subscripted floating-point array `b`.

ANS: `scanf("%f", &b[4]);`

c) Initialize each of the 5 elements of single-subscripted integer array `g` to 8.

ANS:

```
for ( loop = 0; loop <= 4; loop++ )
    g[ loop ] = 8;
```

d) Total the elements of floating-point array `c` of 100 elements.

ANS:

```
for ( loop = 0; loop <= 99; loop++ )
    sum += c[ loop ];
```

e) Copy array `a` into the first portion of array `b`. Assume `double a[11], b[34]`;

ANS:

```
for ( loop = 0; loop <= 10; loop++ )
    b[ loop ] = a[ loop ];
```

f) Determine and print the smallest and largest values contained in 99-element floating-point array `w`.

ANS:

```
smallest = largest = w[ 0 ];

for ( loop = 1; loop <= 98; loop++ )
    if ( w[ loop ] < smallest )
        smallest = w[ loop ];
    else if ( w[ loop ] > largest )
        largest = w[ loop ];
```

6.9 Consider a 2-by-5 integer array `t`.

a) Write a definition for `t`.

ANS: `int t[2][5];`

b) How many rows does `t` have?

ANS: 2

c) How many columns does `t` have?

ANS: 5

d) How many elements does `t` have?

ANS: 10

e) Write the names of all the elements in the second row of `t`.

ANS: `t[1][0]`, `t[1][1]`, `t[1][2]`, `t[1][3]`, `t[1][4]`.

f) Write the names of all the elements in the third column of `t`.

ANS: `t[0][2]`, `t[1][2]`.

g) Write a single statement that sets the element of `t` in row 1 and column 2 to zero.

ANS: `t[1][2] = 0;`

h) Write a series of statements that initialize each element of `t` to zero. Do not use a repetition structure.

ANS:

```
t[ 0 ][ 0 ] = 0;
t[ 0 ][ 1 ] = 0;
t[ 0 ][ 2 ] = 0;
t[ 0 ][ 3 ] = 0;
t[ 0 ][ 4 ] = 0;
t[ 1 ][ 0 ] = 0;
```

```
t[ 1 ][ 1 ] = 0;
t[ 1 ][ 2 ] = 0;
t[ 1 ][ 3 ] = 0;
t[ 1 ][ 4 ] = 0;
```

i) Write a nested for statement that initializes each element of t to zero.

ANS:

```
for ( i = 0; i <= 1; i++ )
    for ( j = 0; j <= 4; j++ )
        t[ i ][ j ] = 0;
```

j) Write a statement that inputs the values for the elements of t from the terminal.

ANS:

```
for ( i = 0; i <= 1; i++ )
    for ( j = 0; j <= 4; j++ ) {
        printf( "Enter an integer: " );
        scanf( "%d", &t[ i ][ j ] )
    }
```

k) Write a series of statements that determine and print the smallest value in array t.

ANS:

```
smallest = t[ 0 ][ 0 ];

for ( i = 0; i <= 1; i++ )
    for ( j = 0; j <= 4; j++ )
        if ( t[ i ][ j ] < smallest )
            smallest = t[ i ][ j ];

printf( " smallest is %d\n", smallest );
```

l) Write a statement that displays the elements of the first row of t.

ANS:

```
for ( i = 0; i <= 4; i++ )
    printf( "%d ", t[ 0 ][ i ] );
```

m) Write a statement that totals the elements of the fourth column of t.

ANS: sum = t[0][3] + t[1][3];

n) Write a series of statements that print the array t in tabular format. List the column subscripts as headings across the top and list the row subscripts at the left of each row.

ANS:

```
printf( "  0\t1\t2\t3\t4\n" );

for ( i = 0; i <= 1; i++ ) {
    printf( "%d ", i );

    for ( j = 0; j <= 4; j++ )
        printf( "%d\t", t[ i ][ j ] );

    printf( "\n" );
}
```

6.10 Use a single-subscripted array to solve the following problem. A company pays its salespeople on a commission basis. The salespeople receive \$200 per week plus 9 percent of their gross sales for that week. For example, a salesperson who grosses \$3000 in sales in a week receives \$200 plus 9 percent of \$3000, or a total of \$470. Write a C program (using an array of counters) that determines how many of the salespeople earned salaries in each of the following ranges (assume that each salesperson's salary is truncated to an integer amount):

- a) \$200–299
- b) \$300–399
- c) \$400–499
- d) \$500–599
- e) \$600–699
- f) \$700–799
- g) \$800–899
- h) \$900–999
- i) \$1000 and over

ANS:

```

1  /* Exercise 6.10 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int salaries[ 11 ] = { 0 }; /* array to hold salary counts */
7      int sales;                 /* current employee's sales */
8      double salary;            /* current employee's salary */
9      double i = 0.09;         /* commission percentage */
10
11     /* prompt user for gross sales */
12     printf( "Enter employee gross sales ( -1 to end ): " );
13     scanf( "%d", &sales );
14
15     /* while sentinel value not read from user */
16     while ( sales != -1 ) {
17
18         /* calculate salary based on sales */
19         salary = 200.0 + sales * i;
20         printf( "Employee Commission is $%.2f\n", salary );
21
22         /* update appropriate salary range */
23         if ( salary >= 200 && salary < 1000 ) {
24             ++salaries[ ( int ) salary / 100 ];
25         } /* end if */
26         else if ( salary >= 1000 ) {
27             ++salaries[ 10 ];
28         } /* end else if */
29
30         /* prompt user for another employee sales amount */
31         printf( "\nEnter employee gross sales ( -1 to end ): " );
32         scanf( "%d", &sales );
33     } /* end while */
34
35     /* display table of ranges and employees in each range */
36     printf( "\nEmployees in the range:\n" );
37     printf( "$200-$299 : %d\n", salaries[ 2 ] );
38     printf( "$300-$399 : %d\n", salaries[ 3 ] );
39     printf( "$400-$499 : %d\n", salaries[ 4 ] );
40     printf( "$500-$599 : %d\n", salaries[ 5 ] );
41     printf( "$600-$699 : %d\n", salaries[ 6 ] );
42     printf( "$700-$799 : %d\n", salaries[ 7 ] );
43     printf( "$800-$899 : %d\n", salaries[ 8 ] );
44     printf( "$900-$999 : %d\n", salaries[ 9 ] );
45     printf( "Over $1000: %d\n", salaries[ 10 ] );
46
47     return 0; /* indicate successful termination */
48
49 } /* end main */

```

```
Enter employee gross sales ( -1 to end ): 3000
Employee Commission is $470.00

Enter employee gross sales ( -1 to end ): 1000
Employee Commission is $290.00

Enter employee gross sales ( -1 to end ): 10000
Employee Commission is $1100.00

Enter employee gross sales ( -1 to end ): 8000
Employee Commission is $920.00

Enter employee gross sales ( -1 to end ): 200
Employee Commission is $218.00

Enter employee gross sales ( -1 to end ): 7000
Employee Commission is $830.00

Enter employee gross sales ( -1 to end ): -1

Employees in the range:
$200-$299 : 2
$300-$399 : 0
$400-$499 : 1
$500-$599 : 0
$600-$699 : 0
$700-$799 : 0
$800-$899 : 1
$900-$999 : 1
Over $1000: 1
```


6.11 The bubble sort presented in Fig. 6.15 is inefficient for large arrays. Make the following simple modifications to improve the performance of the bubble sort.

- After the first pass, the largest number is guaranteed to be in the highest-numbered element of the array; after the second pass, the two highest numbers are “in place,” and so on. Instead of making nine comparisons on every pass, modify the bubble sort to make eight comparisons on the second pass, seven on the third pass and so on.
- The data in the array may already be in the proper order or near-proper order, so why make nine passes if fewer will suffice? Modify the sort to check at the end of each pass if any swaps have been made. If none has been made, then the data must already be in the proper order, so the program should terminate. If swaps have been made, then at least one more pass is needed.

ANS:

```

1  /* Exercise 6.11 Solution */
2  #include <stdio.h>
3  #define MAX 10
4
5  int main()
6  {
7
8      /* initialize array a with initializer list */
9      int a[ MAX ] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
10     int i; /* loop counter */
11     int pass; /* loop counter */
12     int hold; /* temporary variable for swapping */
13     int swap; /* flag to break loop if elements are sorted */
14
15     printf( "Data items in original order\n" );
16
17     /* display original, unsorted array */
18     for ( i = 0; i < MAX; i++ ) {
19         printf( "%4d", a[ i ] );
20     } /* end for */
21
22     printf( "\n\n" );
23
24     /* begin sorting the array */
25     for ( pass = 1; pass < MAX; pass++ ) {
26         swap = 0;
27
28         /* traverse and compare unsorted part of array */
29         for ( i = 0; i < MAX - pass; i++ ) {
30
31             /* compare adjacent array elements */
32             if ( a[ i ] > a[ i + 1 ] ) {
33                 swap = 1; /* raise flag if any elements are swapped */
34                 hold = a[ i ];
35                 a[ i ] = a[ i + 1 ];
36                 a[ i + 1 ] = hold;
37             } /* end if */
38
39         } /* end for */
40
41         printf( "After Pass %d: ", pass );
42
43         /* display array after each pass */
44         for ( i = 0; i <= MAX-pass; i++ ) {
45             printf( " %d", a[ i ] );
46         } /* end for */
47
48         printf( "\n" );
49

```

```

50     /* break loop if array is sorted */
51     if ( !swap ) {
52         break;
53     } /* end if */
54
55 } /* end for */
56
57 printf( "\nData items in ascending order\n" );
58
59 /* display array in sorted order */
60 for ( i = 0; i < 10; i++ ) {
61     printf( "%4d", a[ i ] );
62 } /* end for */
63
64 printf( "\n" );
65
66 return 0; /* indicate successful termination */
67
68 } /* end main */

```

```

Data items in original order
10  9  8  7  6  5  4  3  2  1

After Pass 1:  9  8  7  6  5  4  3  2  1 10
After Pass 2:  8  7  6  5  4  3  2  1  9
After Pass 3:  7  6  5  4  3  2  1  8
After Pass 4:  6  5  4  3  2  1  7
After Pass 5:  5  4  3  2  1  6
After Pass 6:  4  3  2  1  5
After Pass 7:  3  2  1  4
After Pass 8:  2  1  3
After Pass 9:  1  2

Data items in ascending order
 1  2  3  4  5  6  7  8  9 10

```

6.12 Write single statements that perform each of the following single-subscripted array operations:

- a) Initialize the 10 elements of integer array `counts` to zeros.

ANS:

```
for ( i = 0; i <= 9; i++ )
    counts[ i ] = 0;
```

- b) Add 1 to each of the 15 elements of integer array `bonus`.

ANS:

```
for ( i = 0; i <= 14; i++ )
    ++bonus[ i ];
```

- c) Read the 12 values of floating-point array `monthlyTemperatures` from the keyboard.

ANS:

```
for ( i = 0; i <= 11; i++ ) {
    printf( "Enter a temperature: " );
    scanf( "%f", &monthlyTemperatures[ i ] );
}
```

- d) Print the 5 values of integer array `bestScores` in column format.

ANS:

```
for ( i = 0; i <= 4; i++ ) {
    printf( "%d\t", bestScores[ i ] );
}
```

6.13 Find the error(s) in each of the following statements:

a) Assume: `char str[5];`
`scanf("%s", str);` */* User types hello */*

ANS: `str` needs a minimum length of 6; one element for each letter in `hello` and an element for the terminating NULL character.

b) Assume: `int a[3];`
`printf("$d %d %d\n", a[1], a[2], a[3]);`

ANS: `printf("%d %d %d\n", a[0], a[1], a[2]);`

c) `double f[3] = { 1.1, 10.01, 100.001, 1000.0001 };`

ANS: Too many variables defined.

`double f[3] = { 1.1, 10.01, 100.01 };`

d) Assume: `double d[2][10];`

`d[1, 9] = 2.345;`

ANS: `d[1][9] = 2.345;`

6.14 Modify the program of Fig. 6.16 so function `mode` is capable of handling a tie for the mode value. Also modify function `median` so the two middle elements are averaged in an array with an even number of elements.

ANS:

```

1  /* Exercise 6.14 Solution */
2  #include <stdio.h>
3  #define SIZE 100
4
5  void mean( int answer[] );           /* function prototype */
6  void median( int answer[] );        /* function prototype */
7  void mode( int freq[], int answer[] ); /* function prototype */
8
9  int main()
10 {
11
12     /* array of responses */
13     int response[ SIZE ] = { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
14                             7, 8, 9, 5, 9, 8, 7, 8, 7, 1,
15                             6, 7, 8, 9, 3, 9, 8, 7, 1, 7,
16                             7, 8, 9, 8, 9, 8, 9, 7, 1, 9,
17                             6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
18                             7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
19                             5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
20                             7, 8, 9, 6, 8, 7, 8, 9, 7, 1,
21                             7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
22                             4, 5, 6, 1, 6, 5, 7, 8, 7, 9};
23     int frequency[ 10 ] = { 0}; /* array of response frequencies */
24
25     mean( response ); /* process mean */
26     median( response ); /* process median */
27     mode( frequency, response ); /* process mode */
28
29     return 0; /* indicates successful termination */
30 } /* end main */
31
32 /* calculate average of all response values */
33 void mean( int answer[] )
34 {
35     int j; /* loop counter */
36     int total = 0; /* total of all response values */
37
38     printf( "%s\n%s\n%s\n", "*****", " Mean", "*****" );
39
40     /* total response values */
41     for ( j = 0; j <= SIZE - 1; j++ ) {
42         total += answer[ j ];
43     } /* end for */
44
45     /* output results */
46     printf( "The mean is the average value of the data\n" );
47     printf( "items. The mean is equal to the total of\n" );
48     printf( "all the data items divided by the number\n" );
49     printf( "of data items ( %d ).", SIZE );
50     printf( "The mean value for this run is: " );
51     printf( "%d / %d = %.4f\n\n", total, SIZE, ( double ) total / SIZE );
52 } /* end function mean */
53
54 /*sort an array and determine median element's value */
55 void median( int answer[] )
56 {
57     int loop; /* loop counter */
58     int pass; /* loop counter */
59     int hold; /* temporary variable for swapping */
60     int firstRow; /* flag to indicate first row of array */
61
62     printf( "\n%s\n%s\n%s\n", "*****", "Median", "*****" );

```

```

64     printf( "The unsorted array of responses is\n" );
65
66     /* display unsorted array */
67     for ( loop = 0, firstRow = 1; loop <= SIZE - 1; loop++ ) {
68
69         /* start a new line */
70         if ( loop % 20 == 0 && !firstRow ) {
71             printf( "\n" );
72         } /* end if */
73
74         printf( "%2d", answer[ loop ] );
75         firstRow = 0;
76     } /* end for */
77
78     printf( "\n\n" );
79
80     /* sort array */
81     for ( pass = 0; pass <= SIZE - 2; pass++ ) {
82
83         /* compare elements and swap if necessary */
84         for ( loop = 0; loop <= SIZE - 2; loop++ ) {
85
86             /* swap elements */
87             if ( answer[ loop ] > answer[ loop + 1 ] ) {
88                 hold = answer[ loop ];
89                 answer[ loop ] = answer[ loop + 1 ];
90                 answer[ loop + 1 ] = hold;
91             } /* end if */
92
93         } /* end for */
94
95     } /* end for */
96
97     printf( "The sorted array is\n" );
98
99     /* display sorted array */
100    for ( loop = 0, firstRow = 1; loop <= SIZE - 1; loop++ ) {
101
102        /* start a new line */
103        if ( loop % 20 == 0 && !firstRow ) {
104            printf( "\n" );
105        } /* end if */
106
107        printf( "%2d", answer[ loop ] );
108        firstRow = 0;
109    } /* end for */
110
111    printf( "\n\n" );
112
113    /* even number of elements */
114    if ( SIZE % 2 == 0 ) {
115        printf( "The median is the average of elements %d",
116              ( SIZE + 1 ) / 2 );
117        printf( " and %d of", 1 + ( SIZE + 1 ) / 2 );
118        printf( " the sorted %d element array.\n", SIZE );
119        printf( "For this run the median is " );
120        printf( "%.1f\n\n", ( double )( answer[ ( SIZE + 1 ) / 2 ] +
121                                       answer[ ( SIZE + 1 ) / 2 + 1 ] ) / 2 );
122    } /* end if */
123    else { /* odd number of elements */
124        printf( "The median is element %d of ", ( SIZE + 1 ) / 2 );
125        printf( "the sorted %d element array.\n", SIZE );
126        printf( "For this run the median is " );
127        printf( "%d\n\n", answer[ ( SIZE + 1 ) / 2 - 1 ] );
128    } /* end else */
129
130 } /* end function median */
131

```

```

132 /* determine most frequent response */
133 void mode( int freq[], int answer[] )
134 {
135     int rating;           /* loop counter */
136     int loop;            /* loop counter */
137     int largest = 0;     /* represents largest frequency */
138     int array[ 10 ] = { 0 }; /* array used to hold largest frequencies */
139     int count = 0;      /* flag to count number of modes */
140
141     printf( "\n%s\n%s\n", "*****", " Mode", "*****" );
142
143     /* set all frequencies to 0 */
144     for ( rating = 1; rating <= 9; rating++ ) {
145         freq[ rating ] = 0;
146     } /* end for */
147
148     /* traverse array and increment corresponding frequency */
149     for ( loop = 0; loop <= SIZE - 1; loop++ ) {
150         ++freq[ answer[ loop ] ];
151     } /* end for */
152
153     printf( "%s%11s%19s\n\n", "Response", "Frequency", "Histogram" );
154     printf( "%54s\n", "1   1   2   2" );
155     printf( "%54s\n", "5   0   5   0   5" );
156
157     /* display values and frequency */
158     for ( rating = 1; rating <= 9; rating++ ) {
159         printf( "%8d%11d", rating, freq[ rating ] );
160
161         /* test if current frequency is greater than largest frequency */
162         if ( freq[ rating ] > largest ) {
163             largest = freq[ rating ];
164
165             /* set values of array to 0 */
166             for ( loop = 0; loop < 10; loop++ ) {
167                 array[ loop ] = 0;
168             } /* end for */
169
170             /* add new largest frequency to array */
171             array[ rating ] = largest;
172             ++count;
173         } /* end if */
174         /* if current frequency equals largest, add current to array */
175         else if ( freq[ rating ] == largest ) {
176             array[ rating ] = largest;
177             ++count;
178         } /* end else if */
179
180         /* display histogram */
181         for ( loop = 1; loop <= freq[ rating ]; loop++ ) {
182             printf( "*" );
183         } /* end for */
184
185         printf( "\n" );
186     } /* end for */
187
188     printf( "\n" );
189
190     /* if more than one mode */
191     if ( count > 1 ) {
192         printf( "The modes are: " );
193     } /* end if */
194     else { /* only one mode */
195         printf( "The mode is: " );
196     } /* end else */
197
198     /* display mode(s) */
199     for ( loop = 1; loop <= 9; loop++ ) {

```

```

200
201     if ( array[ loop ] != 0 ) {
202         printf( "%d with a frequency of %d\n\t\t", loop, array[ loop ] );
203     } /* end if */
204
205 } /* end for */
206
207 printf( "\n" );
208 } /* end function mode */

```

Mean

The mean is the average value of the data items. The mean is equal to the total of all the data items divided by the number of data items (100). ,The mean value for this run is: 662 / 100 = 6.6200

Median

The unsorted array of responses is

```

6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 1
6 7 8 9 3 9 8 7 1 7 7 8 9 8 9 8 9 7 1 9
6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 1
7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7 9

```

The sorted array is

```

1 1 1 1 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5
5 5 5 5 5 6 6 6 6 6 6 6 6 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

```

The median is the average of elements 50 and 51 of the sorted 100 element array.

For this run the median is 7.0

Mode

Response	Frequency	Histogram
		5 1 1 2 2
		0 5 0 5
1	5	*****
2	3	***
3	4	****
4	5	*****
5	8	*****
6	9	*****
7	23	*****
8	23	*****
9	20	*****

The modes are: 7 with a frequency of 23
8 with a frequency of 23

6.15 Use a single-subscripted array to solve the following problem. Read in 20 numbers, each of which is between 10 and 100, inclusive. As each number is read, print it only if it is not a duplicate of a number already read. Provide for the “worst case” in which all 20 numbers are different. Use the smallest possible array to solve this problem.

ANS:

```

1  /* Exercise 6.15 Solution */
2  #include <stdio.h>
3  #define MAX 20
4
5  int main()
6  {
7      int a[ MAX ] = { 0 }; /* array for user input */
8      int i;                /* loop counter */
9      int j;                /* loop counter */
10     int k = 0;            /* number of values currently entered */
11     int duplicate;        /* flag for duplicate values */
12     int value;           /* current value */
13
14     printf( "Enter 20 integers between 10 and 100:\n" );
15
16     /* get 20 integers from user */
17     for ( i = 0; i <= MAX - 1; i++ ) {
18         duplicate = 0;
19         scanf( "%d", &value );
20
21         /* test if integer is a duplicate */
22         for ( j = 0; j < k; j++ ) {
23
24             /* if duplicate, raise flag and break loop */
25             if ( value == a[ j ] ) {
26                 duplicate = 1;
27                 break;
28             } /* end if */
29
30         } /* end for */
31
32         /* if number is not a duplicate enter it in array */
33         if ( !duplicate ) {
34             a[ k++ ] = value;
35         } /* end if */
36
37     } /* end for */
38
39     printf( "\nThe nonduplicate values are:\n" );
40
41     /* display array of nonduplicates */
42     for ( i = 0; a[ i ] != 0; i++ ) {
43         printf( "%d ", a[ i ] );
44     } /* end for */
45
46     printf( "\n" );
47
48     return 0; /* indicate successful termination */
49
50 } /* end main */

```

```

Enter 20 integers between 10 and 100:
10 11 12 13 14 15 16 17 18 19 20 21 10 11 12 13 14 15 16 17

The nonduplicate values are:
10 11 12 13 14 15 16 17 18 19 20 21

```


6.16 Label the elements of 3-by-5 double-subscripted array `sales` to indicate the order in which they are set to zero by the following program segment:

```
for ( row = 0; row <= 2; row++ )
    for ( column = 0; column <= 4; column++ )
        sales[ row ][ column ] = 0;
```

ANS:

- 1) sales[0][0]
- 2) sales[0][1]
- 3) sales[0][2]
- 4) sales[0][3]
- 5) sales[0][4]
- 6) sales[1][0]
- 7) sales[1][1]
- 8) sales[1][2]
- 9) sales[1][3]
- 10) sales[1][4]
- 11) sales[2][0]
- 12) sales[2][1]
- 13) sales[2][2]
- 14) sales[2][3]
- 15) sales[2][4]

6.17 What does the following program do?

```
1  /* ex06_17.c */
2  /* What does this program do? */
3  #include <stdio.h>
4  #define SIZE 10
5
6  int whatIsThis( const int b[], int p ); /* function prototype */
7
8  /* function main begins program execution */
9  int main()
10 {
11     int x; /* holds return value of function whatIsThis */
12
13     /* initialize array a */
14     int a[ SIZE ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
15
16     x = whatIsThis( a, SIZE );
17
18     printf( "Result is %d\n", x );
19
20     return 0; /* indicates successful termination */
21 } /* end main */
22
23
24 /* what does this function do? */
25 int whatIsThis( const int b[], int p )
26 {
27     /* base case */
28     if ( p == 1 ) {
29         return b[ 0 ];
30     } /* end if */
31     else { /* recursion step */
32
33         return b[ p - 1 ] + whatIsThis( b, p - 1 );
34     } /* end else */
35 }
36 } /* end function whatIsThis */
```

ANS: The program recursively sums the elements in a.

Result is 55

6.18 What does the following program do?

```
1  /* ex06_18.c */
2  /* What does this program do? */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function prototype */
7  void someFunction( const int b[], int startIndex, int size );
8
9  /* function main begins program execution */
10 int main()
11 {
12     int a[ SIZE ] = { 8, 3, 1, 2, 6, 0, 9, 7, 4, 5 }; /* initialize a */
13
14     printf( "Answer is:\n" );
15     someFunction( a, 0, SIZE );
16     printf( "\n" );
17
18     return 0; /* indicates successful termination */
19
20 } /* end main */
21
22 /* What does this function do? */
23 void someFunction( const int b[], int startIndex, int size )
24 {
25     if ( startIndex < size ) {
26         someFunction( b, startIndex + 1, size );
27         printf( "%d ", b[ startIndex ] );
28     } /* end if */
29
30 } /* end function someFunction */
```

ANS: The program recursively outputs the values of a in reverse order.

```
Answer is:
5 4 7 9 0 6 2 1 3 8
```

6.19 Write a program that simulates the rolling of two dice. The program should use `rand` to roll the first die, and should use `rand` again to roll the second die. The sum of the two values should then be calculated. [Note: Since each die can show an integer value from 1 to 6, then the sum of the two values will vary from 2 to 12 with 7 being the most frequent sum and 2 and 12 being the least frequent sums.] Figure 6.23 shows the 36 possible combinations of the two dice. Your program should roll the two dice 36,000 times. Use a single-subscripted array to tally the numbers of times each possible sum appears. Print the results in a tabular format. Also, determine if the totals are reasonable; i.e., there are six ways to roll a 7, so approximately one sixth of all the rolls should be 7.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Fig. 6.23 The 36 possible outcomes of rolling two dice.

ANS:

```

1  /* Exercise 6.19 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  int main()
7  {
8      long i;           /* loop counter */
9      int j;           /* loop counter */
10     int x;           /* first die */
11     int y;           /* second die */
12     int sum[ 13 ] = { 0 }; /* count occurrences of each combination */
13
14     /* array expected contains counts for the expected
15        number of times each sum occurs in 36 rolls of the dice */
16     int expected[ 13 ] = { 0, 0, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1 };
17
18     srand( time( NULL ) ); /* seed random number generator */
19
20     /* roll dice 36,000 times */
21     for ( i = 1; i <= 36000; i++ ) {
22         x = 1 + rand() % 6;
23         y = 1 + rand() % 6;
24         ++sum[ x + y ];
25     } /* end for */
26
27     printf( "%10s%10s%10s%10s\n", "Sum", "Total", "Expected", "Actual" );
28
29     /* display results of rolling dice */
30     for ( j = 2; j <= 12; j++ ) {
31         printf( "%10d%10d%9.3f%%%9.3f%%\n", j, sum[ j ],
32             100.0 * expected[ j ] / 36, 100.0 * sum[ j ] / 36000 );
33     } /* end for */
34
35     return 0; /* indicate successful termination */
36
37 } /* end main */

```

Sum	Total	Expected	Actual
2	1018	2.778%	2.828%
3	2008	5.556%	5.578%
4	3020	8.333%	8.389%
5	4024	11.111%	11.178%
6	4891	13.889%	13.586%
7	6011	16.667%	16.697%
8	5065	13.889%	14.069%
9	3984	11.111%	11.067%
10	2970	8.333%	8.250%
11	1989	5.556%	5.525%
12	1020	2.778%	2.833%

- 6.20** Write a program that runs 1000 games of craps (without human intervention) and answers each of the following questions:
- How many games are won on the first roll, second roll, ..., twentieth roll and after the twentieth roll?
 - How many games are lost on the first roll, second roll, ..., twentieth roll and after the twentieth roll?
 - What are the chances of winning at craps? [Note: You should discover that craps is one of the fairest casino games. What do you suppose this means?]
 - What is the average length of a game of craps?
 - Do the chances of winning improve with the length of the game?
- ANS:**

```

1  /* Exercise 6.20 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  enum Outcome { CONTINUE, WIN, LOSE };
7
8  int rollDice( void ); /* function prototype */
9
10 int main()
11 {
12     enum Outcome gameStatus; /* game status indicator */
13     int sum; /* sum of rolled dice */
14     int myPoint; /* current point */
15     int i; /* game counter */
16     int roll; /* roll counter */
17     int length = 0; /* average length of game */
18     int wins[ 22 ] = { 0 }; /* wins per roll */
19     int losses[ 22 ] = { 0 }; /* losses per roll */
20     int winSum = 0; /* total wins */
21     int loseSum = 0; /* total losses */
22
23     srand( time( NULL ) );
24
25     /* play 1000 times */
26     for ( i = 1; i <= 1000; i++ ) {
27         sum = rollDice();
28         roll = 1;
29
30         /* test if game won or lost on first roll */
31         switch ( sum ) {
32
33             case 7:
34             case 11:
35                 gameStatus = WIN;
36                 break; /* exit switch */
37
38             case 2:
39             case 3:
40             case 12:
41                 gameStatus = LOSE;
42                 break; /* exit switch */
43
44             default:
45                 gameStatus = CONTINUE;
46                 myPoint = sum;
47                 break; /* exit switch */
48         } /* end switch */
49
50         /* continue while game not won or lost */
51         while ( gameStatus == 0 ) {
52             sum = rollDice();
53             roll++;
54
55             /* win on next roll */
56             if ( sum == myPoint ) {
57                 gameStatus = WIN;
58             } /* end if */

```

```

59     else { /* lose on next roll */
60
61         if ( sum == 7 ) {
62             gameStatus = LOSE;
63         } /* end if */
64
65     } /* end else */
66
67 } /* end while */
68
69 /* if more than 21 rolls taken, set number of rolls to 21 */
70 if ( roll > 21 ) {
71     roll = 21;
72 } /* end if */
73
74 /* determine how many rolls were taken and increment
75    corresponding counter in wins or losses array */
76 if ( gameStatus == WIN ) {
77     wins[ roll ]++;
78     winSum++;
79 } /* end if */
80 else {
81     losses[ roll ]++;
82     loseSum++;
83 } /* end else */
84
85 } /* end for */
86
87 printf( "Games won or lost after the 20th roll\n"
88        "are displayed as the 21st roll.\n\n" );
89
90 /* display number of games won and lost for each number of rolls */
91 for ( i = 1; i <= 21; i++ ) {
92     printf( "%3d games won and %3d games lost on roll %d.\n",
93           wins[ i ], losses[ i ], i );
94 } /* end for */
95
96 /* calculate chances of winning */
97 printf( "\nThe chances of winning are %d/%d = %.2f%%\n", winSum,
98        winSum + loseSum, 100.0 * winSum / ( winSum + loseSum ) );
99
100 /* calculate average length of game */
101 for ( i = 1; i <= 21; i++ ) {
102     length += wins[ i ] * i + losses[ i ] * i;
103 } /* end for */
104
105 printf( "The average game length is %.2f rolls.\n",
106        length / 1000.0 );
107
108 return 0; /* indicate successful termination */
109
110 } /* end main */
111
112 /* function to simulate dice rolling */
113 int rollDice( void )
114 {
115     int die1; /* first die */
116     int die2; /* second die */
117     int workSum; /* dice sum */
118
119     die1 = 1 + rand() % 6;
120     die2 = 1 + rand() % 6;
121     workSum = die1 + die2;
122
123     return workSum; /* return total of two dice */
124 } /* end function rollDice */

```

Games won or lost after the 20th roll
are displayed as the 21st roll.

```
212 games won and 102 games lost on roll 1.  
63 games won and 109 games lost on roll 2.  
54 games won and 92 games lost on roll 3.  
45 games won and 70 games lost on roll 4.  
40 games won and 54 games lost on roll 5.  
17 games won and 34 games lost on roll 6.  
9 games won and 21 games lost on roll 7.  
10 games won and 11 games lost on roll 8.  
7 games won and 9 games lost on roll 9.  
3 games won and 2 games lost on roll 10.  
6 games won and 12 games lost on roll 11.  
4 games won and 4 games lost on roll 12.  
1 games won and 1 games lost on roll 13.  
1 games won and 0 games lost on roll 14.  
0 games won and 1 games lost on roll 15.  
1 games won and 1 games lost on roll 16.  
0 games won and 0 games lost on roll 17.  
0 games won and 1 games lost on roll 18.  
1 games won and 1 games lost on roll 19.  
0 games won and 0 games lost on roll 20.  
0 games won and 1 games lost on roll 21.
```

The chances of winning are $474/1000 = 47.40\%$
The average game length is 3.36 rolls.

6.21 (*Airline Reservations System*) A small airline has just purchased a computer for its new automated reservations system. The president has asked you to program the new system. You are to write a program to assign seats on each flight of the airline's only plane (capacity: 10 seats).

Your program should display the following menu of alternatives:

```
Please type 1 for "first class"
Please type 2 for "economy"
```

If the person types 1, then your program should assign a seat in the first class section (seats 1-5). If the person types 2, then your program should assign a seat in the economy section (seats 6-10). Your program should then print a boarding pass indicating the person's seat number and whether it is in the first class or economy section of the plane.

Use a single-subscripted array to represent the seating chart of the plane. Initialize all the elements of the array to 0 to indicate that all seats are empty. As each seat is assigned, set the corresponding elements of the array to 1 to indicate that the seat is no longer available.

Your program should, of course, never assign a seat that has already been assigned. When the first class section is full, your program should ask the person if it is acceptable to be placed in the economy section (and vice versa). If yes, then make the appropriate seat assignment. If no, then print the message "Next flight leaves in 3 hours."

ANS:

```
1  /* Exercise 6.21 Solution */
2  #include <stdio.h>
3  #include <ctype.h>
4
5  int main()
6  {
7      int plane[ 11 ] = { 0 }; /* seats on the plane */
8      int i = 0;                /* counter */
9      int firstClass= 1;        /* first class seats start at 1 */
10     int economy = 6;          /* economy seats start at 6 */
11     int choice;               /* user's choice */
12     char response[ 2 ];       /* user's response */
13
14     /* loop 10 times */
15     while ( i < 10 ) {
16         printf( "\n%s\n%s\n? ", "Please type 1 for \"first class\"",
17                "Please type 2 for \"economy\"");
18         scanf( "%d", &choice );
19
20         /* if user selects first class */
21         if ( choice == 1 ) {
22
23             /* if seat are available in first class */
24             if ( !plane[ firstClass ] && firstClass <= 5 ) {
25                 printf( "Your seat assignment is %d\n", firstClass );
26                 plane[ firstClass++ ] = 1;
27                 i++;
28             } /* end if */
29             /* if no first class seats, but economy seats available */
30             else if ( firstClass > 5 && economy <= 10 ) {
31
32                 /* ask if passenger would like to sit in economy */
33                 printf( "The first class section is full.\n" );
34                 printf( "Would you like to sit in the economy" );
35                 printf( " section ( Y or N )? " );
36                 scanf( "%s", response );
37
38                 /* if response is yes, then assign seat */
39                 if ( toupper( response[ 0 ] ) == 'Y' ) {
40                     printf( "Your seat assignment is %d\n", economy );
41                     plane[ economy++ ] = 1;
42                     i++;
43                 } /* end if */
44                 else { /* print next departure */
45                     printf( "Next flight leaves in 3 hours.\n" );
46                 } /* end else */

```

```
47
48     } /* end else if */
49     else { /* print next departure */
50         printf( "Next flight leaves in 3 hours.\n" );
51     } /* end else */
52
53 } /* end if */
54 else { /* if user selects economy */
55
56     /* if seats available, assign seat */
57     if ( !plane[ economy ] && economy <= 10 ) {
58         printf( "Your seat assignment is %d\n", economy );
59         plane[ economy++ ] = 1;
60         i++;
61     } /* end if */
62     /* if only first class seats are available */
63     else if ( economy > 10 && firstClass <= 5 ) {
64
65         /* ask if first class is suitable */
66         printf( "The economy section is full.\n" );
67         printf( "Would you like to sit in first class " );
68         printf( "section ( Y or N )? " );
69         scanf( "%s", response );
70
71         /* if response is yes, assign seat */
72         if ( toupper( response[ 0 ] ) == 'Y' ) {
73             printf( "Your seat assignment is %d\n", firstClass );
74             plane[ firstClass++ ] = 1;
75             i++;
76         } /* end if */
77         else { /* print next departure */
78             printf( "Next flight leaves in 3 hours.\n" );
79         } /* end else */
80
81     } /* end else if */
82     else { /* print next departure */
83         printf( "Next flight leaves in 3 hours.\n" );
84     } /* end else */
85
86 } /* end else */
87
88 } /* end while */
89
90 printf( "\nAll seats for this flight are sold.\n" );
91
92 return 0; /* indicate successful termination */
93
94 } /* end main */
```

```
Please type 1 for "first class"
Please type 2 for "economy"
? 2
Your seat assignment is 6

Please type 1 for "first class"
Please type 2 for "economy"
? 1
Your seat assignment is 1

Please type 1 for "first class"
Please type 2 for "economy"
? 2
Your seat assignment is 7
.
.
.
Please type 1 for "first class"
Please type 2 for "economy"
? 1
The first class section is full.
Would you like to sit in the economy section ( Y or N )? n
Next flight leaves in 3 hours.

Please type 1 for "first class"
Please type 2 for "economy"
? 1
The first class section is full.
Would you like to sit in the economy section ( Y or N )? y
Your seat assignment is 9

Please type 1 for "first class"
Please type 2 for "economy"
? 2
Your seat assignment is 10

All seats for this flight are sold.
```

6.22 Use a double-subscripted array to solve the following problem. A company has four salespeople (1 to 4) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each different type of product sold. Each slip contains:

- a) The salesperson number
- b) The product number
- c) The total dollar value of that product sold that day

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all of the slips for last month is available. Write a program that will read all this information for last month's sales and summarize the total sales by salesperson by product. All totals should be stored in the double-subscripted array `sales`. After processing all the information for last month, print the results in tabular format with each of the columns representing a particular salesperson and each of the rows representing a particular product. Cross total each row to get the total sales of each product for last month; cross total each column to get the total sales by salesperson for last month. Your tabular printout should include these cross totals to the right of the totaled rows and to the bottom of the totaled columns.

ANS:

```

1  /* Exercise 6.22 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6
7      /* total sales for each salesperson and each product */
8      double sales[ 4 ][ 5 ] = { 0.0 };
9      double productSales[ 5 ] = { 0.0 }; /* total product sales */
10     double value; /* current sales */
11     double totalSales; /* total overall sales */
12     int salesPerson; /* current salesperson */
13     int product; /* current product */
14     int i; /* loop counter */
15     int j; /* loop counter */
16
17     printf( "Enter the salesperson, product, and total sales.\n" );
18     printf( "Enter -1 for the salesperson to end input.\n" );
19     scanf( "%d", &salesPerson );
20
21     /* continue receiving input for each salesperson
22     while -1 is not entered */
23     while ( salesPerson != -1 ) {
24         scanf( "%d%lf", &product, &value );
25         sales[ salesPerson ][ product ] = value;
26         scanf( "%d", &salesPerson );
27     } /* end while */
28
29     /* display table */
30     printf( "\n\n%s\n%s\n%s\n%s\n", "The total sales for each salesperson",
31           "are displayed at the end of each", "row, and the total sales for each",
32           "product are displayed at the bottom ", "of each column.\n" );
33     printf( " %8d%8d%8d%8d%8d\n", 1, 2, 3, 4, 5 );
34
35     /* display salespeople and sales */
36     for ( i = 0; i <= 3; i++ ) {
37         totalSales = 0.0;
38         printf( "%d", i);
39
40         /* add total sales and display individual sales */
41         for ( j = 0; j <= 4; j++ ) {
42             totalSales += sales[ i ][ j ];
43             printf( "%8.2f", sales[ i ][ j ] );
44             productSales[ j ] += sales[ i ][ j ];
45         } /* end for */
46
47         printf( "%8.2f\n", totalSales );
48     } /* end for */
49
50     printf( " " );
51

```

```

52  /* display total product sales */
53  for ( j = 0; j <= 4; j++ ) {
54      printf( "%8.2f", productSales[ j ] );
55  } /* end for */
56
57  return 0; /* indicate successful termination */
58
59  } /* end main */

```

Enter the salesperson, product, and total sales.
Enter -1 for the salesperson to end input.

```

0 0 1.00
0 1 2.00
0 2 3.00
0 3 4.00
0 4 5.00
1 0 1.00
1 1 2.00
1 2 3.00
1 3 4.00
1 4 5.00
2 0 1.00
2 1 2.00
2 2 3.00
2 3 4.00
2 4 5.00
3 0 1.00
3 1 2.00
3 2 3.00
3 3 4.00
3 4 5.00
-1

```

The total sales for each salesperson are displayed at the end of each row, and the total sales for each product are displayed at the bottom of each column.

	1	2	3	4	5	
0	1.00	2.00	3.00	4.00	5.00	15.00
1	1.00	2.00	3.00	4.00	5.00	15.00
2	1.00	2.00	3.00	4.00	5.00	15.00
3	1.00	2.00	3.00	4.00	5.00	15.00
	4.00	8.00	12.00	16.00	20.00	

6.23 (*Turtle Graphics*) The Logo language, which is particularly popular among personal computer users, made the concept of *turtle graphics* famous. Imagine a mechanical turtle that walks around the room under the control of a C program. The turtle holds a pen in one of two positions, up or down. While the pen is down, the turtle traces out shapes as it moves; while the pen is up, the turtle moves about freely without writing anything. In this problem you will simulate the operation of the turtle and create a computerized sketchpad as well.

Use a 50-by-50 array `floor` which is initialized to zeros. Read commands from an array that contains them. Keep track of the current position of the turtle at all times and whether the pen is currently up or down. Assume that the turtle always starts at position 0,0 of the floor with its pen up. The set of turtle commands your program must process are shown in Fig. 6.24.

Command	Meaning
1	Pen up
2	Pen down
3	Turn right
4	Turn left
5,10	Move forward 10 spaces (or a number other than 10)
6	Print the 20-by-20 array
9	End of data (sentinel)

Suppose that the turtle is somewhere near the center of the floor. The following “program” would draw and print a 12-by-12-square:

```

2
5,12
3
5,12
3
5,12
3
5,12
1
6
9

```

As the turtle moves with the pen down, set the appropriate elements of array `floor` to 1s. When the 6 command (print) is given, wherever there is a 1 in the array, display an asterisk, or some other character you choose. Wherever there is a zero, display a blank. Write a program to implement the turtle graphics capabilities discussed here. Write several turtle graphics programs to draw interesting shapes. Add other commands to increase the power of your turtle graphics language.

ANS:

```

1  /* Exercise 6.23 Solution */
2  #include <stdio.h>
3
4  #define TRUE 1
5  #define FALSE 0
6  #define MAX 100 /* the maximum number of commands */
7
8  /* function prototypes */
9  void getCommands( int commands[][ 2 ] );
10 int turnRight( int d );
11 int turnLeft( int d );
12 void movePen( int down, int a[][ 50 ], int dir, int dist );
13 void printArray( int a[][ 50 ] );
14
15 int main()
16 {
17     int floor[ 50 ][ 50 ] = { 0 }; /* floor grid */
18     int penDown = FALSE; /* pen down flag */
19     int command; /* current command */

```

```

20     int direction = 0;                /* direction indicator */
21     int commandArray[ MAX ][ 2 ] = { 0 }; /* array of commands */
22     int distance;                    /* distance to move */
23     int count = 0;                   /* command counter */
24
25     getCommands( commandArray );
26     command = commandArray[ count ][ 0 ];
27
28     /* continue receiving input while -9 is not entered */
29     while ( command != 9 ) {
30
31         /* determine what command was entered and perform action */
32         switch ( command ) {
33
34             case 1:
35                 penDown = FALSE;
36                 break; /* exit switch */
37
38             case 2:
39                 penDown = TRUE;
40                 break; /* exit switch */
41
42             case 3:
43                 direction = turnRight( direction );
44                 break; /* exit switch */
45
46             case 4:
47                 direction = turnLeft( direction );
48                 break; /* exit switch */
49
50             case 5:
51                 distance = commandArray[ count ][ 1 ];
52                 movePen( penDown, floor, direction, distance );
53                 break; /* exit switch */
54
55             case 6:
56                 printf( "\nThe drawing is:\n\n" );
57                 printArray( floor );
58                 break; /* exit switch */
59         } /* end switch */
60
61         command = commandArray[ ++count ][ 0 ];
62     } /* end while */
63
64     return 0; /* indicate successful termination */
65
66 } /* end main */
67
68 /* getCommands prompts user for commands */
69 void getCommands( int commands[][ 2 ] )
70 {
71     int i;                /* counter */
72     int tempCommand; /* temporary command holder */
73
74     printf( "Enter command ( 9 to end input ): " );
75     scanf( "%d", &tempCommand );
76
77     /* receive commands until -9 or 100 commands are entered */
78     for ( i = 0; tempCommand != 9 && i < MAX; i++ ) {
79         commands[ i ][ 0 ] = tempCommand;
80
81         /* ignore comma after 5 is entered */
82         if ( tempCommand == 5 ) {
83             scanf( ",%d", &commands[ i ][ 1 ] );
84         } /* end if */
85
86         printf( "Enter command ( 9 to end input ): " );
87         scanf( "%d", &tempCommand );
88     } /* end for */

```

```

89
90     commands[ i ][ 0 ] = 9; /* last command */
91 } /* end function getCommands */
92
93 /* turnRight turns turtle to the right */
94 int turnRight( int d )
95 {
96     return ++d > 3 ? 0 : d;
97 }
98 } /* end function turnRight */
99
100 /* turnLeft turns turtle to the left */
101 int turnLeft( int d )
102 {
103     return --d < 0 ? 3 : d;
104 }
105 } /* end function turnLeft */
106
107 /* movePen moves the pen */
108 void movePen( int down, int a[][ 50 ], int dir, int dist )
109 {
110     int i;           /* loop counter */
111     int j;           /* loop counter */
112     static int xPos = 0; /* x coordinate */
113     static int yPos = 0; /* y coordinate */
114
115     /* determine which way to move pen */
116     switch ( dir ) {
117
118     case 0: /* move to the right */
119
120         /* move dist spaces or until edge of floor */
121         for ( j = 1; j <= dist && yPos + j < 50; j++ ) {
122
123             /* draw 1 if pen is down */
124             if ( down ) {
125                 a[ xPos ][ yPos + j ] = 1;
126             } /* end if */
127
128         } /* end for */
129
130         yPos += j - 1;
131         break; /* exit switch */
132
133     case 1: /* move down */
134
135         /* move dist spaces or until edge of floor */
136         for ( i = 1; i <= dist && xPos + i < 50; i++ ) {
137
138             /* draw 1 if pen is down */
139             if ( down ) {
140                 a[ xPos + i ][ yPos ] = 1;
141             } /* end if */
142
143         } /* end for */
144
145         xPos += i - 1;
146         break; /* exit switch */
147
148     case 2: /* move to the left */
149
150         /* move dist spaces or until edge of floor */
151         for ( j = 1; j <= dist && yPos - j >= 0; j++ ) {
152
153             /* draw 1 if pen is down */
154             if ( down ) {
155                 a[ xPos ][ yPos - j ] = 1;
156             } /* end if */

```



```

157         } /* end for */
158     } /* end for */
159
160     yPos -= j - 1;
161     break; /* exit switch */
162
163     case 3: /* move up */
164
165         /* move dist spaces or until edge of floor */
166         for ( i = 1; i <= dist && xPos - i >= 0; i++ ) {
167
168             /* draw 1 if pen is down */
169             if ( down ) {
170                 a[ xPos - i ][ yPos ] = 1;
171             } /* end if */
172
173         } /* end for */
174
175         xPos -= i - 1;
176         break; /* exit switch */
177     } /* end switch */
178 } /* end function movePen */
179
180
181 /* printArray prints array drawing */
182 void printArray( int a[][ 50 ] )
183 {
184     int i; /* counter */
185     int j; /* counter */
186
187     /* loop through array */
188     for ( i = 0; i < 50; i++ ) {
189
190         /* loop through array */
191         for ( j = 0; j < 50; j++ ) {
192             putchar( a[ i ][ j ] ? '*' : ' ' );
193         } /* end for */
194
195         putchar( '\n' );
196     } /* end for */
197 } /* end function printArray */

```

```

Enter command ( 9 to end input ): 2
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 3
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 3
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 3
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 1
Enter command ( 9 to end input ): 6
Enter command ( 9 to end input ): 9

```

The drawing is:

```

*****
*      *
*      *
*      *
*      *
*      *
*      *
*      *
*      *
*      *
*      *
*      *
*****

```

6.24 (*Knight's Tour*) One of the more interesting puzzlers for chess buffs is the Knight's Tour problem, originally proposed by the mathematician Euler. The question is this: Can the chess piece called the knight move around an empty chessboard and touch each of the 64 squares once and only once? We study this intriguing problem in depth here.

The knight makes L-shaped moves (over two in one direction and then over one in a perpendicular direction). Thus, from a square in the middle of an empty chessboard, the knight can make eight different moves (numbered 0 through 7) as shown in Fig. 6.25.

- Draw an 8-by-8 chessboard on a sheet of paper and attempt a Knight's Tour by hand. Put a 1 in the first square you move to, a 2 in the second square, a 3 in the third, etc. Before starting the tour, estimate how far you think you will get, remembering that a full tour consists of 64 moves. How far did you get? Were you close to the estimate?
- Now let us develop a program that will move the knight around a chessboard. The board itself is represented by an 8-by-8 double-subscripted array `board`. Each of the squares is initialized to zero. We describe each of the eight possible moves in terms of both their horizontal and vertical components. For example, a move of type 0 as shown in Fig. 6.25 consists of moving two squares horizontally to the right and one square vertically upward. Move 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative numbers. The eight moves may be described by two single-subscripted arrays, `horizontal` and `vertical`, as follows:

	0	1	2	3	4	5	6	7
0								
1				2		1		
2			3				0	
3					K			
4			4				7	
5				5		6		
6								
7								

```
horizontal[ 0 ] = 2
horizontal[ 1 ] = 1
horizontal[ 2 ] = -1
horizontal[ 3 ] = -2
horizontal[ 4 ] = -2
horizontal[ 5 ] = -1
horizontal[ 6 ] = 1
horizontal[ 7 ] = 2
```

```
vertical[ 0 ] = -1
vertical[ 1 ] = -2
vertical[ 2 ] = -2
vertical[ 3 ] = -1
vertical[ 4 ] = 1
vertical[ 5 ] = 2
vertical[ 6 ] = 2
vertical[ 7 ] = 1
```

Let the variables `currentRow` and `currentColumn` indicate the row and column of the knight's current position on the board. To make a move of type `moveNumber`, where `moveNumber` is between 0 and 7, your program uses the statements

```
currentRow += vertical[ moveNumber ];
currentColumn += horizontal[ moveNumber ];
```

Keep a counter that varies from 1 to 64. Record the latest count in each square the knight moves to. Remember to test each potential move to see if the knight has already visited that square. And, of course, test every potential move to make sure that the knight does not land off the chessboard. Now write a program to move the knight around the chessboard. Run the program. How many moves did the knight make?

- c) After attempting to write and run a Knight's Tour program, you have probably developed some valuable insights. We will use these to develop a *heuristic* (or strategy) for moving the knight. Heuristics do not guarantee success, but a carefully developed heuristic greatly improves the chance of success. You may have observed that the outer squares are in some sense more troublesome than the squares nearer the center of the board. In fact, the most troublesome, or inaccessible, squares are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to so that when the board gets congested near the end of the tour there will be a greater chance of success.

We may develop an "accessibility heuristic" by classifying each of the squares according to how accessible they are and always moving the knight to the square (within the knight's L-shaped moves, of course) that is most inaccessible. We label a double-subscripted array `accessibility` with numbers indicating from how many squares each particular square is accessible. On a blank chessboard, the center squares are therefore rated as 8s, the corner squares are rated as 2s, and the other squares have accessibility numbers of 3, 4, or 6 as follows:

```
  2  3  4  4  4  4  3  2
  3  4  6  6  6  6  4  3
  4  6  8  8  8  8  6  4
  4  6  8  8  8  8  6  4
  4  6  8  8  8  8  6  4
  4  6  8  8  8  8  6  4
  3  4  6  6  6  6  4  3
  2  3  4  4  4  4  3  2
```

Now write a version of the Knight's Tour program using the accessibility heuristic. At any time, the knight should move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied squares. Therefore, the tour may begin in any of the four corners. [Note: As the knight moves around the chessboard, your program should reduce the accessibility numbers as more and more squares become occupied. In this way, at any given time during the tour, each available square's accessibility number will remain equal to precisely the number of squares from which that square may be reached.] Run this version of your program. Did you get a full tour? Now modify the program to run 64 tours, one from each square of the chessboard. How many full tours did you get?

- d) Write a version of the Knight's Tour program which, when encountering a tie between two or more squares, decides what square to choose by looking ahead to those squares reachable from the "tied" squares. Your program should move to the square for which the next move would arrive at a square with the lowest accessibility number.

ANS:

```
1  /* Exercise 6.24 Part C Solution */
2  /* Knight's Tour - access version */
3  /* runs one tour */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <time.h>
8
9  #define TRUE 1
10 #define FALSE 0
11
12 /* function prototypes */
13 void clearBoard( int workBoard[][ 8 ] );
14 void printBoard( int workBoard[][ 8 ] );
15 int validMove( int row, int column, int workBoard[][ 8 ] );
16
17 int main()
18 {
19     int board[ 8 ][ 8 ]; /* chess board */
```

```

20
21  /* array of accessibility */
22  int access[ 8 ][ 8 ] = { 2, 3, 4, 4, 4, 4, 3, 2,
23                        3, 4, 6, 6, 6, 6, 4, 3,
24                        4, 6, 8, 8, 8, 8, 6, 4,
25                        4, 6, 8, 8, 8, 8, 6, 4,
26                        4, 6, 8, 8, 8, 8, 6, 4,
27                        4, 6, 8, 8, 8, 8, 6, 4,
28                        3, 4, 6, 6, 6, 6, 4, 3,
29                        2, 3, 4, 4, 4, 4, 3, 2 };
30
31  /* eight horizontal and vertical moves for the knight */
32  int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
33  int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
34  int currentRow; /* current row */
35  int currentColumn; /* current column */
36  int moveNumber = 0; /* move counter */
37  int testRow; /* possible next row */
38  int testColumn; /* possible next column */
39  int minRow; /* row with minimum access number */
40  int minColumn; /* column with minimum access number */
41  int minAccess = 9; /* impossible access number */
42  int accessNumber; /* current access number */
43  int moveType; /* current move type */
44  int done; /* flag to indicate end */
45
46  srand( time( NULL ) );
47
48  clearBoard( board ); /* initialize array board */
49  currentRow = rand() % 8;
50  currentColumn = rand() % 8;
51  board[ currentRow ][ currentColumn ] = ++moveNumber;
52  done = FALSE;
53
54  /* continue while knight still has valid moves */
55  while ( !done ) {
56      accessNumber = minAccess;
57
58      /* loop through all move types */
59      for ( moveType = 0; moveType < 8; moveType++ ) {
60          testRow = currentRow + vertical[ moveType ];
61          testColumn = currentColumn + horizontal[ moveType ];
62
63          /* make sure move is valid */
64          if ( validMove( testRow, testColumn, board ) ) {
65
66              /* if move is valid and has lowest accessNumber,
67               set square to accessNumber */
68              if ( access[ testRow ][ testColumn ] < accessNumber ) {
69                  accessNumber = access[ testRow ][ testColumn ];
70                  minRow = testRow;
71                  minColumn = testColumn;
72              } /* end if */
73
74              --access[ testRow ][ testColumn ];
75          } /* end if */
76
77      } /* end for */
78
79      /* end if knight has no moves */
80      if ( accessNumber == minAccess ) {
81          done = TRUE;
82      } /* end if */
83      else {
84          currentRow = minRow;
85          currentColumn = minColumn;
86          board[ currentRow ][ currentColumn ] = ++moveNumber;
87      } /* end else */

```

```

88
89     } /* end while */
90
91     printf( "The tour ended with %d moves.\n", moveNumber );
92
93     /* determine and print if a full tour was made */
94     if ( moveNumber == 64 ) {
95         printf( "This was a full tour!\n\n" );
96     } /* end if */
97     else {
98         printf( "This was not a full tour.\n\n" );
99     } /* end else */
100
101     printf( "The board for this test is:\n\n" );
102     printBoard( board );
103
104     return 0; /* indicate successful termination */
105 } /* end main */
106
107
108 /* function to clear chess board */
109 void clearBoard( int workBoard[][ 8 ] )
110 {
111     int row; /* row counter */
112     int col; /* column counter */
113
114     /* set all squares to zero */
115     for ( row = 0; row < 8; row++ ) {
116
117         for ( col = 0; col < 8; col++ ) {
118             workBoard[ row ][ col ] = 0;
119         } /* end for */
120     } /* end for */
121 } /* end function clearBoard */
122
123
124 /* function to print chess board */
125 void printBoard( int workBoard[][ 8 ] )
126 {
127     int row; /* row counter */
128     int col; /* column counter */
129
130     printf( "  0 1 2 3 4 5 6 7\n" );
131
132     /* print squares */
133     for ( row = 0; row < 8; row++ ) {
134         printf( "%d", row );
135
136         for ( col = 0; col < 8; col++ ) {
137             printf( "%3d", workBoard[ row ][ col ] );
138         } /* end for */
139
140         printf( "\n" );
141     } /* end for */
142
143     printf( "\n" );
144 } /* end function printBoard */
145
146
147 /* function to determine if move is legal */
148 int validMove( int row, int column, int workBoard[][ 8 ] )
149 {
150
151     /* NOTE: This test stops as soon as it becomes false */
152     return ( row >= 0 && row <= 7 && column >= 0 &&
153             column <= 7 && workBoard[ row ][ column ] == 0 );
154 } /* end function validMove */
155

```

The tour ended with 64 moves.
This was a full tour!

The board for this test is:

```

  0  1  2  3  4  5  6  7
0 33 30 19  4 23 28 17  2
1 20  5 32 29 18  3 24 27
2 31 34 49 22 37 26  1 16
3  6 21 36 59 50 47 38 25
4 35 60 51 48 39 58 15 46
5 10  7 62 57 54 45 40 43
6 61 52  9 12 63 42 55 14
7  8 11 64 53 56 13 44 41

```

6.25 (*Knight's Tour: Brute Force Approaches*) In Exercise 6.24 we developed a solution to the Knight's Tour problem. The approach used, called the "accessibility heuristic," generates many solutions and executes efficiently.

As computers continue increasing in power, we will be able to solve many problems with sheer computer power and relatively unsophisticated algorithms. Let us call this approach "brute force" problem solving.

- a) Use random number generation to enable the knight to walk around the chess board (in its legitimate L-shaped moves, of course) at random. Your program should run one tour and print the final chessboard. How far did the knight get?

ANS:

```

1  /* Exercise 6.25 Part A Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define NO 0
7  #define YES 1
8
9  /* function prototypes */
10 int validMove( int row, int column, int workBoard[][ 8 ] );
11 void printBoard( int board[][ 8 ] );
12
13 int main()
14 {
15     int currentRow;    /* current row */
16     int currentColumn; /* current column */
17     int moveType;     /* current move type */
18     int moveNumber = 0; /* move counter */
19     int testRow;      /* possible next row */
20     int testColumn;   /* possible next column */
21     int count;        /* counter */
22     int done;         /* flag to indicate end */
23     int goodMove;     /* result of call to validMove */
24
25     /* horizontal and vertical moves for the knight, and board */
26     int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
27     int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
28     int board[ 8 ][ 8 ] = { 0 };
29
30     srand( time( NULL ) );
31
32     currentRow = rand() % 8;
33     currentColumn = rand() % 8;
34     board[ currentRow ][ currentColumn ] = ++moveNumber;
35     done = NO;
36
37     /* continue while knight can still move */
38     while ( !done ) {
39         moveType = rand() % 8;
40         testRow = currentRow + vertical[ moveType ];
41         testColumn = currentColumn + horizontal[ moveType ];

```

```

42     goodMove = validMove( testRow, testColumn, board );
43
44     /* test if desired move is valid */
45     if ( goodMove ) {
46         currentRow = testRow;
47         currentColumn = testColumn;
48         board[ currentRow ][ currentColumn ] = ++moveNumber;
49     } /* end if */
50     else {
51
52         /* if move is not legal try another random move */
53         for ( count = 0; count < 7 && !goodMove; count++ ) {
54             moveType = ++moveType % 8;
55             testRow = currentRow + vertical[ moveType ];
56             testColumn = currentColumn + horizontal[ moveType ];
57             goodMove = validMove( testRow, testColumn, board );
58
59             /* test if new move is good */
60             if ( goodMove ) {
61                 currentRow = testRow;
62                 currentColumn = testColumn;
63                 board[ currentRow ][ currentColumn ] = ++moveNumber;
64             } /* end if */
65
66             } /* end for */
67
68         /* if no valid moves, knight can no longer move */
69         if ( !goodMove ) {
70             done = YES;
71         } /* end if */
72
73     } /* end else */
74
75     /* if 64 moves have been made, a full tour is complete */
76     if ( moveNumber == 64 ) {
77         done = YES;
78     } /* end if */
79
80 } /* end while */
81
82 printf( "The tour has ended with %d moves.\n", moveNumber );
83
84 /* test if full tour was made */
85 if ( moveNumber == 64 ) {
86     printf( "This was a full tour!\n" );
87 } /* end if */
88 else {
89     printf( "This was not a full tour.\n" );
90 } /* end else */
91
92 printf( "The board for this random test was:\n\n" );
93 printBoard( board ); /* print the board */
94
95 return 0; /* indicate successful termination */
96
97 } /* end main */
98
99 /* function to test whether a square is on the board
100 and has not been visited yet */
101 int validMove( int row, int column, int workBoard[][ 8 ] )
102 {
103     /* NOTE: This test stops as soon as it becomes false */
104     return ( row >= 0 && row < 8 && column >= 0 &&
105             column < 8 && workBoard[ row ][ column ] == 0 );
106 } /* end function validMove */
107
108

```

```

109 /* function to print the chess board */
110 void printBoard( int board[][ 8 ] )
111 {
112     int row; /* row counter */
113     int col; /* column counter */
114
115     printf( "  0 1 2 3 4 5 6 7\n" );
116
117
118     /* print the rows and columns of the chess board */
119     for ( row = 0; row < 8; row++ ) {
120         printf( "%d", row );
121
122         for ( col = 0; col < 8; col++ ) {
123             printf( "%3d", board[ row ][ col ] );
124         } /* end for */
125
126         printf( "\n" );
127     } /* end for */
128
129     printf( "\n" );
130 } /* end function printBoard */

```

The tour has ended with 32 moves.
This was not a full tour.
The board for this random test was:

```

  0 1 2 3 4 5 6 7
0 13 0 0 0 11 28 0 32
1 0 0 12 29 24 31 8 0
2 0 14 0 0 27 10 25 6
3 18 0 16 23 30 7 0 9
4 15 0 19 0 0 26 5 0
5 20 17 0 0 22 0 2 0
6 0 0 21 0 0 0 0 4
7 0 0 0 0 0 3 0 1

```

- b) Most likely, the preceding program produced a relatively short tour. Now modify your program to attempt 1000 tours. Use a single-subscripted array to keep track of the number of tours of each length. When your program finishes attempting the 1000 tours, it should print this information in neat tabular format. What was the best result?

ANS:

```

1  /* Exercise 6.25 Part B Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define NO 0
7  #define YES 1
8
9  int validMove( int, int, int [][] [ 8 ] );
10
11 int main()
12 {
13     int currentRow; /* current row */
14     int currentColumn; /* current column */
15     int moveType; /* current move type */
16     int moveNumber; /* move counter */
17     int testRow; /* possible next row */
18     int testColumn; /* possible next column */
19     int count; /* counter */
20     int i; /* counter */
21     int row; /* row */
22     int col; /* column */

```



```

23     int done;           /* flag to indicate end */
24     int goodMove;      /* result of call to validMove */
25     int board[ 8 ][ 8 ]; /* chess board */
26     int moveTotal[ 65 ] = { 0 }; /* array of tour totals */
27
28     /* horizontal and vertical moves for the knight */
29     int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
30     int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
31
32     srand( time( NULL ) );
33
34     /* attempt 1000 tours */
35     for ( i = 0; i < 1000; i++ ) {
36
37         /* set all squares equal to 0 */
38         for ( row = 0; row < 8; row++ ) {
39
40             for ( col = 0; col < 8; col++ ) {
41                 board[ row ][ col ] = 0;
42             } /* end for */
43
44         } /* end for */
45
46         moveNumber = 0;
47
48         currentRow = rand() % 8;
49         currentColumn = rand() % 8;
50         board[ currentRow ][ currentColumn ] = ++moveNumber;
51         done = NO;
52
53         /* continue while knight still has valid moves */
54         while ( !done ) {
55             moveType = rand() % 8;
56             testRow = currentRow + vertical[ moveType ];
57             testColumn = currentColumn + horizontal[ moveType ];
58             goodMove = validMove( testRow, testColumn, board );
59
60             /* if desired move is valid, move knight to square */
61             if ( goodMove ) {
62                 currentRow = testRow;
63                 currentColumn = testColumn;
64                 board[ currentRow ][ currentColumn ] = ++moveNumber;
65             } /* end if */
66             else {
67
68                 /* if move is invalid, test other possible moves */
69                 for ( count = 0; count < 7 && !goodMove; count++ ) {
70                     moveType = ++moveType % 8;
71                     testRow = currentRow + vertical[ moveType ];
72                     testColumn = currentColumn + horizontal[ moveType ];
73                     goodMove = validMove( testRow, testColumn, board );
74
75                     /* if move is valid, move knight to square */
76                     if ( goodMove ) {
77                         currentRow = testRow;
78                         currentColumn = testColumn;
79                         board[ currentRow ][ currentColumn ] = ++moveNumber;
80                     } /* end if */
81
82                 } /* end for */
83
84                 /* if no valid moves, while loop exits */
85                 if ( !goodMove ) {
86                     done = YES;
87                 } /* end if */
88
89             } /* end else */
90

```

```
91     /* if full tour is made, while loop exits */
92     if ( moveNumber == 64 ) {
93         done = YES;
94     } /* end if */
95
96     } /* end while */
97
98     ++moveTotal[ moveNumber ];
99 } /* end for */
100
101 /* display how many tours of each move number were made */
102 for ( i = 1; i < 65; i++ ) {
103
104     if ( moveTotal[ i ] ) {
105         printf( "There were %d tours of %d moves.\n",
106             moveTotal[ i ], i );
107     } /* end if */
108
109 } /* end for */
110
111 return 0; /* indicate successful termination */
112
113 } /* end main */
114
115 /* function to determine if a move is legal */
116 int validMove( int testRow, int testColumn, int board[][ 8 ] )
117 {
118
119     /* test if square is on board and if knight has previously
120     visited it */
121     if ( testRow >= 0 && testRow < 8 && testColumn >= 0 &&
122         testColumn < 8 ) {
123         return board[ testRow ][ testColumn ] != 0 ? NO : YES;
124     } /* end if */
125     else {
126         return NO;
127     } /* end else */
128
129 } /* end function validMove */
130
```

```
There were 1 tours of 4 moves.
There were 3 tours of 5 moves.
There were 2 tours of 6 moves.
There were 3 tours of 7 moves.
There were 2 tours of 8 moves.
There were 4 tours of 10 moves.
There were 5 tours of 11 moves.
There were 4 tours of 12 moves.
There were 5 tours of 13 moves.
There were 7 tours of 14 moves.
There were 7 tours of 15 moves.
There were 9 tours of 16 moves.
There were 8 tours of 17 moves.
There were 10 tours of 18 moves.
There were 9 tours of 19 moves.
There were 9 tours of 20 moves.
There were 11 tours of 21 moves.
There were 19 tours of 22 moves.
There were 17 tours of 23 moves.
There were 18 tours of 24 moves.
There were 12 tours of 25 moves.
There were 20 tours of 26 moves.
There were 14 tours of 27 moves.
There were 18 tours of 28 moves.
There were 22 tours of 29 moves.
There were 21 tours of 30 moves.
There were 31 tours of 31 moves.
There were 28 tours of 32 moves.
There were 25 tours of 33 moves.
There were 32 tours of 34 moves.
There were 26 tours of 35 moves.
There were 40 tours of 36 moves.
There were 38 tours of 37 moves.
There were 38 tours of 38 moves.
There were 37 tours of 39 moves.
There were 33 tours of 40 moves.
There were 35 tours of 41 moves.
There were 34 tours of 42 moves.
There were 33 tours of 43 moves.
There were 36 tours of 44 moves.
There were 30 tours of 45 moves.
There were 35 tours of 46 moves.
There were 26 tours of 47 moves.
There were 37 tours of 48 moves.
There were 22 tours of 49 moves.
There were 17 tours of 50 moves.
There were 20 tours of 51 moves.
There were 21 tours of 52 moves.
There were 17 tours of 53 moves.
There were 19 tours of 54 moves.
There were 14 tours of 55 moves.
There were 3 tours of 56 moves.
There were 7 tours of 57 moves.
There were 3 tours of 59 moves.
There were 3 tours of 60 moves.
```

- c) Most likely, the preceding program gave you some “respectable” tours but no full tours. Now “pull all the stops out” and simply let your program run until it produces a full tour. [*Caution:* This version of the program could run for hours on a powerful computer.] Once again, keep a table of the number of tours of each length and print this table when the first full tour is found. How many tours did your program attempt before producing a full tour? How much time did it take?
- d) Compare the brute force version of the Knight's Tour with the accessibility heuristic version. Which required a more careful study of the problem? Which algorithm was more difficult to develop? Which required more computer power? Could we be certain (in advance) of obtaining a full tour with the accessibility heuristic approach? Could we be certain (in advance) of obtaining a full tour with the brute force approach? Argue the pros and cons of brute force problem solving in general.

6.26 (*Eight Queens*) Another puzzler for chess buffs is the Eight Queens problem. Simply stated: Is it possible to place eight queens on an empty chessboard so that no queen is “attacking” any other—that is, so that no two queens are in the same row, the same column, or along the same diagonal? Use the kind of thinking developed in Exercise 6.24 to formulate a heuristic for solving the Eight Queens problem. Run your program. [*Hint*: It is possible to assign a numeric value to each square of the chessboard indicating how many squares of an empty chessboard are “eliminated” once a queen is placed in that square. For example, each of the four corners would be assigned the value 22, as in Fig. Fig. 6.26.]

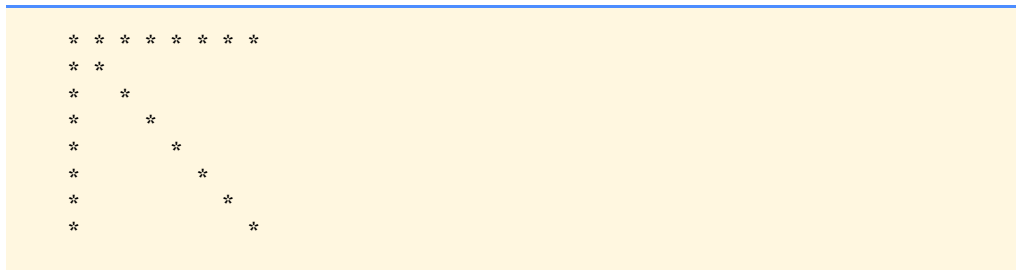


Fig. 6.26 The 22 squares eliminated by placing a queen in the upper-left corner.

Once these “elimination numbers” are placed in all 64 squares, an appropriate heuristic might be: Place the next queen in the square with the smallest elimination number. Why is this strategy intuitively appealing?

6.27 (*Eight Queens: Brute Force Approaches*) In this problem you will develop several brute force approaches to solving the Eight Queens problem introduced in Exercise 6.26.

- a) Solve the Eight Queens problem, using the random brute force technique developed in Exercise 6.25.
- b) Use an exhaustive technique (i.e., try all possible combinations of eight queens on the chessboard).
- c) Why do you suppose the exhaustive brute force approach may not be appropriate for solving the Knight's Tour problem?
- d) Compare and contrast the random brute force and exhaustive brute force approaches in general.

6.28 (*Duplicate elimination*) In Chapter 12, we explore the high-speed binary search tree data structure. One feature of a binary search tree is that duplicate values are discarded when insertions are made into the tree. This is referred to as duplicate elimination. Write a program that produces 20 random numbers between 1 and 20. The program should store all nonduplicate values in an array. Use the smallest possible array to accomplish this task.

```

1  /* Exercise 6.28 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define SIZE 20
7
8  int main()
9  {
10     int loop;           /* loop counter */
11     int randNumber;    /* current random number */
12     int loop2;         /* loop counter */
13     int subscript = 0; /* array subscript counter */
14     int duplicate;     /* duplicate flag */
15     int array[ SIZE ] = { 0 }; /* array of random numbers */
16
17     srand( time( NULL ) );
18
19     /* loop 20 times */
20     for ( loop = 0; loop <= SIZE - 1; loop++ ) {
21         duplicate = 0;
22         randNumber = 1 + rand() % 20; /* generate random number */
23
24         /* loop through current numbers in array */
25         for ( loop2 = 0; loop2 <= subscript; loop2++ ) {
26
27             /* compare randNumber with previous numbers */
28             if ( randNumber == array[ loop2 ] ) {
29                 duplicate = 1;
30                 break;
31             } /* end if */
32
33         } /* end for */
34
35         /* if not a duplicate */
36         if ( !duplicate ) {
37             array[ subscript++ ] = randNumber;
38         } /* end if */
39     } /* end while */
40
41     printf( "Non-repetitive array values are:\n" );
42
43     /* display array */
44     for ( loop = 0; array[ loop ] != 0; loop++ ) {
45         printf( "\t\t\t\tArray[ %d ] = %d\n", loop, array[ loop ] );
46     } /* end for */
47
48     return 0; /* indicate successful termination */
49
50 } /* end main */

```

Non-repetitive array values are:

```
Array[ 0 ] = 11  
Array[ 1 ] = 17  
Array[ 2 ] = 3  
Array[ 3 ] = 18  
Array[ 4 ] = 9  
Array[ 5 ] = 2  
Array[ 6 ] = 20  
Array[ 7 ] = 4  
Array[ 8 ] = 1  
Array[ 9 ] = 10  
Array[ 10 ] = 7  
Array[ 11 ] = 13  
Array[ 12 ] = 19  
Array[ 13 ] = 6  
Array[ 14 ] = 8  
Array[ 15 ] = 16
```

6.29 (*Knight's Tour: Closed Tour Test*) In the Knight's Tour, a full tour is when the knight makes 64 moves touching each square of the chess board once and only once. A closed tour occurs when the 64th move is one move away from the location in which the knight started the tour. Modify the Knight's Tour program you wrote in Exercise 6.24 to test for a closed tour if a full tour has occurred.

ANS:

```

1  /* Exercise 6.29 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define TRUE 1
7  #define FALSE 0
8
9  /* function prototypes */
10 void clearBoard( int workBoard[][ 8 ] );
11 void printBoard( int workBoard[][ 8 ] );
12 int validMove( int row, int column, int workBoard[][ 8 ] );
13
14 int main( void )
15 {
16
17     int firstMoveRow;    /* starting row */
18     int firstMoveCol;   /* starting column */
19     int closedTour = 0; /* closed tour flag */
20     int currentRow;     /* current row */
21     int currentColumn; /* current column */
22     int moveNumber = 0; /* move counter */
23     int testRow;       /* possible next row */
24     int testColumn;    /* possible next column */
25     int minRow;        /* minimum row access number */
26     int minColumn;     /* minimum column access number */
27     int minAccess = 9; /* access number reset */
28     int accessNumber;  /* current access number */
29     int moveType;      /* current move type */
30     int done;          /* flag to indicate end */
31     int board[ 8 ][ 8 ]; /* chess board */
32
33     /* horizontal and vertical moves for the knight */
34     int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
35     int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
36
37     /* access grid */
38     int access[ 8 ][ 8 ] = { 2, 3, 4, 4, 4, 4, 3, 2,
39                             3, 4, 6, 6, 6, 6, 4, 3,
40                             4, 6, 8, 8, 8, 8, 6, 4,
41                             4, 6, 8, 8, 8, 8, 6, 4,
42                             4, 6, 8, 8, 8, 8, 6, 4,
43                             4, 6, 8, 8, 8, 8, 6, 4,
44                             3, 4, 6, 6, 6, 6, 4, 3,
45                             2, 3, 4, 4, 4, 4, 3, 2 };
46
47
48     srand( time( NULL ) );
49
50     clearBoard( board ); /* initialize array board */
51     currentRow = rand() % 8;
52     currentColumn = rand() % 8;
53     firstMoveRow = currentRow; /* store first moves row */
54     firstMoveCol = currentColumn; /* store first moves col */
55
56     board[ currentRow ][ currentColumn ] = ++moveNumber;
57     done = FALSE;
58
59     /* loop while knight can still move */
60     while ( !done ) {
61         accessNumber = minAccess;

```

```

62
63      /* test what moves knight can make */
64      for ( moveType = 0; moveType < 8; moveType++ ) {
65          testRow = currentRow + vertical[ moveType ];
66          testColumn = currentColumn + horizontal[ moveType ];
67
68          /* if the knight can make a valid move */
69          if ( validMove( testRow, testColumn, board ) ) {
70
71              /* if move has lowest accessNumber, move to that space */
72              if ( access[ testRow ][ testColumn ] < accessNumber ) {
73                  accessNumber = access[ testRow ][ testColumn ];
74                  minRow = testRow;
75                  minColumn = testColumn;
76              } /* end if */
77
78              --access[ testRow ][ testColumn ];
79          } /* end if */
80
81      } /* end for */
82
83      /* if knight cannot access any more squares, loop terminates */
84      if ( accessNumber == minAccess ) {
85          done = TRUE;
86      } /* end if */
87      else {
88          currentRow = minRow;
89          currentColumn = minColumn;
90          board[ currentRow ][ currentColumn ] = ++moveNumber;
91
92          /* check for closed tour */
93          if ( moveNumber == 64 ) {
94
95              /* loop through possible next moves */
96              for ( moveType = 0; moveType < 8; moveType++ ) {
97                  testRow = currentRow + vertical[ moveType ];
98                  testColumn = currentColumn + horizontal[ moveType ];
99
100                 /* test if knight is one move away from start */
101                 if ( testRow == firstMoveRow && testColumn ==
102                     firstMoveCol ) {
103                     closedTour = 1;
104                 } /* end if */
105
106             } /* end for */
107
108         } /* end if */
109
110     } /* end else */
111
112 } /* end while */
113
114 printf( "The tour ended with %d moves.\n", moveNumber );
115
116 /* display results of tour */
117 if ( moveNumber == 64 && closedTour == 1 ) {
118     printf( "This was a closed tour!\n\n" );
119 } /* end if */
120 else if ( moveNumber == 64 ) {
121     printf( "This was a full tour!\n\n" );
122 } /* end else if */
123 else {
124     printf( "This was not a full tour.\n\n" );
125 } /* end else */
126
127 printf( "The board for this test is:\n\n" );
128 printBoard( board );
129

```



```

130     return 0; /* indicate successful termination */
131
132 } /* end main */
133
134 /* function to clear the chess board */
135 void clearBoard( int workBoard[][ 8 ] )
136 {
137     int row; /* row counter */
138     int col; /* col counter */
139
140     /* set all values on board to 0 */
141     for ( row = 0; row < 8; row++ ) {
142
143         for ( col = 0; col < 8; col++ ) {
144             workBoard[ row ][ col ] = 0;
145         } /* end for */
146     } /* end for */
147 } /* end function clearBoard */
148
149 } /* end function clearBoard */
150
151 /* function to print the chessboard */
152 void printBoard( int workBoard[][ 8 ] )
153 {
154     int row; /* row counter */
155     int col; /* column counter */
156
157     printf( "  0 1 2 3 4 5 6 7\n" );
158
159     /* print rows of chessboard */
160     for ( row = 0; row < 8; row++ ) {
161         printf( "%d", row );
162
163         /* print columns of chess board */
164         for ( col = 0; col < 8; col++ ) {
165             printf( "%3d", workBoard[ row ][ col ] );
166         } /* end for */
167
168         printf( "\n" );
169     } /* end for */
170
171     printf( "\n" );
172 } /* end function printBoard */
173
174 /* function to determine if a move is valid */
175 int validMove( int row, int column, int workBoard[][ 8 ] )
176 {
177
178     /* NOTE: This test stops as soon as it becomes false */
179     return ( row >= 0 && row < 8 && column >= 0 &&
180             column < 8 && workBoard[ row ][ column ] == 0 );
181 } /* end function validMove */
182

```

The tour ended with 64 moves.
This was a full tour!

The board for this test is:

```

  0 1 2 3 4 5 6 7
0 32 13 34 57 30 15 42 19
1 35 58 31 14 47 18 29 16
2 12 33 60 49 56 41 20 43
3 59 36 55 46 25 48 17 28
4 54 11 50 61 40 27 44 21
5 37 62 53 26 45 24 3 6
6 10 51 64 39 8 5 22 1
7 63 38 9 52 23 2 7 4

```

6.30 (*The Sieve of Eratosthenes*) A prime integer is any integer that can be divided evenly only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It works as follows:

- 1) Create an array with all elements initialized to 1 (true). Array elements with prime subscripts will remain 1. All other array elements will eventually be set to zero.
- 2) Starting with array subscript 2 (subscript 1 must be prime), every time an array element is found whose value is 1, loop through the remainder of the array and set to zero every element whose subscript is a multiple of the subscript for the element with value 1. For array subscript 2, all elements beyond 2 in the array that are multiples of 2 will be set to zero (subscripts 4, 6, 8, 10, etc.). For array subscript 3, all elements beyond 3 in the array that are multiples of 3 will be set to zero (subscripts 6, 9, 12, 15, etc.).

When this process is complete, the array elements that are still set to one indicate that the subscript is a prime number. These subscripts can then be printed. Write a program that uses an array of 1000 elements to determine and print the prime numbers between 1 and 999. Ignore element 0 of the array.

ANS:

```

1  /* Exercise 6.30 Solution */
2  #include <stdio.h>
3  #define SIZE 1000
4
5  int main()
6  {
7      int array[ SIZE ]; /* array to indicate prime numbers */
8      int loop;          /* loop counter */
9      int loop2;        /* loop counter */
10     int count = 0;     /* total prime numbers */
11
12     /* set all array elements to 1 */
13     for ( loop = 0; loop < SIZE; loop++ ) {
14         array[ loop ] = 1;
15     } /* end for */
16
17     /* test for multiples of current subscript */
18     for ( loop = 1; loop < SIZE; loop++ ) {
19
20         /* start with array subscript two */
21         if ( array[ loop ] == 1 && loop != 1 ) {
22
23             /* loop through remainder of array */
24             for ( loop2 = loop; loop2 <= SIZE; loop2++ ) {
25
26                 /* set to zero all multiples of loop */
27                 if ( loop2 % loop == 0 && loop2 != loop ) {
28                     array[ loop2 ] = 0;
29                 } /* end if */
30
31             } /* end for */
32
33         } /* end if */
34
35     } /* end for */
36
37     /* display prime numbers in the range 2 - 197 */
38     for ( loop = 2; loop < SIZE; loop++ ) {
39
40         if ( array[ loop ] == 1 ) {
41             printf( "%3d is a prime number.\n", loop );
42             ++count;
43         } /* end if */
44
45     } /* end for */
46
47     printf( "A total of %d prime numbers were found.\n", count );
48
49     return 0; /* indicate successful termination */
50
51 } /* end main */

```

```

2 is a prime number.
3 is a prime number.
5 is a prime number.
7 is a prime number.
11 is a prime number.
13 is a prime number.
17 is a prime number.
19 is a prime number.
.
.
.
971 is a prime number.
977 is a prime number.
983 is a prime number.
991 is a prime number.
997 is a prime number.
A total of 168 prime numbers were found.

```

6.31 (*Bucket Sort*) A bucket sort begins with an single-subscripted array of positive integers to be sorted, and a double-subscripted array of integers with rows subscripted from 0 to 9 and columns subscripted from 0 to $n - 1$ where n is the number of values in the array to be sorted. Each row of the double-subscripted array is referred to as a bucket. Write a function `bucketSort` that takes an integer array and the array size as arguments.

The algorithm is as follows:

- 1) Loop through the single-subscripted array and place each of its values in a row of the bucket array based on its ones digit. For example, 97 is placed in row 7, 3 is placed in row 3 and 100 is placed in row 0.
- 2) Loop through the bucket array and copy the values back to the original array. The new order of the above values in the single-subscripted array is 100, 3 and 97.
- 3) Repeat this process for each subsequent digit position (tens, hundreds, thousands, etc.) and stop when the leftmost digit of the largest number has been processed.

On the second pass of the array, 100 is placed in row 0, 3 is placed in row 0 (it had only one digit) and 97 is placed in row 9. The order of the values in the single-subscripted array is 100, 3 and 97. On the third pass, 100 is placed in row 1, 3 is placed in row zero and 97 is placed in row zero (after 3). The bucket sort is guaranteed to have all the values properly sorted after processing the leftmost digit of the largest number. The bucket sort knows it is done when all the values are copied into row zero of the double-subscripted array.

Note that the double-subscripted array of buckets is ten times the size of the integer array being sorted. This sorting technique provides better performance than a bubble sort, but requires much larger storage capacity. Bubble sort requires only one additional memory location for the type of data being sorted. Bucket sort is an example of a space-time trade-off. It uses more memory, but performs better. This version of the bucket sort requires copying all the data back to the original array on each pass. Another possibility is to create a second double-subscripted bucket array and repeatedly move the data between the two bucket arrays until all the data is copied into row zero of one of the arrays. Row zero then contains the sorted array.

ANS:

```

1  /* Exercise 6.31 Solution */
2  #include <stdio.h>
3
4  /* symbolic constant SIZE must be defined as the array size
5   for bucketSort to work */
6  #define SIZE 12
7
8  /* function prototypes */
9  void bucketSort( int a[] );
10 void distributeElements( int a[], int buckets[][ SIZE ], int digit );
11 void collectElements( int a[], int buckets[][ SIZE ] );
12 int numberOfDigits( int b[], int arraySize );
13 void zeroBucket( int buckets[][ SIZE ] );
14
15 int main()
16 {
17
18     /* array to be sorted */
19     int array[ SIZE ] = { 19, 13, 5, 27, 1, 26, 31, 16, 2, 9, 11, 21 };

```

```

20     int i; /* loop counter */
21
22     printf( "Array elements in original order:\n" );
23
24     /* display the unsorted array */
25     for ( i = 0; i < SIZE; i++ ) {
26         printf( "%3d", array[ i ] );
27     } /* end for */
28
29     putchar( '\n' );
30     bucketSort( array ); /* sort the array */
31
32     printf( "\nArray elements in sorted order:\n" );
33
34     /* display sorted array */
35     for ( i = 0; i < SIZE; i++ ) {
36         printf( "%3d", array[ i ] );
37     } /* end for */
38
39     putchar( '\n' );
40
41     return 0; /* indicate successful termination */
42
43 } /* end main */
44
45 /* Perform the bucket sort algorithm */
46 void bucketSort( int a[] )
47 {
48     int totalDigits;          /* largest # of digits in array */
49     int i;                   /* loop counter */
50     int bucket[ 10 ][ SIZE ] = { 0 }; /* initialize bucket array */
51
52     totalDigits = numberOfDigits( a, SIZE );
53
54     /* put elements in buckets for sorting
55     one sorted, get elements from buckets */
56     for ( i = 1; i <= totalDigits; i++ ) {
57         distributeElements( a, bucket, i );
58         collectElements( a, bucket );
59
60         /* set all bucket contents to zero */
61         if ( i != totalDigits ) {
62             zeroBucket( bucket );
63         } /* end if */
64     } /* end for */
65
66 } /* end function bucketSort */
67
68
69 /* Determine the number of digits in the largest number */
70 int numberOfDigits( int b[], int arraySize )
71 {
72     int largest = b[ 0 ]; /* assume first element is largest */
73     int i;               /* loop counter */
74     int digits = 0;      /* total number of digits */
75
76     /* find largest array element */
77     for ( i = 1; i < arraySize; i++ ) {
78
79         if ( b[ i ] > largest ) {
80             largest = b[ i ];
81         } /* end if */
82     } /* end for */
83
84     /* find number of digits of largest element */
85     while ( largest != 0 ) {
86         ++digits;

```

```

88     largest /= 10;
89 } /* end while */
90
91     return digits; /* return number of digits */
92
93 } /* end function numberOfDigits */
94
95 /* Distribute elements into buckets based on specified digit */
96 void distributeElements( int a[], int buckets[][ SIZE ], int digit )
97 {
98     int divisor = 10; /* used to get specific digit */
99     int i; /* loop counter */
100    int bucketNumber; /* current bucket number */
101    int elementNumber; /* current element number */
102
103    /* determine the divisor */
104    for ( i = 1; i < digit; i++ ) {
105        divisor *= 10;
106    } /* end for */
107
108    /* bucketNumber example for hundreds digit: */
109    /* ( 1234 % 1000 - 1234 % 100 ) / 100 --> 2 */
110    for ( i = 0; i < SIZE; i++ ) {
111        bucketNumber = ( a[ i ] % divisor - a[ i ] % ( divisor / 10 ) ) /
112            ( divisor / 10 );
113
114        /* retrieve value in buckets[ bucketNumber ][ 0 ] to determine */
115        /* which element of the row to store a[ i ] in. */
116        elementNumber = ++buckets[ bucketNumber ][ 0 ];
117        buckets[ bucketNumber ][ elementNumber ] = a[ i ];
118    } /* end for */
119
120 } /* end function distributeElements */
121
122 /* Return elements to original array */
123 void collectElements( int a[], int buckets[][ SIZE ] )
124 {
125     int i; /* loop counter */
126     int j; /* loop counter */
127     int subscript = 0; /* current subscript */
128
129     /* retrieve elements from buckets */
130     for ( i = 0; i < 10; i++ ) {
131
132         for ( j = 1; j <= buckets[ i ][ 0 ]; j++ ) {
133             a[ subscript++ ] = buckets[ i ][ j ];
134         } /* end for */
135
136     } /* end for */
137
138 } /* end function collectElements */
139
140 /* Set all buckets to zero */
141 void zeroBucket( int buckets[][ SIZE ] )
142 {
143     int i; /* loop counter */
144     int j; /* loop counter */
145
146     for ( i = 0; i < 10; i++ ) {
147
148         for ( j = 0; j < SIZE; j++ ) {
149             buckets[ i ][ j ] = 0;
150         } /* end for */
151
152     } /* end for */
153
154 } /* end function zeroBucket */

```

```

Array elements in original order:
19 13 5 27 1 26 31 16 2 9 11 21

Array elements in sorted order:
1 2 5 9 11 13 16 19 21 26 27 31

```

RECURSION EXERCISES

6.32 (*Selection Sort*) A selection sort searches an array looking for the smallest element in the array. When the smallest element is found, it is swapped with the first element of the array. The process is then repeated for the subarray beginning with the second element of the array. Each pass of the array results in one element being placed in its proper location. This sort requires similar processing capabilities to the bubble sort—for an array of n elements, $n - 1$ passes must be made, and for each subarray, $n - 1$ comparisons must be made to find the smallest value. When the subarray being processed contains one element, the array is sorted. Write a recursive function `selectionSort` to perform this algorithm.

ANS:

```

1  /* Exercise 6.32 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define MAXRANGE 1000
7  #define SIZE 10
8
9  void selectionSort( int array[], int size ); /* function prototype */
10
11 int main()
12 {
13     int sortThisArray[ SIZE ] = { 0 }; /* array to be sorted */
14     int loop; /* loop counter */
15
16     srand( time( NULL ) ); /* seed random number generator */
17
18     /* fill array with random numbers between 1-1000 */
19     for ( loop = 0; loop < SIZE; loop++ ) {
20         sortThisArray[ loop ] = 1 + rand() % MAXRANGE;
21     } /* end for */
22
23     printf( "\nUnsorted array is:\n" );
24
25     /* display unsorted array */
26     for ( loop = 0; loop < SIZE; loop++ ) {
27         printf( " %d ", sortThisArray[ loop ] );
28     } /* end for */
29
30     selectionSort( sortThisArray, SIZE ); /* sort array */
31
32     printf( "\n\nSorted array is:\n" );
33
34     /* display sorted array */
35     for ( loop = 0; loop < SIZE; loop++ ) {
36         printf( " %d ", sortThisArray[ loop ] );
37     } /* end for */
38
39     printf( "\n\n" );
40
41     return 0; /* indicate successful termination */
42 } /* end main */
43
44 /* function to sort an array */
45 void selectionSort( int array[], int size )
46 {
47     int temp; /* temporary variable used for swapping */

```

```
49     int loop; /* loop counter */
50
51     /* sort array until only one element is left */
52     if ( size >= 1 ) {
53
54         /* find smallest element and put it in first position */
55         for ( loop = 0; loop <= size - 1; loop++ ) {
56
57             /* swap elements */
58             if ( array[ loop ] < array[ 0 ] ) {
59                 temp = array[ loop ];
60                 array[ loop ] = array[ 0 ];
61                 array[ 0 ] = temp;
62             } /* end if */
63
64         } /* end for */
65
66         /* recursive call to selectionSort */
67         selectionSort( &array[ 1 ], size - 1 );
68     } /* end for */
69
70 } /* end function selectionSort */
```

```
Unsorted array is:
629 748 87 955 484 505 799 377 11 287

Sorted array is:
11 87 287 377 484 505 629 748 799 955
```

6.33 (*Palindromes*) A palindrome is a string that is spelled the same way forwards and backwards. Some examples of palindromes are: “radar,” “able was i ere i saw elba,” and, if you ignore blanks, “a man a plan a canal panama.” Write a recursive function `testPalindrome` that returns 1 if the string stored in the array is a palindrome and 0 otherwise. The function should ignore spaces and punctuation in the string.

ANS:

```

1  /* Exercise 6.33 solution */
2  #include <stdio.h>
3  #define SIZE 80
4
5  /* function prototype */
6  int testPalindrome( char array[], int left, int right );
7
8  int main()
9  {
10     char c; /* temporarily holds keyboard input */
11     char string[ SIZE ]; /* original string */
12     char copy[ SIZE ]; /* copy of string without spaces */
13     int count = 0; /* length of string */
14     int copyCount; /* length of copy */
15     int i; /* counter */
16
17     printf( "Enter a sentence:\n" );
18
19     /* get sentence to test from user */
20     while ( ( c = getchar() ) != '\n' && count < SIZE ) {
21         string[ count++ ] = c;
22     } /* end while */
23
24     string[ count ] = '\0'; /* terminate string */
25
26     /* make a copy of string without spaces */
27     for ( copyCount = 0, i = 0; string[ i ] != '\0'; i++ ) {
28
29         if ( string[ i ] != ' ' ) {
30             copy[ copyCount++ ] = string[ i ];
31         } /* end if */
32
33     } /* end for */
34
35     /* print whether or not the sentence is a palindrome */
36     if ( testPalindrome( copy, 0, copyCount - 1 ) ) {
37         printf( "\"%s\" is a palindrome\n", string );
38     } /* end if */
39     else {
40         printf( "\"%s\" is not a palindrome\n", string );
41     } /* end else */
42
43     return 0; /* indicate successful termination */
44
45 } /* end main */
46
47 /* function to see if the sentence is a palindrome */
48 int testPalindrome( char array[], int left, int right )
49 {
50
51     /* test array to see if a palindrome */
52     if ( left == right || left > right ) {
53         return 1;
54     } /* end if */
55     else if ( array[ left ] != array[ right ] ) {
56         return 0;
57     } /* end else if */
58     else {
59         return testPalindrome( array, left + 1, right - 1 );
60     } /* end else */
61
62 } /* end function testPalindrome */

```



```
Enter a sentence:  
able was i ere i saw elba  
"able was i ere i saw elba" is a palindrome
```

```
Enter a sentence:  
hi there  
"hi there" is not a palindrome
```

6.34 (*Linear Search*) Modify the program of Fig. 6.18 to use a recursive `linearSearch` function to perform the linear search of the array. The function should receive an integer array and the size of the array as arguments. If the search key is found, return the array subscript; otherwise, return `-1`.

ANS:

```

1  /* Exercise 6.34 Solution */
2  #include <stdio.h>
3  #define SIZE 100
4
5  /* function prototypes */
6  int linearSearch( int array[], int key, int low, int high );
7
8  int main()
9  {
10     int array[ SIZE ]; /* array to be searched */
11     int loop;          /* loop counter */
12     int searchKey;    /* element to search for */
13     int element;      /* result of linear search */
14
15     /* initialize array elements */
16     for ( loop = 0; loop < SIZE; loop++ ) {
17         array[ loop ] = 2 * loop;
18     } /* end for */
19
20     /* obtain search key from user */
21     printf( "Enter the integer search key: " );
22     scanf( "%d", &searchKey );
23
24     /* search array for search key */
25     element = linearSearch( array, searchKey, 0, SIZE - 1 );
26
27     /* display message if search key was found */
28     if ( element != -1 ) {
29         printf( "Found value in element %d\n", element );
30     } /* end if */
31     else {
32         printf( "Value not found\n" );
33     } /* end else */
34
35     return 0; /* indicate successful termination */
36
37 } /* end main */
38
39 /* function to search array for specified key */
40 int linearSearch( int array[], int key, int low, int high )
41 {
42
43     /* recursively search array */
44     if ( array[ low ] == key ) {
45         return low;
46     } /* end if */
47     else if ( low == high ) {
48         return -1;
49     } /* end else if */
50     else { /* recursive call */
51         return linearSearch( array, key, low + 1, high );
52     } /* end else */
53
54 } /* end function linearSearch */

```

```

Enter the integer search key: 8
Found value in element 4

```

```
Enter the integer search key: 48  
Found value in element 24
```

```
Enter the integer search key: 99  
Value not found
```

6.35 (*Binary Search*) Modify the program of Fig. 6.19 to use a recursive `binarySearch` function to perform the binary search of the array. The function should receive an integer array and the starting subscript and ending subscript as arguments. If the search key is found, return the array subscript; otherwise, return `-1`.

ANS:

```

1  /* Exercise 6.35 Solution */
2  #include <stdio.h>
3  #define SIZE 15
4
5  /* function prototypes */
6  int binarySearch( int b[], int searchKey, int low, int high );
7  void printHeader( void );
8  void printRow( int b[], int low, int mid, int high );
9
10 int main()
11 {
12     int a[ SIZE ]; /* array to be searched */
13     int i;         /* loop counter */
14     int key;       /* search key */
15     int result;    /* result of search */
16
17     /* initialize array elements */
18     for ( i = 0; i < SIZE; i++ ) {
19         a[ i ] = 2 * i;
20     } /* end for */
21
22     /* obtain key from user */
23     printf( "Enter a number between 0 and 28: " );
24     scanf( "%d", &key );
25
26     printHeader();
27
28     /* search array for key */
29     result = binarySearch( a, key, 0, SIZE - 1 );
30
31     /* display results of the search */
32     if ( result != -1 ) {
33         printf( "\n%d found in array element %d\n", key, result );
34     } /* end if */
35     else {
36         printf( "\n%d not found\n", key );
37     } /* end else */
38
39     return 0; /* indicate successful termination */
40 } /* end main */
41
42
43 /* function to search array for specified key */
44 int binarySearch( int b[], int searchKey, int low, int high )
45 {
46     int middle; /* middle of array */
47
48     /* find middle of array and print current subarray */
49     if ( low <= high ) {
50         middle = ( low + high ) / 2;
51         printRow( b, low, middle, high );
52
53         /* determine if middle element is the key and if not,
54            recursively call binarySearch */
55         if ( searchKey == b[ middle ] ) {
56             return middle;
57         } /* end if */
58         else if ( searchKey < b[ middle ] ) {
59             /* recursive call on bottom half of array */
60             return binarySearch( b, searchKey, low, middle - 1 );
61         } /* end else if */

```

```

62     else {
63         /* recursive call on upper half of array */
64         return binarySearch( b, searchKey, middle + 1, high );
65     } /* end else */
66
67 } /* end if */
68
69 return -1; /* searchKey not found */
70
71 } /* end function binarySearch */
72
73 /* Print a header for the output */
74 void printHeader( void )
75 {
76     int i; /* loop counter */
77
78     printf( "\nSubscripts:\n" );
79
80     /* print subscripts of array */
81     for ( i = 0; i < SIZE; i++ ) {
82         printf( "%3d ", i );
83     } /* end for */
84
85     printf( "\n" );
86
87     /* print dividing line */
88     for ( i = 1; i <= 4 * SIZE; i++ ) {
89         printf( "-" );
90     } /* end for */
91
92     printf( "\n" );
93 } /* end function printHeader */
94
95 /* print one row of output showing the current
96    part of the array being processed. */
97 void printRow( int b[], int low, int mid, int high )
98 {
99     int i; /* loop counter */
100
101     /* print subarray currently being processed */
102     for ( i = 0; i < SIZE; i++ ) {
103
104         if ( i < low || i > high ) {
105             printf( "   " );
106         } /* end if */
107         else if ( i == mid ) { /* mark middle value */
108             printf( "%3d*", b[ i ] );
109         } /* end else if */
110         else {
111             printf( "%3d ", b[ i ] );
112         } /* end else */
113
114     } /* end for */
115
116     printf( "\n" );
117 } /* end function printRow */

```

Enter a number between 0 and 28: 17

Subscripts:

```

0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
-----
0  2  4  6  8 10 12 14* 16 18 20 22 24 26 28
                    16 18 20 22* 24 26 28
                        16 18* 20
                            16*

```

17 not found

```
Enter a number between 0 and 28: 10
```

```
Subscripts:
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
-----
0 2 4 6 8 10 12 14* 16 18 20 22 24 26 28
0 2 4 6* 8 10 12
      8 10* 12
```

```
10 found in array element 5
```

6.36 (*Eight Queens*) Modify the Eight Queens program you created in Exercise 6.26 to solve the problem recursively.

6.37 (*Print an array*) Write a recursive function `printArray` that takes an array and the size of the array as arguments, and returns nothing. The function should stop processing and return when it receives an array of size zero.

ANS:

```

1  /* Exercise 6.37 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define SIZE 10
7
8  /* function prototype */
9  void printArray( int array[], int low, int high );
10
11 int main()
12 {
13     int array[ SIZE ]; /* array to be printed */
14     int loop;          /* loop counter */
15
16     srand( time( NULL ) );
17
18     /* initialize array elements to random numbers */
19     for ( loop = 0; loop < SIZE; loop++ ) {
20         array[ loop ] = 1 + rand() % 500;
21     } /* end for */
22
23     printf( "Array values printed in main:\n" );
24
25     /* print array elements */
26     for ( loop = 0; loop < SIZE; loop++ ) {
27         printf( "%d ", array[ loop ] );
28     } /* end for */
29
30     printf( "\n\nArray values printed in printArray:\n" );
31     printArray( array, 0, SIZE - 1 );
32     printf( "\n" );
33
34     return 0; /* indicate successful termination */
35 } /* end main */
36
37
38 /* function to recursively print an array */
39 void printArray( int array[], int low, int high )
40 {
41     /* print first element of array passed */
42     printf( "%d ", array[ low ] );
43
44     /* return if array only has 1 element */
45     if ( low == high ) {
46         return;
47     } /* end if */
48     else { /* call printArray with new subarray */
49         printArray( array, low + 1, high );
50     } /* end else */
51 } /* end function printArray */
52

```

```

Array values printed in main:
22 180 7 321 486 366 69 304 273 213

Array values printed in printArray:
22 180 7 321 486 366 69 304 273 213

```

6.38 (*Print a string backwards*) Write a recursive function `stringReverse` that takes a character array as an argument, prints it back to front and returns nothing. The function should stop processing and return when the terminating null character of the string is encountered.

ANS:

```

1  /* Exercise 6.38 Solution */
2  #include <stdio.h>
3  #define SIZE 30
4
5  void stringReverse( char strArray[] ); /* function prototype */
6
7  int main()
8  {
9      int loop; /* loop counter */
10
11     /* initialize string strArray */
12     char strArray[ SIZE ] = "Print this string backwards.";
13
14     /* display original string */
15     for ( loop = 0; loop < SIZE; loop++ ) {
16         printf( "%c", strArray[ loop ] );
17     } /* end for */
18
19     printf( "\n" );
20     stringReverse( strArray ); /* reverse the string */
21     printf( "\n" );
22
23     return 0; /* indicate successful termination */
24
25 } /* end main */
26
27 /* function to reverse a string */
28 void stringReverse( char strArray[] )
29 {
30
31     /* return when null character is encountered */
32     if ( strArray[ 0 ] == '\0' ) {
33         return;
34     } /* end if */
35
36     /* recursively call stringReverse with new substring */
37     stringReverse( &strArray[ 1 ] );
38     printf( "%c", strArray[ 0 ] ); /* output string elements */
39 } /* end function stringReverse */

```

```

Print this string backwards.
.sdrawkcab gnirts siht tnirP

```


6.39 (Find the minimum value in an array) Write a recursive function `recursiveMinimum` that takes an integer array and the array size as arguments and returns the smallest element of the array. The function should stop processing and return when it receives an array of one element.

ANS:

```

1  /* Exercise 6.39 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #define SIZE 10
6  #define MAXRANGE 1000
7
8  /* function prototype */
9  int recursiveMinimum( int array[], int low, int high );
10
11 int main()
12 {
13     int array[ SIZE ]; /* array to be searched */
14     int loop;          /* loop counter */
15     int smallest;      /* smallest element */
16
17     srand( time( NULL ) );
18
19     /* initialize elements of array to random numbers */
20     for ( loop = 0; loop < SIZE; loop++ ) {
21         array[ loop ] = 1 + rand() % MAXRANGE;
22     } /* end for */
23
24     printf( "Array members are:\n" );
25
26     /* display array */
27     for ( loop = 0; loop < SIZE; loop++ ) {
28         printf( " %d ", array[ loop ] );
29     } /* end for */
30
31     /* find and display smallest array element */
32     printf( "\n" );
33     smallest = recursiveMinimum( array, 0, SIZE - 1 );
34     printf( "\nSmallest element is: %d\n", smallest );
35
36     return 0; /* indicate successful termination */
37 } /* end main */
38
39 /* function to recursively find minimum array element */
40 int recursiveMinimum( int array[], int low, int high )
41 {
42     {
43         static int smallest = MAXRANGE; /* largest possible value */
44
45         /* if first element of array is smallest so far,
46            set smallest equal to that element */
47         if ( array[ low ] < smallest ) {
48             smallest = array[ low ];
49         } /* end if */
50
51         /* if only one element in array, return smallest */
52         if ( low == high ) {
53             return smallest;
54         } /* end if */
55         else { /* recursively call recursiveMinimum with new subarray */
56             return recursiveMinimum( array, low + 1, high );
57         } /* end else */
58     }
59 } /* end function recursiveMinimum */

```

```
Array members are:  
666 251 624 359 577 837 992 197 249 492  
Smallest element is: 197
```


7

Pointers: Solutions

SOLUTIONS

7.7 Answer each of the following:

a) The _____ operator returns the location in memory where its operand is stored.

ANS: address (&).

b) The _____ operator returns the value of the object to which its operand points.

ANS: indirection (*).

c) To simulate call-by-reference when passing a nonarray variable to a function, it is necessary to pass the _____ of the variable to the function.

ANS: address.

7.8 State whether the following are *true* or *false*. If *false*, explain why.

a) Two pointers that point to different arrays cannot be compared meaningfully.

ANS: True. It is not possible to know where these arrays will be stored in advance.

b) Because the name of an array is a pointer to the first element of the array, array names may be manipulated in precisely the same manner as pointers.

ANS: False. Array names cannot be modified to point to another location in memory.

7.9 Answer each of the following. Assume that unsigned integers are stored in 2 bytes and that the starting address of the array is at location 1002500 in memory.

a) Define an array of type `unsigned int` called `values` with five elements, and initialize the elements to the even integers from 2 to 10. Assume the symbolic constant `SIZE` has been defined as 5.

ANS: `unsigned int values[SIZE] = { 2, 4, 6, 8, 10 };`

b) Define a pointer `vPtr` that points to an object of type `unsigned int`.

ANS: `unsigned int *vPtr;`

c) Print the elements of array `values` using array subscript notation. Use a `for` statement and assume integer control variable `i` has been defined.

ANS:

```
for ( i = 0; i < SIZE; i++ )
    printf( "%d ", values[ i ] );
```

d) Give two separate statements that assign the starting address of array `values` to pointer variable `vPtr`.

ANS:

1) `vPtr = values;`

2) `vPtr = &values[0];`

e) Print the elements of array `values` using pointer/offset notation.

ANS:

```
for ( i = 0; i < SIZE; i++ )
    printf( "%d", *( vPtr + i ) );
```

f) Print the elements of array `values` using pointer/offset notation with the array name as the pointer.

ANS:

```
for ( i = 0; i < SIZE; i++ )
    printf( "%d", *( values + i ) );
```

g) Print the elements of array `values` by subscripting the pointer to the array.

ANS:

```
for ( i = 0; i < SIZE; i++ )
    printf( "%d", vPtr[ i ] );
```

h) Refer to element 5 of array `values` using array subscript notation, pointer/offset notation with the array name as the pointer, pointer subscript notation, and pointer/offset notation.

ANS: `values[4]`, `*(values + 4)`, `vPtr[4]`, `*(vPtr + 4)`.

i) What address is referenced by `vPtr + 3`? What value is stored at that location?

ANS: 1002506; 8.

j) Assuming `vPtr` points to `values[4]`, what address is referenced by `vPtr -= 4`. What value is stored at that location?

ANS: 1002500; 2.

7.10 For each of the following, write a single statement that performs the indicated task. Assume that long integer variables `value1` and `value2` have been defined and that `value1` has been initialized to 200000.

a) Define the variable `lPtr` to be a pointer to an object of type `long`.

ANS: `long *lPtr;`

b) Assign the address of variable `value1` to pointer variable `lPtr`.

ANS: `lPtr = &value1;`

c) Print the value of the object pointed to by `lPtr`.

ANS: `printf("%ld\n", *lPtr);`

d) Assign the value of the object pointed to by `lPtr` to variable `value2`.

ANS: `value2 = *lPtr;`

e) Print the value of `value2`.

ANS: `printf("%ld\n", value2);`

f) Print the address of `value1`.

ANS: `printf("%p\n", &value1);`

g) Print the address stored in `lPtr`. Is the value printed the same as the address of `value1`?

ANS: `printf("%p\n", lPtr); /* The value is the same */`

7.11 Do each of the following.

a) Write the function header for function `zero`, which takes a long integer array parameter `bigIntegers` and does not return a value.

ANS: `void zero(long int *bigIntegers);`

b) Write the function prototype for the function in *Part a*.

ANS: `void zero(long int *);`

c) Write the function header for function `add1AndSum`, which takes an integer array parameter `oneTooSmall` and returns an integer.

ANS: `int add1AndSum(int *oneTooSmall);`

d) Write the function prototype for the function described in *Part c*.

ANS: `int add1AndSum(int *);`

Note: Exercise 7.12 through Exercise 7.15 are reasonably challenging. Once you have done these problems, you ought to be able to implement most popular card games easily.

7.12 Modify the program in Fig. 7.24 so that the card-dealing function deals a five-card poker hand. Then write the following additional functions:

- Determine if the hand contains a pair.
- Determine if the hand contains two pairs.
- Determine if the hand contains three of a kind (e.g., three jacks).
- Determine if the hand contains four of a kind (e.g., four aces).
- Determine if the hand contains a flush (i.e., all five cards of the same suit).
- Determine if the hand contains a straight (i.e., five cards of consecutive face values).

7.13 Use the functions developed in Exercise 7.12 to write a program that deals two five-card poker hands, evaluates each hand, and determines which is the better hand.

7.14 Modify the program developed in Exercise 7.13 so that it can simulate the dealer. The dealer's five-card hand is dealt “face down” so the player cannot see it. The program should then evaluate the dealer's hand, and based on the quality of the hand, the dealer should draw one, two or three more cards to replace the corresponding number of unneeded cards in the original hand. The program should then re-evaluate the dealer's hand. [*Caution:* This is a difficult problem!]

7.15 Modify the program developed in Exercise 7.14 so that it can handle the dealer's hand automatically, but the player is allowed to decide which cards of the player's hand to replace. The program should then evaluate both hands and determine who wins. Now use this new program to play 20 games against the computer. Who wins more games, you or the computer? Have one of your friends play 20 games against the computer. Who wins more games? Based on the results of these games, make appropriate modifications to refine your poker playing program (this, too, is a difficult problem). Play 20 more games. Does your modified program play a better game?

7.16 In the card shuffling and dealing program of Fig. 7.24, we intentionally used an inefficient shuffling algorithm that introduced the possibility of indefinite postponement. In this problem, you will create a high-performance shuffling algorithm that avoids indefinite postponement.

Modify the program of Fig. 7.24 as follows. Begin by initializing the deck array as shown in Fig. 7.29. Modify the `shuffle` function to loop row-by-row and column-by-column through the array touching every element once. Each element should be swapped with a randomly selected element of the array.

Print the resulting array to determine if the deck is satisfactorily shuffled (as in Fig. 7.30, for example). You may want your program to call the `shuffle` function several times to ensure a satisfactory shuffle.

Unshuffled deck array													
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	14	15	16	17	18	19	20	21	22	23	24	25	26
2	27	28	29	30	31	32	33	34	35	36	37	38	39
3	40	41	42	43	44	45	46	47	48	49	50	51	52

Fig. 7.29 Unshuffled deck array.

Sample shuffled deck array													
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	19	40	27	25	36	46	10	34	35	41	18	2	44
1	13	28	14	16	21	30	8	11	31	17	24	7	1
2	12	33	15	42	43	23	45	3	29	32	4	47	26
3	50	38	52	39	48	51	9	5	37	49	22	6	20

Fig. 7.30 Sample shuffled deck array.

Note that although the approach in this problem improves the shuffling algorithm, the dealing algorithm still requires searching the deck array for card 1, then card 2, then card 3, and so on. Worse yet, even after the dealing algorithm locates and deals the card, the algorithm continues searching through the remainder of the deck. Modify the program of Fig. 7.24 so that once a card is dealt, no further attempts are made to match that card number, and the program immediately proceeds with dealing the next card. In Chapter 10, we develop a dealing algorithm that requires only one operation per card.

ANS:

```

1  /* Exercise 7.16 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* function prototypes */
7  void shuffle( int workDeck[][ 13 ] );
8  void deal( int workDeck[][ 13 ], char *workFace[], char *workSuit[] );
9
10 int main()
11 {
12     int card = 1;          /* card counter */
13     int row;              /* loop counter */
14     int column;          /* loop counter */
15     int deck[ 4 ][ 13 ]; /* array of cards */
16
17     /* define arrays of card suits and faces */
18     char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
19     char *face[ 13 ] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
20         "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
21
22     srand( time( NULL ) );
23
24     /* initialize deck */
25     for ( row = 0; row <= 3; row++ ) {
26
27         for ( column = 0; column <= 12; column++ ) {
28             deck[ row ][ column ] = card++;
29         } /* end for */
30     } /* end for */
31
32     shuffle( deck );
33     deal( deck, face, suit );
34
35     return 0; /* indicate successful termination */
36 } /* end main */
37
38 /* introduce another way to shuffle */
39 void shuffle( int workDeck[][ 13 ] )
40 {
41     int temp;          /* temporary holder */
42     int row;          /* loop counter */
43     int column;      /* loop counter */
44     int randRow;     /* random suit */
45     int randColumn; /* random face */
46
47     /* run through the loop and touch every element once */
48     for ( row = 0; row <= 3; row++ ) {
49
50         for ( column = 0; column <= 12; column++ ) {
51
52
53

```

```
54     /* generate a random card */
55     randRow = rand() % 4;
56     randColumn = rand() % 13;
57
58     /* swap random card with current card */
59     temp = workDeck[ row ][ column ];
60     workDeck[ row ][ column ] = workDeck[ randRow ][ randColumn ];
61     workDeck[ randRow ][ randColumn ] = temp;
62 } /* end for */
63
64 } /* end for */
65
66 } /* end function shuffle */
67
68 /* deal the cards */
69 void deal( int workDeck2[][ 13 ], char *workFace[], char *workSuit[] )
70 {
71     int card; /* card counter */
72     int row; /* loop counter */
73     int column; /* loop counter */
74
75     /* loop through and print the cards */
76     for ( card = 1; card <= 52; card++ ) {
77
78         /* loop through rows */
79         for ( row = 0; row <= 3; row++ ) {
80
81             /* loop through columns */
82             for ( column = 0; column <= 12; column++ ) {
83
84                 /* if current card equals card then deal */
85                 if ( workDeck2[ row ][ column ] == card ) {
86                     printf( "%5s of %-8s", workFace[ column ], workSuit[ row ] );
87                     card % 2 == 0 ? putchar( '\n' ) : putchar( '\t' );
88                     break; /* break loop */
89                 } /* end if */
90
91             } /* end for */
92
93         } /* end for */
94
95     } /* end for */
96
97 } /* end function deal */
```


Eight of Spades	Ace of Spades
Five of Hearts	Ace of Hearts
Eight of Diamonds	Queen of Spades
Deuce of Hearts	Seven of Hearts
Seven of Clubs	Six of Hearts
Four of Clubs	Ace of Clubs
Six of Spades	Ten of Diamonds
Ten of Hearts	King of Hearts
Four of Diamonds	Four of Hearts
Jack of Diamonds	Three of Diamonds
Deuce of Spades	Queen of Clubs
Three of Hearts	Six of Clubs
Nine of Hearts	Nine of Diamonds
King of Spades	Seven of Diamonds
Five of Spades	Seven of Spades
Four of Spades	Ten of Spades
King of Diamonds	Nine of Spades
Deuce of Clubs	Jack of Hearts
Ace of Diamonds	Ten of Clubs
Eight of Hearts	Six of Diamonds
Nine of Clubs	Five of Diamonds
Three of Clubs	Deuce of Diamonds
Queen of Hearts	King of Clubs
Queen of Diamonds	Jack of Clubs
Five of Clubs	Three of Spades
Jack of Spades	Eight of Clubs

7.17 (*Simulation: The Tortoise and the Hare*) In this problem, you will recreate one of the truly great moments in history, namely the classic race of the tortoise and the hare. You will use random number generation to develop a simulation of this memorable event.

Our contenders begin the race at “square 1” of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground.

There is a clock that ticks once per second. With each tick of the clock, your program should adjust the position of the animals according to the rules of Fig. 7.31.

Animal	Move type	Percentage of the time	Actual move
Tortoise	Fast plod	50%	3 squares to the right
	Slip	20%	6 squares to the left
	Slow plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big hop	20%	9 squares to the right
	Big slip	10%	12 squares to the left
	Small hop	30%	1 square to the right
	Small slip	20%	2 squares to the left

Use variables to keep track of the positions of the animals (i.e., position numbers are 1–70). Start each animal at position 1 (i.e., the “starting gate”). If an animal slips left before square 1, move the animal back to square 1.

Generate the percentages in the preceding table by producing a random integer, i , in the range $1 \leq i \leq 10$. For the tortoise, perform a “fast plod” when $1 \leq i \leq 5$, a “slip” when $6 \leq i \leq 7$, or a “slow plod” when $8 \leq i \leq 10$. Use a similar technique to move the hare.

Begin the race by printing

```
BANG !!!!!
AND THEY'RE OFF !!!!!
```

Then, for each tick of the clock (i.e., each repetition of a loop), print a 70 position line showing the letter T in the position of the tortoise and the letter H in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare and your program should print OUCH!!! beginning at that position. All print positions other than the T, the H, or the OUCH!!! (in case of a tie) should be blank.

After each line is printed, test if either animal has reached or passed square 70. If so, then print the winner and terminate the simulation. If the tortoise wins, print TORTOISE WINS!!! YAY!!! If the hare wins, print Hare wins. Yuch. If both animals win on the same tick of the clock, you may want to favor the turtle (the “underdog”), or you may want to print It's a tie. If neither animal wins, perform the loop again to simulate the next tick of the clock. When you are ready to run your program, assemble a group of fans to watch the race. You'll be amazed at how involved your audience gets!

ANS:

```
1  /* Exercise 7.17 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* function prototypes */
7  void moveTortoise( int *turtlePtr );
8  void moveHare( int *rabbitPtr );
9  void printCurrentPositions( int *snapperPtr, int *bunnyPtr );
10
11 int main()
12 {
13     int tortoise = 1; /* tortoise current position */
14     int hare = 1;    /* hare current position */
15     int timer = 0;  /* time elapsed during race */
16
```

```
17     srand( time( NULL ) );
18
19     printf( "ON YOUR MARK, GET SET\n" );
20     printf( "BANG          !!!!\n" );
21     printf( "AND THEY'RE OFF  !!!!\n" );
22
23     /* loop through the events */
24     while ( tortoise != 70 && hare != 70 ) {
25         moveTortoise( &tortoise );
26         moveHare( &hare );
27         printCurrentPositions( &tortoise, &hare );
28         ++timer;
29     } /* end while */
30
31     /* determine the winner and print message */
32     if ( tortoise >= hare ) {
33         printf( "\nTORTOISE WINS!!! YAY!!!\n" );
34     } /* end if */
35     else {
36         printf( "Hare wins. Yuch.\n" );
37     } /* end else */
38
39     printf( "TIME ELAPSED = %d seconds", timer );
40
41     return 0; /* indicate successful termination */
42
43 } /* end main */
44
45 /* progress for the tortoise */
46 void moveTortoise( int *turtlePtr )
47 {
48     int x; /* random number */
49
50     x = rand() % 10 + 1; /* generate random number from 1-10 */
51
52     /* determine progress */
53     if ( x >= 1 && x <= 5 ) { /* fast plod */
54         *turtlePtr += 3;
55     } /* end if */
56     else if ( x == 6 || x == 7 ) { /* slip */
57         *turtlePtr -= 6;
58     } /* end else if */
59     else { /* slow plod */
60         ++( *turtlePtr );
61     } /* end else */
62
63     /* check boundaries */
64     if ( *turtlePtr < 1 ) {
65         *turtlePtr = 1;
66     } /* end if */
67     if ( *turtlePtr > 70 ) {
68         *turtlePtr = 70;
69     } /* end if */
70
71 } /* end function moveTortoise */
72
73 /* progress for the hare */
74 void moveHare( int *rabbitPtr )
75 {
76     int y; /* random number */
77
```

```
78     y = rand() % 10 + 1; /* generate random number from 1-10 */
79
80     /* determine progress */
81     if ( y == 3 || y == 4 ) { /* big hop */
82         *rabbitPtr += 9;
83     } /* end if */
84     else if ( y == 5 ) { /* big slip */
85         *rabbitPtr -= 12;
86     } /* end else if */
87     else if ( y >= 6 && y <= 8 ) { /* small hop */
88         ++( *rabbitPtr );
89     } /* end else if */
90     else if ( y == 10 ) { /* small slip */
91         *rabbitPtr -= 2;
92     } /* end else if */
93
94     /* check boundaries */
95     if ( *rabbitPtr < 1 ) {
96         *rabbitPtr = 1;
97     } /* end if */
98
99     if ( *rabbitPtr > 70 ) {
100        *rabbitPtr = 70;
101    } /* end if */
102
103 } /* end function moveHare */
104
105 /* display new position */
106 void printCurrentPositions( int *snapperPtr, int *bunnyPtr )
107 {
108     int count; /* counter */
109
110     /* loop through race */
111     for ( count = 1; count <= 70; count++ )
112
113         /* print current leader */
114         if ( count == *snapperPtr && count == *bunnyPtr ) {
115             printf( "OUCH!!!" );
116         } /* end if */
117         else if ( count == *bunnyPtr ) {
118             printf( "H" );
119         } /* end else if */
120         else if ( count == *snapperPtr ) {
121             printf( "T" );
122         } /* end else if */
123         else {
124             printf( " " );
125         } /* end else */
126
127     printf( "\n" );
128 } /* end function printCurrentPositions */
```


SPECIAL SECTION: BUILDING YOUR OWN COMPUTER

In the next several problems, we take a temporary diversion away from the world of high-level language programming. We “peel open” a computer and look at its internal structure. We introduce machine language programming and write several machine language programs. To make this an especially valuable experience, we then build a computer (through the technique of software-based *simulation*) on which you can execute your machine language programs!

7.18 (*Machine Language Programming*) Let us create a computer we will call the Simpletron. As its name implies, it is a simple machine, but as we will soon see, a powerful one as well. The Simpletron runs programs written in the only language it directly understands—that is, Simpletron Machine Language, or SML for short.

The Simpletron contains an *accumulator*—a “special register” in which information is put before the Simpletron uses that information in calculations or examines it in various ways. All information in the Simpletron is handled in terms of *words*. A word is a signed four-digit decimal number such as +3364, -1293, +0007, -0001, etc. The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers 00, 01, ..., 99.

Before running an SML program, we must *load* or place the program into memory. The first instruction (or statement) of every SML program is always placed in location 00.

Each instruction written in SML occupies one word of the Simpletron's memory (and hence instructions are signed four-digit decimal numbers). We assume that the sign of an SML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the Simpletron's memory may contain either an instruction, a data value used by a program or an unused (and hence undefined) area of memory. The first two digits of each SML instruction are the *operation code*, which specifies the operation to be performed. SML operation codes are summarized in Fig. Fig. 7.32.

Operation code	Meaning
<i>Input/output operations:</i>	
<code>#define READ 10</code>	Read a word from the terminal into a specific location in memory.
<code>#define WRITE 11</code>	Write a word from a specific location in memory to the terminal.
<i>Load/store operations:</i>	
<code>#define LOAD 20</code>	Load a word from a specific location in memory into the accumulator.
<code>#define STORE 21</code>	Store a word from the accumulator into a specific location in memory.
<i>Arithmetic operations:</i>	
<code>#define ADD 30</code>	Add a word from a specific location in memory to the word in the accumulator (leave result in accumulator).
<code>#define SUBTRACT 31</code>	Subtract a word from a specific location in memory from the word in the accumulator (leave result in accumulator).
<code>#define DIVIDE 32</code>	Divide a word from a specific location in memory into the word in the accumulator (leave result in accumulator).
<code>#define MULTIPLY 33</code>	Multiply a word from a specific location in memory by the word in the accumulator (leave result in accumulator).
<i>Transfer of control operations:</i>	
<code>#define BRANCH 40</code>	Branch to a specific location in memory.
<code>#define BRANCHNEG 41</code>	Branch to a specific location in memory if the accumulator is negative.
<code>#define BRANCHZERO 42</code>	Branch to a specific location in memory if the accumulator is zero.
<code>#define HALT 43</code>	Halt—i.e., the program has completed its task.

Fig. 7.32 Simpletron Machine Language (SML) operation codes.

The last two digits of an SML instruction are the *operand*, which is the address of the memory location containing the word to which the operation applies. Now let us consider several simple SML programs.

Example 1 Location	Number	Instruction
00	+1007	<i>(Read A)</i>
01	+1008	<i>(Read B)</i>
02	+2007	<i>(Load A)</i>
03	+3008	<i>(Add B)</i>
04	+2109	<i>(Store C)</i>
05	+1109	<i>(Write C)</i>
06	+4300	<i>(Halt)</i>
07	+0000	<i>(Variable A)</i>
08	+0000	<i>(Variable B)</i>
09	+0000	<i>(Result C)</i>

The preceding SML program reads two numbers from the keyboard, and computes and prints their sum. The instruction +1007 reads the first number from the keyboard and places it into location 07 (which has been initialized to zero). Then +1008 reads the next number into location 08. The *load* instruction, +2007, puts the first number into the accumulator, and the *add* instruction, +3008, adds the second number to the number in the accumulator. *All SML arithmetic instructions leave their results in the accumulator.* The *store* instruction, +2109, places the result back into memory location 09 from which the *write* instruction, +1109, takes the number and prints it (as a signed four-digit decimal number). The *halt* instruction, +4300, terminates execution.

Example 2 Location	Number	Instruction
00	+1009	<i>(Read A)</i>
01	+1010	<i>(Read B)</i>
02	+2009	<i>(Load A)</i>
03	+3110	<i>(Subtract B)</i>
04	+4107	<i>(Branch negative to 07)</i>
05	+1109	<i>(Write A)</i>
06	+4300	<i>(Halt)</i>
07	+1110	<i>(Write B)</i>
08	+4300	<i>(Halt)</i>
09	+0000	<i>(Variable A)</i>
10	+0000	<i>(Variable B)</i>

The preceding SML program reads two numbers from the keyboard, and determines and prints the larger value. Note the use of the instruction +4107 as a conditional transfer of control, much the same as C's `if` statement. Now write SML programs to accomplish each of the following tasks.

- a) Use a sentinel-controlled loop to read 10 positive integers and compute and print their sum.

ANS:

```

00 +1009 (Read Value)
01 +2009 (Load Value)
02 +4106 (Branch negative to 06)
03 +3008 (Add Sum)
04 +2108 (Store Sum)
05 +4000 (Branch 00)
06 +1108 (Write Sum)
07 +4300 (Halt)
08 +0000 (Variable Sum)
09 +0000 (Variable Value)

```

- b) Use a counter-controlled loop to read seven numbers, some positive and some negative, and compute and print their average.

ANS:

```

00 +2018 (Load Counter)
01 +3121 (Subtract Termination)
02 +4211 (Branch zero to 11)
03 +2018 (Load Counter)
04 +3019 (Add Increment)
05 +2118 (Store Counter)
06 +1017 (Read Value)
07 +2016 (Load Sum)
08 +3017 (Add Value)
09 +2116 (Store Sum)
10 +4000 (Branch 00)
11 +2016 (Load Sum)
12 +3218 (Divide Counter)
13 +2120 (Store Result)
14 +1120 (Write Result)
15 +4300 (Halt)
16 +0000 (Variable Sum)
17 +0000 (Variable Value)
18 +0000 (Variable Counter)
19 +0001 (Variable Increment)
20 +0000 (Variable Result)
21 +0007 (Variable Termination)

```

- c) Read a series of numbers and determine and print the largest number. The first number read indicates how many numbers should be processed.

ANS:

```

00 +1017 (Read Endvalue)
01 +2018 (Load Counter)
02 +3117 (Subtract Endvalue)
03 +4215 (Branch zero to 15)
04 +2018 (Load Counter)
05 +3021 (Add Increment)
06 +2118 (Store Counter)
07 +1019 (Read Value)
08 +2020 (Load Largest)
09 +3119 (Subtract Value)
10 +4112 (Branch negative to 12)
11 +4001 (Branch 01)
12 +2019 (Load Value)
13 +2120 (Store Largest)
14 +4001 (Branch 01)
15 +1120 (Write Largest)
16 +4300 (Halt)
17 +0000 (Variable Endvalue)
18 +0000 (Variable Counter)
19 +0000 (Variable Value)
20 +0000 (Variable Largest)
21 +0001 (Variable Increment)

```


7.19 (*A Computer Simulator*) It may at first seem outrageous, but in this problem you are going to build your own computer. No, you will not be soldering components together. Rather, you will use the powerful technique of *software-based simulation* to create a *software model* of the Simpletron. You will not be disappointed. Your Simpletron simulator will turn the computer you are using into a Simpletron, and you will actually be able to run, test and debug the SML programs you wrote in Exercise 7.18.

When you run your Simpletron simulator, it should begin by printing:

```
*** Welcome to Simpletron! ***
*** Please enter your program one instruction ***
*** (or data word) at a time. I will type the ***
*** location number and a question mark (?). ***
*** You then type the word for that location. ***
*** Type the sentinel -99999 to stop entering ***
*** your program. ***
```

Simulate the memory of the Simpletron with a single-subscripted array `memory` that has 100 elements. Now assume that the simulator is running, and let us examine the dialog as we enter the program of Example 2 of Exercise 7.18:

```
00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300
07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
11 ? -99999
*** Program loading completed ***
*** Program execution begins ***
```

The SML program has now been placed (or loaded) into the array memory. Now the Simpletron executes your SML program. Execution begins with the instruction in location 00 and, like C, continues sequentially, unless directed to some other part of the program by a transfer of control.

Use the variable `accumulator` to represent the accumulator register. Use the variable `instructionCounter` to keep track of the location in memory that contains the instruction being performed. Use the variable `operationCode` to indicate the operation currently being performed—i.e., the left two digits of the instruction word. Use the variable `operand` to indicate the memory location on which the current instruction operates. Thus, `operand` is the rightmost two digits of the instruction currently being performed. Do not execute instructions directly from memory. Rather, transfer the next instruction to be performed from memory to a variable called `instructionRegister`. Then “pick off” the left two digits and place them in the variable `operationCode`, and “pick off” the right two digits and place them in `operand`.

When Simpletron begins execution, the special registers are initialized as follows:

```
accumulator          +0000
instructionCounter    00
instructionRegister    +0000
operationCode         00
operand              00
```

Now let us “walk through” the execution of the first SML instruction, +1009 in memory location 00. This is called an *instruction execution cycle*.

The `instructionCounter` tells us the location of the next instruction to be performed. We *fetch* the contents of that location from memory by using the C statement

```
instructionRegister = memory[ instructionCounter ];
```

The operation code and the operand are extracted from the instruction register by the statements

```
operationCode = instructionRegister / 100;
operand = instructionRegister % 100;
```

Now the Simpletron must determine that the operation code is actually a *read* (versus a *write*, a *load*, etc.). A `switch` differentiates among the twelve operations of SML.

In the `switch` statement, the behavior of various SML instructions is simulated as follows (we leave the others to the reader):

```

read:  scanf( "%d", &memory[ operand ] );
load:  accumulator = memory[ operand ];
add:   accumulator += memory[ operand ];
Various branch instructions: We'll discuss these shortly.
halt:  This instruction prints the message

*** Simpletron execution terminated ***

```

then prints the name and contents of each register as well as the complete contents of memory. Such a printout is often called a *computer dump*. To help you program your dump function, a sample dump format is shown in Fig. 7.33. Note that a dump after executing a Simpletron program would show the actual values of instructions and data values at the moment execution terminated.

REGISTERS:										
accumulator										+0000
instructionCounter										00
instructionRegister										+0000
operationCode										00
operand										00
MEMORY:										
	0	1	2	3	4	5	6	7	8	9
0	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
10	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
20	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
30	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
40	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
50	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
60	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
70	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
80	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
90	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000

Fig. 7.33 Sample dump of Simpletron's memory.

Let us proceed with the execution of our program's first instruction, namely the +1009 in location 00. As we have indicated, the `switch` statement simulates this by performing the C statement

```
scanf( "%d", &memory[ operand ] );
```

A question mark (?) should be displayed on the screen before the `scanf` is executed to prompt the user for input. The Simpletron waits for the user to type a value and then press the *Return* key. The value is then read into location 09.

At this point, simulation of the first instruction is completed. All that remains is to prepare the Simpletron to execute the next instruction. Since the instruction just performed was not a transfer of control, we need merely increment the instruction counter register as follows:

```
++instructionCounter;
```

This completes the simulated execution of the first instruction. The entire process (i.e., the instruction execution cycle) begins anew with the fetch of the next instruction to be executed.

Now let us consider how the branching instructions—the transfers of control—are simulated. All we need to do is adjust the value in the instruction counter appropriately. Therefore, the unconditional branch instruction (40) is simulated within the `switch` as

```
instructionCounter = operand;
```

The conditional “branch if accumulator is zero” instruction is simulated as

```

    if ( accumulator == 0 )
        instructionCounter = operand;

```

At this point, you should implement your Simpletron simulator and run the SML programs you wrote in Exercise 7.18. You may embellish SML with additional features and provide for these in your simulator.

Your simulator should check for various types of errors. During the program loading phase, for example, each number the user types into the Simpletron's memory must be in the range -9999 to +9999. Your simulator should use a `while` loop to test that each number entered is in this range, and, if not, keep prompting the user to reenter the number until the user enters a correct number.

During the execution phase, your simulator should check for various serious errors, such as attempts to divide by zero, attempts to execute invalid operation codes and accumulator overflows (i.e., arithmetic operations resulting in values larger than +9999 or smaller than -9999). Such serious errors are called *fatal errors*. When a fatal error is detected, your simulator should print an error message such as:

```

*** Attempt to divide by zero ***
*** Simpletron execution abnormally terminated ***

```

and should print a full computer dump in the format we have discussed previously. This will help the user locate the error in the program.

ANS:

```

1  /* Exercise 7.19 Solution */
2  #include <stdio.h>
3
4  /* define commands */
5  #define SIZE 100
6  #define SENTINEL -99999
7  #define TRUE 1
8  #define FALSE 0
9  #define READ 10
10 #define WRITE 11
11 #define LOAD 20
12 #define STORE 21
13 #define ADD 30
14 #define SUBTRACT 31
15 #define DIVIDE 32
16 #define MULTIPLY 33
17 #define BRANCH 40
18 #define BRANCHNEG 41
19 #define BRANCHZERO 42
20 #define HALT 43
21
22 /* function prototypes */
23 void load( int *loadMemory );
24 void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
25             int *opCodePtr, int *opPtr );
26 void dump( int *memory, int accumulator, int instructionCounter,
27           int instructionRegister, int operationCode,
28           int operand );
29 int validWord( int word );
30
31 int main()
32 {
33     int memory[ SIZE ]; /* define memory array */
34     int ac = 0;         /* accumulator */
35     int ic = 0;         /* instruction counter */
36     int opCode = 0;     /* operation code */
37     int op = 0;         /* operand */
38     int ir = 0;         /* instruction register */
39     int i;              /* counter */
40

```

```

41     /* clear memory */
42     for ( i = 0; i < SIZE; i++ ) {
43         memory[ i ] = 0;
44     } /* end for */
45
46     load( memory );
47     execute( memory, &ac, &ic, &ir, &opCode, &op );
48     dump( memory, ac, ic, ir, opCode, op );
49
50     return 0; /* indicate successful termination */
51 } /* end main */
52
53
54 /* function loads instructions */
55 void load( int *loadMemory )
56 {
57     long int instruction; /* current instruction */
58     int i = 0;           /* indexing variable */
59
60     printf( "%s\n\n%s\n\n%s\n\n%s\n\n%s\n\n",
61           "**** Welcome to Simpletron ****",
62           "**** Please enter your program one instruction ****",
63           "**** ( or data word ) at a time. I will type the ****",
64           "**** location number and a question mark ( ? ). ****",
65           "**** You then type the word for that location. ****",
66           "**** Type the sentinel -99999 to stop entering ****",
67           "**** your program. ****" );
68
69     printf( "00 ? " );
70     scanf( "%ld", &instruction ); /* read instruction */
71
72     /* while sentinel is not read from user */
73     while ( instruction != SENTINEL ) {
74
75         /* test instruction for validity */
76         if ( !validWord( instruction ) ) {
77             printf( "Number out of range. Please enter again.\n" );
78         } /* end if */
79         else { /* load instruction */
80             loadMemory[ i++ ] = instruction;
81         } /* end else */
82
83         printf( "%02d ? ", i );
84         scanf( "%ld", &instruction );
85     } /* end while */
86
87 } /* end function load */
88
89 /* carry out the commands */
90 void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
91             int *opCodePtr, int *opPtr )
92 {
93     int fatal = FALSE; /* fatal error flag */
94     int temp;         /* temporary holding space */
95
96     printf( "\n*****START SIMPLETRON EXECUTION*****\n\n" );
97
98     /* separate operation code and operand */
99     *irPtr = memory[ *icPtr ];
100    *opCodePtr = *irPtr / 100;
101    *opPtr = *irPtr % 100;

```

```

102
103 /* loop while command is not HALT or fatal */
104 while ( *opCodePtr != HALT && !fatal ) {
105
106     /* determine appropriate action */
107     switch ( *opCodePtr ) {
108
109         /* read data into location in memory */
110         case READ:
111             printf( "Enter an integer: " );
112             scanf( "%d", &temp );
113
114             /* check for validity */
115             while ( !validWord( temp ) ) {
116                 printf( "Number out of range. Please enter again: " );
117                 scanf( "%d", &temp );
118             } /* end while */
119
120             memory[ *opPtr ] = temp; /* write to memory */
121             ++( *icPtr );
122             break; /* exit switch */
123
124         /* write data from memory to screen */
125         case WRITE:
126             printf( "Contents of %02d: %d\n", *opPtr, memory[ *opPtr ] );
127             ++( *icPtr );
128             break; /* exit switch */
129
130         /* load data from memory into accumulator */
131         case LOAD:
132             *acPtr = memory[ *opPtr ];
133             ++( *icPtr );
134             break; /* exit switch */
135
136         /* store data from accumulator into memory */
137         case STORE:
138             memory[ *opPtr ] = *acPtr;
139             ++( *icPtr );
140             break; /* exit switch */
141
142         /* add data from memory to data in accumulator */
143         case ADD:
144             temp = *acPtr + memory[ *opPtr ];
145
146             /* check validity */
147             if ( !validWord( temp ) ) {
148                 printf( "**** FATAL ERROR: Accumulator overflow      ***\n" );
149                 printf( "**** Simpletron execution abnormally terminated ***\n" );
150                 fatal = TRUE;
151             } /* end if */
152             else {
153                 *acPtr = temp;
154                 ++( *icPtr );
155             } /* end else */
156
157             break; /* exit switch */
158
159         /* subtract data in memory from data in accumulator */
160         case SUBTRACT:
161             temp = *acPtr - memory[ *opPtr ];
162

```

```

163     /* check validity */
164     if ( !validWord( temp ) ) {
165         printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
166         printf( "*** Simpletron execution abnormally terminated ***\n" );
167         fatal = TRUE;
168     } /* end if */
169     else {
170         *acPtr = temp;
171         ++( *icPtr );
172     } /* end else */
173
174     break; /* exit switch */
175
176     /* divide data in memory into data in accumulator */
177     case DIVIDE:
178
179         /* check for divide by zero error */
180         if ( memory[ *opPtr ] == 0 ) {
181             printf( "*** FATAL ERROR: Attempt to divide by zero    ***\n" );
182             printf( "*** Simpletron execution abnormally terminated ***\n" );
183             fatal = TRUE;
184         } /* end if */
185         else {
186             *acPtr /= memory[ *opPtr ];
187             ++( *icPtr );
188         } /* end else */
189
190         break; /* exit switch */
191
192     /* multiple data in memory by data in accumulator */
193     case MULTIPLY:
194         temp = *acPtr * memory[ *opPtr ];
195
196         /* check validity */
197         if ( !validWord( temp ) ) {
198             printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
199             printf( "*** Simpletron execution abnormally terminated ***\n" );
200             fatal = TRUE;
201         } /* end if */
202         else {
203             *acPtr = temp;
204             ++( *icPtr );
205         } /* end else */
206
207         break; /* exit switch */
208
209     /* branch to specific location in memory */
210     case BRANCH:
211         *icPtr = *opPtr;
212         break; /* exit switch */
213
214     /* branch to location in memory if accumulator is negative */
215     case BRANCHNEG:
216
217         /* if accumulator is negative */
218         if ( *acPtr < 0 ) {
219             *icPtr = *opPtr;
220         } /* end if */
221         else {
222             ++( *icPtr );
223         } /* end else */

```

```

224         break; /* exit switch */
225
226     /* branch to location in memory if accumulator is zero */
227     case BRANCHZERO:
228
229         /* if accumulator is zero */
230         if ( *acPtr == 0 ) {
231             *icPtr = *opPtr;
232         } /* end if */
233         else {
234             ++( *icPtr );
235         } /* end else */
236
237         break; /* exit switch */
238
239     default:
240         printf( "*** FATAL ERROR: Invalid opcode detected      ***\n" );
241         printf( "*** Simpletron execution abnormally terminated ***\n" );
242         fatal = TRUE;
243         break; /* exit switch */
244     } /* end switch */
245 } /* end while */
246
247 /* separate next operation code and operand */
248 *irPtr = memory[ *icPtr ];
249 *opCodePtr = *irPtr / 100;
250 *opPtr = *irPtr % 100;
251 } /* end while */
252
253 printf( "\n*****END SIMPLETRON EXECUTION*****\n" );
254 } /* end function execute */
255
256 /* print out name and content of each register and memory */
257 void dump( int *memory, int accumulator, int instructionCounter,
258           int instructionRegister, int operationCode,
259           int operand )
260 {
261     int i; /* counter */
262
263     printf( "\n%s\n%-23s%+05d\n%-23s%5.2d\n%-23s%+05d\n%-23s%5.2d\n%-23s%5.2d",
264           "REGISTERS:", "accumulator", accumulator, "instructioncounter",
265           instructionCounter, "instructionregister", instructionRegister,
266           "operationcode", operationCode, "operand", operand );
267
268     printf( "\n\nMEMORY:\n  " );
269
270     /* print column headers */
271     for ( i = 0; i <= 9; i++ ) {
272         printf( "%5d ", i );
273     } /* end for */
274
275     /* print row headers and memory contents */
276     for ( i = 0; i < SIZE; i++ ) {
277
278         /* print in increments of 10 */
279         if ( i % 10 == 0 ) {
280             printf( "\n%2d ", i );
281         } /* end if */
282
283         printf( "%+05d ", memory[ i ] );
284     } /* end for */

```

```

285
286     printf( "\n" );
287 } /* end function dump */
288
289 /* function tests validity of word */
290 int validWord( int word )
291 {
292     return word >= -9999 && word <= 9999;
293 }
294 } /* end function validWord */
295

```

```

***           Welcome to Simpletron           ***

*** Please enter your program one instruction ***
*** ( or data word ) at a time. I will type the ***
*** location number and a question mark ( ? ). ***
*** You then type the word for that location. ***
*** Type the sentinel -99999 to stop entering ***
*** your program.                               ***

00 ? 1007
01 ? 1008
02 ? 2007
03 ? 3008
04 ? 2109
05 ? 1109
06 ? 4300
07 ? 0000
08 ? 0000
09 ? 0000
10 ? -99999

*****START SIMPLETRON EXECUTION*****

Enter an integer: 23
Enter an integer: 17
Contents of 09: 40

*****END SIMPLETRON EXECUTION*****

REGISTERS:
accumulator           +0040
instructioncounter      06
instructionregister     +4300
operationcode          43
operand                00

MEMORY:
   0  +1007 +1008 +2007 +3008 +2109 +1109 +4300 +0023 +0017 +0040
  10  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  20  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  30  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  40  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  50  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  60  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  70  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  80  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  90  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

```


7.20 Modify the card shuffling and dealing program of Fig. 7.24 so the shuffling and dealing operations are performed by the same function (`shuffleAndDeal`). The function should contain one nested looping structure that is similar to function `shuffle` in Fig. 7.24.

ANS:

```

1  /* Exercise 7.20 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* function prototype */
7  void shuffleAndDeal( int workdeck[][ 13 ], char *workface[],
8                      char *worksuit[] );
9
10 int main()
11 {
12
13     /* define card suit array and card face array */
14     char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
15     char *face[ 13 ] = { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
16                          "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
17     int deck[ 4 ][ 13 ] = { 0 }; /* array of cards */
18
19     srand( time( NULL ) );
20
21     shuffleAndDeal( deck, face, suit );
22
23     return 0; /* indicate successful termination */
24 } /* end main */
25
26
27 /* integrate shuffling and dealing operation */
28 void shuffleAndDeal( int workdeck[][ 13 ], char *workface[],
29                     char *worksuit[] )
30 {
31     int card; /* card loop counter */
32     int row; /* current suit */
33     int column; /* current face */
34
35     /* loop through the deck of cards, shuffle and print */
36     for ( card = 1; card <= 52; card++ ) {
37
38         /* choose random card until not equal to zero */
39         do {
40             row = rand() % 4;
41             column = rand() % 13;
42         } while( workdeck[ row ][ column ] != 0 ); /* end do...while */
43
44         workdeck[ row ][ column ] = card;
45
46         /* deal card */
47         printf( "%5s of %-8s", workface[ column ], worksuit[ row ] );
48
49         card % 2 == 0 ? printf( "\n" ) : printf( "\t" );
50     } /* end for */
51 } /* end function shuffleAndDeal */
52

```

Seven of Spades	King of Diamonds
Six of Spades	King of Hearts
Three of Clubs	Three of Diamonds
Jack of Diamonds	Jack of Spades
Queen of Clubs	Eight of Hearts
Four of Hearts	Deuce of Clubs
Six of Clubs	Eight of Spades
Three of Hearts	Five of Hearts
Seven of Clubs	Ace of Hearts
Ten of Hearts	Five of Diamonds
Queen of Hearts	Eight of Clubs
Five of Clubs	Deuce of Diamonds
Deuce of Spades	Jack of Hearts
Ace of Clubs	Nine of Diamonds
Five of Spades	Nine of Clubs
Deuce of Hearts	King of Spades
Nine of Hearts	Queen of Spades
King of Clubs	Four of Spades
Seven of Diamonds	Ace of Diamonds
Six of Hearts	Ten of Diamonds
Nine of Spades	Queen of Diamonds
Three of Spades	Jack of Clubs
Four of Diamonds	Ace of Spades
Ten of Spades	Ten of Clubs
Four of Clubs	Eight of Diamonds
Six of Diamonds	Seven of Hearts

7.21 What does this program do?

```
1  /* ex07_21.c */
2  /* What does this program do? */
3  #include <stdio.h>
4
5  void mystery1( char *s1, const char *s2 ); /* prototype */
6
7  int main()
8  {
9      char string1[ 80 ]; /* create char array */
10     char string2[ 80 ]; /* create char array */
11
12     printf( "Enter two strings: " );
13     scanf( "%s%s" , string1, string2 );
14
15     mystery1( string1, string2 );
16
17     printf("%s", string1 );
18
19     return 0; /* indicates successful termination */
20
21 } /* end main */
22
23 /* What does this function do? */
24 void mystery1( char *s1, const char *s2 )
25 {
26     while ( *s1 != '\0' ) {
27         s1++;
28     } /* end while */
29
30     for ( ; *s1 = *s2; s1++, s2++ ) {
31         ; /* empty statement */
32     } /* end for */
33
34 } /* end function mystery1 */
```

ANS: Concatenates strings.

```
Enter two strings: string1 string2
string1string2
```

7.22 What does this program do?

```
1  /* ex07_22.c */
2  /* what does this program do? */
3  #include <stdio.h>
4
5  int mystery2( const char *s ); /* prototype */
6
7  int main()
8  {
9      char string[ 80 ]; /* create char array */
10
11     printf( "Enter a string: ");
12     scanf( "%s", string );
13
14     printf( "%d\n", mystery2( string ) );
15
16     return 0; /* indicates successful termination */
17
18 } /* end main */
19
20 /* What does this function do? */
21 int mystery2( const char *s )
22 {
23     int x; /* counter */
24
25     /* loop through string */
26     for ( x = 0; *s != '\0'; s++ ) {
27         x++;
28     } /* end for */
29
30     return x;
31
32 } /* end function mystery2 */
```

ANS: Determines the length of a string.

```
Enter a string: string1
7
```

7.23 Find the error in each of the following program segments. If the error can be corrected, explain how.

a) `int *number;`
`printf("%d\n", *number);`

ANS: number has not been assigned to point to a location in memory.

b) `float *realPtr;`
`long *integerPtr;`
`integerPtr = realPtr;`

ANS: A pointer cannot be assigned to a different type, other than void *.

c) `int * x, y;`
`x = y;`

ANS: There are two possible solutions. 1) The indirection operator (*) is not distributive and would be required for y, which would result in a valid pointer assignment. 2) y as it is defined is a valid integer variable, and would require the address operator (&) in the pointer assignment statement.

d) `char s[] = "this is a character array";`
`int count;`
`for (; *s != '\0'; s++)`
`printf("%c ", *s);`

ANS: s should be defined as char *, a constant pointer cannot be moved.

e) `short *numPtr, result;`
`void *genericPtr = numPtr;`
`result = *genericPtr + 7;`

ANS: A void * pointer cannot be dereferenced.

f) `float x = 19.34;`
`float xPtr = &x;`
`printf("%f\n", xPtr);`

ANS: xPtr is not defined as a pointer so it should be dereferenced as well.

g) `char *s;`
`printf("%s\n", s);`

ANS: s has not been assigned a value, it does not point to anything.

7.24 (*Quicksort*) In the examples and exercises of Chapter 6, we discussed the sorting techniques bubble sort, bucket sort and selection sort. We now present the recursive sorting technique called Quicksort. The basic algorithm for a single-subscripted array of values is as follows:

- a) *Partitioning Step*: Take the first element of the unsorted array and determine its final location in the sorted array (i.e., all values to the left of the element in the array are less than the element, and all values to the right of the element in the array are greater than the element). We now have one element in its proper location and two unsorted subarrays.
- b) *Recursive Step*: Perform *Step 1* on each unsorted subarray.

Each time *Step 1* is performed on a subarray, another element is placed in its final location of the sorted array, and two unsorted subarrays are created. When a subarray consists of one element, it must be sorted; therefore, that element is in its final location.

The basic algorithm seems simple enough, but how do we determine the final position of the first element of each subarray. As an example, consider the following set of values (the element in bold is the partitioning element—it will be placed in its final location in the sorted array):

37 2 6 4 89 8 10 12 68 45

- a) Starting from the rightmost element of the array, compare each element with **37** until an element less than **37** is found. Then swap **37** and that element. The first element less than **37** is 12, so **37** and 12 are swapped. The new array is

12 2 6 4 89 8 10 **37** 68 45

Element 12 is in italic to indicate that it was just swapped with **37**.

- b) Starting from the left of the array, but beginning with the element after 12, compare each element with **37** until an element greater than **37** is found. Then swap **37** and that element. The first element greater than **37** is 89, so **37** and 89 are swapped. The new array is

12 2 6 4 **37** 8 10 89 68 45

- c) Starting from the right, but beginning with the element before 89, compare each element with **37** until an element less than **37** is found. Then swap **37** and that element. The first element less than **37** is 10, so **37** and 10 are swapped. The new array is

12 2 6 4 10 8 **37** 89 68 45

- d) Starting from the left, but beginning with the element after 10, compare each element with **37** until an element greater than **37** is found. Then swap **37** and that element. There are no more elements greater than **37**, so when we compare **37** with itself, we know that **37** has been placed in its final location of the sorted array.

Once the partition has been applied to the array, there are two unsorted subarrays. The subarray with values less than 37 contains 12, 2, 6, 4, 10 and 8. The subarray with values greater than 37 contains 89, 68 and 45. The sort continues by partitioning both subarrays in the same manner as the original array.

Write recursive function `quicksort` to sort a single-subscripted integer array. The function should receive as arguments an integer array, a starting subscript and an ending subscript. Function `partition` should be called by `quicksort` to perform the partitioning step.

ANS:

```

1  /* Exercise 7.24 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define MAX 10
7
8  /* function prototypes */
9  void quicksort( int *array, int first, int last );
10 int partition( int *array, int left, int right );
11 void swap( int *ptr1, int *ptr2 );
12
13 int main()
14 {
15     int loop; /* loop counter */
16     int arrayToBeSorted[ MAX ] = { 0 }; /* array to sort */

```

```
17
18     srand( time( NULL ) );
19
20     /* randomly generate content */
21     for ( loop = 0; loop < MAX; loop++ ) {
22         arrayToBeSorted[ loop ] = rand() % 1000;
23     } /* end for */
24
25     printf( "Initial array values are: \n" );
26
27     /* print out values of the array */
28     for ( loop = 0; loop < MAX; loop++ ) {
29         printf( "%4d", arrayToBeSorted[ loop ] );
30     } /* end for */
31
32     printf( "\n\n" );
33
34     /* if there is only one element */
35     if ( MAX == 1 ) {
36         printf( "Array is sorted: %d\n", arrayToBeSorted[ 0 ] );
37     } /* end if */
38     else { /* call quicksort */
39         quicksort( arrayToBeSorted, 0, MAX - 1 );
40         printf( "The sorted array values are:\n" );
41
42         /* display sorted array */
43         for ( loop = 0; loop < MAX; loop++ ) {
44             printf( "%4d", arrayToBeSorted[ loop ] );
45         } /* end for */
46
47         printf( "\n" );
48     } /* end else */
49
50     return 0; /* indicate successful termination */
51 } /* end main */
52
53
54 /* recursive function to sort array */
55 void quicksort( int array[], int first, int last )
56 {
57     int currentLocation; /* current location in array */
58
59     /* if array is sorted, return */
60     if ( first >= last ) {
61         return;
62     } /* end if */
63
64     currentLocation = partition( array, first, last ); /* place an element */
65     quicksort( array, first, currentLocation - 1 ); /* sort left side */
66     quicksort( array, currentLocation + 1, last ); /* sort right side */
67
68 } /* end function quicksort */
69
70 /* partition the array into multiple sections */
71 int partition( int array[], int left, int right )
72 {
73     int position = left; /* final location of first element */
74
75     /* infinite loop */
76     while ( 1 ) {
77
```

```

78     /* loop through the portion of the array */
79     while ( array[ position ] <= array[ right ] &&
80            position != right ) {
81         --right;
82     } /* end while */
83
84     /* if correct position is found */
85     if ( position == right ) {
86         return position ;
87     } /* end if */
88
89     /* swap positions */
90     if ( array[ position ] > array[ right ] ) {
91         swap( &array[ position ], &array[ right ] );
92         position = right;
93     } /* end if */
94
95     /* loop through the portion of the array */
96     while ( array[ left ] <= array[ position ] &&
97            left != position ) {
98         ++left;
99     } /* end while */
100
101     /* if correct position is found */
102     if ( position == left ) {
103         return position;
104     } /* end if */
105
106     /* swap positions */
107     if ( array[ left ] > array[ position ] ) {
108         swap( &array[ position ], &array[ left ] );
109         position = left;
110     } /* end if */
111
112 } /* end while */
113
114 } /* end function partition */
115
116 /* swap locations */
117 void swap( int *ptr1, int *ptr2 )
118 {
119     int temp; /* temporary holder */
120
121     temp = *ptr1;
122     *ptr1 = *ptr2;
123     *ptr2 = temp;
124 } /* end function swap */

```

Initial array values are:
276 980 550 654 811 764 571 469 12 161

The sorted array values are:
12 161 276 469 550 571 654 764 811 980

7.25 (*Maze Traversal*) The following grid is a double-subscripted array representation of a maze.

```

# # # # # # # # # # #
# . . . # . . . . . #
. . # . # . # # # . #
# # # . # . . . # . #
# . . . . # # # . # .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . # . . . #
# # # # # # # # # # #

```

The # symbols represent the walls of the maze, and the periods (.) represent squares in the possible paths through the maze.

There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming there is an exit). If there is not an exit, you will arrive at the starting location again. Place your right hand on the wall to your right and begin walking forward. Never remove your hand from the wall. If the maze turns to the right, you follow the wall to the right. As long as you do not remove your hand from the wall, eventually you will arrive at the exit of the maze. There may be a shorter path than the one you have taken, but you are guaranteed to get out of the maze.

Write recursive function `mazeTraverse` to walk through the maze. The function should receive as arguments a 12-by-12 character array representing the maze and the starting location of the maze. As `mazeTraverse` attempts to locate the exit from the maze, it should place the character X in each square in the path. The function should display the maze after each move so the user can watch as the maze is solved.

ANS:

```

1  /* Exercise 7.25 Solution */
2  /* This solution assumes that there is only one */
3  /* entrance and one exit for a given maze, and */
4  /* these are the only two zeroes on the borders.*/
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  #define DOWN 0 /* move down */
9  #define RIGHT 1 /* move right */
10 #define UP 2 /* move up */
11 #define LEFT 3 /* move left */
12
13 #define X_START 2 /* starting X and Y coordinate for maze */
14 #define Y_START 0
15
16 /* function prototypes */
17 void mazeTraverse( char maze[ 12 ][ 12 ], int xCoord, int yCoord,
18                 int direction );
19 void printMaze( const char maze[][ 12 ] );
20 int validMove( const char maze[][ 12 ], int r, int c );
21 int coordsAreEdge( int x, int y );
22
23 int main()
24 {
25
26     /* maze grid */
27     char maze[ 12 ][ 12 ] =
28     { { '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1' },
29       { '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1' },
30       { '0', '0', '1', '0', '1', '0', '1', '1', '1', '1', '0', '1' },
31       { '1', '1', '1', '0', '1', '0', '0', '0', '0', '1', '0', '1' },

```

```

32  { '1', '0', '0', '0', '0', '1', '1', '1', '0', '1', '0', '0'},
33  { '1', '1', '1', '1', '0', '1', '0', '1', '0', '1', '0', '1'},
34  { '1', '0', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1'},
35  { '1', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1'},
36  { '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '1'},
37  { '1', '1', '1', '1', '1', '1', '0', '1', '1', '1', '0', '1'},
38  { '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '1'},
39  { '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1' } };
40
41  mazeTraversal( maze, X_START, Y_START, RIGHT );
42
43  return 0; /* indicate successful termination */
44
45 } /* end main */
46
47 /* Assume that there is exactly 1 entrance and
48    exactly 1 exit to the maze. */
49 void mazeTraversal( char maze[ 12 ][ 12 ], int xCoord, int yCoord,
50                   int direction )
51 {
52     static int flag = 0; /* starting position flag */
53
54     maze[ xCoord ][ yCoord ] = 'X'; /* mark current point */
55     printMaze( maze );
56
57     /* if maze completed */
58     if ( coordsAreEdge( xCoord, yCoord ) && xCoord != X_START &&
59         yCoord != Y_START ) {
60         printf( "\nMaze successfully exited!\n\n" );
61         return;
62     } /* end if */
63     else if ( xCoord == X_START && yCoord == Y_START && flag == 1 ) {
64         printf( "\nArrived back at the starting location.\n\n" );
65         return;
66     } /* end else if */
67     else { /* make next move */
68         int move; /* next move */
69         int count; /* counter */
70
71         flag = 1;
72
73         /* loop 4 times and find first valid move */
74         for ( move = direction, count = 0; count < 4; ++count,
75             ++move, move %= 4 ) {
76
77             /* choose valid move */
78             switch( move ) {
79
80                 case DOWN: /* move down */
81
82                     /* if move is valid, call mazeTraversal */
83                     if ( validMove( maze, xCoord + 1, yCoord ) ) {
84                         mazeTraversal( maze, xCoord + 1, yCoord, LEFT );
85                         return;
86                     } /* end if */
87
88                     break; /* exit switch */
89
90                 case RIGHT: /* move right */
91

```

```

92         /* if move is valid, call mazeTraversal */
93         if ( validMove( maze, xCoord, yCoord + 1 ) ) {
94             mazeTraversal( maze, xCoord, yCoord + 1, DOWN );
95             return;
96         } /* end if */
97
98         break; /* exit switch */
99
100     case UP: /* move up */
101
102         /* if move is valid, call mazeTraversal */
103         if ( validMove( maze, xCoord - 1, yCoord ) ) {
104             mazeTraversal( maze, xCoord - 1, yCoord, RIGHT );
105             return;
106         } /* end if */
107
108         break; /* exit switch */
109
110     case LEFT: /* move left */
111
112         /* if move is valid, call mazeTraversal */
113         if ( validMove( maze, xCoord, yCoord - 1 ) ) { /* move left */
114             mazeTraversal( maze, xCoord, yCoord - 1, UP );
115             return;
116         } /* end if */
117
118         break; /* exit switch */
119     } /* end switch */
120
121     } /* end for */
122
123     } /* end else */
124
125 } /* end function mazeTraversal */
126
127 /* validate move */
128 int validMove( const char maze[][ 12 ], int r, int c )
129 {
130     return ( r >= 0 && r <= 11 && c >= 0 && c <= 11 &&
131             maze[ r ][ c ] != '1' );
132 } /* end function validMove */
133
134
135 /* function to check coordinates */
136 int coordsAreEdge( int x, int y )
137 {
138
139     /* if coordinate is not valid */
140     if ( ( x == 0 || x == 11 ) && ( y >= 0 && y <= 11 ) ) {
141         return 1;
142     } /* end if */
143     else if ( ( y == 0 || y == 11 ) && ( x >= 0 && x <= 11 ) ) {
144         return 1;
145     } /* end else if */
146     else { /* coordinate is valid */
147         return 0;
148     } /* end else */
149
150 } /* end function coordsAreEdge */
151

```

```
152 /* print the current state of the maze */
153 void printMaze( const char maze[][ 12 ] )
154 {
155     int x; /* row counter */
156     int y; /* column counter */
157
158     /* iterate through the maze */
159     for ( x = 0; x < 12; x++ ) {
160
161         for ( y = 0; y < 12; y++ ) {
162             printf( "%c ", maze[ x ][ y ] );
163         } /* end for */
164
165         printf( "\n" );
166     } /* end for */
167
168     printf( "\nHit return to see next move" );
169     getchar();
170 } /* end function printMaze */
```

Hit return to see next move

```

1 1 1 1 1 1 1 1 1 1 1
1 X X X 1 X X X X X 1
X X 1 X 1 X 1 1 1 1 0 1
1 1 1 X 1 X X X X 1 0 1
1 X X X X 1 1 1 X 1 0 0
1 1 1 1 X 1 0 1 X 1 0 1
1 X X 1 X 1 0 1 X 1 0 1
1 1 X 1 X 1 0 1 X 1 0 1
1 X X X X X X X X 1 0 1
1 1 1 1 1 1 X 1 1 1 0 1
1 X X X X X X 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1

```

Hit return to see next move

```

1 1 1 1 1 1 1 1 1 1 1
1 X X X 1 X X X X X 1
X X 1 X 1 X 1 1 1 1 X 1
1 1 1 X 1 X X X X 1 0 1
1 X X X X 1 1 1 X 1 0 0
1 1 1 1 X 1 0 1 X 1 0 1
1 X X 1 X 1 0 1 X 1 0 1
1 1 X 1 X 1 0 1 X 1 0 1
1 X X X X X X X X 1 0 1
1 1 1 1 1 1 X 1 1 1 0 1
1 X X X X X X 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1

```

...

Hit return to see next move

```

1 1 1 1 1 1 1 1 1 1 1
1 X X X 1 X X X X X 1
X X 1 X 1 X 1 1 1 1 X 1
1 1 1 X 1 X X X X 1 X 1
1 X X X X 1 1 1 X 1 X X
1 1 1 1 X 1 0 1 X 1 X 1
1 X X 1 X 1 0 1 X 1 X 1
1 1 X 1 X 1 0 1 X 1 X 1
1 X X X X X X X X 1 X 1
1 1 1 1 1 1 X 1 1 1 X 1
1 X X X X X X 1 X X X 1
1 1 1 1 1 1 1 1 1 1 1

```

Hit return to see next move

Maze successfully exited!

7.26 (*Generating Mazes Randomly*) Write a function `mazeGenerator` that takes as an argument a double-subscripted 12-by-12 character array and randomly produces a maze. The function should also provide the starting and ending locations of the maze. Try your function `mazeTraverse` from Exercise 7.25 using several randomly generated mazes.

ANS:

```

1  /* Exercise 7.26 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define DOWN 0 /* move down */
7  #define RIGHT 1 /* move right */
8  #define UP 2 /* move up */
9  #define LEFT 3 /* move left */
10 #define POSSIBLE_ZEROS 100 /* maximum possible zeroes */
11
12 /* function prototypes */
13 void mazeTraverse( char maze[ 12 ][ 12 ], const int xCoord,
14                  const int yCoord, int row, int col, int direction );
15 void mazeGenerator( char maze[][ 12 ], int *xPtr, int *yPtr );
16 void printMaze( const char maze[][ 12 ] );
17 int validMove( const char maze[][ 12 ], int r, int c );
18 int coordsAreEdge( int x, int y );
19
20 int main()
21 {
22     char maze[ 12 ][ 12 ]; /* maze grid */
23     int loop; /* row counter */
24     int loop2; /* column counter */
25     int xStart; /* starting x coordinate */
26     int yStart; /* starting y coordinate */
27     int x; /* current x coordinate */
28     int y; /* current y coordinate */
29
30     /* initialize maze grid to 1's */
31     for ( loop = 0; loop < 12; loop++ ) {
32
33         for ( loop2 = 0; loop2 < 12; loop2++ ) {
34             maze[ loop ][ loop2 ] = '1';
35         } /* end for */
36
37     } /* end for */
38
39     /* generate the maze */
40     mazeGenerator( maze, &xStart, &yStart );
41
42     x = xStart; /* starting row */
43     y = yStart; /* starting col */
44
45     mazeTraverse( maze, xStart, yStart, x, y, RIGHT );
46
47     return 0; /* indicate successful termination */
48
49 } /* end main */
50
51 /* Assume that there is exactly 1 entrance and
52 exactly 1 exit to the maze. */
53 void mazeTraverse( char maze[ 12 ][ 12 ], const int xCoord,
54                  const int yCoord, int row, int col, int direction )
55 {
56     static int flag = 0; /* starting position flag */

```

```

57
58 maze[ row ][ col ] = 'X'; /* insert X at current location */
59 printMaze( maze );
60
61 /* if maze completed */
62 if ( coordsAreEdge( row, col ) && row != xCoord && col != yCoord ) {
63     printf( "\nMaze successfully exited!\n\n" );
64     return;
65 } /* end if */
66 else if ( row == xCoord && col == yCoord && flag == 1 ) {
67     printf( "\nArrived back at the starting location.\n\n" );
68     return;
69 } /* end else if */
70 else { /* make next move */
71     int move; /* next move */
72     int count; /* counter */
73
74     flag = 1;
75
76     /* loop 4 times and find first valid move */
77     for ( move = direction, count = 0; count < 4; ++count,
78         ++move, move %= 4 ) {
79
80         /* choose valid move */
81         switch( move ) {
82
83             case DOWN: /* move down */
84
85                 /* if move is valid, call mazeTraversal */
86                 if ( validMove( maze, row + 1, col ) ) {
87                     mazeTraversal( maze, xCoord, yCoord, row + 1,
88                         col, LEFT );
89                     return;
90                 } /* end if */
91
92                 break; /* exit switch */
93
94             case RIGHT: /* move right */
95
96                 /* if move is valid, call mazeTraversal */
97                 if ( validMove( maze, row, col + 1 ) ) {
98                     mazeTraversal( maze, xCoord, yCoord, row,
99                         col + 1, DOWN );
100                    return;
101                } /* end if */
102
103                break; /* exit switch */
104
105             case UP: /* move up */
106
107                 /* if move is valid, call mazeTraversal */
108                 if ( validMove( maze, row - 1, col ) ) {
109                     mazeTraversal( maze, xCoord, yCoord, row - 1,
110                         col, RIGHT );
111                     return;
112                 } /* end if */
113
114                 break; /* exit switch */
115
116             case LEFT: /* move left */
117

```

```

118         /* if move is valid, call mazeTraversal */
119         if ( validMove( maze, row, col - 1 ) ) {
120             mazeTraversal( maze, xCoord, yCoord, row,
121                 col - 1, UP );
122             return;
123         } /* end if */
124
125         break; /* exit switch */
126     } /* end switch */
127
128     } /* end for */
129
130 } /* end else */
131
132 } /* end function mazeTraversal */
133
134 /* validate move */
135 int validMove( const char maze[][ 12 ], int r, int c )
136 {
137     return ( r >= 0 && r <= 11 && c >= 0 && c <= 11 &&
138         maze[ r ][ c ] != '1' );
139 }
140 /* end function validMove */
141
142 /* check boundaries of coordinates */
143 int coordsAreEdge( int x, int y )
144 {
145
146     /* if coordinates not valid */
147     if ( ( x == 0 || x == 11 ) && ( y >= 0 && y <= 11 ) ) {
148         return 1;
149     } /* end if */
150     else if ( ( y == 0 || y == 11 ) && ( x >= 0 && x <= 11 ) ) {
151         return 1;
152     } /* end else if */
153     else { /* coordinates valid */
154         return 0;
155     } /* end else */
156 }
157 /* end function coordsAreEdge */
158
159 /* print the maze */
160 void printMaze( const char maze[][ 12 ] )
161 {
162     int x; /* row counter */
163     int y; /* column counter */
164
165     /* loop through maze grid */
166     for ( x = 0; x < 12; x++ ) {
167
168         for ( y = 0; y < 12; y++ ) {
169             printf( "%c ", maze[ x ][ y ] );
170         } /* end for */
171
172         printf( "\n" );
173     } /* end for */
174
175     printf( "\nHit return to see next move" );
176     getchar();
177 } /* end function printMaze */
178

```



```
179 /* random maze generator */
180 void mazeGenerator( char maze[][ 12 ], int *xPtr, int *yPtr )
181 {
182     int a;      /* random number */
183     int x;      /* random number */
184     int y;      /* random number */
185     int entry; /* random entry */
186     int exit;  /* random exit */
187     int loop;  /* loop counter */
188
189     srand( time( NULL ) );
190
191     /* generate random entry and exit positions */
192     do {
193         entry = rand() % 4;
194         exit = rand() % 4;
195     } while ( entry == exit ); /* end do...while */
196
197     /* Determine entry position while avoiding corners */
198     if ( entry == 0 ) {
199         *xPtr = 1 + rand() % 10;
200         *yPtr = 0;
201         maze[ *xPtr ][ 0 ] = '0';
202     } /* end if */
203     else if ( entry == 1 ) {
204         *xPtr = 0;
205         *yPtr = 1 + rand() % 10;
206         maze[ 0 ][ *yPtr ] = '0';
207     } /* end else if */
208     else if ( entry == 2 ) {
209         *xPtr = 1 + rand() % 10;
210         *yPtr = 11;
211         maze[ *xPtr ][ 11 ] = '0';
212     } /* end else if */
213     else {
214         *xPtr = 11;
215         *yPtr = 1 + rand() % 10;
216         maze[ 11 ][ *yPtr ] = '0';
217     } /* end else */
218
219     /* Determine exit location */
220     if ( exit == 0 ) {
221         a = 1 + rand() % 10;
222         maze[ a ][ 0 ] = '0';
223     } /* end if */
224     else if ( exit == 1 ) {
225         a = 1 + rand() % 10;
226         maze[ 0 ][ a ] = '0';
227     } /* end else if */
228     else if ( exit == 2 ) {
229         a = 1 + rand() % 10;
230         maze[ a ][ 11 ] = '0';
231     } /* end else if */
232     else {
233         a = 1 + rand() % 10;
234         maze[ 11 ][ a ] = '0';
235     } /* end else */
236
237     /* randomly add zeroes to maze grid */
238     for ( loop = 1; loop < POSSIBLE_ZEROS; loop++ ) {
239         x = 1 + rand() % 10;
```

```

240     y = 1 + rand() % 10;
241     maze[ x ][ y ] = '0';
242 } /* end for */
243
244 } /* end function mazeGenerator */

```

Hit return to see next move

```

1 1 1 1 0 1 1 1 1 1 1
1 0 1 1 0 1 0 1 X X X 1
1 1 1 0 0 0 1 1 1 1 X 1
X X 1 0 0 1 X X X X X 1
1 X X X 1 X X 1 X X 1 1
1 X 1 X X X 1 0 1 X 1 1
1 1 X 1 X X X 1 X X 1 1
1 1 X X X X 1 1 X 1 0 1
1 X X 0 0 X 1 X X X 1 1
1 1 X 0 1 X 1 X X 1 0 1
1 1 X X X X X 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1

```

Hit return to see next move

```

1 1 1 1 0 1 1 1 1 1 1
1 0 1 1 0 1 0 1 X X X 1
1 1 1 0 0 0 1 1 1 1 X 1
X X 1 0 0 1 X X X X X 1
1 X X X 1 X X 1 X X 1 1
1 X 1 X X X 1 0 1 X 1 1
1 1 X 1 X X X 1 X X 1 1
1 1 X X X X 1 1 X 1 0 1
1 X X 0 0 X 1 X X X 1 1
1 1 X 0 1 X 1 X X 1 0 1
1 1 X X X X X 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1

```

...

Hit return to see next move

```

1 1 1 1 X 1 1 1 1 1 1 1
1 0 1 1 X 1 0 1 X X X 1
1 1 1 0 X X 1 1 1 1 X 1
X X 1 X X 1 X X X X X 1
1 X X X 1 X X 1 X X 1 1
1 X 1 X X X 1 0 1 X 1 1
1 1 X 1 X X X 1 X X 1 1
1 1 X X X X 1 1 X 1 0 1
1 X X 0 0 X 1 X X X 1 1
1 1 X 0 1 X 1 X X 1 0 1
1 1 X X X X X 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1

```

Hit return to see next move

Maze successfully exited!

7.27 (*Mazes of Any Size*) Generalize functions `mazeTraverse` and `mazeGenerator` of Exercise 7.25 and Exercise 7.26 to process mazes of any width and height.

ANS:

```

1  /* Exercise 7.27 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define ROW 10 /* height */
7  #define COL 10 /* width */
8  #define DOWN 0 /* move down */
9  #define RIGHT 1 /* move right */
10 #define UP 2 /* move up */
11 #define LEFT 3 /* move left */
12
13 /* function prototypes */
14 void mazeTraverse( char maze[ ROW ][ COL ], const int xCoord,
15                  const int yCoord, int row, int col, int direction );
16 void mazeGenerator( char maze[][ COL ], int *xPtr, int *yPtr );
17 void printMaze( const char maze[][ COL ] );
18 int validMove( const char maze[][ COL ], int r, int c );
19 int coordsAreEdge( int x, int y );
20
21 int main()
22 {
23     char maze[ ROW ][ COL ]; /* maze grid */
24     int loop; /* row counter */
25     int loop2; /* column counter */
26     int xStart; /* starting x coordinate */
27     int yStart; /* starting y coordinate */
28     int x; /* current x coordinate */
29     int y; /* current y coordinate */
30
31     /* initialize maze grid to 1's */
32     for ( loop = 0; loop < ROW; loop++ ) {
33
34         for ( loop2 = 0; loop2 < COL; loop2++ ) {
35             maze[ loop ][ loop2 ] = '1';
36         } /* end for */
37     } /* end for */
38
39     /* generate the maze */
40     mazeGenerator( maze, &xStart, &yStart );
41
42     x = xStart; /* starting row */
43     y = yStart; /* starting col */
44
45     mazeTraverse( maze, xStart, yStart, x, y, RIGHT );
46
47     return 0; /* indicate successful termination */
48 } /* end main */
49
50
51
52 /* Assume that there is exactly 1 entrance and
53 exactly 1 exit to the maze. */
54 void mazeTraverse( char maze[ ROW ][ COL ], const int xCoord,
55                  const int yCoord, int row, int col, int direction )
56 {
57     static int flag = 0; /* starting position flag */

```

```

58
59 maze[ row ][ col ] = 'X'; /* insert X at current location */
60 printMaze( maze );
61
62 /* if maze completed */
63 if ( coordsAreEdge( row, col ) && row != xCoord && col != yCoord ) {
64     printf( "\nMaze successfully exited!\n\n" );
65     return;
66 } /* end if */
67 else if ( row == xCoord && col == yCoord && flag == 1 ) {
68     printf( "\nArrived back at the starting location.\n\n" );
69     return;
70 } /* end else if */
71 else { /* make next move */
72     int move; /* next move */
73     int count; /* counter */
74
75     flag = 1;
76
77     /* loop 4 times and find first valid move */
78     for ( move = direction, count = 0; count < 4; ++count,
79         ++move, move %= 4 ) {
80
81         /* choose valid move */
82         switch( move ) {
83
84             case DOWN: /* move down */
85
86                 /* if move is valid, call mazeTraversal */
87                 if ( validMove( maze, row + 1, col ) ) {
88                     mazeTraversal( maze, xCoord, yCoord, row + 1,
89                         col, LEFT );
90                     return;
91                 } /* end if */
92
93                 break; /* exit switch */
94
95             case RIGHT: /* move right */
96
97                 /* if move is valid, call mazeTraversal */
98                 if ( validMove( maze, row, col + 1 ) ) {
99                     mazeTraversal( maze, xCoord, yCoord, row,
100                         col + 1, DOWN );
101                     return;
102                 } /* end if */
103
104                 break; /* exit switch */
105
106             case UP: /* move up */
107
108                 /* if move is valid, call mazeTraversal */
109                 if ( validMove( maze, row - 1, col ) ) {
110                     mazeTraversal( maze, xCoord, yCoord, row - 1,
111                         col, RIGHT );
112                     return;
113                 } /* end if */
114
115                 break; /* exit switch */
116
117             case LEFT: /* move left */
118

```

```

119         /* if move is valid, call mazeTraversal */
120         if ( validMove( maze, row, col - 1 ) ) {
121             mazeTraversal( maze, xCoord, yCoord, row,
122                 col - 1, UP );
123             return;
124         } /* end if */
125
126         break; /* exit switch */
127     } /* end switch */
128
129     } /* end for */
130
131     } /* end else */
132 } /* end function mazeTraversal */
133
134 /* validate move */
135 int validMove( const char maze[][ COL ], int r, int c )
136 {
137     return ( r >= 0 && r <= ROW - 1 && c >= 0 && c <= COL - 1 &&
138         maze[ r ][ c ] != '1' ); /* a valid move */
139 } /* end function validMove */
140
141 /* check boundaries of coordinates */
142 int coordsAreEdge( int x, int y )
143 {
144     /* if coordinates not valid */
145     if ( ( x == 0 || x == ROW - 1 ) && ( y >= 0 && y <= COL - 1 ) ) {
146         return 1;
147     } /* end if */
148     else if ( ( y == 0 || y == COL - 1 ) && ( x >= 0 &&
149         x <= ROW - 1 ) ) {
150         return 1;
151     } /* end else if */
152     else { /* coordinates valid */
153         return 0;
154     } /* end else */
155 } /* end function coordsAreEdge */
156
157 /* print the maze */
158 void printMaze( const char maze[][ COL ] )
159 {
160     int x; /* row counter */
161     int y; /* column counter */
162
163     /* loop through maze grid */
164     for ( x = 0; x < ROW; x++ ) {
165         for ( y = 0; y < COL; y++ ) {
166             printf( "%c ", maze[ x ][ y ] );
167         } /* end for */
168
169         printf( "\n" );
170     } /* end for */
171
172     printf( "\nHit return to see next move" );
173     getchar();
174 } /* end function printMaze */

```

```

180
181 /* random maze generator */
182 void mazeGenerator( char maze[][ COL ], int *xPtr, int *yPtr )
183 {
184     int a;      /* random number */
185     int x;      /* random number */
186     int y;      /* random number */
187     int entry; /* random entry */
188     int exit;  /* random exit */
189     int loop;  /* loop counter */
190
191     srand( time( NULL ) );
192
193     /* generate random entry and exit positions */
194     do {
195         entry = rand() % 4;
196         exit = rand() % 4;
197     } while ( entry == exit ); /* end do...while */
198
199     /* Determine entry position while avoiding corners */
200     if ( entry == 0 ) {
201         *xPtr = 1 + rand() % ( ROW - 2 );
202         *yPtr = 0;
203         maze[ *xPtr ][ *yPtr ] = '0';
204     } /* end if */
205     else if ( entry == 1 ) {
206         *xPtr = 0;
207         *yPtr = 1 + rand() % ( COL - 2 );
208         maze[ *xPtr ][ *yPtr ] = '0';
209     } /* end else if */
210     else if ( entry == 2 ) {
211         *xPtr = 1 + rand() % ( ROW - 2 );
212         *yPtr = COL - 1;
213         maze[ *xPtr ][ *yPtr ] = '0';
214     } /* end else if */
215     else {
216         *xPtr = ROW - 1;
217         *yPtr = 1 + rand() % ( COL - 2 );
218         maze[ *xPtr ][ *yPtr ] = '0';
219     } /* end else */
220
221     /* Determine exit location */
222     if ( exit == 0 ) {
223         a = 1 + rand() % ( ROW - 2 );
224         maze[ a ][ 0 ] = '0';
225     } /* end if */
226     else if ( exit == 1 ) {
227         a = 1 + rand() % ( COL - 2 );
228         maze[ 0 ][ a ] = '0';
229     } /* end else if */
230     else if ( exit == 2 ) {
231         a = 1 + rand() % ( ROW - 2 );
232         maze[ a ][ COL - 1 ] = '0';
233     } /* end else if */
234     else {
235         a = 1 + rand() % ( COL - 2 );
236         maze[ ROW - 1 ][ a ] = '0';
237     } /* end else */
238
239     /* randomly add zeroes to maze grid */
240     for ( loop = 1; loop < ( ROW - 2 ) * ( COL - 2 ); loop++ ) {

```

```

241     x = 1 + rand() % ( ROW - 2 );
242     y = 1 + rand() % ( COL - 2 );
243     maze[ x ][ y ] = '0';
244 } /* end for */
245
246 } /* end function mazeGenerator */

```

```

1 1 X 1 1 1 1 1 1 1
1 0 0 1 1 0 1 0 0 1
0 0 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 0 1 1
1 1 1 0 0 0 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 1 0 1
1 0 0 0 1 1 0 0 0 1
1 0 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1

```

Hit return to see next move

```

1 1 X 1 1 1 1 1 1 1
1 0 X 1 1 0 1 0 0 1
0 0 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 0 1 1
1 1 1 0 0 0 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 1 0 1
1 0 0 0 1 1 0 0 0 1
1 0 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1

```

...

Hit return to see next move

```

1 1 X 1 1 1 1 1 1 1
1 X X 1 1 0 1 0 0 1
X X 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 0 1 1
1 1 1 0 0 0 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 1 0 1
1 0 0 0 1 1 0 0 0 1
1 0 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1

```

Hit return to see next move

Maze successfully exited!

7.28 (*Arrays of Pointers to Functions*) Rewrite the program of Fig. 6.22 to use a menu driven interface. The program should offer the user four options as follows:

```

Enter a choice:
0 Print the array of grades
1 Find the minimum grade
2 Find the maximum grade
3 Print the average on all tests for each student
4 End program

```

One restriction on using arrays of pointers to functions is that all the pointers must have the same type. The pointers must be to functions of the same return type that receive arguments of the same type. For this reason, the functions in Fig. 6.22 must be modified so that they each return the same type and take the same parameters. Modify functions `minimum` and `maximum` to print the minimum or maximum value and return nothing. For option 3, modify function `average` of Fig. 6.22 to output the average for each student (not a specific student). Function `average` should return nothing and take the same parameters as `printArray`, `minimum` and `maximum`. Store the pointers to the four functions in array `processGrades` and use the choice made by the user as the subscript into the array for calling each function.

ANS:

```

1  /* Exercise 7.28 Solution */
2  #include <stdio.h>
3  #define STUDENTS 3
4  #define EXAMS 4
5
6  /* function prototypes */
7  void minimum( int grades[][ EXAMS ], int pupils, int tests );
8  void maximum( int grades[][ EXAMS ], int pupils, int tests );
9  void average( int grades[][ EXAMS ], int pupils, int tests );
10 void printArray( int grades[][ EXAMS ], int pupils, int tests );
11 void printMenu( void );
12
13 int main()
14 {
15
16     /* pointer to a function that takes as parameters a
17        two-dimensional array and two integer values */
18     void ( *processGrades[ 4 ] )( int [][] EXAMS ], int, int )
19         = { printArray, minimum, maximum, average };
20
21     int choice = 0; /* menu choice */
22
23     /* array of student grades */
24     int studentGrades[ STUDENTS ][ EXAMS ] = { { 77, 68, 86, 73 },
25                                                { 96, 87, 89, 78 },
26                                                { 70, 90, 86, 81 } };
27
28     /* loop while user does not choose option 4 */
29     while ( choice != 4 ) {
30
31         /* display menu and read user's choice */
32         do {
33             printMenu();
34             scanf( "%d", &choice );
35         } while ( choice < 0 || choice > 4 ); /* end do...while */
36
37         /* pass choice into the array */
38         if ( choice != 4 ) {
39             ( *processGrades[ choice ] )( studentGrades, STUDENTS, EXAMS );
40         } /* end if */

```



```
41     else {
42         printf( "Program Ended.\n" );
43     } /* end else */
44
45 } /* end while */
46
47 return 0; /* indicate successful termination */
48
49 } /* end main */
50
51 /* search for the minimum value */
52 void minimum( int grades[][ EXAMS ], int pupils, int tests )
53 {
54     int i;           /* loop counter */
55     int j;           /* loop counter */
56     int lowGrade = 100; /* set lowGrade to highest possible score */
57
58     /* loop through rows */
59     for ( i = 0; i <= pupils - 1; i++ ) {
60
61         /* loop through columns */
62         for ( j = 0; j <= tests - 1; j++ ) {
63
64             /* if current grade is lower than lowGrade */
65             if ( grades[ i ][ j ] < lowGrade ) {
66                 lowGrade = grades[ i ][ j ];
67             } /* end if */
68
69         } /* end for */
70
71     } /* end for */
72
73     printf( "\n\tThe lowest grade is %d\n", lowGrade );
74 } /* end function minimum */
75
76 /* search for maximum value */
77 void maximum( int grades[][ EXAMS ], int pupils, int tests )
78 {
79     int i;           /* loop counter */
80     int j;           /* loop counter */
81     int highGrade = 0; /* set highGrade to lowest possible score */
82
83     /* loop through rows */
84     for ( i = 0; i <= pupils - 1; i++ ) {
85
86         /* loop through columns */
87         for ( j = 0; j <= tests - 1; j++ ) {
88
89             /* if current grade is higher than highGrade */
90             if ( grades[ i ][ j ] > highGrade ) {
91                 highGrade = grades[ i ][ j ];
92             } /* end if */
93
94         } /* end for */
95
96     } /* end for */
97
98     printf( "\n\tThe highest grade is %d\n", highGrade );
99 } /* end function maximum */
100
```

```

101 /* calculate average */
102 void average( int grades[][ EXAMS ], int pupils, int tests )
103 {
104     int i;      /* loop counter */
105     int j;      /* loop counter */
106     int total; /* sum of all grades */
107
108     printf( "\n" );
109
110     /* loop through rows */
111     for ( i = 0; i <= pupils - 1; i++ ) {
112         total = 0;
113
114         /* loop through columns */
115         for ( j = 0; j <= tests - 1; j++ ) {
116             total += grades[ i ][ j ];
117         } /* end for */
118
119         printf( "\tThe average for student %d is %.1f\n",
120             i + 1, ( double ) total / tests );
121     } /* end for */
122 } /* end function average */
123
124
125 /* print the contents of the array */
126 void printArray( int grades[][ EXAMS ], int pupils, int tests )
127 {
128     int i; /* loop counter */
129     int j; /* loop counter */
130
131     printf( "\n\t          [ 0 ] [ 1 ] [ 2 ] [ 3 ]" );
132
133     /* loop through rows */
134     for ( i = 0; i <= pupils - 1; i++ ) {
135         printf( "\n\tstudentGrades[ %d ] ", i );
136
137         /* loop through columns */
138         for ( j = 0; j <= tests - 1; j++ ) {
139             printf( "%-7d", grades[ i ][ j ] );
140         } /* end for */
141     } /* end for */
142
143     printf( "\n" );
144 } /* end function printArray */
145
146
147 /* display the menu */
148 void printMenu( void )
149 {
150     printf( "\n\tEnter a choice:\n"
151         "\t 0 Print the array of grades\n"
152         "\t 1 Find the minimum grade\n"
153         "\t 2 Find the maximum grade\n"
154         "\t 3 Print the average on all"
155         " tests for each student\n"
156         "\t 4 End program\n"
157         "\t? " );
158 } /* end function printMenu */

```

```
Enter a choice:
0 Print the array of grades
1 Find the minimum grade
2 Find the maximum grade
3 Print the average on all tests for each student
4 End program
? 0
```

```
          [ 0 ] [ 1 ] [ 2 ] [ 3 ]
studentGrades[ 0 ] 77   68   86   73
studentGrades[ 1 ] 96   87   89   78
studentGrades[ 2 ] 70   90   86   81
```

```
Enter a choice:
0 Print the array of grades
1 Find the minimum grade
2 Find the maximum grade
3 Print the average on all tests for each student
4 End program
? 1
```

The lowest grade is 68

```
Enter a choice:
0 Print the array of grades
1 Find the minimum grade
2 Find the maximum grade
3 Print the average on all tests for each student
4 End program
? 2
```

The highest grade is 96

```
Enter a choice:
0 Print the array of grades
1 Find the minimum grade
2 Find the maximum grade
3 Print the average on all tests for each student
4 End program
? 3
```

The average for student 1 is 76.0
The average for student 2 is 87.5
The average for student 3 is 81.8

```
Enter a choice:
0 Print the array of grades
1 Find the minimum grade
2 Find the maximum grade
3 Print the average on all tests for each student
4 End program
? 4
```

Program Ended.

7.29 (*Modifications to the Simpletron Simulator*) In Exercise 7.19, you wrote a software simulation of a computer that executes programs written in Simpletron Machine Language (SML). In this exercise, we propose several modifications and enhancements to the Simpletron Simulator. In Exercises 12.26 and 12.27, we propose building a compiler that converts programs written in a high-level programming language (a variation of BASIC) to Simpletron Machine Language. Some of the following modifications and enhancements may be required to execute the programs produced by the compiler.

- a) Extend the Simpletron Simulator's memory to contain 1000 memory locations to enable the Simpletron to handle larger programs.
- b) Allow the simulator to perform remainder calculations. This requires an additional Simpletron Machine Language instruction.
- c) Allow the simulator to perform exponentiation calculations. This requires an additional Simpletron Machine Language instruction.
- d) Modify the simulator to use hexadecimal values rather than integer values to represent Simpletron Machine Language instructions.
- e) Modify the simulator to allow output of a newline. This requires an additional Simpletron Machine Language instruction.
- f) Modify the simulator to process floating-point values in addition to integer values.
- g) Modify the simulator to handle string input. [*Hint*: Each Simpletron word can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the ASCII decimal equivalent of a character. Add a machine language instruction that will input a string and store the string beginning at a specific Simpletron memory location. The first half of the word at that location will be a count of the number of characters in the string (i.e., the length of the string). Each succeeding half word contains one ASCII character expressed as two decimal digits. The machine language instruction converts each character into its ASCII equivalent and assigns it to a half word.]
- h) Modify the simulator to handle output of strings stored in the format of part (g). [*Hint*: Add a machine language instruction that prints a string beginning at a specified Simpletron memory location. The first half of the word at that location is the length of the string in characters. Each succeeding half word contains one ASCII character expressed as two decimal digits. The machine language instruction checks the length and prints the string by translating each two-digit number into its equivalent character.]

7.30 What does this program do?

```
1  /* ex07_30.c */
2  /* What does this program do? */
3  #include <stdio.h>
4
5  int mystery3( const char *s1, const char *s2 ); /* prototype */
6
7  int main()
8  {
9      char string1[ 80 ]; /* create char array */
10     char string2[ 80 ]; /* create char array */
11
12     printf( "Enter two strings: " );
13     scanf( "%s%s", string1 , string2 );
14
15     printf( "The result is %d\n", mystery3( string1, string2 ) );
16
17     return 0; /* indicates successful termination */
18 } /* end main */
19
20
21 int mystery3( const char *s1, const char *s2 )
22 {
23     for ( ; *s1 != '\0' && *s2 != '\0'; s1++, s2++ ) {
24
25         if ( *s1 != *s2 ) {
26             return 0;
27         } /* end if */
28
29     } /* end for */
30
31     return 1;
32 } /* end function mystery3 */
```

ANS: The Program compares two strings, element by element, for equality.

```
Enter two strings: string1 string2
The result is 0
```

```
Enter two strings: string2 string2
The result is 1
```

8

C Characters and Strings: Solutions

SOLUTIONS

8.5 Write a program that inputs a character from the keyboard and tests the character with each of the functions in the character handling library. The program should print the value returned by each function.

ANS:

```
1  /* Exercise 8.5 Solution */
2  #include <stdio.h>
3  #include <ctype.h>
4
5  int main()
6  {
7      int c; /* character input by user */
8
9      printf( "Enter a character: " );
10     c = getchar();
11
12     /* test each function of the character handling library */
13     printf( "isdigit( \'%c\' ) = %d\n", c, isdigit( c ) );
14     printf( "isalpha( \'%c\' ) = %d\n", c, isalpha( c ) );
15     printf( "isalnum( \'%c\' ) = %d\n", c, isalnum( c ) );
16     printf( "isxdigit( \'%c\' ) = %d\n", c, isxdigit( c ) );
17     printf( "islower( \'%c\' ) = %d\n", c, islower( c ) );
18     printf( "isupper( \'%c\' ) = %d\n", c, isupper( c ) );
19     printf( "tolower( \'%c\' ) = %d\n", c, tolower( c ) );
20     printf( "toupper( \'%c\' ) = %d\n", c, toupper( c ) );
21     printf( "isspace( \'%c\' ) = %d\n", c, isspace( c ) );
22     printf( "iscntrl( \'%c\' ) = %d\n", c, iscntrl( c ) );
23     printf( "ispunct( \'%c\' ) = %d\n", c, ispunct( c ) );
24     printf( "isprint( \'%c\' ) = %d\n", c, isprint( c ) );
25     printf( "isgraph( \'%c\' ) = %d\n", c, isgraph( c ) );
26
27     return 0; /* indicate successful termination */
28
29 } /* end main */
```

```

Enter a character: h
isdigit( 'h' ) = 0
isalpha( 'h' ) = 2
isalnum( 'h' ) = 2
isxdigit( 'h' ) = 0
islower( 'h' ) = 2
isupper( 'h' ) = 0
tolower( 'h' ) = 104
toupper( 'h' ) = 72
isspace( 'h' ) = 0
iscntrl( 'h' ) = 0
ispunct( 'h' ) = 0
isprint( 'h' ) = 2
isgraph( 'h' ) = 2

```

8.6 Write a program that inputs a line of text with function `gets` into char array `s[100]`. Output the line in uppercase letters and in lowercase letters.

ANS:

```

1  /* Exercise 8.6 Solution */
2  #include <stdio.h>
3  #include <ctype.h>
4
5  int main()
6  {
7      char s[ 100 ]; /* define character array of size 100 */
8      int i; /* loop counter */
9
10     /* use gets to get text from user */
11     printf( "Enter a line of text:\n" );
12     gets( s );
13     printf( "\nThe line in uppercase is:\n" );
14
15     /* convert each character to uppercase and output */
16     for ( i = 0; s[ i ] != '\0'; i++ ) {
17         printf( "%c", toupper( s[ i ] ) );
18     } /* end for */
19
20     printf( "\n\nThe line in lowercase is:\n" );
21
22     /* convert each character to lowercase and output */
23     for ( i = 0; s[ i ] != '\0'; i++ ) {
24         printf( "%c", tolower( s[ i ] ) );
25     } /* end for */
26
27     return 0; /* indicate successful termination */
28
29 } /* end main */

```

```

Enter a line of text:
A line with UPPER- and lowercase LeTters

The line in uppercase is:
A LINE WITH UPPER- AND LOWERCASE LETTERS

The line in lowercase is:
a line with upper- and lowercase letters

```

8.7 Write a program that inputs four strings that represent integers, converts the strings to integers, sums the values and prints the total of the four values.

ANS:

```
1  /* Exercise 8.7 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      char stringValue[ 6 ]; /* integer string input by user */
8      int sum = 0;           /* result of four integers */
9      int i;                 /* loop counter */
10
11     /* loop 4 times */
12     for ( i = 1; i <= 4; i++ ) {
13         printf( "Enter an integer string: " );
14         scanf( "%s", stringValue );
15
16         /* atoi converts stringValue to integer */
17         sum += atoi( stringValue );
18     } /* end for */
19
20     printf( "\nThe total of the values is %d\n", sum );
21
22     return 0; /* indicate successful termination */
23
24 } /* end main */
```

```
Enter an integer string: 43
Enter an integer string: 77
Enter an integer string: 120
Enter an integer string: 9999

The total of the values is 10239
```

8.8 Write a program that inputs four strings that represent floating-point values, converts the strings to double values, sums the values and prints the total of the four values.

ANS:

```
1  /* Exercise 8.8 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      char stringValue[ 15 ]; /* string input by user */
8      double sum = 0.0;       /* sum of all four values */
9      int i;                 /* loop counter */
10
11     /* loop 4 times */
12     for ( i = 1; i <= 4; i++ ) {
13         printf( "Enter a doubleing point string: " );
14         gets( stringValue );
15
16         /* atof converts stringValue to a floating-point value */
17         sum += atof( stringValue );
18     } /* end for */
19
20     printf( "\nThe total of the values is %f\n", sum );
21
22     return 0; /* indicate successful termination */
23
24 } /* end main */
```



```
Enter a doubleing point string: 1.2
Enter a doubleing point string: 2.3
Enter a doubleing point string: 3.4
Enter a doubleing point string: 4.5
```

```
The total of the values is 11.400000
```

8.9 Write a program that uses function `strcmp` to compare two strings input by the user. The program should state whether the first string is less than, equal to or greater than the second string.

ANS:

```
1  /* Exercise 8.9 Solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      char string1[ 20 ]; /* first string input by user */
8      char string2[ 20 ]; /* second string input by user */
9      int result;        /* result of comparing two strings */
10
11     printf( "Enter two strings: " );
12     scanf( "%s%s", string1, string2 ); /* read two strings */
13
14     result = strcmp( string1, string2 );
15
16     /* display appropriate message for result */
17     if ( result > 0 ) {
18         printf( "\"%s\" is greater than \"%s\"\n", string1, string2 );
19     } /* end if */
20     else if ( result == 0 ) {
21         printf( "\"%s\" is equal to \"%s\"\n", string1, string2 );
22     } /* end else if */
23     else {
24         printf( "\"%s\" is less than \"%s\"\n", string1, string2 );
25     } /* end else */
26
27     return 0; /* indicate successful termination */
28
29 } /* end main */
```

```
Enter two strings: Greg Dave
"Greg" is greater than "Dave"
```

```
Enter two strings: Bill Bill
"Bill" is equal to "Bill"
```

```
Enter two strings: Pete Tim
"Pete" is less than "Tim"
```

8.10 Write a program that uses function `strncmp` to compare two strings input by the user. The program should input the number of characters to be compared. The program should state whether the first string is less than, equal to or greater than the second string.

ANS:

```

1  /* Exercise 8.10 Solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      char string1[ 20 ]; /* first string input by user */
8      char string2[ 20 ]; /* second string input by user */
9      int result;        /* result of using strncmp */
10     int compareCount;  /* how many characters to be compared */
11
12     /* get two strings from user */
13     printf( "Enter two strings: " );
14     scanf( "%s%s", string1, string2 );
15
16     /* get number of characters to compare */
17     printf( "How many characters should be compared: " );
18     scanf( "%d", &compareCount );
19
20     result = strncmp( string1, string2, compareCount );
21
22     /* display appropriate message for result */
23     if ( result > 0 ) {
24         printf( "\"%s\" is greater than \"%s\" up to %d characters\n",
25               string1, string2, compareCount );
26     } /* end if */
27     else if ( result == 0 ) {
28         printf( "\"%s\" is equal to \"%s\" up to %d characters\n",
29               string1, string2, compareCount );
30     } /* end else if */
31     else {
32         printf( "\"%s\" is less than \"%s\" up to %d characters\n",
33               string1, string2, compareCount );
34     } /* end else */
35
36     return 0; /* indicate successful termination */
37
38 } /* end main */

```

```

Enter two strings: ABCDEFG ABCDEFH
How many characters should be compared: 6
"ABCDEFG" is less than "ABCDEFH" up to 6 characters

```

```

Enter two strings: ABCDEFG ABCDEFH
How many characters should be compared: 7
"ABCDEFG" is less than "ABCDEFH" up to 7 characters

```

```

Enter two strings: ABCEFG ABCDFG
How many characters should be compared: 4
"ABCEFG" is greater than "ABCDFG" up to 4 characters

```

8.11 Write a program that uses random number generation to create sentences. The program should use four arrays of pointers to char called `article`, `noun`, `verb` and `preposition`. The program should create a sentence by selecting a word at random from each array in the following order: `article`, `noun`, `verb`, `preposition`, `article` and `noun`. As each word is picked, it should be concatenated to the previous words in an array large enough to hold the entire sentence. The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period. The program should generate 20 such sentences.

The arrays should be filled as follows: The `article` array should contain the articles "the", "a", "one", "some" and "any"; the `noun` array should contain the nouns "boy", "girl", "dog", "town" and "car"; the `verb` array should contain the verbs "drove", "jumped", "ran", "walked" and "skipped"; the `preposition` array should contain the prepositions "to", "from", "over", "under" and "on".

After the preceding program is written and working, modify the program to produce a short story consisting of several of these sentences. (How about the possibility of a random term paper writer?)

ANS:

```

1  /* Exercise 8.11 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #include <string.h>
6  #include <ctype.h>
7
8  int main()
9  {
10     /* initialize 4 arrays of char pointers */
11     char *article[] = { "the", "a", "one", "some", "any" };
12     char *noun[] = { "boy", "girl", "dog", "town", "car" };
13     char *verb[] = { "drove", "jumped", "ran", "walked", "skipped" };
14     char *preposition[] = { "to", "from", "over", "under", "on" };
15     char sentence[ 100 ] = ""; /* completed sentence */
16     int i; /* loop counter */
17
18     /* create 20 sentences */
19     for ( i = 1; i <= 20; i++ ) {
20
21         /* randomly choose pieces of sentence */
22         strcat( sentence, article[ rand() % 5 ] );
23         strcat( sentence, " " );
24
25         strcat( sentence, noun[ rand() % 5 ] );
26         strcat( sentence, " " );
27
28         strcat( sentence, verb[ rand() % 5 ] );
29         strcat( sentence, " " );
30
31         strcat( sentence, preposition[ rand() % 5 ] );
32         strcat( sentence, " " );
33
34         strcat( sentence, article[ rand() % 5 ] );
35         strcat( sentence, " " );
36
37         strcat( sentence, noun[ rand() % 5 ] );
38
39         /* capitalize first letter and print sentence */
40         putchar( toupper( sentence[ 0 ] ) );
41         printf( "%s.\n", &sentence[ 1 ] );
42         sentence[ 0 ] = '\0';
43     } /* end for */
44
45     return 0; /* indicate successful termination */
46
47 } /* end main */

```

A dog skipped to any car.
Some town ran on the boy.
A dog jumped from the dog.
One girl jumped on one town.
One dog jumped from some boy.
One girl jumped under any dog.
One car drove on some girl.
One town walked on a girl.
Some town ran on one dog.
One car walked from any town.
A boy drove over some girl.
The dog skipped under a boy.
The car drove to a girl.
Some town skipped under any car.
A boy jumped from a town.
Any car jumped under one town.
Some dog skipped from some boy.
Any town skipped to one girl.
Some girl jumped to any dog.
The car ran under one dog.

8.12 (*Limericks*) A limerick is a humorous five-line verse in which the first and second lines rhyme with the fifth, and the third line rhymes with the fourth. Using techniques similar to those developed in Exercise 8.11, write a program that produces random limericks. Polishing this program to produce good limericks is a challenging problem, but the result will be worth the effort!

8.13 Write a program that encodes English language phrases into pig Latin. Pig Latin is a form of coded language often used for amusement. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following algorithm:

To form a pig Latin phrase from an English language phrase, tokenize the phrase into words with function `strtok`. To translate each English word into a pig Latin word, place the first letter of the English word at the end of the English word, and add the letters “ay.” Thus the word “jump” becomes “umpjay,” the word “the” becomes “hetay” and the word “computer” becomes “omputercay.” Blanks between words remain as blanks. Assume the following: The English phrase consists of words separated by blanks, there are no punctuation marks, and all words have two or more letters. Function `printLatinWord` should display each word. [*Hint*: Each time a token is found in a call to `strtok`, pass the token pointer to function `printLatinWord`, and print the pig Latin word.]

ANS:

```

1  /* Exercise 8.13 Solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  void printLatinWord( char *word ); /* function prototype */
6
7  int main()
8  {
9      char sentence[ 80 ]; /* sentence input by user */
10     char *tokenPtr;      /* pointer to current token */
11
12     printf( "Enter a sentence:\n" );
13     gets( sentence );
14     printf( "\nThe sentence in Pig Latin is:\n" );
15
16     /* call function strtok to alter the sentence */
17     tokenPtr = strtok( sentence, " .,;" );
18
19     /* if tokenPtr does not equal NULL */
20     while ( tokenPtr ) {
21
22         /* pass the token to printLatinWord and get next token */
23         printLatinWord( tokenPtr );
24         tokenPtr = strtok( NULL, " .,;" );
25
26         /* if tokenPtr not NULL, print space */
27         if ( tokenPtr ) {
28             printf( " " );
29         } /* end if */
30
31     } /* end while */
32
33     printf( "." );
34
35     return 0; /* indicates successful termination */
36
37 } /* end main */
38
39 /* print out the English word in pig Latin form */
40 void printLatinWord( char *word )
41 {
42     unsigned int i; /* loop counter */
43
44     /* loop through the word */
45     for ( i = 1; i < strlen( word ); i++ ) {
46         printf( "%c", word[ i ] );
47     } /* end for */
48
49     printf( "%c%s", word[ 0 ], "ay" );
50 } /* end function printLatinWord */

```

Enter a sentence:
characters and strings

The sentence in Pig Latin is:
haracterscay ndaay tringssay.

8.14 Write a program that inputs a telephone number as a string in the form (555) 555-5555. The program should use function `strtok` to extract the area code as a token, the first three digits of the phone number as a token and the last four digits of the phone number as a token. The seven digits of the phone number should be concatenated into one string. The program should convert the area-code string to `int` and convert the phone number string to `long`. Both the area code and the phone number should be printed.

ANS:

```

1  /* Exercise 8.14 Solution */
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8      char p[ 20 ];
9      char phoneNumber[ 10 ] = { '\0' }; /* complete phone number */
10     char *tokenPtr; /* store temporary token */
11     int areaCode; /* store area code */
12     long phone; /* store phone number */
13
14     printf( "Enter a phone number in the form ( 555 )"
15            " 555-5555:\n" );
16     gets( p );
17
18     /* convert area code token to an integer */
19     areaCode = atoi( strtok( p, "(" ) );
20
21     /* take next token and copy to phoneNumber */
22     tokenPtr = strtok( NULL, "-" );
23     strcpy( phoneNumber, tokenPtr );
24
25     /* take last token and concatenate to phoneNumber */
26     tokenPtr = strtok( NULL, "" );
27     strcat( phoneNumber, tokenPtr );
28
29     /* convert phoneNumber to long integer */
30     phone = atol( phoneNumber );
31
32     printf( "\nThe integer area code is %d\n", areaCode );
33     printf( "The long integer phone number is %ld\n", phone );
34
35     return 0; /* indicate successful termination */
36
37 } /* end main */

```

Enter a phone number in the form (555) 555-5555:
(800) 555-1212

The integer area code is 800
The long integer phone number is 5551212

- 8.15 Write a program that inputs a line of text, tokenizes the line with function `strtok` and outputs the tokens in reverse order.
ANS:

```
1  /* Exercise 8.15 solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  void reverseTokens( char *sentence ); /* function prototype */
6
7  int main()
8  {
9      char text[ 80 ]; /* line of text from user */
10
11     printf( "Enter a line of text:\n" );
12     gets( text );
13
14     reverseTokens( text ); /* call to function reverseTokens */
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */
19
20 /* function to reverse the individual tokens */
21 void reverseTokens( char *sentence )
22 {
23     char *pointers[ 50 ]; /* array to store entire sentence */
24     char *temp;          /* pointer to each token */
25     int count = 0;       /* token counter */
26     int i;               /* loop counter */
27
28     /* function strtok takes first word of sentence */
29     temp = strtok( sentence, " " );
30
31     /* while temp does not equal NULL */
32     while ( temp ) {
33
34         /* add the word into the array and get next token */
35         pointers[ count++ ] = temp;
36         temp = strtok( NULL, " " );
37     } /* end while */
38
39     printf( "The tokens in reverse order are:\n" );
40
41     /* loop through the array backwards */
42     for ( i = count - 1; i >= 0; i-- ) {
43         printf( "%s ", pointers[ i ] );
44     } /* end for */
45
46 } /* end function reverseTokens */
```

```
Enter a line of text:
testing 1 2 3
The tokens in reverse order are:
3 2 1 testing
```

8.16 Write a program that inputs a line of text and a search string from the keyboard. Using function `strstr`, locate the first occurrence of the search string in the line of text, and assign the location to variable `searchPtr` of type `char *`. If the search string is found, print the remainder of the line of text beginning with the search string. Then, use `strstr` again to locate the next occurrence of the search string in the line of text. If a second occurrence is found, print the remainder of the line of text beginning with the second occurrence. [Hint: The second call to `strstr` should contain `searchPtr + 1` as its first argument.]

ANS:

```

1  /* Exercise 8.16 Solution */
2  #include <stdio.h>
3  #include <string.h>
4  int main()
5  {
6      char text[ 80 ]; /* line of text */
7      char search[ 15 ]; /* search string */
8      char *searchPtr; /* pointer to search string */
9
10     /* get line of text from user */
11     printf( "Enter a line of text:\n" );
12     gets( text );
13
14     /* get search string from user */
15     printf( "Enter a search string: " );
16     scanf( "%s", search );
17
18     /* search for search string in text */
19     searchPtr = strstr( text, search );
20
21     /* if searchPtr is not NULL */
22     if ( searchPtr ) {
23         printf( "\n%s\n%s\n%s":\n%s\n",
24             "The remainder of the line beginning with",
25             "the first occurrence of ", search, searchPtr );
26
27         /* search for a second occurrence */
28         searchPtr = strstr( searchPtr + 1, search );
29
30         /* if searchPtr is not NULL */
31         if ( searchPtr ) {
32             printf( "\n%s\n%s\n%s":\n%s\n",
33                 "The remainder of the line beginning with",
34                 "the second occurrence of ", search, searchPtr );
35         } /* end if */
36         else {
37             printf( "The search string appeared only once.\n" );
38         } /* end else */
39     } /* end if */
40     else {
41         printf( "\"%s\" not found.\n", search );
42     } /* end else */
43
44     return 0; /* indicate successful termination */
45 } /* end main */

```



```
Enter a line of text:
To be or not to be; that is the question.
Enter a search string: be
```

```
The remainder of the line beginning with
the first occurrence of "be":
be or not to be; that is the question.
```

```
The remainder of the line beginning with
the second occurrence of "be":
be; that is the question.
```

8.17 Write a program based on the program of Exercise 8.16 that inputs several lines of text and a search string, and uses function `strstr` to determine the total occurrences of the string in the lines of text. Print the result.

ANS:

```
1  /* Exercise 8.17 Solution */
2  #include <stdio.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  int main()
7  {
8      char text[ 3 ][ 80 ]; /* array to hold text entered by user */
9      char search[ 20 ]; /* search string */
10     char *searchPtr; /* pointer to search string */
11     int count = 0; /* total occurrences of search string */
12     int i; /* loop counter */
13     int j; /* loop counter */
14
15     printf( "Enter three lines of text:\n" );
16
17     /* read in 3 lines of text */
18     for ( i = 0; i <= 2; i++ ) {
19         gets( &text[ i ][ 0 ] );
20     } /* end for */
21
22     /* make all characters lowercase */
23     for ( i = 0; i <= 2; i++ ) {
24
25         /* loop through each character */
26         for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {
27             text[ i ][ j ] = tolower( text[ i ][ j ] );
28         } /* end for */
29     } /* end for */
30
31     printf( "\nEnter a search string: " ); /* get search string */
32     scanf( "%s", search );
33
34     /* loop through all three strings */
35     for ( i = 0; i <= 2; i++ ) {
36
37         /* set pointer to first character of string */
38         searchPtr = &text[ i ][ 0 ];
39
40         /* loop while strstr does not return NULL */
41         while ( searchPtr = strstr( searchPtr, search ) ) {
42             ++count;
43             searchPtr++;
44         } /* end while */
45     } /* end for */
46
47     } /* end for */
48
```

```

49     printf( "\nThe total occurrences of \"%s\" in the text is %d\n",
50             search, count );
51
52     return 0; /* indicate successful termination */
53
54 } /* end main */

```

Enter three lines of text:
 This program inputs three lines of text
 and counts the number of occurrences of
 the search string in the three lines of text.

Enter a search string: th

The total occurrences of "th" in the text is 6

8.18 Write a program that inputs several lines of text and a search character, and uses function `strchr` to determine the total occurrences of the character in the lines of text.

ANS:

```

1  /* Exercise 8.18 Solution */
2  #include <stdio.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  int main()
7  {
8      char text[ 3 ][ 80 ]; /* array to hold text entered by user */
9      char search;         /* search character */
10     char *searchPtr;     /* pointer to search character */
11     int count = 0;       /* total search characters found */
12     int i;               /* loop counter */
13     int j;               /* loop counter */
14
15     printf( "Enter three lines of text:\n" );
16
17     /* read 3 lines of text */
18     for ( i = 0; i <= 2; i++ ) {
19         gets( &text[ i ][ 0 ] );
20     } /* end for */
21
22     /* convert all letters to lowercase */
23     for ( i = 0; i <= 2; i++ ) {
24
25         /* loop through each character */
26         for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {
27             text[ i ][ j ] = tolower( text[ i ][ j ] );
28         } /* end for */
29     } /* end for */
30
31     /* get search character */
32     printf( "\nEnter a search character: " );
33     scanf( "%c", &search );
34
35     /* loop through 3 lines of text */
36     for ( i = 0; i <= 2; i++ ) {
37
38         /* set pointer to first character in line */
39         searchPtr = &text[ i ][ 0 ];
40
41         /* loop while strchr does not return NULL */
42         while ( searchPtr = strchr( searchPtr, search ) ) {
43             ++count;
44         }

```

```
45     searchPtr++;
46     } /* end while */
47
48 } /* end for */
49
50 printf( "\nThe total occurrences of '%c' in the text is %d\n",
51         search, count );
52
53 return 0; /* indicate successful termination */
54
55 } /* end main */
```

```
Enter three lines of text:
This program inputs three lines of text
and counts the number of occurrences of
the specified search character in the text
```

```
Enter a search character: e
```

```
The total occurrences of 'e' in the text is 15
```

8.19 Write a program based on the program of Exercise 8.18 that inputs several lines of text and uses function `strchr` to determine the total occurrences of each letter of the alphabet in the lines of text. Uppercase and lowercase letters should be counted together. Store the totals for each letter in an array and print the values in tabular format after the totals have been determined.

ANS:

```

1  /* Exercise 8.19 Solution */
2  #include <stdio.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  int main()
7  {
8      char text[ 3 ][ 80 ];          /* 3 lines of text */
9      char *searchPtr;              /* pointer to search character */
10     char characters[ 26 ] = { 0 }; /* totals for each letter */
11     int count = 0;                 /* total for current letter */
12     int i;                          /* loop counter */
13     int j;                          /* loop counter */
14
15     printf( "Enter three lines of text:\n" );
16
17     /* read three lines of text */
18     for ( i = 0; i <= 2; i++ ) {
19         gets( &text[ i ][ 0 ] );
20     } /* end for */
21
22     /* convert letters to lowercase */
23     for ( i = 0; i <= 2; i++ ) {
24
25         /* loop through each character of line */
26         for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {
27             text[ i ][ j ] = tolower( text[ i ][ j ] );
28         } /* end for */
29
30     } /* end for */
31
32     /* loop through alphabet */
33     for ( i = 0; i <= 25; i++ ) {
34
35         /* loop through 3 lines of text */
36         for ( j = 0, count = 0; j <= 2; j++ ) {
37             searchPtr = &text[ j ][ 0 ];
38
39             /* while strchr does not return NULL */
40             while ( searchPtr = strchr( searchPtr, 'a' + i ) ) {
41                 ++count;
42                 searchPtr++;
43             } /* end while */
44
45         } /* end for */
46
47         characters[ i ] = count;
48     } /* end for */
49
50     printf( "\nThe total occurrences of each character:\n" );
51
52     /* display totals for each character */
53     for ( i = 0; i <= 25; i++ ) {
54         printf( "%c:%3d\n", 'a' + i, characters[ i ] );
55     } /* end for */
56
57     return 0; /* indicate successful termination */
58
59 } /* end main */

```

```
Enter three lines of text:  
This program inputs three lines of text  
and determines the number of occurrences  
of each character in the three lines.
```

```
The total occurrences of each character:
```

```
a: 5  
b: 1  
c: 6  
d: 2  
e: 17  
f: 3  
g: 1  
h: 7  
i: 6  
j: 0  
k: 0  
l: 2  
m: 3  
n: 8  
o: 5  
p: 2  
q: 0  
r: 10  
s: 6  
t: 10  
u: 3  
v: 0  
w: 0  
x: 1  
y: 0  
z: 0
```

8.20 Write a program that inputs several lines of text and uses `strtok` to count the total number of words. Assume that the words are separated either by spaces or newline characters.

ANS:

```
1  /* Exercise 8.20 Solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      char text[ 4 ][ 80 ]; /* text entered by user */
8      char *tokenPtr;      /* pointer to current token */
9      int i;               /* loop counter */
10     int counter = 0;     /* token counter */
11
12     printf( "Enter 4 lines of text: \n" );
13
14     /* read 4 lines of text */
15     for ( i = 0; i <= 3; i++ ) {
16         gets( &text[ i ][ 0 ] );
17     } /* end for */
18
19     /* loop through 4 lines of text */
20     for ( i = 0; i <= 3; i++ ) {
21
22         /* get first token */
23         tokenPtr = strtok( &text[ i ][ 0 ], " \n" );
24
25         /* while tokenPtr does not equal NULL */
26         while ( tokenPtr ) {
27             ++counter;
28             tokenPtr = strtok( NULL, " \n" ); /* get next token */
29         } /* end while */
30     } /* end for */
31
32     printf( "\nThe total number of words is %d\n", counter );
33
34     return 0; /* indicate successful termination */
35
36 } /* end main */
```

```
Enter 4 lines of text:
This line of text has seven words
This line has five words
There are two words on the next line
I am
```

```
The total number of words is 22
```

8.21 Use the string comparison functions discussed in Section 8.6 and the techniques for sorting arrays developed in Chapter 6 to write a program that alphabetizes a list of strings. Use the names of 10 or 15 towns in your area as data for your program.

ANS:

```
1  /* Exercise 8.21 solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  void bubbleSort( char a[][ 50 ] ); /* function prototype */
6
7  int main()
8  {
9      char array[ 10 ][ 50 ]; /* 10 lines of text from user */
10     int i; /* counter */
11
12     /* read in 10 lines of text */
13     for ( i = 0; i <= 9; i++ ) {
14         printf( "Enter a string: " );
15         scanf( "%s", &array[ i ][ 0 ] );
16     } /* end for */
17
18     bubbleSort( array ); /* sort the array of strings */
19     printf( "\nThe strings in sorted order are:\n" );
20
21     /* display text in sorted order */
22     for ( i = 0; i <= 9; i++ ) {
23         printf( "%s\n", &array[ i ][ 0 ] );
24     } /* end for */
25
26     return 0; /* indicate successful termination */
27 } /* end main */
28
29 /* sort the array */
30 void bubbleSort( char a[][ 50 ] )
31 {
32     int i; /* loop counter */
33     int j; /* loop counter */
34     char temp[ 50 ]; /* temporary array */
35
36     /* make 9 passes */
37     for ( i = 0; i <= 8; i++ ) {
38         for ( j = 0; j <= 8; j++ ) {
39
40             /* swap strings if necessary */
41             if ( strcmp( &a[ j ][ 0 ], &a[ j + 1 ][ 0 ] ) > 0 ) {
42                 strcpy( temp, &a[ j ][ 0 ] );
43                 strcpy( &a[ j ][ 0 ], &a[ j + 1 ][ 0 ] );
44                 strcpy( &a[ j + 1 ][ 0 ], temp );
45             } /* end if */
46
47         } /* end for */
48     } /* end for */
49 } /* end function bubbleSort */
```

```
Enter a string: Westborough
Enter a string: Wellesley
Enter a string: Natick
Enter a string: Waltham
Enter a string: Framingham
Enter a string: Marlborough
Enter a string: Boston
Enter a string: Ashland
Enter a string: Hopkington
Enter a string: Shrewsbury
```

The strings in sorted order are:

```
Ashland
Boston
Framingham
Hopkington
Marlborough
Natick
Shrewsbury
Waltham
Wellesley
Westborough
```

8.22 The chart in Appendix D shows the numeric code representations for the characters in the ASCII character set. Study this chart and then state whether each of the following is *true* or *false*.

a) The letter “A” comes before the letter “B.”

ANS: True.

b) The digit “9” comes before the digit “0.”

ANS: False.

c) The commonly used symbols for addition, subtraction, multiplication and division all come before any of the digits.

ANS: True.

d) The digits come before the letters.

ANS: True.

e) If a sort program sorts strings into ascending sequence, then the program will place the symbol for a right parenthesis before the symbol for a left parenthesis.

ANS: False.

- 8.23** Write a program that reads a series of strings and prints only those strings beginning with the letter “b.”
ANS:

```
1  /* Exercise 8.23 solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int i; /* loop counter */
7      char array[ 5 ][ 20 ]; /* 5 strings from user */
8
9      /* read 5 strings from user */
10     for ( i = 0; i <= 4; i++ ) {
11         printf( "Enter a string: " );
12         scanf( "%s", &array[ i ][ 0 ] );
13     } /* end for */
14
15     printf( "\nThe strings starting with 'b' are:\n" );
16
17     /* loop through strings */
18     for ( i = 0; i <= 4; i++ ) {
19
20         /* print if first character is 'b' */
21         if ( array[ i ][ 0 ] == 'b' ) {
22             printf( "%s\n", &array[ i ][ 0 ] );
23         } /* end if */
24     } /* end for */
25
26     return 0; /* indicate successful termination */
27
28
29 }
```

```
Enter a string: the
Enter a string: big
Enter a string: bad
Enter a string: boy
Enter a string: sings
```

```
The strings starting with 'b' are:
big
bad
boy
```

- 8.24 Write a program that reads a series of strings and prints only those strings that end with the letters “ed.”
ANS:

```
1  /* Exercise 8.24 solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      int i;                /* loop counter */
8      int length;          /* length of current string */
9      char array[ 5 ][ 20 ]; /* 5 strings from user */
10
11     /* read in 5 strings from user */
12     for ( i = 0; i <= 4; i++ ) {
13         printf( "Enter a string: " );
14         scanf( "%s", &array[ i ][ 0 ] );
15     } /* end for */
16
17     printf( "\nThe strings ending with \"ED\" are:\n" );
18
19     /* loop through 5 strings */
20     for ( i = 0; i <= 4; i++ ) {
21
22         /* find length of current string */
23         length = strlen( &array[ i ][ 0 ] );
24
25         /* print string if it ends with "ED" */
26         if ( strcmp( &array[ i ][ length - 2 ], "ED" ) == 0 ) {
27             printf( "%s\n", &array[ i ][ 0 ] );
28         } /* end if */
29
30     } /* end for */
31
32     return 0; /* indicate successful termination */
33
34 } /* end main */
```

```
Enter a string: WALKED
Enter a string: SKIPPED
Enter a string: JUMPED
Enter a string: FLEW
Enter a string: DROVE
```

```
The strings ending with "ED" are:
WALKED
SKIPPED
JUMPED
```

8.25 Write a program that inputs an ASCII code and prints the corresponding character. Modify this program so that it generates all possible three-digit codes in the range 000 to 255 and attempts to print the corresponding characters. What happens when this program is run?

ANS:

```
1  /* Exercise 8.25 solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int c; /* ASCII character */
7
8      printf( "Enter an ASCII character code ( EOF to end ): " );
9
10     /* while user does not enter EOF */
11     while ( scanf( "%d", &c ) != EOF ) {
12
13         /* check if character code is valid */
14         if ( c >= 0 && c <= 255 ) {
15             printf( "The corresponding character is '%c'\n", c );
16         } /* end if */
17         else {
18             printf( "Invalid character code\n" );
19         } /* end else */
20
21         printf( "\nEnter an ASCII character code ( EOF to end ): " );
22     } /* end while */
23
24     return 0; /* indicate successful termination */
25
26 } /* end main */
```

```
Enter an ASCII character code ( EOF to end ): 90
The corresponding character is 'Z'
```

```
Enter an ASCII character code ( EOF to end ): 116
The corresponding character is 't'
```

```
Enter an ASCII character code ( EOF to end ): 130
The corresponding character is 'é'
```

```
Enter an ASCII character code ( EOF to end ): 45
The corresponding character is '-'
```

```
Enter an ASCII character code ( EOF to end ): 40
The corresponding character is '('
```

```
Enter an ASCII character code ( EOF to end ): ^Z
```

8.26 Using the ASCII character chart in Appendix D as a guide, write your own versions of the character handling functions in Fig. 8.1.

ANS:

```

1  /* Exercise 8.26 Solution */
2  #include <stdio.h>
3
4  /* function prototypes */
5  int isDigit( int c );
6  int isAlpha( int c );
7  int isAlNum( int c );
8  int isLower( int c );
9  int isUpper( int c );
10 int toLower( int c );
11 int isSpace( int c );
12 int isPunct( int c );
13 int isPrint( int c );
14 int isGraph( int c );
15 int toLower( int c );
16 int toUpper( int c );
17
18 int main()
19 {
20     int v; /* function result */
21     char array[ 2 ] = { '\0' }; /* character from user */
22
23     /* read a character from the user */
24     printf( "Enter a character: " );
25     scanf( "%c", &array[ 0 ] );
26
27     /* test isDigit function */
28     v = isDigit( ( int ) array[ 0 ] );
29     printf( "According to isDigit" );
30     v == 0 ? printf( " %c is not a digit\n", array[ 0 ] ) :
31             printf( " %c is a digit\n", array[ 0 ] );
32
33     /* test isAlpha function */
34     v = isAlpha( ( int ) array[ 0 ] );
35     printf( "According to isAlpha" );
36     v == 0 ? printf( " %c is not a letter\n", array[ 0 ] ) :
37             printf( " %c is a letter\n", array[ 0 ] );
38
39     /* test isAlNum function */
40     v = isAlNum( ( int ) array[ 0 ] );
41     printf( "According to isAlNum" );
42     v == 0 ? printf( " %c is not a letter or digit\n", array[ 0 ] ) :
43             printf( " %c is a letter or digit\n", array[ 0 ] );
44
45     /* test isLower function */
46     v = isLower( ( int ) array[ 0 ] );
47     printf( "According to isLower" );
48     v == 0 ? printf( " %c is not a lowercase letter\n", array[ 0 ] ) :
49             printf( " %c is a lowercase letter\n", array[ 0 ] );
50
51     /* test isUpper function */
52     v = isUpper( ( int ) array[ 0 ] );
53     printf( "According to isUpper" );
54     v == 0 ? printf( " %c is not an uppercase letter\n", array[ 0 ] ) :
55             printf( " %c is an uppercase letter\n", array[ 0 ] );
56
57     /* test isSpace function */
58     v = isSpace( ( int ) array[ 0 ] );
59     printf( "According to isSpace" );
60     v == 0 ? printf( " %c is not a white-space character\n", array[ 0 ] ) :
61             printf( " character is a white-space character\n" );
62

```

```

63  /* test isPunct function */
64  v = isPunct( ( int ) array[ 0 ] );
65  printf( "According to isPunct" );
66  v == 0 ? printf( " %c is not a punctuation character\n", array[ 0 ] ):
67          printf( " %c is a punctuation character\n", array[ 0 ] );
68
69  /* test isPrint function */
70  v = isPrint( ( int ) array[ 0 ] );
71  printf( "According to isPrint" );
72  v == 0 ? printf( " %c is not a printing character\n", array[ 0 ] ):
73          printf( " %c is a printing character\n", array[ 0 ] );
74
75  /* test isGraph function */
76  v = isGraph( ( int ) array[ 0 ] );
77  printf( "According to isGraph" );
78  v == 0 ? printf( " %c is not a printing character\n", array[ 0 ] ):
79          printf( " %c is a printing character other than space\n", array[ 0 ] );
80
81  /* test toLower function */
82  v = toLower( ( int ) array[ 0 ] );
83  printf( "According to toLower" );
84  v == 0 ? printf( " %c is unchanged\n", array[ 0 ] ):
85          printf( " %c has been converted to lowercase\n", v );
86
87  /* test toUpper function */
88  v = toUpper( ( int ) array[ 0 ] );
89  printf( "According to toUpper" );
90  v == 0 ? printf( " %c is unchanged\n", array[ 0 ] ):
91          printf( " %c has been converted to uppercase\n", v );
92
93  return 0; /* indicate successful termination */
94
95 } /* end main */
96
97 /* determines whether argument is a digit */
98 int isDigit( int c )
99 {
100     return ( c >= 48 && c <= 57 ) ? 1 : 0;
101 } /* end function isDigit */
102
103 /* determines whether argument is a letter */
104 int isAlpha( int c )
105 {
106     return ( ( c >= 65 && c <= 90 ) || ( c >= 97 && c <= 122 ) ) ? 1 : 0;
107 } /* end function isAlpha */
108
109 /* determines whether argument is a letter or digit */
110 int isAlNum( int c )
111 {
112     return ( isDigit( c ) == 1 || isAlpha( c ) == 1 ) ? 1 : 0;
113 } /* end function isAlNum */
114
115 /* determines whether argument is a lowercase letter */
116 int isLower( int c )
117 {
118     return ( c >= 97 && c <= 122 ) ? 1 : 0;
119 } /* end function isLower */
120
121 /* determines whether argument is an uppercase letter */
122 int isUpper( int c )
123 {
124     return ( c >= 65 && c <= 90 ) ? 1 : 0;
125 } /* end function isUpper */

```

```
131
132 /* determines whether argument is a whitespace character */
133 int isSpace( int c )
134 {
135     return ( ( c == 32 ) || ( c >= 9 && c <= 13 ) ) ? 1 : 0;
136 } /* end function isSpace */
137
138
139 /* determines whether argument is a printing character
140 other than a space, a digit or a letter */
141 int isPunct( int c )
142 {
143     return ( isAlNum( c ) == 0 && isSpace( c ) == 0 ) ? 1 : 0;
144 } /* end function isPunct */
145
146
147 /* determines whether argument is a printing character
148 including the space character */
149 int isPrint( int c )
150 {
151     return ( c >= 32 && c <= 126 ) ? 1 : 0;
152 } /* end function isPrint */
153
154
155 /* determines whether argument is a printing character
156 other than the space character */
157 int isGraph( int c )
158 {
159     return ( c >= 33 && c <= 126 ) ? 1 : 0;
160 } /* end function isGraph */
161
162
163 /* converts and uppercase letter to lowercase */
164 int toLower( int c )
165 {
166     return ( isUpper( c ) == 1 ) ? c + 32 : c;
167 } /* end function toLower */
168
169
170 /* converts a lowercase letter to uppercase */
171 int toUpper( int c )
172 {
173     return ( isLower( c ) == 1 ) ? c - 32 : c;
174 } /* end function toUpper */
175
176
```

```
Enter a character: m
According to isDigit m is not a digit
According to isAlpha m is a letter
According to isAlNum m is a letter or digit
According to isLower m is a lowercase letter
According to isUpper m is not an uppercase letter
According to isSpace m is not a white-space character
According to isPunct m is not a punctuation character
According to isPrint m is a printing character
According to isGraph m is a printing character other than space
According to toLower m has been converted to lowercase
According to toUpper M has been converted to uppercase
```

```

Enter a character: *
According to isDigit * is not a digit
According to isAlpha * is not a letter
According to isAlNum * is not a letter or digit
According to isLower * is not a lowercase letter
According to isUpper * is not an uppercase letter
According to isSpace * is not a white-space character
According to isPunct * is a punctuation character
According to isPrint * is a printing character
According to isGraph * is a printing character other than space
According to toLower * has been converted to lowercase
According to toUpper * has been converted to uppercase

```

- 8.27** Write your own versions of the functions in Fig. 8.5 for converting strings to numbers.
- 8.28** Write two versions of each of the string copy and string concatenation functions in Fig. 8.17. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.
- 8.29** Write your own versions of the functions `getchar`, `gets`, `putchar` and `puts` described in Fig. 8.12.
- 8.30** Write two versions of each string comparison function in Fig. 8.20. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.
- 8.31** Write your own versions of the functions in Fig. 8.22 for searching strings.
- 8.32** Write your own versions of the functions in Fig. 8.30 for manipulating blocks of memory.
- 8.33** Write two versions of function `strlen` in Fig. 8.36. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.

SPECIAL SECTION: ADVANCED STRING MANIPULATION EXERCISES

The preceding exercises are keyed to the text and designed to test the reader's understanding of fundamental string manipulation concepts. This section includes a collection of intermediate and advanced problems. The reader should find these problems challenging yet enjoyable. The problems vary considerably in difficulty. Some require an hour or two of program writing and implementation. Others are useful for lab assignments that might require two or three weeks of study and implementation. Some are challenging term projects.

8.34 (*Text Analysis*) The availability of computers with string manipulation capabilities has resulted in some rather interesting approaches to analyzing the writings of great authors. Much attention has been focused on whether William Shakespeare ever lived. Some scholars believe that there is substantial evidence indicating that Christopher Marlowe actually penned the masterpieces attributed to Shakespeare. Researchers have used computers to find similarities in the writings of these two authors. This exercise examines three methods for analyzing texts with a computer.

- a) Write a program that reads several lines of text and prints a table indicating the number of occurrences of each letter of the alphabet in the text. For example, the phrase

To be, or not to be: that is the question:

contains one "a," two "b's," no "c's," etc.

ANS:

```

1  /* Exercise 8.34 Part A Solution */
2  #include <stdio.h>
3  #include <ctype.h>
4
5  int main()
6  {
7      char letters[ 26 ] = { 0 }; /* letters of the alphabet */
8      char text[ 3 ][ 80 ];      /* three lines of text */
9      int i;                    /* loop counter */
10     int j;                    /* loop counter */
11
12     printf( "Enter three lines of text:\n" );

```

```

13
14     /* read 3 lines of text */
15     for ( i = 0; i <= 2; i++ ) {
16         gets( &text[ i ][ 0 ] );
17     } /* end for */
18
19     /* loop through 3 strings */
20     for ( i = 0; i <= 2; i++ ) {
21
22         /* loop through each character */
23         for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {
24
25             /* if letter, update corresponding array element */
26             if ( isalpha( text[ i ][ j ] ) ) {
27                 ++letters[ tolower( text[ i ][ j ] ) - 'a' ];
28             } /* end if */
29
30         } /* end for */
31
32     } /* end for */
33
34     printf( "\nTotal letter counts:\n" );
35
36     /* print letter totals */
37     for ( i = 0; i <= 25; i++ ) {
38         printf( "%c:%3d\n", 'a' + i, letters[ i ] );
39     } /* end for */
40
41     return 0; /* indicate successful termination */
42
43 } /* end main */

```

Enter three lines of text:
This program counts the occurrences of each
letter of the alphabet in the input text. Then,
it prints a summary of the occurrences.

Total letter counts:

```

a: 6
b: 1
c: 8
d: 0
e: 14
f: 3
g: 1
h: 8
i: 5
j: 0
k: 0
l: 2
m: 3
n: 7
o: 7
p: 4
q: 0
r: 9
s: 6
t: 15
u: 5
v: 0
w: 0
x: 1
y: 1
z: 0

```


- b) Write a program that reads several lines of text and prints a table indicating the number of one-letter words, two-letter words, three-letter words, etc., appearing in the text. For example, the phrase

Whether 'tis nobler in the mind to suffer

contains

Word length	Occurrences
1	0
2	2
3	1
4	2 (including 'tis)
5	0
6	2
7	1

ANS:

```

1  /* Exercise 8.34 Part B solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      char text[ 3 ][ 80 ];      /* 3 strings from user */
8      char *temp;              /* token pointer */
9      int lengths[ 20 ] = { 0 }; /* array of length counts */
10     int i;                    /* loop counter */
11
12     printf( "Enter three lines of text:\n" );
13
14     /* read 3 lines of text */
15     for ( i = 0; i <= 2; i++ ) {
16         gets( &text[ i ][ 0 ] );
17     } /* end for */
18
19     /* loop through each string */
20     for ( i = 0; i <= 2; i++ ) {
21
22         /* get first token */
23         temp = strtok( &text[ i ][ 0 ], ". \n" );
24
25         /* while temp does not equal NULL */
26         while ( temp ) {
27
28             /* increment corresponding array element */
29             ++lengths[ strlen( temp ) ];
30             temp = strtok( NULL, ". \n" );
31         } /* end while */
32     } /* end for */
33
34     putchar( '\n' );
35
36     /* display results in array */
37     for ( i = 1; i <= 19; i++ ) {
38
39         /* if length is not zero */
40         if ( lengths[ i ] ) {
41             printf( "%d word%s of length %d\n",
42                 lengths[ i ], lengths[ i ] == 1 ? "" : "s", i );
43         }
44     }

```

```

44     } /* end if */
45     } /* end for */
46     } /* end for */
47     return 0; /* indicate successful termination */
48 } /* end main */

```

Enter three lines of text:
This program determines the length of each word
in the input text. The input text here has words
of several different lengths.

```

3 words of length 2
4 words of length 3
6 words of length 4
3 words of length 5
1 word of length 6
3 words of length 7
1 word of length 9
1 word of length 10

```

- c) Write a program that reads several lines of text and prints a table indicating the number of occurrences of each different word in the text. The first version of your program should include the words in the table in the same order in which they appear in the text. A more interesting (and useful) printout should then be attempted in which the words are sorted alphabetically. For example, the lines

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer

contain the words “to” three times, the word “be” two times, the word “or” once, etc.

ANS:

```

1  /* Exercise 8.34 Part C solution */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      char text[ 3 ][ 80 ];          /* 3 string from user */
8      char *temp;                  /* token pointer */
9      char words[ 100 ][ 20 ] = { "" }; /* array of words */
10     int i;                        /* loop counter */
11     int j;                        /* loop counter */
12     int count[ 100 ] = { 0 };     /* array of word counts */
13
14     printf( "Enter three lines of text:\n" );
15
16     /* read three lines of text */
17     for ( i = 0; i <= 2; i++ ) {
18         gets( &text[ i ][ 0 ] );
19     } /* end for */
20
21     /* loop through 3 strings */
22     for ( i = 0; i <= 2; i++ ) {
23
24         /* get first token */
25         temp = strtok( &text[ i ][ 0 ], ". \n" );
26
27         /* while temp does not equal NULL */
28         while ( temp ) {
29

```

```

30     /* loop through words for match */
31     for ( j = 0; words[ j ][ 0 ] && strcmp( temp,
32         &words[ j ][ 0 ] ) != 0; j++ ) {
33         ; /* empty body */
34     } /* end for */
35
36     ++count[ j ]; /* increment count */
37
38     /* if temp could not be found in words array */
39     if ( !words[ j ][ 0 ] ) {
40         strcpy( &words[ j ][ 0 ], temp );
41     } /* end if */
42
43     temp = strtok( NULL, ". \n" );
44 } /* end while */
45
46 } /* end for */
47
48 putchar( '\n' );
49
50 /* loop through words array */
51 for ( j = 0; words[ j ][ 0 ] != '\0' && j <= 99; j++ ) {
52     printf( "\'%s\' appeared %d time%s\n",
53         &words[ j ][ 0 ], count[ j ], count[ j ] == 1 ? "" : "s" );
54 } /* end for */
55
56 return 0; /* indicate successful termination */
57
58 } /* end main */

```

Enter three lines of text:
This program counts the number
of occurrences of each word in
the input text.

```

"This" appeared 1 time
"program" appeared 1 time
"counts" appeared 1 time
"the" appeared 2 times
"number" appeared 1 time
"of" appeared 2 times
"occurrences" appeared 1 time
"each" appeared 1 time
"word" appeared 1 time
"in" appeared 1 time
"input" appeared 1 time
"text" appeared 1 time

```

8.35 (*Word Processing*) The detailed treatment of string manipulation in this text is greatly attributable to the exciting growth in word processing in recent years. One important function in word processing systems is *type-justification*—the alignment of words to both the left and right margins of a page. This generates a professional-looking document that gives the appearance of being set in type, rather than prepared on a typewriter. Type-justification can be accomplished on computer systems by inserting one or more blank characters between each of the words in a line so that the rightmost word aligns with the right margin.

Write a program that reads several lines of text and prints this text in type-justified format. Assume that the text is to be printed on 8 1/2-inch-wide paper and that one-inch margins are to be allowed on both the left and right sides of the printed page. Assume that the computer prints 10 characters to the horizontal inch. Therefore, your program should print 6 1/2 inches of text or 65 characters per line.

8.36 (*Printing Dates in Various Formats*) Dates are commonly printed in several different formats in business correspondence. Two of the more common formats are

07/21/2003 and July 21, 2003

Write a program that reads a date in the first format and prints that date in the second format.

ANS:

```

1  /* Exercise 8.36 solution */
2  #include <stdio.h>
3
4  int main()
5  {
6
7      /* array of month names */
8      char *months[ 13 ] = { "", "January", "February", "March",
9                             "April", "May", "June", "July",
10                            "August", "September", "October",
11                            "November", "December"};
12
13     int m; /* integer month */
14     int d; /* integer day */
15     int y; /* integer year */
16
17     /* read a date from user */
18     printf( "Enter a date in the form mm/dd/yyyy: " );
19     scanf( "%d/%d/%d", &m, &d, &y );
20
21     /* output date in new format */
22     printf( "The date is: %s %d, %d\n", months[ m ], d, y );
23
24     return 0; /* indicate successful termination */
25 } /* end main */

```

```

Enter a date in the form mm/dd/yyyy: 06/18/2003
The date is: June 18, 2003

```

8.37 (*Check Protection*) Computers are frequently used in check-writing systems, such as payroll and accounts payable applications. Many strange stories circulate regarding weekly paychecks being printed (by mistake) for amounts in excess of \$1 million. Weird amounts are printed by computerized check-writing systems because of human error and/or machine failure. Systems designers, of course, make every effort to build controls into their systems to prevent erroneous checks from being issued.

Another serious problem is the intentional alteration of a check amount by someone who intends to cash a check fraudulently. To prevent a dollar amount from being altered, most computerized check-writing systems employ a technique called *check protection*.

Checks designed for imprinting by computer contain a fixed number of spaces in which the computer may print an amount. Suppose a paycheck contains nine blank spaces in which the computer is supposed to print the amount of a weekly paycheck. If the amount is large, then all nine of those spaces will be filled, for example:

```

11,230.60 (check amount)
-----
123456789 (position numbers)

   99.87
-----
123456789

```

contains three blank spaces. If a check is printed with blank spaces, it is easier for someone to alter the amount of the check. To prevent a check from being altered, many check-writing systems insert *leading asterisks* to protect the amount as follows:

```

****99.87
-----
123456789

```

Write a program that inputs a dollar amount to be printed on a check and then prints the amount in check-protected format with leading asterisks if necessary. Assume that nine spaces are available for printing an amount.

ANS:

```

1  /* Exercise 8.37 solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      double amount;          /* check amount */
7      double base = 100000.0; /* base to check number of digits */
8      int i;                  /* loop counter */
9      int j;                  /* loop counter */
10
11     /* get check amount */
12     printf( "Enter check amount: " );
13     scanf( "%lf", &amount );
14
15     printf( "The protected amount is $" );
16
17     /* loop until amount is less than base */
18     for ( i = 0; amount < base; i++ ) {
19         base /= 10;
20     } /* end for */
21
22     /* print i leading asterisks */
23     for ( j = 1; j <= i; j++ ) {
24         printf( "*" );
25     } /* end for */
26
27     printf( "%.2f\n", 9 - i, amount );
28
29     return 0; /* indicate successful termination */
30
31 } /* end main */

```

```

Enter check amount: 234.83
The protected amount is $***234.83

```

```

Enter check amount: 14892.98
The protected amount is $*14892.98

```

```

Enter check amount: 1.54
The protected amount is $*****1.54

```

8.38 (*Writing the Word Equivalent of a Check Amount*) Continuing the discussion of the previous example, we reiterate the importance of designing check-writing systems to prevent alteration of check amounts. One common security method requires that the check amount be both written in numbers and “spelled out” in words. Even if someone is able to alter the numerical amount of the check, it is extremely difficult to change the amount in words.

Many computerized check-writing systems do not print the amount of the check in words. Perhaps the main reason for this omission is the fact that most high-level languages used in commercial applications do not contain adequate string manipulation features. Another reason is that the logic for writing word equivalents of check amounts is somewhat involved.

Write a program that inputs a numeric check amount and writes the word equivalent of the amount. For example, the amount 112.43 should be written as

ONE HUNDRED TWELVE and 43/100

ANS:

```

1  /* Exercise 8.38 solution */
2  /* NOTE THAT THIS PROGRAM ONLY HANDLES VALUES UP TO $99.99 */
3  /* The program is easily modified to process larger values */
4  #include <stdio.h>
5
6  int main()
7  {
8
9      /* word equivalents of single digits */
10     char *digits[ 10 ] = { "", "ONE", "TWO", "THREE", "FOUR",
11                          "FIVE", "SIX", "SEVEN", "EIGHT", "NINE"};
12
13     /* word equivalents of 10-19 */
14     char *teens[ 10 ] = { "TEN", "ELEVEN", "TWELVE", "THIRTEEN",
15                         "FOURTEEN", "FIFTEEN", "SIXTEEN",
16                         "SEVENTEEN", "EIGHTEEN", "NINETEEN"};
17
18     /* word equivalents of tens digits */
19     char *tens[ 10 ] = { "", "TEN", "TWENTY", "THIRTY", "FORTY",
20                       "FIFTY", "SIXTY", "SEVENTY", "EIGHTY",
21                       "NINETY"};
22
23     int dollars; /* check dollar amount */
24     int cents; /* check cents amount */
25     int digit1; /* ones digit */
26     int digit2; /* tens digit */
27
28     /* get check amount */
29     printf( "Enter the check amount ( 0.00 to 99.99 ): " );
30     scanf( "%d.%d", &dollars, &cents );
31     printf( "\nThe check amount in words is:\n" );
32
33     /* print equivalent words */
34     if ( dollars < 10 ) {
35         printf( "%s ", digits[ dollars ] );
36     } /* end if */
37     else if ( dollars < 20 ) {
38         printf( "%s ", teens[ dollars - 10 ] );
39     } /* end else if */
40     else {
41         digit1 = dollars / 10; /* ones digit */
42         digit2 = dollars % 10; /* tens digit */
43
44         /* if ones digit is zero */
45         if ( digit2 == 0 ) {
46             printf( "%s ", tens[ digit1 ] );
47         } /* end if */
48         else {
49             printf( "%s-%s ", tens[ digit1 ], digits[ digit2 ] );
50         } /* end else */
51     }
52     } /* end else */
53
54     printf( "and %d/100\n", cents );
55
56     return 0; /* indicate successful termination */
57
58 } /* end main */

```

Enter the check amount (0.00 to 99.99): 72.63

The check amount in words is:
SEVENTY-TWO and 63/100

```
Enter the check amount ( 0.00 to 99.99 ): 13.22
```

```
The check amount in words is:
THIRTEEN and 22/100
```

```
Enter the check amount ( 0.00 to 99.99 ): 5.75
```

```
The check amount in words is:
FIVE and 75/100
```

ANS:

8.39 (*Morse Code*) Perhaps the most famous of all coding schemes is Morse code, developed by Samuel Morse in 1832 for use with the telegraph system. Morse code assigns a series of dots and dashes to each letter of the alphabet, each digit, and a few special characters (such as period, comma, colon and semicolon). In sound-oriented systems, the dot represents a short sound and the dash represents a long sound. Other representations of dots and dashes are used with light-oriented systems and signal-flag systems.

Separation between words is indicated by a space,—quite simply, the absence of a dot or dash. In a sound-oriented system, a space is indicated by a short period of time during which no sound is transmitted. The international version of Morse code appears in Fig. 8.39.

Write a program that reads an English-language phrase and encodes the phrase into Morse code. Also write a program that reads a phrase in Morse code and converts the phrase into the English-language equivalent. Use one blank between each Morse-coded letter and three blanks between each Morse-coded word.

Character	Code	Character	Code
A	.-	T	-
B	-...	U	..-
C	-.-.	V	...-
D	-..	W	.-
E	.	X	-..-
F	..-.	Y	-.-
G	--.	Z	--..
H		
I	..	<i>Digits</i>	
J	.-	1	.-
K	-.-	2	..
L	.-..	3	...-
M	--	4-
N	-.	5
O	---	6	-....
P	.-.	7	--...
Q	--.-	8	---..
R	.-.	9	----.
S	...	0	-----

Fig. 8.1 The letters of the alphabet as expressed in international Morse code.

8.40 (*A Metric Conversion Program*) Write a program that will assist the user with metric conversions. Your program should allow the user to specify the names of the units as strings (i.e., centimeters, liters, grams, etc., for the metric system and inches, quarts, pounds, etc., for the English system) and should respond to simple questions such as

```
"How many inches are in 2 meters?"  
"How many liters are in 10 quarts?"
```

Your program should recognize invalid conversions. For example, the question

```
"How many feet in 5 kilograms?"
```

is not meaningful, because "feet" are units of length while "kilograms" are units of mass.

8.41 (*Dunning Letters*) Many businesses spend a great deal of time and money collecting overdue debts. *Dunning* is the process of making repeated and insistent demands upon a debtor in an attempt to collect a debt.

Computers are often used to generate dunning letters automatically and in increasing degrees of severity as a debt ages. The theory is that as a debt becomes older, it becomes more difficult to collect, and therefore the dunning letters must become more threatening.

Write a program that contains the texts of five dunning letters of increasing severity. Your program should accept as input the following:

- a) Debtor's name
- b) Debtor's address
- c) Debtor's account
- d) Amount owed
- e) Age of the amount owed (i.e., one month overdue, two months overdue, etc.).

Use the age of the amount owed to select one of the five message texts, and then print the dunning letter inserting the other user-supplied information where appropriate.

A CHALLENGING STRING MANIPULATION PROJECT

8.42 (*A Crossword-Puzzle Generator*) Most people have worked a crossword puzzle at one time or another, but few have ever attempted to generate one. Generating a crossword puzzle is a difficult problem. It is suggested here as a string manipulation project requiring substantial sophistication and effort. There are many issues the programmer must resolve to get even the simplest crossword-puzzle generator program working. For example, how does one represent the grid of a crossword puzzle inside the computer? Should one use a series of strings, or should double-subscripted arrays be used? The programmer needs a source of words (i.e., a computerized dictionary) that can be directly referenced by the program. In what form should these words be stored to facilitate the complex manipulations required by the program? The really ambitious reader will want to generate the "clues" portion of the puzzle in which the brief hints for each "across" word and each "down" word are printed for the puzzle worker. Merely printing a version of the blank puzzle itself is not a simple problem.

9

C Formatted Input/Output: Solutions

SOLUTIONS

9.4 Write a `printf` or `scanf` statement for each of the following:

a) Print unsigned integer 40000 left justified in a 15-digit field with 8 digits.

ANS: `printf("%-15.8u", (unsigned) 40000);`

b) Read a hexadecimal value into variable `hex`.

ANS: `scanf("%x", hex);`

c) Print 200 with and without a sign.

ANS: `printf("+d %d\n", 200, 200);`

d) Print 100 in hexadecimal form preceded by 0x.

ANS: `printf("%#x\n", 100);`

e) Read characters into array `s` until the letter `p` is encountered.

ANS: `scanf("[%p]", s);`

f) Print 1.234 in a 9-digit field with preceding zeros.

ANS: `printf("%09.3f\n", 1.234);`

g) Read a time of the form `hh:mm:ss`, storing the parts of the time in the integer variables `hour`, `minute` and `second`.

Skip the colons (`:`) in the input stream. Use the assignment suppression character.

ANS: `scanf("%d%c%d%c%d", &hour, &minute, &second);`

h) Read a string of the form "characters" from the standard input. Store the string in character array `s`. Eliminate the quotation marks from the input stream.

ANS: `scanf("\"%[^\"]\"", s);`

i) Read a time of the form `hh:mm:ss`, storing the parts of the time in the integer variables `hour`, `minute` and `second`.

Skip the colons (`:`) in the input stream. Do not use the assignment-suppression character.

ANS: `scanf("%d:%d:%d:", &hour, &minute, &second);`

9.5 Show what is printed by each of the following statements. If a statement is incorrect, indicate why.

a) `printf("%-10d\n", 10000);`

ANS: 10000

b) `printf("%c\n", "This is a string");`

ANS: A string cannot be printed with the `%c` specifier.

c) `printf("%*.*lf\n", 8, 3, 1024.987654);`

ANS: 1024.988

d) `printf("%#o\n%#X\n%#e\n", 17, 17, 1008.83689);`

ANS:

021

```

    0X11
    1.008837e+03
e) printf( "% 1d\n%+1d\n", 1000000, 1000000 );
ANS:
    1000000
    +1000000
f) printf( "%10.2E\n", 444.93738 );
ANS:  4.45E+02  preceded by two spaces
g) printf( "%10.2g\n", 444.93738 );
ANS:  4.4e+02  preceded by three spaces
h) printf( "%d\n", 10.987 );
ANS: A floating point value cannot be printed with the %d conversion specifier.

```

9.6 Find the error(s) in each of the following program segments. Explain how each error can be corrected.

```

a) printf( "%s\n", 'Happy Birthday' );
ANS: printf( "%s\n", "Happy Birthday" );
b) printf( "%c\n", 'Hello' );
ANS: printf( "%s\n", "Hello" );
c) printf( "%c\n", "This is a string" );
ANS: printf( "%s\n", "This is a string" );
d) The following statement should print "Bon Voyage":
    printf( "%s", "Bon Voyage" );
ANS: printf( "\ %s", "Bon Voyage" );
e) char day[] = "Sunday";
    printf( "%s\n", day[ 3 ] );
ANS: printf( "%s\n", day );
f) printf( 'Enter your name: ' );
ANS: printf( "Enter your name: " );
g) printf( %f, 123.456 );
ANS: printf( "%F", 123.456 );
h) The following statement should print the characters 'O' and 'K':
    printf( "%s%s\n", 'O', 'K' );
ANS: printf( "%c%c\n", 'O', 'K' );
i) char s[ 10 ];
    scanf( "%c", s[ 7 ] );
ANS: scanf( "%c", &s[ 7 ] );

```

9.7 Write a program that loads 10-element array `number` with random integers from 1 to 1000. For each value, print the value and a running total of the number of characters printed. Use the `%n` conversion specifier to determine the number of characters output for each value. Print the total number of characters output for all values up to and including the current value each time the current value is printed. The output should have the following format:

Value	Total characters
342	3
1000	7
963	10
6	11
etc.	

ANS:

```

1  /* Exercise 9.7 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  int main()
7  {

```

```

8   int a[ 10 ] = { 0 }; /* random integers from 1 to 1000 */
9   int i;                /* loop counter */
10  int count;           /* number of characters in current value */
11  int totalCount = 0;  /* total characters in array */
12
13  srand( time( NULL ) );
14
15  /* fill the array with random numbers */
16  for ( i = 0; i <= 9; i++ ) {
17      a[ i ] = 1 + rand() % 1000;
18  } /* end for */
19
20  /* print table headers */
21  printf( "%s\t%s\n", "Value", "Total characters" );
22
23  /* loop through 10 elements */
24  for ( i = 0; i <= 9; i++ ) {
25      printf( "%d\n", a[ i ], &count );
26      totalCount+= count; /* update totalCount */
27      printf( "\t%d\n", totalCount );
28  } /* end for */
29
30  return 0; /* indicate successful termination */
31
32 } /* end main */

```

Value	Total characters
842	3
18	5
220	8
658	11
275	14
647	17
657	20
623	23
242	26
471	29

9.8 Write a program to test the difference between the %d and %i conversion specifiers when used in scanf statements. Use the statements

```

scanf( "%i%d", &x, &y );
printf( "%d %d\n", x, y );

```

to input and print the values. Test the program with the following sets of input data:

```

10    10
-10   -10
010   010
0x10  0x10

```

ANS:

```

1  /* Exercise 9.8 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int i; /* loop counter */
7      int x; /* first integer from user */
8      int y; /* second integer from user */
9

```

```

10  /* loop four times */
11  for ( i = 1; i <= 4; i++ ) {
12      printf( "\nEnter two integers: " );
13      scanf( "%i%d", &x, &y );
14      printf( "%d %d\n", x, y );
15  } /* end for */
16
17  return 0; /* indicate successful termination */
18
19  } /* end main */

```

```

Enter two integers: 10 10
10 10

Enter two integers: -10 -10
-10 -10

Enter two integers: 010 010
8 10

Enter two integers: 0x10 0x10
16 0

```

9.9 Write a program that prints pointer values using all the integer conversion specifiers and the %p conversion specifier. Which ones print strange values? Which ones cause errors? In which format does the %p conversion specifier display the address on your system?

ANS:

```

1  /* Exercise 9.9 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int x; /* define x for testing */
7
8      printf( "%o\n", &x );
9      printf( "%lo\n", &x );
10     printf( "%d\n", &x );
11     printf( "%ld\n", &x );
12     printf( "%x\n", &x );
13     printf( "%lx\n", &x );
14     printf( "%p\n", &x );
15
16     return 0; /* indicate successful termination */
17
18 } /* end main */

```

```

4577574
4577574
1245052
1245052
12ff7c
12ff7c
0012FF7C

```

9.10 Write a program to test the results of printing the integer value 12345 and the floating-point value 1.2345 in various size fields. What happens when the values are printed in fields containing fewer digits than the values?

ANS:

```

1  /* Exercise 9.10 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6
7      /* print the integer 12345 */
8      printf( "%10d\n", 12345 );
9      printf( "%5d\n", 12345 );
10     printf( "%2d\n\n", 12345 );
11
12     /* print the floating-point value 1.2345 */
13     printf( "%10f\n", 1.2345 );
14     printf( "%6f\n", 1.2345 );
15     printf( "%2f\n", 1.2345 );
16
17     return 0; /* indicate successful termination */
18
19 } /* end main */

```

```

      12345
12345
12345

      1.234500
1.234500
1.234500

```

9.11 Write a program that prints the value 100.453627 rounded to the nearest digit, tenth, hundredth, thousandth and ten thousandth.

ANS:

```

1  /* Exercise 9.11 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      printf( "%.0f\n", 100.453627 );
7      printf( "%.1f\n", 100.453627 );
8      printf( "%.2f\n", 100.453627 );
9      printf( "%.3f\n", 100.453627 );
10     printf( "%.4f\n", 100.453627 );
11
12     return 0; /* indicate successful termination */
13
14 } /* end main */

```

```

100
100.5
100.45
100.454
100.4536

```

9.12 Write a program that inputs a string from the keyboard and determines the length of the string. Print the string using twice the length as the field width.

ANS:

```
1  /* Exercise 9.12 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int count;          /* length of string */
7      char string[ 20 ]; /* string entered by user */
8
9      /* read string from user and find length */
10     printf( "Enter a string:\n" );
11     scanf( "%s%n", string, &count );
12
13     printf( "%*s\n", 2 * count, string ); /* print the string */
14
15     return 0; /* indicate successful termination */
16
17 } /* end main */
```

```
Enter a string:
hello
      hello
```

9.13 Write a program that converts integer Fahrenheit temperatures from 0 to 212 degrees to floating-point Celsius temperatures with 3 digits of precision. Use the formula

$$\text{celsius} = 5.0 / 9.0 * (\text{fahrenheit} - 32);$$

to perform the calculation. The output should be printed in two right-justified columns of 10 characters each, and the Celsius temperatures should be preceded by a sign for both positive and negative values.

ANS:

```

1  /* Exercise 9.13 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int fahrenheit; /* holds fahrenheit temperature */
7      double celcius; /* holds celcius temperature */
8
9      printf( "%10s%12s\n", "Fahrenheit", "Celcius" );
10
11     /* convert fahrenheit to celsius and display temperatures
12      showing the sign for celsius temperatures */
13     for ( fahrenheit = 0; fahrenheit <= 212; fahrenheit++ ) {
14         celcius = 5.0 / 9.0 * ( fahrenheit - 32 );
15         printf( "%10d%+12.3f\n", fahrenheit, celcius );
16     } /* end for */
17
18     return 0; /* indicate successful termination */
19
20 } /* end main */

```

Fahrenheit	Celcius
0	-17.778
1	-17.222
2	-16.667
3	-16.111
4	-15.556
5	-15.000
6	-14.444
7	-13.889
.	.
.	.
204	+95.556
205	+96.111
206	+96.667
207	+97.222
208	+97.778
209	+98.333
210	+98.889
211	+99.444
212	+100.000

9.14 Write a program to test all the escape sequences in Figure 9.16. For the escape sequences that move the cursor, print a character before and after printing the escape sequence so it is clear where the cursor has moved.

ANS:

```

1  /* Exercise 9.14 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6
7      /* test all escape sequences */
8      printf( "The single quote : '\\n" );
9      printf( "The double quote : '\"n" );
10     printf( "The question mark: '\\n" );
11     printf( "The backslash   : '\\n" );
12
13     printf( "The bell. \\a\\n\\n" );
14
15     printf( "Move cursor back one position on current line. *\\b*\\n" );
16     printf( "Move cursor to start of next logical page. *\\f*\\n" );
17
18     printf( "Move cursor to the beginning of next line. *\\n*\\n" );
19     printf( "Move cursor to the beginning of current line. *\\r*\\n" );
20
21     printf( "Move cursor to the next horizontal tab position. *\\t*\\n" );
22     printf( "Move cursor to the next vertical tab position. *\\v*\\n" );
23
24     return 0; /* indicate successful termination */
25
26 } /* end main */

```

```

The single quote : '
The double quote : "
The question mark: ?
The backslash   : \
The bell.

```

```

Move cursor back one position on current line. *
Move cursor to start of next logical page. *?*
Move cursor to the beginning of next line. *
*
*ove cursor to the beginning of current line. *
Move cursor to the next horizontal tab position. *      *
Move cursor to the next vertical tab position. *?*

```

9.15 Write a program that determines whether ? can be printed as part of a printf format control string as a literal character rather than using the \? escape sequence.

ANS:

```
1  /* Exercise 9.15 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      printf( "Did the \? print at the end of the sentence?\n" );
7
8      return 0; /* indicate successful termination */
9
10 } /* end main */
```

```
Did the ? print at the end of the sentence?
```

9.16 Write a program that inputs the value 437 using each of the scanf integer conversion specifiers. Print each input value using all the integer conversion specifiers.

ANS:

```

1  /* Exercise 9.16 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int array[ 5 ]; /* holds the value 437 five times */
7      int loop;      /* loop counter */
8
9      /* array of table headers */
10     char *s[] = { "Read with %d:", "Read with %i:", "Read with %o:",
11                  "Read with %u:", "Read with %x:"};
12
13     /* prompt the user and read 5 values */
14     printf( "Enter the value 437 five times: " );
15     scanf( "%d%i%o%u%x", &array[ 0 ], &array[ 1 ], &array[ 2 ],
16           &array[ 3 ], &array[ 4 ] );
17
18     /* loop through all 5 values */
19     for ( loop = 0; loop <= 4; loop++ ) {
20
21         /* print each of the 5 values */
22         printf( "%s\n%d %i %o %u %x\n\n", s[ loop ], array[ loop ],
23               array[ loop ], array[ loop ], array[ loop ] );
24     } /* end for */
25
26     return 0; /* indicate successful termination */
27
28 } /* end main */

```

```

Enter the value 437 five times: 437 437 437 437 437
Read with %d:
437 437 665 437 1b5

Read with %i:
437 437 665 437 1b5

Read with %o:
287 287 437 287 11f

Read with %u:
437 437 665 437 1b5

Read with %x:
1079 1079 2067 1079 437

```

9.17 Write a program that uses each of the conversion specifiers `e`, `f` and `g` to input the value 1.2345. Print the values of each variable to prove that each conversion specifier can be used to input this same value.

ANS:

```
1  /* Exercise 9.17 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      float a[ 3 ]; /* holds the value 1.2345 three times */
7
8      /* array of table headers */
9      char *s[] = { "Read with %e:", "Read with %f:", "Read with %g:" };
10
11     /* prompt the user and read 3 values */
12     printf( "Enter the value 1.2345 three times: " );
13     scanf( "%e%f%g", &a[ 0 ], &a[ 1 ], &a[ 2 ] );
14
15     printf( "%s%e\n\n", s[ 0 ], a[ 0 ] );
16     printf( "%s%f\n\n", s[ 1 ], a[ 1 ] );
17     printf( "%s%g\n\n", s[ 2 ], a[ 2 ] );
18
19     return 0; /* indicate successful termination */
20
21 } /* end main */
```

```
Enter the value 1.2345 three times: 1.2345 1.2345 1.2345
Read with %e:1.234500e+000

Read with %f:1.234500

Read with %g:1.2345
```

9.18 In some programming languages, strings are entered surrounded by either single *or* double quotation marks. Write a program that reads the three strings `suzy`, `"suzy"` and `'suzy'`. Are the single and double quotes ignored by C or read as part of the string?

ANS:

```
22 /* Exercise 9.18 Solution */
23 #include <stdio.h>
24
25 int main()
26 {
27     char a[ 10 ]; /* first string */
28     char b[ 10 ]; /* second string */
29     char c[ 10 ]; /* third string */
30
31     /* prompt user and read three strings */
32     printf( "Enter the strings suzy, \"suzy\", and 'suzy':\n" );
33     scanf( "%s%s%s", a, b, c );
34
35     printf( "%s %s %s\n", a, b, c ); /* display strings */
36
37     return 0; /* indicate successful termination */
38
39 }
```

```
Enter the strings suzy, "suzy", and 'suzy':
suzy
"suzy"
'suzy'
suzy "suzy" 'suzy'
```

9.19 Write a program that determines whether `?` can be printed as the character constant `'?'` rather than the character constant escape sequence `'\?'` using conversion specifier `%c` in the format control string of a `printf` statement.

ANS:

```
1 /* Exercise 9.19 Solution */
2 #include <stdio.h>
3
4 int main()
5 {
6     const char questionMark = '?'; /* define '?' as a char constant */
7
8     printf( "This %c can be printed without using the \\?\n",
9           questionMark );
10
11     return 0; /* indicate successful termination */
12
13 }
```

```
This ? can be printed without using the \?
```

9.20 Write a program that uses the conversion specifier `g` to output the value `9876.12345`. Print the value with precisions ranging from 1 to 9.

ANS:

```
1  /* Exercise 9.20 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6
7      /* output the value 9876.12345 with precisions from 1 to 9 */
8      printf( "Precision: %d, value = %.1g\n", 1, 9876.12345 );
9      printf( "Precision: %d, value = %.2g\n", 2, 9876.12345 );
10     printf( "Precision: %d, value = %.3g\n", 3, 9876.12345 );
11     printf( "Precision: %d, value = %.4g\n", 4, 9876.12345 );
12     printf( "Precision: %d, value = %.5g\n", 5, 9876.12345 );
13     printf( "Precision: %d, value = %.6g\n", 6, 9876.12345 );
14     printf( "Precision: %d, value = %.7g\n", 7, 9876.12345 );
15     printf( "Precision: %d, value = %.8g\n", 8, 9876.12345 );
16     printf( "Precision: %d, value = %.9g\n", 9, 9876.12345 );
17
18     return 0; /* indicate successful termination */
19
20 } /* end main */
```

```
Precision: 1, value = 1e+004
Precision: 2, value = 9.9e+003
Precision: 3, value = 9.88e+003
Precision: 4, value = 9876
Precision: 5, value = 9876.1
Precision: 6, value = 9876.12
Precision: 7, value = 9876.123
Precision: 8, value = 9876.1234
Precision: 9, value = 9876.12345
```


10

Structures, Unions, Bit Manipulations and Enumerations: Solutions

Solutions

10.5 Provide the definition for each of the following structures and unions:

- a) Structure `inventory` containing character array `partName[30]`, integer `partNumber`, floating point `price`, integer `stock` and integer `reorder`.

ANS:

```
struct inventory {
    char partName[ 30 ];
    int partNumber;
    float price;
    int stock;
    int reorder;
};
```

- b) Union `data` containing `char c`, `short s`, `long l`, `float f` and `double d`.

ANS:

```
union data {
    char c;
    short s;
    long l;
    float f;
    double d;
};
```

- c) A structure called `address` that contains character arrays `streetAddress[25]`, `city[20]`, `state[3]` and `zipCode[6]`.

ANS:

```
struct address {
    char streetAddress[ 25 ];
    char city[ 20 ];
    char state[ 3 ];
    char zipCode[ 6 ];
};
```

- d) Structure `student` that contains arrays `firstName[15]` and `lastName[15]` and variable `homeAddress` of type `struct address` from part (c).

ANS:

```
struct student {
    char firstName[ 15 ];
```



```

    char lastName[ 15 ];
    struct address homeAddress;
};

```

e) Structure test containing 16 bit fields with widths of 1 bit. The names of the bit fields are the letters a to p.

ANS:

```

struct test {
    unsigned a:1, b:1, c:1, d:1, e:1, f:1, g:1, h:1,
           i:1, j:1, k:1, l:1, m:1, n:1, o:1, p:1;
};

```

10.6 Given the following structure and variable definitions,

```

struct customer {
    char lastName[ 15 ];
    char firstName[ 15 ];
    int customerNumber;

    struct {
        char phoneNumber[ 11 ];
        char address[ 50 ];
        char city[ 15 ];
        char state[ 3 ];
        char zipCode[ 6 ];
    } personal;
} customerRecord, *customerPtr;

customerPtr = &customerRecord;

```

write an expression that can be used to access the structure members in each of the following parts:

a) Member lastName of structure customerRecord.

ANS: customerRecord.lastName

b) Member lastName of the structure pointed to by customerPtr.

ANS: customerPtr->lastName

c) Member firstName of structure customerRecord.

ANS: customerRecord.firstName

d) Member firstName of the structure pointed to by customerPtr.

ANS: customerPtr->firstName

e) Member customerNumber of structure customerRecord.

ANS: customerRecord.customerNumber

f) Member customerNumber of the structure pointed to by customerPtr.

ANS: customerRecord->customerNumber

g) Member phoneNumber of member personal of structure customerRecord.

ANS: customerRecord.personal.phoneNumber

h) Member phoneNumber of member personal of the structure pointed to by customerPtr.

ANS: customerRecord->personal.phoneNumber

i) Member address of member personal of structure customerRecord.

ANS: customerRecord.personal.address

j) Member address of member personal of the structure pointed to by customerPtr.

ANS: customerRecord->personal.address

k) Member city of member personal of structure customerRecord.

ANS: customerRecord.personal.city

l) Member city of member personal of the structure pointed to by customerPtr.

ANS: customerRecord->personal.city

m) Member state of member personal of structure customerRecord.

ANS: customerRecord.personal.state

n) Member state of member personal of the structure pointed to by customerPtr.

ANS: customerRecord->personal.state

o) Member zipCode of member personal of customerRecord.

ANS: customerRecord.personal.zipCode

p) Member zipCode of member personal of the structure pointed to by customerPtr.

ANS: customerRecord->personal.zipCode

10.7 Modify the program of Fig. 10.16 to shuffle the cards using a high performance shuffle (as shown in Fig. 10.3). Print the resulting deck in two column format as in Fig. 10.4. Precede each card with its color.

ANS:

```

1  /* Exercise 10.7 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* bitCard structure definition */
7  struct bitCard {
8      unsigned face : 4; /* 4 bits; 0-15 */
9      unsigned suit : 2; /* 2 bits; 0-3 */
10     unsigned color : 1; /* 1 bit; 0-1 */
11 }; /* end structure bitCard */
12
13 /* new type name Card */
14 typedef struct bitCard Card;
15
16 /* prototypes */
17 void fillDeck( Card *wDeck );
18 void shuffle( Card *wDeck );
19 void deal( Card *wDeck2 );
20
21 int main()
22 {
23     Card deck[ 52 ]; /* create array of Cards */
24
25     srand( time( NULL ) ); /* randomize */
26
27     fillDeck( deck );
28     shuffle( deck );
29     deal( deck );
30
31     return 0; /* indicate successful termination */
32 } /* end main */
33
34 /* create 52 cards */
35 void fillDeck( Card *wDeck )
36 {
37     int i; /* loop counter */
38
39     /* loop 52 times and create cards */
40     for ( i = 0; i <= 51; i++ ) {
41         wDeck[ i ].face = i % 13;
42         wDeck[ i ].suit = i / 13;
43         wDeck[ i ].color = i / 26;
44     } /* end for */
45 } /* end function fillDeck */
46
47 /* shuffle cards */
48 void shuffle( Card *wDeck )
49 {
50     int i; /* current card */
51     int j; /* random card to swap with current card */
52     Card temp; /* temporary Card */
53 }
54
55

```

```
56  /* loop through deck */
57  for ( i = 0; i <= 51; i++ ) {
58      j = rand() % 52;
59
60      /* swap cards if not equal */
61      if ( i != j ) {
62          temp = wDeck[ i ];
63          wDeck[ i ] = wDeck[ j ];
64          wDeck[ j ] = temp;
65      } /* end if */
66
67  } /* end for */
68
69 } /* end function shuffle */
70
71 /* deal the cards */
72 void deal( Card *wDeck2 )
73 {
74
75     /* arrays face, suit and color hold all possible string
76        descriptions of the cards */
77     char *face[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
78                    "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
79     char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades" };
80     char *color[] = { "Red", "Black" };
81     int i; /* loop counter */
82
83     /* loop through deck and print string description of each card */
84     for ( i = 0; i <= 51; i++ ) {
85         printf( "%5s: %5s of %-8s", color[ wDeck2[ i ].color ],
86               face[ wDeck2[ i ].face ], suit[ wDeck2[ i ].suit ] );
87         putchar( ( i + 1 ) % 2 ? '\t' : '\n' );
88     } /* end for */
89
90 } /* end function deal */
```

Red: Eight of Diamonds	Red: Queen of Hearts
Red: Jack of Hearts	Red: Seven of Hearts
Red: Three of Diamonds	Black: Eight of Spades
Black: Ten of Spades	Black: Three of Clubs
Black: Jack of Spades	Black: Deuce of Spades
Red: Deuce of Diamonds	Red: Ten of Hearts
Red: Queen of Diamonds	Red: King of Diamonds
Black: Nine of Clubs	Black: Ace of Spades
Red: Seven of Diamonds	Red: Three of Hearts
Black: Nine of Spades	Red: Five of Diamonds
Black: Jack of Clubs	Black: Six of Spades
Black: Five of Clubs	Black: Queen of Clubs
Black: Ace of Clubs	Red: Nine of Hearts
Red: Ten of Diamonds	Red: Ace of Diamonds
Black: Deuce of Clubs	Red: Four of Diamonds
Black: Seven of Clubs	Red: King of Hearts
Red: Six of Hearts	Red: Deuce of Hearts
Red: Jack of Diamonds	Black: Three of Spades
Red: Four of Hearts	Black: Four of Clubs
Black: Ten of Clubs	Black: Six of Clubs
Red: Nine of Diamonds	Black: King of Spades
Red: Ace of Hearts	Black: Five of Spades
Black: Four of Spades	Black: Queen of Spades
Black: Seven of Spades	Red: Five of Hearts
Red: Eight of Hearts	Black: King of Clubs
Black: Eight of Clubs	Red: Six of Diamonds

10.8 Create union integer with members char c, short s, int i and long b. Write a program that inputs value of type char, short, int and long and stores the values in union variables of type union integer. Each union variable should be printed as a char, a short, an int and a long. Do the values always print correctly?

ANS:

```

1  /* Exercise 10.8 Solution */
2  /* NOTE: The program output is machine dependent */
3  #include <stdio.h>
4
5  /* integer union definition */
6  union integer {
7      char c; /* character input by user */
8      short s; /* short integer input by user */
9      int i; /* integer input by user */
10     long l; /* long integer input by user */
11 }; /* end union integer */
12
13 int main()
14 {
15     union integer a; /* define union a */
16
17     /* read a character from user into the union */
18     printf( "Enter a character: " );
19     scanf( "%c", &a.c );
20
21     /* print each value of union */
22     printf( "\\'%c'\\ printed as a character is %c\\n", a.c, a.c );
23     printf( "\\'%c'\\ printed as a short integer is %hd\\n", a.c, a.s );
24     printf( "\\'%c'\\ printed as an integer is %d\\n", a.c, a.i );
25     printf( "\\'%c'\\ printed as a long integer is %ld\\n", a.c, a.l );
26
27     /* read a short integer from user into the union */
28     printf( "\\nEnter a short integer: " );
29     scanf( "%hd", &a.s );
30
31     /* print each value of union */
32     printf( "%hd printed as a character is %c\\n", a.s, a.c );
33     printf( "%hd printed as a short integer is %hd\\n", a.s, a.s );
34     printf( "%hd printed as an integer is %d\\n", a.s, a.i );
35     printf( "%hd printed as a long integer is %ld\\n", a.s, a.l );
36
37     /* read an integer from user into the union */
38     printf( "\\nEnter an integer: " );
39     scanf( "%d", &a.i );
40
41     /* print each value of union */
42     printf( "%d printed as a character is %c\\n", a.i, a.c );
43     printf( "%d printed as a short integer is %hd\\n", a.i, a.s );
44     printf( "%d printed as an integer is %d\\n", a.i, a.i );
45     printf( "%d printed as a long integer is %ld\\n", a.i, a.l );
46
47     /* read a long integer from user into the union */
48     printf( "\\nEnter a long integer: " );
49     scanf( "%ld", &a.l );
50
51     /* print each value of union */
52     printf( "%ld printed as a character is %c\\n", a.l, a.c );
53     printf( "%ld printed as a short integer is %hd\\n", a.l, a.s );
54     printf( "%ld printed as an integer is %d\\n", a.l, a.i );
55     printf( "%ld printed as a long integer is %ld\\n", a.l, a.l );
56

```

```
57     return 0; /* indicate successful termination */
58
59 } /* end main */
```

```
Enter a character: A
'A' printed as a character is A
'A' printed as a short integer is -13247
'A' printed as an integer is -858993599
'A' printed as a long integer is -858993599

Enter a short integer: 97
97 printed as a character is a
97 printed as a short integer is 97
97 printed as an integer is -859045791
97 printed as a long integer is -859045791

Enter an integer: 32700
32700 printed as a character is +
32700 printed as a short integer is 32700
32700 printed as an integer is 32700
32700 printed as a long integer is 32700

Enter a long integer: 10000000
10000000 printed as a character is Ç
10000000 printed as a short integer is -27008
10000000 printed as an integer is 10000000
10000000 printed as a long integer is 10000000
```

10.9 Create union `floatingPoint` with members `float f`, `double d` and `long double x`. Write a program that inputs value of type `float`, `double` and `long double` and stores the values in union variables of type union `floatingPoint`. Each union variable should be printed as a `float`, a `double` and a `long double`. Do the values always print correctly?

ANS:

```

1  /* Exercise 10.9 Solution */
2  /* NOTE: The program output is machine dependent */
3  #include <stdio.h>
4
5  /* floatingPoint union definition */
6  union floatingPoint {
7      float f;      /* floating-point value input by user */
8      double d;    /* double value input by user */
9      long double l; /* long double value input by user */
10 }; /* end union floatingPoint */
11
12 int main()
13 {
14     union floatingPoint a; /* define union a */
15
16     /* read a floating-point value from user into the union */
17     printf( "Enter a float: " );
18     scanf( "%f", &a.f );
19
20     /* print each value of union */
21     printf( "%f printed as a float is %f\n", a.f, a.f );
22     printf( "%f printed as a double is %f\n", a.f, a.d );
23     printf( "%f printed as a long double is %Lf\n", a.f, a.l );
24
25     /* read a double value from user into the union */
26     printf( "\nEnter a double: " );
27     scanf( "%lf", &a.d );
28
29     /* print each value of union */
30     printf( "%lf printed as a float is %f\n", a.d, a.f );
31     printf( "%lf printed as a double is %f\n", a.d, a.d );
32     printf( "%lf printed as a long double is %Lf\n", a.d, a.l );
33
34     /* read a long double value from user into the union */
35     printf( "\nEnter a long double: " );
36     scanf( "%Lf", &a.l );
37
38     /* print each value of union */
39     printf( "%Lf printed as a float is %f\n", a.l, a.f );
40     printf( "%Lf printed as a double is %f\n", a.l, a.d );
41     printf( "%Lf printed as a long double is %Lf\n", a.l, a.l );
42
43     return 0; /* indicate successful termination */
44
45 } /* end main */

```

10.10 Write a program that right shifts an integer variable 4 bits. The program should print the integer in bits before and after the shift operation. Does your system place 0s or 1s in the vacated bits?

ANS:

```

1  /* Exercise 10.10 Solution */
2  #include <stdio.h>
3
4  void displayBits( unsigned value ); /* prototype */
5
6  int main()
7  {
8      unsigned val; /* value from user */
9
10     /* prompt user and read value */
11     printf( "Enter an integer: " );
12     scanf( "%u", &val );
13
14     /* display value before shifting */
15     printf( "%u before right shifting 4 bits is:\n", val );
16     displayBits( val );
17
18     /* display value after shifting */
19     printf( "%u after right shifting 4 bits is:\n", val );
20     displayBits( val >> 4 );
21
22     return 0; /* indicate successful termination */
23
24 } /* end main */
25
26 /* function displayBits prints each bit of value */
27 void displayBits( unsigned value )
28 {
29     unsigned c; /* bit counter */
30     unsigned displayMask = 1 << 15; /* bit mask */
31
32     printf( "%7u = ", value );
33
34     /* loop through bits */
35     for ( c = 1; c <= 16; c++ ) {
36         value & displayMask ? putchar( '1' ) : putchar( '0' );
37         value <<= 1; /* shift value 1 bit to the left */
38
39         if ( c % 8 == 0 ) { /* print a space */
40             putchar( ' ' );
41         } /* end if */
42     } /* end for */
43
44     putchar( '\n' );
45 } /* end function displayBits */

```

```

Enter an integer: 1234
1234 before right shifting 4 bits is:
1234 = 00000100 11010010
1234 after right shifting 4 bits is:
77 = 00000000 01001101

```

10.11 If your computer uses 2-byte integers, modify the program of Fig. 10.7 so that it works with 2-byte integers.

10.12 Left shifting an unsigned integer by 1 bit is equivalent to multiplying the value 2. Write function power2 that takes two integer arguments number and pow and calculates

$$\text{number} * 2^{\text{pow}}$$

Use the shift operator to calculate the result. Print the values as integers and as bits.

ANS:

```

1  /* Exercise 10.12 Solution */
2  #include <stdio.h>
3
4  /* prototypes */
5  void displayBits( unsigned value );
6  unsigned power2( unsigned n, unsigned p );
7
8  int main()
9  {
10     unsigned number; /* value from user */
11     unsigned pow;    /* number of bits to left shift */
12     unsigned result; /* result of shift */
13
14     /* prompt user and read two integers */
15     printf( "Enter two integers: " );
16     scanf( "%u%u", &number, &pow );
17
18     /* display bits of number */
19     printf( "number:\n" );
20     displayBits( number );
21
22     /* display bits of pow */
23     printf( "\npow:\n" );
24     displayBits( pow );
25
26     /* perform shift and display results */
27     result = power2( number, pow );
28     printf( "\n%u * 2^%u = %u\n", number, pow, result );
29     displayBits( result );
30
31     return 0; /* indicate successful termination */
32 }
33 /* end main */
34
35 /* function power2 left shifts n by p */
36 unsigned power2( unsigned n, unsigned p )
37 {
38     return n << p;
39 }
40 /* end function power2 */
41
42 /* display the bits of value */
43 void displayBits( unsigned value )
44 {
45     unsigned c; /* bit counter */
46     unsigned displayMask = 1 << 15; /* bit mask */
47
48     printf( "%7u = ", value );
49
50     /* loop through bits */
51     for ( c = 1; c <= 16; c++ ) {
52         value & displayMask ? putchar( '1' ) : putchar( '0' );

```

```

53     value <<= 1; /* shift value 1 bit to the left */
54
55     if ( c % 8 == 0 ) { /* print a space */
56         putchar( ' ' );
57     } /* end if */
58
59 } /* end for */
60
61 putchar( '\n' );
62 } /* end function displayBits */

```

```

Enter two integers: 10 3
number:
    10 = 00000000 00001010

pow:
    3 = 00000000 00000011

10 * 2^3 = 80
    80 = 00000000 01010000

```

10.13 The left-shift operator can be used to pack two character values into an unsigned integer variable. Write a program that inputs two characters from the keyboard and passes them to function `packCharacters`. To pack two characters into an unsigned integer variable, assign the first character to the unsigned variable, shift the unsigned variable left by 8 bit positions and combine the unsigned variable with the second character using the bitwise inclusive OR operator. The program should output the characters in their bit format before and after they are packed into the unsigned integer to prove that the characters are in fact packed correctly in the unsigned variable.

ANS:

```

1  /* Exercise 10.13 Solution */
2  #include <stdio.h>
3
4  /* prototypes */
5  unsigned packCharacters( char x, char y );
6  void displayBits( unsigned value );
7
8  int main()
9  {
10     char a;          /* first character from user */
11     char b;          /* second character from user */
12     unsigned result; /* result of packing both characters */
13
14     /* prompt user and read two characters */
15     printf( "Enter two characters: " );
16     scanf( "%c %c", &a, &b );
17
18     /* display first character as bits */
19     printf( "\\'%c\\' in bits as an unsigned integers is:\n", a );
20     displayBits( a );
21
22     /* display second character as bits */
23     printf( "\\n\\'%c\\' in bits as an unsigned integers is:\n", b );
24     displayBits( b );
25
26     /* pack characters and display result */
27     result = packCharacters( a, b );
28     printf( "\\n\\'%c\\' and \\'%c\\' packed in an unsigned integer:\n",
29            a, b );

```

```
30     displayBits( result );
31
32     return 0; /* indicate successful termination */
33
34 } /* end main */
35
36 /* function packCharacters packs two characters into an unsigned int */
37 unsigned packCharacters( char x, char y )
38 {
39     unsigned pack = x; /* initialize pack to x */
40
41     pack <<= 8; /* shift pack 8 bits to the left */
42     pack |= y; /* pack y using inclusive OR operator */
43     return pack;
44
45 } /* end function packCharacters */
46
47 /* display the bits of value */
48 void displayBits( unsigned value )
49 {
50     unsigned c; /* bit counter */
51     unsigned displayMask = 1 << 15; /* bit mask */
52
53     printf( "%7u = ", value );
54
55     /* loop through bits */
56     for ( c = 1; c <= 16; c++ ) {
57         value & displayMask ? putchar( '1' ) : putchar( '0' );
58         value <<= 1; /* shift value 1 bit to the left */
59
60         if ( c % 8 == 0 ) { /* print a space */
61             putchar( ' ' );
62         } /* end if */
63
64     } /* end for */
65
66     putchar( '\n' );
67 } /* end function displayBits */
```

```
Enter two characters: A B
'A' in bits as an unsigned integers is:
65 = 00000000 01000001

'B' in bits as an unsigned integers is:
66 = 00000000 01000010

'A' and 'B' packed in an unsigned integer:
16706 = 01000001 01000010
```

10.14 Using the right-shift operator, the bitwise AND operator and a mask, write function `unpackCharacters` that takes the unsigned integer from Exercise 10.13 and unpacks it into two characters. To unpack two characters from an unsigned integer, combine the unsigned integer with the mask 65280 (00000000 00000000 11111111 00000000) and right shift the result 8 bits. Assign the resulting value to a char variable. Then combine the unsigned integer with the mask 255 (00000000 00000000 00000000 11111111). Assign the result to another char variable. The program should print the unsigned integer in bits before it is unpacked, then print the characters in bits to confirm that they were unpacked correctly.

ANS:

```

1  /* Exercise 10.14 Solution */
2  #include <stdio.h>
3
4  /* prototypes */
5  void unpackCharacters( char *aPtr, char *bPtr, unsigned pack );
6  void displayBits( unsigned value );
7
8  int main()
9  {
10     char a; /* first character unpacked */
11     char b; /* second character unpacked */
12     unsigned packed = 16706; /* initialize packed value */
13
14     /* display bits of packed */
15     printf( "The packed character representation is:\n" );
16     displayBits( packed );
17
18     /* unpack packed and display results */
19     unpackCharacters( &a, &b, packed );
20     printf( "\nThe unpacked characters are '%c' and '%c'\n", a, b );
21     displayBits( a );
22     displayBits( b );
23
24     return 0; /* indicate successful termination */
25
26 } /* end main */
27
28 /* unpack two characters from pack */
29 void unpackCharacters( char *aPtr, char *bPtr, unsigned pack )
30 {
31     unsigned mask1 = 65280; /* mask for first character */
32     unsigned mask2 = 255; /* mask for second character */
33
34     *aPtr = ( pack & mask1 ) >> 8; /* separate first character */
35     *bPtr = ( pack & mask2 ); /* separate second character */
36 } /* end function unpackCharacters */
37
38 /* display the bits of value */
39 void displayBits( unsigned value )
40 {
41     unsigned c; /* bit counter */
42     unsigned displayMask = 1 << 15; /* bit mask */
43
44     printf( "%7u = ", value );
45
46     /* loop through bits */
47     for ( c = 1; c <= 16; c++ ) {
48         value & displayMask ? putchar( '1' ) : putchar( '0' );
49         value <<= 1; /* shift value 1 bit to the left */
50
51         if ( c % 8 == 0 ) { /* print a space */
52             putchar( ' ' );
53         } /* end if */

```

```
54
55     } /* end for */
56
57     putchar( '\n' );
58 } /* end function displayBits */
```

The packed character representation is:
16706 = 01000001 01000010

The unpacked characters are 'A' and 'B'
65 = 00000000 01000001
66 = 00000000 01000010

10.15 If your system uses 4-byte integers, rewrite the program of Exercise 10.13 to pack 4 characters.

10.16 If your system uses 4-byte integers, rewrite the function `unpackCharacters` of Exercise 10.14 to unpack 4 characters. Create the masks you need to unpack the 4 characters by left shifting the value 255 in the mask variable by 8 bits 0, 1, 2 or 3 times (depending on the byte you are unpacking).

10.17 Write a program that reverses the order of the bits in an unsigned integer value. The program should input the value from the user and call function `reverseBits` to print the bits in reverse order. Print the value in bits both before and after the bits are reversed to confirm that the bits are reversed properly.

ANS:

```

1  /* Exercise 10.17 Solution */
2  #include <stdio.h>
3
4  /* prototypes */
5  unsigned reverseBits( unsigned value );
6  void displayBits( unsigned value );
7
8  int main()
9  {
10     unsigned a; /* unsigned integer from user */
11
12     /* prompt user and read value */
13     printf( "Enter an unsigned integer: " );
14     scanf( "%u", &a );
15
16     /* display bits of a before reversed */
17     printf( "\nBefore bits are reversed:\n" );
18     displayBits( a );
19
20     /* reverse bits and display results */
21     a = reverseBits( a );
22     printf( "\nAfter bits are reversed:\n" );
23     displayBits( a );
24
25     return 0; /* indicate successful termination */
26 } /* end main */
27
28
29 /* reverseBits reverses the bits of value */
30 unsigned reverseBits( unsigned value )
31 {
32     unsigned mask = 1; /* bit mask */
33     unsigned temp = 0; /* reversed bits */
34     int i; /* loop counter */
35
36     /* loop through bits of value */
37     for ( i = 0; i <= 15; i++ ) {
38         temp <<= 1; /* right shift 1 bit */
39         temp |= ( value & mask ); /* separate bit and place in temp */
40         value >>= 1; /* left shift 1 bit */
41     } /* end for */
42
43     return temp;
44 } /* end function reverseBits */
45
46
47 /* display the bits of value */
48 void displayBits( unsigned value )
49 {
50     unsigned c; /* bit counter */
51     unsigned displayMask = 1 << 15; /* bit mask */
52
53     printf( "%7u = ", value );
54
55     /* loop through bits */
56     for ( c = 1; c <= 16; c++ ) {

```

```
57     value & displayMask ? putchar( '1' ) : putchar( '0' );
58     value <<= 1; /* shift value 1 bit to the left */
59
60     if ( c % 8 == 0 ) { /* print a space */
61         putchar( ' ' );
62     } /* end if */
63
64 } /* end for */
65
66     putchar( '\n' );
67 } /* end function displayBits */
```

Enter an unsigned integer: 2127

Before bits are reversed:

2127 = 00001000 01001111

After bits are reversed:

61968 = 11110010 00010000

10.18 Modify function `displayBits` of Fig. 10.7 so it is portable between systems using 2-byte integers and systems using 4-byte integers. [*Hint*: Use the `sizeof` operator to determine the size of an integer on a particular machine.]

ANS:

```

1  /* Exercise 10.18 Solution */
2  #include <stdio.h>
3
4  void displayBits( unsigned value ); /* prototype */
5
6  int main()
7  {
8      unsigned x; /* value from user */
9
10     /* prompt user and read value */
11     printf( "Enter an unsigned integer: " );
12     scanf( "%u", &x );
13     displayBits( x );
14
15     return 0; /* indicate successful termination */
16
17 } /* end main */
18
19 /* display the bits of value */
20 void displayBits( unsigned value )
21 {
22     unsigned c; /* bit counter */
23     unsigned displayMask; /* bit mask */
24
25     /* if system uses 4-byte integers */
26     if ( sizeof( int ) == 4 ) {
27         displayMask = 1 << 31;
28     } /* end if */
29     else { /* assume default of 2-byte integers */
30         displayMask = 1 << 15;
31     } /* end else */
32
33     printf( "%7u = ", value );
34
35     /* loop through bits */
36     for ( c = 1; c <= sizeof( int ) * 8; c++ ) {
37         putchar( value & displayMask ? '1' : '0' );
38         value <<= 1; /* shift value 1 bit to the left */
39
40         if ( c % 8 == 0 ) { /* print a space */
41             putchar( ' ' );
42         } /* end if */
43
44     } /* end for */
45
46     putchar( '\n' );
47 } /* end function displayBits */

```

```

Enter an unsigned integer: 2345
2345 = 00000000 00000000 00001001 00101001

```

10.19 The following program uses function `multiple` to determine if the integer entered from the keyboard is a multiple of some integer `X`. Examine the function `multiple`, then determine the value of `X`.

```
1  /* ex10_19.c */
2  /* This program determines if a value is a multiple of X. */
3  #include <stdio.h>
4
5  int multiple( int num ); /* prototype */
6
7  int main()
8  {
9      int y; /* y will hold an integer entered by the user */
10
11     printf( "Enter an integer between 1 and 32000: " );
12     scanf( "%d", &y );
13
14     /* if y is a multiple of X */
15     if ( multiple( y ) ) {
16         printf( "%d is a multiple of X\n", y );
17     } /* end if */
18     else {
19         printf( "%d is not a multiple of X\n", y );
20     } /* end else */
21
22     return 0; /* indicates successful termination */
23 } /* end main */
24
25 /* determine if num is a multiple of X */
26 int multiple( int num )
27 {
28     int i;          /* counter */
29     int mask = 1; /* initialize mask */
30     int mult = 1; /* initialize mult */
31
32     for ( i = 1; i <= 10; i++, mask <<= 1 ) {
33
34         if ( ( num & mask ) != 0 ) {
35             mult = 0;
36             break;
37         } /* end if */
38
39     } /* end for */
40
41     return mult;
42 } /* end function multiple */
```

ANS:

```
Enter an integer between 1 and 32000: 1024
1024 is a multiple of X
```

10.20 What does the following program do?

```
1  /* ex10_20.c */
2  #include <stdio.h>
3
4  int mystery( unsigned bits ); /* prototype */
5
6  int main()
7  {
8      unsigned x; /* x will hold an integer entered by the user */
9
10     printf( "Enter an integer: " );
11     scanf( "%u", &x );
12
13     printf( "The result is %d\n", mystery( x ) );
14
15     return 0; /* indicates successful termination */
16 } /* end main */
17
18 /* What does this function do? */
19 int mystery( unsigned bits )
20 {
21     unsigned i;          /* counter */
22     unsigned mask = 1 << 31; /* initialize mask */
23     unsigned total = 0;  /* initialize total */
24
25     for ( i = 1; i <= 32; i++, bits <<= 1 ) {
26
27         if ( ( bits & mask ) == mask ) {
28             total++;
29         } /* end if */
30     } /* end for */
31
32     return !( total % 2 ) ? 1 : 0;
33 } /* end function mystery */
```

ANS:

```
Enter an integer: 5678
The result is 0
```

```
Enter an integer: 65
The result is 1
```

11

C File Processing: Solutions

SOLUTIONS

11.5 Fill in the blanks in each of the following:

a) Computers store large amounts of data on secondary storage devices as _____.

ANS: files.

b) A(n) _____ is composed of several fields.

ANS: record.

c) A field that may contain digits, letters and blanks is called a(n) _____ field.

ANS: alphanumeric.

d) To facilitate the retrieval of specific records from a file, one field in each record is chosen as a(n) _____.

ANS: key.

e) The vast majority of information stored in computer systems is stored in _____ files.

ANS: sequential

f) A group of related characters that conveys meaning is called a(n) _____.

ANS: field.

g) The file pointers for the three files that are opened automatically when program execution begins are named _____, _____ and _____.

ANS: stdin, stdout, stderr.

h) Function _____ writes a character to a specified file.

ANS: fputc.

i) Function _____ writes a line to a specified file.

ANS: fputs.

j) Function _____ is generally used to write data to a random-access file.

ANS: fwrite.

k) Function _____ repositions the file position pointer to the beginning of the file.

ANS: rewind.

11.6 State which of the following are *true* and which are *false*. If *false*, explain why.

a) The impressive functions performed by computers essentially involve the manipulation of zeros and ones.

ANS: True.

b) People prefer to manipulate bits instead of characters and fields because bits are more compact.

ANS: False. People prefer to manipulate characters and fields because they are less cumbersome and more understandable.

c) People specify programs and data items as characters; computers then manipulate and process these characters as groups of zeros and ones.

ANS: True.

d) A person's zip code is an example of a numeric field.

ANS: True.

e) A person's street address is generally considered to be an alphabetic field in computer applications.

ANS: False. A street address is generally considered to be alphanumeric.

f) Data items processed by a computer form a data hierarchy in which data items become larger and more complex as we progress from fields to characters to bits etc.

ANS: Data items process by a computer form a data hierarchy in which data items become larger and more complex as we progress from bits to characters to fields, etc.

g) A record key identifies a record as belonging to a particular field.

ANS: False. A record key identifies a record as belonging to a particular person or entity.

h) Most organizations store all their information in a single file to facilitate computer processing.

ANS: False. Most organizations have many files in which they store their information.

i) Files are always referred to by name in C programs.

ANS: False. A pointer to each file is used to refer to the file.

j) When a program creates a file, the file is automatically retained by the computer for future reference.

ANS: True.

11.7 Exercise 11.3 asked the reader to write a series of single statements. Actually, these statements form the core of an important type of file-processing program, namely, a file-matching program. In commercial data processing, it is common to have several files in each system. In an accounts receivable system, for example, there is generally a master file containing detailed information about each customer such as the customer's name, address, telephone number, outstanding balance, credit limit, discount terms, contract arrangements and possibly a condensed history of recent purchases and cash payments.

As transactions occur (i.e., sales are made and cash payments arrive in the mail), they are entered into a file. At the end of each business period (i.e., a month for some companies, a week for others and a day in some cases) the file of transactions (called "trans.dat" in Exercise 11.3) is applied to the master file (called "oldmast.dat" in Exercise 11.3), thus updating each account's record of purchases and payments. After each of these updatings run, the master file is rewritten as a new file ("newmast.dat"), which is then used at the end of the next business period to begin the updating process again.

File-matching programs must deal with certain problems that do not exist in single-file programs. For example, a match does not always occur. A customer on the master file might not have made any purchases or cash payments in the current business period, and therefore no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payments might have just moved to this community, and the company may not have had a chance to create a master record for this customer.

Use the statements written in Exercise 11.3 as a basis for writing a complete file-matching accounts receivable program. Use the account number on each file as the record key for matching purposes. Assume that each file is a sequential file with records stored in increasing account number order.

When a match occurs (i.e., records with the same account number appear on both the master file and the transaction file), add the dollar amount on the transaction file to the current balance on the master file and write the "newmast.dat" record. (Assume that purchases are indicated by positive amounts on the transaction file, and that payments are indicated by negative amounts.) When there is a master record for a particular account but no corresponding transaction record, merely write the master record to "newmast.dat". When there is a transaction record but no corresponding master record, print the message "Unmatched transaction record for account number ..." (fill in the account number from the transaction record).

ANS:

```

1  /* Exercise 11.7 Solution */
2  /* NOTE: This program was run using the */
3  /* data in Exercise 11.8 */
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  int main()
8  {
9      int masterAccount;           /* account from old master file */
10     int transactionAccount;      /* account from transactions file */
11     double masterBalance;       /* balance from old master file */
12     double transactionBalance;  /* balance from transactions file */
13     char masterName[ 30 ];      /* name from master file */
14     FILE *ofPtr;               /* old master file pointer */

```

```

15 FILE *tfPtr;           /* transactions file pointer */
16 FILE *nfPtr;         /* new master file pointer */
17
18 /* terminate application if old master file cannot be opened */
19 if ( ( ofPtr = fopen( "oldmast.dat", "r" ) ) == NULL ) {
20     printf( "Unable to open oldmast.dat\n" );
21     exit( 1 );
22 } /* end if */
23
24 /* terminate application if transactions file cannot be opened */
25 if ( ( tfPtr = fopen( "trans.dat", "r" ) ) == NULL ) {
26     printf( "Unable to open trans.dat\n" );
27     exit( 1 );
28 } /* end if */
29
30 /* terminate application if new master file cannot be opened */
31 if ( ( nfPtr = fopen( "newmast.dat", "w" ) ) == NULL ) {
32     printf( "Unable to open newmast.dat\n" );
33     exit( 1 );
34 } /* end if */
35
36 /* display account currently being processed */
37 printf( "Processing...\n" );
38 fscanf( tfPtr, "%d%lf", &transactionAccount, &transactionBalance );
39
40 /* while not the end of transactions file */
41 while ( !feof( tfPtr ) ) {
42
43     /* read next record from old master file */
44     fscanf( ofPtr, "%d[^\0-9-]%lf", &masterAccount, masterName,
45           &masterBalance );
46
47     /* display accounts from master file until number of
48        new account is reached */
49     while ( masterAccount < transactionAccount && !feof( ofPtr ) ) {
50         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
51               masterBalance );
52         printf( "%d %s %.2f\n", masterAccount, masterName,
53               masterBalance );
54
55         /* read next record from old master file */
56         fscanf( ofPtr, "%d[^\0-9-]%lf", &masterAccount,
57               masterName, &masterBalance );
58     } /* end while */
59
60     /* if matching account found, update balance and output
61        account info */
62     if ( masterAccount == transactionAccount ) {
63         masterBalance += transactionBalance;
64         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
65               masterBalance );
66         printf( "%d %s %.2f\n", masterAccount, masterName,
67               masterBalance );
68     } /* end if */
69
70     /* tell user if account from transactions file does
71        not match account from master file */
72     else if ( masterAccount > transactionAccount ) {
73         printf( "Unmatched transaction record for account %d\n",
74               transactionAccount );
75         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
76               masterBalance );
77         printf( "%d %s %.2f\n", masterAccount, masterName,
78               masterBalance );
79     } /* end else if */
80     else {
81         printf( "Unmatched transaction record for account %d\n",
82               transactionAccount );
83     } /* end else */

```

```

84
85     /* get next account and balance from transactions file */
86     fscanf( tFPtr, "%d%lf", &transactionAccount, &transactionBalance );
87 } /* end while */
88
89 /* loop through file and display account number, name and balance */
90 while ( !feof( ofPtr ) ) {
91     fscanf( ofPtr, "%d%[^0-9-]%lf", &masterAccount, masterName,
92           &masterBalance );
93     fprintf( nfPtr, "%d %s %.2f", masterAccount, masterName,
94           masterBalance );
95     printf( "%d %s %.2f", masterAccount, masterName, masterBalance );
96 } /* end while */
97
98 fclose( ofPtr ); /* close all file pointers */
99 fclose( tFPtr );
100 fclose( nfPtr );
101
102 return 0; /* indicate successful termination */
103
104 } /* end main */

```

```

Processing...
100 Alan Jones 375.31
300 Mary Smith 89.30
Unmatched transaction record for account 400
500 Sam Sharp 0.00
700 Suzy Green -14.22
Unmatched transaction record for account 900

```

11.8 After writing the program of Exercise 11.7, write a simple program to create some test data for checking out the program of Exercise 11.7. Use the following sample account data:

Master File:		
Account number	Name	Balance
100	Alan Jones	348.17
300	Mary Smith	27.19
500	Sam Sharp	0.00
700	Suzy Green	-14.22

Transaction File:	
Account number	Dollar amount
100	27.14
300	62.11
400	100.56
900	82.17

ANS:

```

1 /* Exercise 11.8 Solution */
2 #include <stdio.h>
3
4 int main()
5 {
6     int account; /* account number */

```

```

7   char name[ 30 ]; /* account name */
8   double balance; /* account balance */
9   double amount; /* transaction amount */
10  FILE *ofPtr; /* old master file pointer */
11  FILE *tfPtr; /* transaction file pointer */
12
13  /* open both files for writing */
14  ofPtr = fopen( "oldmast.dat", "w" );
15  tfPtr = fopen( "trans.dat", "w" );
16
17  /* prompt user for sample data */
18  printf( "Sample data for file oldmast.dat:\n" );
19  printf( "Enter account, name, and balance (EOF to end): " );
20
21  /* loop while EOF character not entered by user */
22  while ( scanf( "%d%[^0-9-]%lf", &account, name,
23              &balance ) != EOF ) {
24
25      /* write data to old master file */
26      fprintf( ofPtr, "%d %s %.2f\n", account, name, balance );
27      printf( "Enter account, name, and balance (EOF to end): " );
28  } /* end while */
29
30  fclose( ofPtr ); /* close file pointer */
31
32  /* prompt user for sample data */
33  printf( "\nSample data for file trans.dat:\n" );
34  printf( "Enter account and transaction amount (EOF to end): " );
35
36  /* loop while EOF character not entered by user */
37  while ( scanf( "%d%lf", &account, &amount ) != EOF ) {
38
39      /* write data to transactions file */
40      fprintf( tfPtr, "%d %.2f\n", account, amount );
41      printf( "Enter account and transaction amount (EOF to end): " );
42  } /* end while */
43
44  fclose( tfPtr ); /* close file pointer */
45
46  return 0; /* indicate successful termination */
47
48 } /* end main */

```

Sample data for file oldmast.dat:

```

Enter account, name, and balance (EOF to end): 100 Alan Jones 348.17
Enter account, name, and balance (EOF to end): 300 Mary Smith 27.19
Enter account, name, and balance (EOF to end): 500 Sam Sharp 0.00
Enter account, name, and balance (EOF to end): 700 Suzy Green -14.22
Enter account, name, and balance (EOF to end): ^Z

```

Sample data for file trans.dat:

```

Enter account and transaction amount (EOF to end): 100 27.14
Enter account and transaction amount (EOF to end): 300 62.11
Enter account and transaction amount (EOF to end): 400 100.56
Enter account and transaction amount (EOF to end): 900 82.17
Enter account and transaction amount (EOF to end): ^Z

```


11.9 Run the program of Exercise 11.7 using the files of test data created in Exercise 11.8. Use the listing program of Section 11.7 to print the new master file. Check the results carefully.

11.10 It is possible (actually common) to have several transaction records with the same record key. This occurs because a particular customer might make several purchases and cash payments during a business period. Rewrite your accounts receivable file-matching program of Exercise 11.7 to provide for the possibility of handling several transaction records with the same record key. Modify the test data of Exercise 11.8 to include the following additional transaction records:

Account number	Dollar amount
300	83.89
700	80.78
700	1.53

ANS:

```

1  /* Exercise 11.10 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      int masterAccount;      /* account from old master file */
8      int transactionAccount; /* account from transactions file */
9      double masterBalance;   /* balance from old master file */
10     double transactionBalance; /* balance from transactions file */
11     char masterName[ 30 ];   /* name from master file */
12     FILE *ofPtr;             /* old master file pointer */
13     FILE *tfPtr;             /* transactions file pointer */
14     FILE *nfPtr;             /* new master file pointer */
15
16     /* terminate application if old master file cannot be opened */
17     if ( ( ofPtr = fopen( "oldmast.dat", "r" ) ) == NULL ) {
18         printf( "Unable to open oldmast.dat\n" );
19         exit( 1 );
20     } /* end if */
21
22     /* terminate application if transactions file cannot be opened */
23     if ( ( tfPtr = fopen( "trans.dat", "r" ) ) == NULL ) {
24         printf( "Unable to open trans.dat\n" );
25         exit( 1 );
26     } /* end if */
27
28     /* terminate application if new master file cannot be opened */
29     if ( ( nfPtr = fopen( "newmast.dat", "w" ) ) == NULL ) {
30         printf( "Unable to open newmast.dat\n" );
31         exit( 1 );
32     } /* end if */
33
34     /* display account currently being processed */
35     printf( "Processing...\n" );
36     fscanf( tfPtr, "%d%lf", &transactionAccount, &transactionBalance );
37
38     /* while not the end of transactions file */
39     while ( !feof( tfPtr ) ) {
40
41         /* read next record from old master file */
42         fscanf( ofPtr, "%d%[^A0-9-]%lf", &masterAccount, masterName,
43             &masterBalance );
44
45         /* display accounts from master file until number of
46            new account is reached */
47         while ( masterAccount < transactionAccount && !feof( ofPtr ) ) {
48             fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
49                 masterBalance );

```

```

50     printf( "%d %s %.2f\n", masterAccount, masterName,
51             masterBalance );
52
53     /* read next record from old master file */
54     fscanf( ofPtr, "%d%[\^0-9-]%1f", &masterAccount,
55     masterName, &masterBalance );
56     } /* end while */
57
58     /* if matching account found, update balance and output
59     account info */
60     if ( masterAccount == transactionAccount ) {
61
62         /* while more transactions exist for current account */
63         while ( masterAccount == transactionAccount &&
64                 !feof( tfPtr ) ) {
65
66             /* update masterBalance and read next record */
67             masterBalance += transactionBalance;
68             fscanf( tfPtr, "%d%1f", &transactionAccount,
69                     &transactionBalance );
70         } /* end while */
71
72         fprintf( nfPtr, "%d %s %.2f\n",
73                 masterAccount, masterName, masterBalance );
74         printf( "%d %s %.2f\n", masterAccount, masterName, masterBalance );
75     } /* end if */
76
77     /* tell user if account from transactions file does
78     not match account from master file */
79     else if ( masterAccount > transactionAccount ) {
80         printf( "Unmatched transaction record for account %d\n",
81                 transactionAccount );
82         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName, masterBalance );
83         printf( "%d %s %.2f\n", masterAccount, masterName, masterBalance );
84         fscanf( tfPtr, "%d%1f", &transactionAccount, &transactionBalance );
85     } /* end else if */
86     else {
87         printf( "Unmatched transaction record for account %d\n",
88                 transactionAccount );
89         fscanf( tfPtr, "%d%1f", &transactionAccount, &transactionBalance );
90     } /* end else */
91
92     } /* end while */
93
94     /* loop through file and display account number, name and balance */
95     while ( !feof( ofPtr ) ) {
96         fscanf( ofPtr, "%d%[\^0-9-]%1f", &masterAccount, masterName,
97                 &masterBalance );
98         fprintf( nfPtr, "%d %s %.2f", masterAccount, masterName,
99                 masterBalance );
100        printf( "%d %s %.2f", masterAccount, masterName, masterBalance );
101    } /* end while */
102
103    fclose( ofPtr ); /* close all file pointers */
104    fclose( tfPtr );
105    fclose( nfPtr );
106
107    return 0; /* indicate successful termination */
108
109 } /* end main */

```

```

Processing...
100 Alan Jones 375.31
300 Mary Smith 173.19
Unmatched transaction record for account 400
500 Sam Sharp 0.00
700 Suzy Green 68.09
Unmatched transaction record for account 900

```

11.11 Write statements that accomplish each of the following. Assume that the structure

```

struct person {
    char lastName[ 15 ];
    char firstName[ 15 ];
    char age[ 4 ];
};

```

has been defined and that the file is already open for writing.

- Initialize the file "nameage.dat" so that there are 100 records with lastName = "unassigned", firstName = "" and age = "0".
- Input 10 last names, first names and ages, and write them to the file.
- Update a record; if there is no information in the record, tell the user "No info".
- Delete a record that has information by reinitializing that particular record.

11.12 You are the owner of a hardware store and need to keep an inventory that can tell you what tools you have, how many you have and the cost of each one. Write a program that initializes the file "hardware.dat" to 100 empty records, lets you input the data concerning each tool, enables you to list all your tools, lets you delete a record for a tool that you no longer have and lets you update *any* information in the file. The tool identification number should be the record number. Use the following information to start your file:

Record #	Tool name	Quantity	Cost
3	Electric sander	7	57.98
17	Hammer	76	11.99
24	Jig saw	21	11.00
39	Lawn mower	3	79.50
56	Power saw	18	99.99
68	Screwdriver	106	6.99
77	Sledge hammer	11	21.50
83	Wrench	34	7.50

11.13 *Telephone Number Word Generator.* Standard telephone keypads contain the digits 0 through 9. The numbers 2 through 9 each have three letters associated with them, as is indicated by the following table:

Digit	Letter
2	A B C
3	D E F
4	G H I
5	J K L
6	M N O
7	P R S
8	T U V
9	W X Y

Many people find it difficult to memorize phone numbers, so they use the correspondence between digits and letters to develop seven-letter words that correspond to their phone numbers. For example, a person whose telephone number is 686-2377 might use the correspondence indicated in the above table to develop the seven-letter word "NUMBERS."

Businesses frequently attempt to get telephone numbers that are easy for their clients to remember. If a business can advertise a simple word for its customers to dial, then no doubt the business will receive a few more calls.

Each seven-letter word corresponds to exactly one seven-digit telephone number. The restaurant wishing to increase its take-home business could surely do so with the number 825-3688 (i.e., "TAKEOUT").

Each seven-digit phone number corresponds to many separate seven-letter words. Unfortunately, most of these represent unrecognizable juxtapositions of letters. It is possible, however, that the owner of a barber shop would be pleased to know that the shop's telephone number, 424-7288, corresponds to "HAIRCUT." The owner of a liquor store would, no doubt, be delighted to find that the store's telephone number, 233-7226, corresponds to "BEERCAN." A veterinarian with the phone number 738-2273 would be pleased to know that the number corresponds to the letters "PETCARE."

Write a C program that, given a seven-digit number, writes to a file every possible seven-letter word corresponding to that number. There are 2187 (3 to the seventh power) such words. Avoid phone numbers with the digits 0 and 1.

ANS:

```

1  /* Exercise 11.13 Solution */
2  #include <stdio.h>
3
4  void wordGenerator( int number[] ); /* prototype */
5
6  int main()
7  {
8      int loop; /* loop counter */
9      int phoneNumber[ 7 ] = { 0 }; /* holds phone number */
10
11     /* prompt user to enter phone number */
12     printf( "Enter a phone number one digit at a time" );
13     printf( " using the digits 2 thru 9:\n" );
14
15     /* loop 7 times to get number */
16     for ( loop = 0; loop <= 6; loop++ ) {
17         printf( "? " );
18         scanf( "%d", &phoneNumber[ loop ] );
19
20         /* test if number is between 0 and 9 */
21         while ( phoneNumber[ loop ] < 2 || phoneNumber[ loop ] > 9 ) {
22             printf( "\nInvalid number entered. Please enter again: " );
23             scanf( "%d", &phoneNumber[ loop ] );
24         } /* end while */
25
26     } /* end for */
27
28     wordGenerator( phoneNumber ); /* form words from phone number */
29
30     return 0; /* indicate successful termination */
31 } /* end main */
32
33 /* function to form words based on phone number */
34 void wordGenerator( int number[] )
35 {
36     {
37         int loop; /* loop counter */
38         int loop1; /* loop counter for first digit of phone number */
39         int loop2; /* loop counter for second digit of phone number */
40         int loop3; /* loop counter for third digit of phone number */
41         int loop4; /* loop counter for fourth digit of phone number */
42         int loop5; /* loop counter for fifth digit of phone number */
43         int loop6; /* loop counter for sixth digit of phone number */
44         int loop7; /* loop counter for seventh digit of phone number */
45         FILE *foutPtr; /* output file pointer */
46

```

```

47  /* letters corresponding to each number */
48  char *phoneLetters[ 10 ] = { "", "", "ABC", "DEF", "GHI", "JKL",
49                               "MNO", "PRS", "TUV", "WXY"};
50
51  /* open output file */
52  if ( ( foutPtr = fopen( "phone.out", "w" ) ) == NULL ) {
53      printf( "Output file was not opened.\n" );
54  } /* end if */
55  else { /* print all possible combinations */
56
57      for ( loop1 = 0; loop1 <= 2; loop1++ ) {
58
59          for ( loop2 = 0; loop2 <= 2; loop2++ ) {
60
61              for ( loop3 = 0; loop3 <= 2; loop3++ ) {
62
63                  for ( loop4 = 0; loop4 <= 2; loop4++ ) {
64
65                      for ( loop5 = 0; loop5 <= 2; loop5++ ) {
66
67                          for ( loop6 = 0; loop6 <= 2; loop6++ ) {
68
69                              for ( loop7 = 0; loop7 <= 2; loop7++ ) {
70                                  fprintf( foutPtr, "%c%c%c%c%c%c%c\n",
71                                          phoneLetters[ number[ 0 ] ][ loop1 ],
72                                          phoneLetters[ number[ 1 ] ][ loop2 ],
73                                          phoneLetters[ number[ 2 ] ][ loop3 ],
74                                          phoneLetters[ number[ 3 ] ][ loop4 ],
75                                          phoneLetters[ number[ 4 ] ][ loop5 ],
76                                          phoneLetters[ number[ 5 ] ][ loop6 ],
77                                          phoneLetters[ number[ 6 ] ][ loop7 ] );
78                                  } /* end for */
79
80                              } /* end for */
81
82                          } /* end for */
83
84                      } /* end for */
85
86                  } /* end for */
87
88              } /* end for */
89
90          } /* end for */
91
92          /* output phone number */
93          fprintf( foutPtr, "\nPhone number is " );
94
95          /* loop through digits */
96          for ( loop = 0; loop <= 6; loop++ ) {
97
98              /* insert hyphen */
99              if ( loop == 3 ) {
100                 fprintf( foutPtr, "-" );
101             } /* end if */
102
103             fprintf( foutPtr, "%d", number[ loop ] );
104         } /* end for */
105
106     } /* end else */
107
108     fclose( foutPtr ); /* close file pointer */
109 } /* end function wordGenerator

```

```
Enter a phone number one digit at a time using the digits 2 thru 9:  
? 8  
? 4  
? 3  
? 2  
? 6  
? 7  
? 7
```

The contents of phone.out are:

```
TGDAMPP  
TGDAMPR  
TGDAMPS  
TGDAMRP  
TGDAMRR  
TGDAMRS  
TGDAMSP  
TGDAMSR  
.  
.  
.  
VIFCORP  
VIFCORR  
VIFCORS  
VIFCOSP  
VIFCOSR  
VIFCOSS
```

```
Phone number is 843-2677
```

11.14 If you have a computerized dictionary available, modify the program you wrote in Exercise 11.13 to look up the words in the dictionary. Some seven-letter combinations created by this program consist of two or more words (the phone number 843-2677 produces “THEBOSS”).

11.15 Modify the example of Fig. 8.14 to use functions `fgetc` and `fputs` rather than `getchar` and `puts`. The program should give the user the option to read from the standard input and write to the standard output or to read from a specified file and write to a specified file. If the user chooses the second option, have the user enter the file names for the input and output files.

ANS:

```

1  /* Exercise 11.15 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      char c; /* current character */
8      char sentence[ 80 ]; /* text from user or input file */
9      char input[ 20 ]; /* input file */
10     char output[ 20 ]; /* output file */
11     char choice[ 2 ]; /* user's menu choice */
12     int i = 0; /* character counter */
13     FILE *infilePtr; /* input file pointer */
14     FILE *outfilePtr; /* output file pointer */
15
16     /* display choices to user */
17     printf( "%s\n%s\n%s", "1 Read from standard input; ",
18            "write to standard output", "2 Read from a file; write to file",
19            "Enter choice: " );
20     scanf( "%s", choice );
21
22     /* while user does not enter a valid choice */
23     while ( choice[ 0 ] != '1' && choice[ 0 ] != '2' ) {
24         printf( "Invalid choice. Choose again: " );
25         scanf( "%s", choice );
26     } /* end while */
27
28     /* if user chooses option 2 */
29     if ( choice[ 0 ] == '2' ) {
30         printf( "Enter input file name: " ); /* get input file name */
31         scanf( "%s", input );
32
33         printf( "Enter output file name: " ); /* get output file name */
34         scanf( "%s", output );
35
36         /* exit program if unable to open input file */
37         if ( ( infilePtr = fopen( input, "r" ) ) == NULL ) {
38             printf( "Unable to open %s\n", input );
39             exit( 1 );
40         } /* end if */
41
42         /* exit program if unable to open output file */
43         else if ( ( outfilePtr = fopen( output, "w" ) ) == NULL ) {
44             printf( "Unable to open %s\n", output );
45             fclose( infilePtr );
46             exit( 1 );
47         } /* end if */
48
49     } /* end if */
50     else { /* if user chooses option 1 */
51         infilePtr = stdin;
52         outfilePtr = stdout;
53     } /* end else */
54
55     /* if user chooses option 1 */
56     if ( choice[ 0 ] == '1' ) {
57
58         /* prompt user for text */
59         printf( "Enter a line of text:\n" );
60         scanf( " " ); /* Eliminate spaces and newlines at the
61                    start of the input stream */
62     } /* end if */

```

```

63
64     /* read each character using fgetc */
65     while ( ( c = fgetc( infilePtr ) ) != '\n' && !feof( infilePtr ) ) {
66         sentence[ i++ ] = c;
67     } /* end while */
68
69     /* add terminating character and output text with fputs */
70     sentence[ i ] = '\0';
71     fprintf( outfilePtr, "\nThe line entered was:\n" );
72     fputs( sentence, outfilePtr );
73
74     /* close file pointers */
75     if ( choice[ 0 ] == '2' ) {
76         fclose( infilePtr );
77         fclose( outfilePtr );
78     } /* end if */
79
80     return 0; /* indicate successful termination */
81
82 } /* end main */

```

```

1 Read from standard input; write to standard output
2 Read from a file; write to file
Enter choice: 1
Enter a line of text:
This is a test.

The line entered was:
This is a test.

```

```

1 Read from standard input; write to standard output
2 Read from a file; write to a file
Enter choice: 2
Enter input file name: test.dat
Enter output file name: output.dat

```

Contents of test.dat

```
This is a test file for exercise 11.15.
```

Contents of output.dat

```
The line entered was:
This is a test file for exercise 11.15.
```


11.16 Write a program that uses the `sizeof` operator to determine the sizes in bytes of the various data types on your computer system. Write the results to the file "datasize.dat" so you may print the results later. The format for the results in the file should be as follows:

Data type	Size
char	1
unsigned char	1
short int	2
unsigned short int	2
int	4
unsigned int	4
long int	4
unsigned long int	4
float	4
double	8
long double	16

[Note: The type sizes on your computer might be different from those listed above.]

ANS:

```

1  /* Exercise 11.16 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      FILE *outPtr; /* output file pointer */
7
8      /* open datasize.dat for writing */
9      outPtr = fopen( "datasize.dat", "w" );
10
11     /* write size of various data types */
12     fprintf( outPtr, "%s%16s\n", "Data type", "Size" );
13     fprintf( outPtr, "%s%21d\n", "char", sizeof( char ) );
14     fprintf( outPtr, "%s%12d\n", "unsigned char",
15             sizeof( unsigned char ) );
16     fprintf( outPtr, "%s%16d\n", "short int", sizeof( short int ) );
17     fprintf( outPtr, "%s%7d\n", "unsigned short int",
18             sizeof( unsigned short int ) );
19     fprintf( outPtr, "%s%22d\n", "int", sizeof( int ) );
20     fprintf( outPtr, "%s%13d\n", "unsigned int",
21             sizeof( unsigned int ) );
22     fprintf( outPtr, "%s%17d\n", "long int", sizeof( long int ) );
23     fprintf( outPtr, "%s%8d\n", "unsigned long int",
24             sizeof( unsigned long int ) );
25     fprintf( outPtr, "%s%20d\n", "float", sizeof( float ) );
26     fprintf( outPtr, "%s%19d\n", "double", sizeof( double ) );
27     fprintf( outPtr, "%s%14d\n", "long double", sizeof( long double ) );
28
29     fclose( outPtr ); /* close file pointer */
30
31     return 0; /* indicate successful termination */
32
33 } /* end main */

```

Contents of datasize.dat

Data type	Size
char	1
unsigned char	1
short int	2
unsigned short int	2
int	4
unsigned int	4
long int	4
unsigned long int	4
float	4
double	8
long double	8

11.17 In Exercise 7.19, you wrote a software simulation of a computer that used a special machine language called Simpletron Machine Language (SML). In the simulation, each time you wanted to run an SML program, you entered the program into the simulator from the keyboard. If you made a mistake while typing the SML program, the simulator was restarted and the SML code was reentered. It would be nice to be able to read the SML program from a file rather than type it each time. This would reduce time and mistakes in preparing to run SML programs.

- a) Modify the simulator you wrote in Exercise 7.19 to read SML programs from a file specified by the user at the keyboard.
- b) After the Simpletron executes, it outputs the contents of its registers and memory on the screen. It would be nice to capture the output in a file, so modify the simulator to write its output to a file in addition to displaying the output on the screen.

ANS:

```

1  /* Exercise 11.17 Solution */
2  #include <stdio.h>
3
4  /* define commands */
5  #define SIZE 100
6  #define TRUE 1
7  #define FALSE 0
8  #define READ 10
9  #define WRITE 11
10 #define LOAD 20
11 #define STORE 21
12 #define ADD 30
13 #define SUBTRACT 31
14 #define DIVIDE 32
15 #define MULTIPLY 33
16 #define BRANCH 40
17 #define BRANCHNEG 41
18 #define BRANCHZERO 42
19 #define HALT 43
20
21 /* function prototype */
22 void load( int *loadMemory );
23 void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
24             int *opCodePtr, int *opPtr );
25 void dump( int *memory, int accumulator, int instructionCounter,
26           int instructionRegister, int operationCode,
27           int operand );
28 int validWord( int word );
29
30 int main()
31 {
32     int memory[ SIZE ]; /* define memory array */
33     int ac = 0;         /* accumulator */
34     int ic = 0;         /* instruction counter */
35     int opCode = 0;     /* operation code */
36     int op = 0;        /* operand */
37     int ir = 0;        /* instruction register */
38     int i;             /* counter */
39
40     /* clear memory */
41     for ( i = 0; i < SIZE; i++ ) {
42         memory[ i ] = 0;
43     } /* end for */
44
45     load( memory );
46     execute( memory, &ac, &ic, &ir, &opCode, &op );
47     dump( memory, ac, ic, ir, opCode, op );
48
49     return 0; /* indicate successful termination */
50 }
51 /* end main */
52
53 /* function loads instructions */
54 void load( int *loadMemory )
55 {

```

```

56     int instruction;        /* current instruction */
57     int i = 0;             /* indexing variable */
58     char fileName[ 36 ];  /* input file name */
59     FILE *finPtr;         /* input file pointer */
60
61     /* prompt user for input file name */
62     printf( "Enter input file: " );
63     scanf( "%s", fileName );
64
65     /* open input file */
66     if ( ( finPtr = fopen( fileName, "r" ) ) == NULL ) {
67         printf( "Data file was NOT opened.\n" );
68     } /* end if */
69     else { /* if file opened correctly */
70         fscanf( finPtr, "%d", &instruction );
71
72         /* while not end of file */
73         while ( !feof( finPtr ) ) {
74
75             /* check if instruction is valid */
76             while ( !validWord( instruction ) ) {
77                 printf( "***DATA ERROR.\n" );
78                 printf( "***check instructions in data file.\n" );
79                 fscanf( finPtr, "%d", &instruction );
80             } /* end while */
81
82             /* load instruction and read next instruction */
83             loadMemory[ i++ ] = instruction;
84             fscanf( finPtr, "%d", &instruction );
85         } /* end while */
86
87     } /* end else */
88
89     fclose( finPtr ); /* close file pointer */
90 } /* end function load */
91
92 /* carry out the commands */
93 void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
94             int *opCodePtr, int *opPtr )
95 {
96     int fatal = FALSE; /* fatal error flag */
97     int temp;          /* temporary holding space */
98
99     printf( "\n*****START SIMPLETRON EXECUTION*****\n\n" );
100
101     /* separate operation code and operand */
102     *irPtr = memory[ *icPtr ];
103     *opCodePtr = *irPtr / 100;
104     *opPtr = *irPtr % 100;
105
106     /* loop while command is not HALT or fatal */
107     while ( *opCodePtr != HALT && !fatal ) {
108
109         /* determine appropriate action */
110         switch ( *opCodePtr ) {
111
112             /* read data into location in memory */
113             case READ:
114                 printf( "Enter an integer: " );
115                 scanf( "%d", &temp );
116
117                 /* check for validity */
118                 while ( !validWord( temp ) ) {
119                     printf( "Number out of range. Please enter again: " );
120                     scanf( "%d", &temp );
121                 } /* end while */
122
123                 memory[ *opPtr ] = temp; /* write to memory */

```

```

124     ++( *icPtr );
125     break; /* exit switch */
126
127     /* write data from memory to screen */
128     case WRITE:
129         printf( "Contents of %02d: %d\n", *opPtr, memory[ *opPtr ] );
130         ++( *icPtr );
131         break; /* exit switch */
132
133     /* load data from memory into accumulator */
134     case LOAD:
135         *acPtr = memory[ *opPtr ];
136         ++( *icPtr );
137         break; /* exit switch */
138
139     /* store data from accumulator into memory */
140     case STORE:
141         memory[ *opPtr ] = *acPtr;
142         ++( *icPtr );
143         break; /* exit switch */
144
145     /* add data from memory to data in accumulator */
146     case ADD:
147         temp = *acPtr + memory[ *opPtr ];
148
149         /* check validity */
150         if ( !validWord( temp ) ) {
151             printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
152             printf( "*** Simpletron execution abnormally terminated ***\n" );
153             fatal = TRUE;
154         } /* end if */
155         else {
156             *acPtr = temp;
157             ++( *icPtr );
158         } /* end else */
159
160         break; /* exit switch */
161
162     /* subtract data in memory from data in accumulator */
163     case SUBTRACT:
164         temp = *acPtr - memory[ *opPtr ];
165
166         /* check validity */
167         if ( !validWord( temp ) ) {
168             printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
169             printf( "*** Simpletron execution abnormally terminated ***\n" );
170             fatal = TRUE;
171         } /* end if */
172         else {
173             *acPtr = temp;
174             ++( *icPtr );
175         } /* end else */
176
177         break; /* exit switch */
178
179     /* divide data in memory into data in accumulator */
180     case DIVIDE:
181
182         /* check for divide by zero error */
183         if ( memory[ *opPtr ] == 0 ) {
184             printf( "*** FATAL ERROR: Attempt to divide by zero      ***\n" );
185             printf( "*** Simpletron execution abnormally terminated ***\n" );
186             fatal = TRUE;
187         } /* end if */
188         else {
189             *acPtr /= memory[ *opPtr ];
190             ++( *icPtr );
191         } /* end else */

```

```

192         break; /* exit switch */
193
194     /* multiple data in memory by data in accumulator */
195     case MULTIPLY:
196         temp = *acPtr * memory[ *opPtr ];
197
198         /* check validity */
199         if ( !validWord( temp ) ) {
200             printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
201             printf( "*** Simpletron execution abnormally terminated ***\n" );
202             fatal = TRUE;
203         } /* end if */
204         else {
205             *acPtr = temp;
206             ++( *icPtr );
207         } /* end else */
208
209         break; /* exit switch */
210
211     /* branch to specific location in memory */
212     case BRANCH:
213         *icPtr = *opPtr;
214         break; /* exit switch */
215
216     /* branch to location in memory if accumulator is negative */
217     case BRANCHNEG:
218
219         /* if accumulator is negative */
220         if ( *acPtr < 0 ) {
221             *icPtr = *opPtr;
222         } /* end if */
223         else {
224             ++( *icPtr );
225         } /* end else */
226
227         break; /* exit switch */
228
229     /* branch to location in memory if accumulator is zero */
230     case BRANCHZERO:
231
232         /* if accumulator is zero */
233         if ( *acPtr == 0 ) {
234             *icPtr = *opPtr;
235         } /* end if */
236         else {
237             ++( *icPtr );
238         } /* end else */
239
240         break; /* exit switch */
241
242     default:
243         printf( "*** FATAL ERROR: Invalid opcode detected          ***\n" );
244         printf( "*** Simpletron execution abnormally terminated ***\n" );
245         fatal = TRUE;
246         break; /* exit switch */
247 } /* end switch */
248
249     /* separate next operation code and operand */
250     *irPtr = memory[ *icPtr ];
251     *opCodePtr = *irPtr / 100;
252     *opPtr = *irPtr % 100;
253 } /* end while */
254
255 printf( "\n*****END SIMPLETRON EXECUTION*****\n" );
256 } /* end function execute */
257
258

```

```

259 /* print out name and content of each register and memory */
260 void dump( const int *memory, int accumulator, int instructionCounter,
261           int instructionRegister, int operationCode, int operand )
262 {
263     int i; /* counter */
264     char outputFile[ 36 ]; /* output file name */
265     FILE *foutPtr; /* output file pointer */
266
267     /* prompt user for output file name */
268     printf( "Enter output file name: " );
269     scanf( "%s", outputFile );
270
271     /* open output file for writing */
272     if ( ( foutPtr = fopen( outputFile, "w" ) ) == NULL ) {
273         printf( "Output file was not opened.\n" );
274     } /* end if */
275     else { /* if file opened correctly, print headers to file */
276         fprintf( foutPtr, "\n%s\n%-23s%+05d\n%-23s%5.2d\n%-23s%+05d\n",
277                "REGISTERS:", "accumulator", accumulator,
278                "instructioncounter", instructionCounter,
279                "instructionregister", instructionRegister );
280         fprintf( foutPtr, "%-23s%5.2d\n%-23s%5.2d",
281                "operationcode", operationCode, "operand", operand );
282         fprintf( foutPtr, "\n\nMEMORY:\n" );
283     } /* end else */
284
285     /* print headers to screen */
286     printf( "\n%s\n%-23s%+05d\n%-23s%5.2d\n%-23s%+05d",
287            "REGISTERS:", "accumulator", accumulator, "instructioncounter",
288            instructionCounter, "instructionregister", instructionRegister);
289     printf( "\n%-23s%5.2d\n%-23s%5.2d",
290            "operationcode", operationCode, "operand", operand );
291     printf( "\n\nMEMORY:\n" );
292
293     /* print column headers */
294     for ( i = 0; i <= 9; i++ ) {
295         printf( "%5d ", i );
296         fprintf( foutPtr, "%5d ", i );
297     } /* end for */
298
299     /* print row headers and memory contents */
300     for ( i = 0; i < SIZE; i++ ) {
301
302         /* print in increments of 10 */
303         if ( i % 10 == 0 ) {
304             printf( "\n%2d ", i );
305             fprintf( foutPtr, "\n%2d ", i );
306         } /* end for */
307
308         printf( "%+05d ", memory[ i ] );
309         fprintf( foutPtr, "%+05d ", memory[ i ] );
310     } /* end for */
311
312     printf( "\n" );
313     fprintf( foutPtr, "\n" );
314     fclose( foutPtr ); /* close file pointer */
315 } /* end function dump */
316
317 /* function tests validity of word */
318 int validWord( int word )
319 {
320     return word >= -9999 && word <= 9999;
321 }
322 } /* end function validWord */

```

```

Enter input file: simple.in

*****START SIMPLETRON EXECUTION*****

Enter an integer: 5
Enter an integer: 2
Contents of 09: 7

*****END SIMPLETRON EXECUTION*****
Enter output file name: simple.out

REGISTERS:
accumulator          +0007
instructioncounter    06
instructionregister   +4300
operationcode         43
operand              00

MEMORY:
   0  +1007 +1008 +2007 +3008 +2109 +1109 +4300 +0005 +0002 +0007
  10  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  20  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  30  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  40  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  50  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  60  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  70  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  80  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
  90  +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

```

Contents of simple.in (a simple addition program)

```

1007
1008
2007
3008
2109
1109
4300
0000
0000
0000

```


12

Data Structures: Solutions

SOLUTIONS

12.6 Write a program that concatenates two linked lists of characters. The program should include function `concatenate` that takes pointers to both lists as arguments and concatenates the second list to the first list.

ANS:

```
1  /* Exercise 12.6 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* ListNode structure definition */
6  struct ListNode {
7      char data;           /* node data */
8      struct ListNode *nextPtr; /* pointer to next node */
9  }; /* end struct ListNode */
10
11 typedef struct ListNode ListNode;
12 typedef ListNode *ListNodePtr;
13
14 /* function prototypes */
15 void concatenate( ListNodePtr a, ListNodePtr b );
16 void insert( ListNodePtr *sPtr, char value );
17 void printList( ListNodePtr currentPtr );
18
19 int main()
20 {
21     ListNodePtr list1Ptr = NULL; /* pointer to first list */
22     ListNodePtr list2Ptr = NULL; /* pointer to second list */
23     char i; /* loop counter */
24
25     /* assign letters from A to C into first list */
26     for ( i = 'A'; i <= 'C'; i++ ) {
27         insert( &list1Ptr, i );
28     } /* end for */
29
30     printf( "List 1 is: " );
```

```

31     printList( list1Ptr );
32
33     /* assign letters from D to F into second list */
34     for ( i = 'D'; i <= 'F'; i++ ) {
35         insert( &list2Ptr, i );
36     } /* end for */
37
38     printf( "List 2 is: " );
39     printList( list2Ptr );
40
41     concatenate( list1Ptr, list2Ptr );
42     printf( "The concatenated list is: " );
43     printList( list1Ptr );
44
45     return 0; /* indicate successful termination */
46
47 } /* end main */
48
49 /* Concatenate two lists */
50 void concatenate( ListNodePtr a, ListNodePtr b )
51 {
52     ListNodePtr currentPtr; /* temporary pointer */
53
54     currentPtr = a; /* set currentPtr to first linked list */
55
56     /* while currentPtr does not equal NULL */
57     while( currentPtr->nextPtr != NULL ) {
58         currentPtr = currentPtr->nextPtr;
59     } /* end while */
60
61     currentPtr->nextPtr = b; /* concatenate both lists */
62 } /* end function concatenate */
63
64 /* Insert a new value into the list in sorted order */
65 void insert( ListNodePtr *sPtr, char value )
66 {
67     ListNodePtr newPtr; /* new node */
68     ListNodePtr previousPtr; /* previous node */
69     ListNodePtr currentPtr; /* current node */
70
71     /* dynamically allocate memory */
72     newPtr = malloc( sizeof( ListNode ) );
73
74     /* if newPtr does not equal NULL */
75     if ( newPtr ) {
76         newPtr->data = value;
77         newPtr->nextPtr = NULL;
78
79         previousPtr = NULL;
80         currentPtr = *sPtr; /* set currentPtr to start of list */
81
82         /* loop to find correct location in list */
83         while ( currentPtr != NULL && value > currentPtr->data ) {
84             previousPtr = currentPtr;
85             currentPtr = currentPtr->nextPtr;
86         } /* end while */
87
88         /* insert at beginning of list */
89         if ( previousPtr == NULL ) {
90             newPtr->nextPtr = *sPtr;
91             *sPtr = newPtr;
92         } /* end if */

```

```
93     else { /* insert node between previousPtr and currentPtr */
94         previousPtr->nextPtr = newPtr;
95         newPtr->nextPtr = currentPtr;
96     } /* end else */
97
98     } /* end if */
99     else {
100         printf( "%c not inserted. No memory available.\n", value );
101     } /* end else */
102
103 } /* end function insert */
104
105 /* Print the list */
106 void printList( ListNodePtr currentPtr )
107 {
108
109     /* if list is empty */
110     if ( !currentPtr ) {
111         printf( "List is empty.\n\n" );
112     } /* end if */
113     else {
114
115         /* loop while currentPtr does not equal NULL */
116         while ( currentPtr ) {
117             printf( "%c ", currentPtr->data );
118             currentPtr = currentPtr->nextPtr;
119         } /* end while */
120
121         printf( "\n\n" );
122     } /* end else */
123
124 } /* end function printList */
```

List 1 is: A B C *

List 2 is: D E F *

The concatenated list is: A B C D E F *

12.7 Write a program that merges two ordered lists of integers into a single ordered list of integers. Function merge should receive pointers to the first node of each of the lists to be merged and should return a pointer to the first node of the merged list.

ANS:

```

1  /* Exercise 12.7 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* ListNode structure definition */
6  struct ListNode {
7      int data;           /* node data */
8      struct ListNode *nextPtr; /* pointer to next node */
9  }; /* end struct ListNode */
10
11 typedef struct ListNode ListNode;
12 typedef ListNode *ListNodePtr;
13
14 /* function prototype */
15 void insert( ListNodePtr *sPtr, int value );
16 void printList( ListNodePtr currentPtr );
17 ListNodePtr merge( ListNodePtr a, ListNodePtr b );
18
19 int main()
20 {
21     ListNodePtr list1Ptr = NULL; /* pointer to first list */
22     ListNodePtr list2Ptr = NULL; /* pointer to second list */
23     ListNodePtr list3Ptr;       /* pointer to merged list */
24     int i; /* loop counter */
25
26     /* build first list */
27     for ( i = 2; i <= 10; i += 2 ) {
28         insert( &list1Ptr, i );
29     } /* end for */
30
31     printf( "List 1 is: " );
32     printList( list1Ptr );
33
34     /* build second list */
35     for ( i = 1; i <= 9; i += 2 ) {
36         insert( &list2Ptr, i );
37     } /* end for */
38
39     printf( "List 2 is: " );
40     printList( list2Ptr );
41
42     /* merge both lists and print results */
43     list3Ptr = merge( list1Ptr, list2Ptr );
44     printf( "The merged list is: " );
45     printList( list3Ptr );
46
47     return 0; /* indicate successful termination */
48 } /* end main */
49
50
51 /* Merge two lists of integers */
52 ListNodePtr merge( ListNodePtr a, ListNodePtr b )
53 {
54     ListNodePtr currentPtr1; /* pointer to first list */
55     ListNodePtr currentPtr2; /* pointer to second list */
56     ListNodePtr c = NULL;    /* pointer to merged list */
57

```

```

58     currentPtr1 = a; /* set currentPtr1 to first linked list */
59     currentPtr2 = b; /* set currentPtr2 to second linked list */
60
61     /* while currentPtr1 does not equal NULL */
62     while ( currentPtr1 != NULL ) {
63
64         /* compare currentPtr1 and currentPtr2, insert lesser node */
65         if ( currentPtr2 == NULL || currentPtr1->data <
66             currentPtr2->data ) {
67
68             /* insert currentPtr1 node */
69             insert( &c, currentPtr1->data );
70             currentPtr1 = currentPtr1->nextPtr;
71         } /* end if */
72         else {
73
74             /* insert currentPtr2 node */
75             insert( &c, currentPtr2->data );
76             currentPtr2 = currentPtr2->nextPtr;
77         } /* end else */
78
79     } /* end while */
80
81     /* insert any remaining nodes in currentPtr2 list */
82     while ( currentPtr2 != NULL ) {
83         insert( &c, currentPtr2->data );
84         currentPtr2 = currentPtr2->nextPtr;
85     } /* end while */
86
87     return c; /* return merged list */
88
89 } /* end function merge */
90
91 /* Insert a new value into the list in sorted order */
92 void insert( ListNodePtr *sPtr, int value )
93 {
94     ListNodePtr newPtr;      /* new node */
95     ListNodePtr previousPtr; /* previous node */
96     ListNodePtr currentPtr;  /* current node */
97
98     /* dynamically allocate memory */
99     newPtr = malloc( sizeof( ListNode ) );
100
101     /* if newPtr does not equal NULL */
102     if ( newPtr ) {
103         newPtr->data = value;
104         newPtr->nextPtr = NULL;
105
106         previousPtr = NULL;
107         currentPtr = *sPtr; /* set currentPtr to start of list */
108
109         /* loop to find correct location in list */
110         while ( currentPtr != NULL && value > currentPtr->data ) {
111             previousPtr = currentPtr;
112             currentPtr = currentPtr->nextPtr;
113         } /* end while */
114
115         /* insert at beginning of list */
116         if ( previousPtr == NULL ) {
117             newPtr->nextPtr = *sPtr;
118             *sPtr = newPtr;
119         } /* end if */

```

```
120     else { /* insert node between previousPtr and currentPtr */
121         previousPtr->nextPtr = newPtr;
122         newPtr->nextPtr = currentPtr;
123     } /* end else */
124
125 } /* end if */
126 else {
127     printf( "%c not inserted. No memory available.\n", value );
128 } /* end else */
129
130 } /* end function insert */
131
132 /* Print the list */
133 void printList( ListNodePtr currentPtr )
134 {
135     /* if list is empty */
136     if ( !currentPtr ) {
137         printf( "List is empty.\n\n" );
138     } /* end if */
139     else {
140         /* loop while currentPtr does not equal NULL */
141         while ( currentPtr ) {
142             printf( "%d ", currentPtr->data );
143             currentPtr = currentPtr->nextPtr;
144         } /* end while */
145
146         printf( "\n\n" );
147     } /* end else */
148 }
149
150 } /* end function printList */
```

List 1 is: 2 4 6 8 10 *

List 2 is: 1 3 5 7 9 *

The merged list is: 1 2 3 4 5 6 7 8 9 10 *

12.8 Write a program that inserts 25 random integers from 0 to 100 in order in a linked list. The program should calculate the sum of the elements and the floating-point average of the elements.

ANS: .

```

1  /* Exercise 12.8 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* ListNode structure definition */
7  typedef struct ListNode {
8      int data; /* node data */
9      struct ListNode *nextPtr; /* pointer to next node */
10 } ListNode; /* end struct ListNode */
11
12 typedef ListNode *ListNodePtr;
13
14 /* function prototypes */
15 int sumList( ListNodePtr a );
16 double averageList( ListNodePtr a );
17 void insert( ListNodePtr *sPtr, int value );
18 void printList( ListNodePtr currentPtr );
19
20 int main()
21 {
22     ListNodePtr listPtr = NULL; /* list pointer */
23     int i; /* loop counter */
24
25     srand( time( NULL ) ); /* randomize */
26
27     /* build list with random numbers from 0 to 100 */
28     for ( i = 1; i <= 25; i++ ) {
29         insert( &listPtr, rand() % 101 );
30     } /* end for */
31
32     printf( "The list is:\n" );
33     printList( listPtr );
34
35     /* calculate and display the sum and average of list values */
36     printf( "The sum is %d\n", sumList( listPtr ) );
37     printf( "The average is %f\n", averageList( listPtr ) );
38
39     return 0; /* indicate successful termination */
40
41 } /* end main */
42
43 /* Sum the integers in a list */
44 int sumList( ListNodePtr a )
45 {
46     ListNodePtr currentPtr; /* temporary pointer to list a */
47     int total = 0; /* sum of node values */
48
49     currentPtr = a; /* set currentPtr to list a */
50
51     /* loop through list */
52     while ( currentPtr != NULL ) {
53
54         /* add node value to total */
55         total += currentPtr->data;
56         currentPtr = currentPtr->nextPtr;
57     } /* end while */

```



```
58
59     return total;
60
61 } /* end function sumList */
62
63 /* Average the integers in a list */
64 double averageList( ListNodePtr a )
65 {
66     ListNodePtr currentPtr; /* temporary pointer to list a */
67     double total = 0.0; /* sum of node values */
68     int count = 0; /* number of nodes in list */
69
70     currentPtr = a; /* set currentPtr to list a */
71
72     /* loop through list */
73     while ( currentPtr != NULL ) {
74         ++count; /* increment count */
75         total += currentPtr->data; /* update total */
76         currentPtr = currentPtr->nextPtr;
77     } /* end while */
78
79     return total / count; /* return average */
80
81 } /* end function averageList */
82
83 /* Insert a new value into the list in sorted order */
84 void insert( ListNodePtr *sPtr, int value )
85 {
86     ListNodePtr newPtr; /* new node */
87     ListNodePtr previousPtr; /* previous node */
88     ListNodePtr currentPtr; /* current node */
89
90     /* dynamically allocate memory */
91     newPtr = malloc( sizeof( ListNode ) );
92
93     /* if newPtr does not equal NULL */
94     if ( newPtr ) {
95         newPtr->data = value;
96         newPtr->nextPtr = NULL;
97
98         previousPtr = NULL;
99         currentPtr = *sPtr; /* set currentPtr to start of list */
100
101         /* loop to find correct location in list */
102         while ( currentPtr != NULL && value > currentPtr->data ) {
103             previousPtr = currentPtr;
104             currentPtr = currentPtr->nextPtr;
105         } /* end while */
106
107         /* insert at beginning of list */
108         if ( previousPtr == NULL ) {
109             newPtr->nextPtr = *sPtr;
110             *sPtr = newPtr;
111         } /* end if */
112         else { /* insert node between previousPtr and currentPtr */
113             previousPtr->nextPtr = newPtr;
114             newPtr->nextPtr = currentPtr;
115         } /* end else */
116
117     } /* end if */
118     else {
```

```
119     printf( "%c not inserted. No memory available.\n", value );
120     } /* end else */
121
122 } /* end function insert */
123
124 /* Print the list */
125 void printList( ListNodePtr currentPtr )
126 {
127
128     /* if list is empty */
129     if ( !currentPtr ) {
130         printf( "List is empty.\n\n" );
131     } /* end if */
132     else {
133
134         /* loop while currentPtr does not equal NULL */
135         while ( currentPtr ) {
136             printf( "%d ", currentPtr->data );
137             currentPtr = currentPtr->nextPtr;
138         } /* end while */
139
140         printf( "\n\n" );
141     } /* end else */
142
143 } /* end function printList */
```

The list is:

6 12 14 20 27 31 31 34 37 38 56 59 63 66 72 73 73 76 77 79 88 94 95 96 97 *

The sum is 1414

The average is 56.560000

- 12.9 Write a program that creates a linked list of 10 characters, then creates a copy of the list in reverse order.
ANS:

```

1  /* Exercise 12.9 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* ListNode structure definition */
6  struct ListNode {
7      char data;           /* node data */
8      struct ListNode *nextPtr; /* pointer to next node */
9  }; /* end struct ListNode */
10
11 typedef struct ListNode ListNode;
12 typedef ListNode *ListNodePtr;
13
14 /* function prototypes */
15 ListNodePtr reverseList( ListNodePtr currentPtr );
16 void insert( ListNodePtr *sPtr, char value );
17 void printList( ListNodePtr currentPtr );
18 void push( ListNodePtr *topPtr, char info );
19
20 int main()
21 {
22     ListNodePtr listPtr = NULL; /* list pointer */
23     char i; /* loop counter */
24
25     /* build list with characters A to J */
26     for ( i = 'A'; i <= 'J'; i++ ) {
27         insert( &listPtr, i );
28     } /* end for */
29
30     printf( "The list is:\n" );
31     printList( listPtr );
32
33     /* reverse the list and display result */
34     printf( "The list in reverse is:\n" );
35     printList( reverseList( listPtr ) );
36
37     return 0; /* indicate successful termination */
38 } /* end main */
39
40
41 /* Create a list in the reverse order of the list argument */
42 ListNodePtr reverseList( ListNodePtr currentPtr )
43 {
44     ListNodePtr stack = NULL; /* pointer to reversed list */
45
46     /* loop through list currentPtr */
47     while ( currentPtr != NULL ) {
48
49         /* push current element on to stack */
50         push( &stack, currentPtr->data );
51         currentPtr = currentPtr->nextPtr;
52     } /* end while */
53
54     return stack; /* return reversed list */
55 } /* end function reverseList */
56
57

```

```
58 /* Insert a new value into the list in sorted order */
59 void insert( ListNodePtr *sPtr, char value )
60 {
61     ListNodePtr newPtr;      /* new node */
62     ListNodePtr previousPtr; /* previous node */
63     ListNodePtr currentPtr;  /* current node */
64
65     /* dynamically allocate memory */
66     newPtr = malloc( sizeof( ListNode ) );
67
68     /* if newPtr does not equal NULL */
69     if ( newPtr ) {
70         newPtr->data = value;
71         newPtr->nextPtr = NULL;
72
73         previousPtr = NULL;
74         currentPtr = *sPtr; /* set currentPtr to start of list */
75
76         /* loop to find correct location in list */
77         while ( currentPtr != NULL && value > currentPtr->data ) {
78             previousPtr = currentPtr;
79             currentPtr = currentPtr->nextPtr;
80         } /* end while */
81
82         /* insert at beginning of list */
83         if ( previousPtr == NULL ) {
84             newPtr->nextPtr = *sPtr;
85             *sPtr = newPtr;
86         } /* end if */
87         else { /* insert node between previousPtr and currentPtr */
88             previousPtr->nextPtr = newPtr;
89             newPtr->nextPtr = currentPtr;
90         } /* end else */
91
92     } /* end if */
93     else {
94         printf( "%c not inserted. No memory available.\n", value );
95     } /* end else */
96
97 } /* end function insert */
98
99 /* Insert a node at the stack top */
100 void push( ListNodePtr *topPtr, char info )
101 {
102     ListNodePtr newPtr; /* temporary node pointer */
103
104     /* dynamically allocate memory */
105     newPtr = malloc( sizeof( ListNode ) );
106
107     /* if memory was allocated, insert node at top of list */
108     if ( newPtr ) {
109         newPtr->data = info;
110         newPtr->nextPtr = *topPtr;
111         *topPtr = newPtr;
112     } /* end if */
113     else {
114         printf( "%c not inserted. No memory available.\n", info );
115     } /* end else */
116
117 } /* end function push */
118
```

```
119 /* Print the list */
120 void printList( ListNodePtr currentPtr )
121 {
122
123     /* if list is empty */
124     if ( !currentPtr ) {
125         printf( "List is empty.\n\n" );
126     } /* end if */
127     else {
128
129         /* loop while currentPtr does not equal NULL */
130         while ( currentPtr ) {
131             printf( "%c ", currentPtr->data );
132             currentPtr = currentPtr->nextPtr;
133         } /* end while */
134
135         printf( "\n\n" );
136     } /* end else */
137
138 } /* end function printList */
```

The list is:
A B C D E F G H I J *

The list in reverse is:
J I H G F E D C B A *

12.10 Write a program that inputs a line of text and uses a stack to print the line reversed.

ANS:

```

1  /* Exercise 12.10 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* stackNode structure definition */
6  struct stackNode {
7      char data;           /* node data */
8      struct stackNode *nextPtr; /* pointer to next node */
9  }; /* end struct stackNode */
10
11 typedef struct stackNode StackNode;
12 typedef StackNode *StackNodePtr;
13
14 /* function prototypes */
15 void push( StackNodePtr *topPtr, char info );
16 char pop( StackNodePtr *topPtr );
17 int isEmpty( StackNodePtr topPtr );
18
19 int main()
20 {
21     StackNodePtr stackPtr = NULL; /* points to the stack top */
22     char c; /* current character from text */
23
24     printf( "Enter a line of text:\n" );
25
26     /* read each letter with getchar and push on stack */
27     while ( ( c = getchar() ) != '\n' ) {
28         push( &stackPtr, c );
29     } /* end while */
30
31     printf( "\nThe line is reverse is:\n" );
32
33     /* while the stack is not empty, pop next character */
34     while ( !isEmpty( stackPtr ) ) {
35         printf( "%c", pop( &stackPtr ) );
36     } /* end while */
37
38     return 0; /* indicate successful termination */
39 } /* end main */
40
41
42 /* Insert a node at the stack top */
43 void push( StackNodePtr *topPtr, char info )
44 {
45     StackNodePtr newPtr; /* temporary node pointer */
46
47     /* dynamically allocate memory */
48     newPtr = malloc( sizeof( StackNode ) );
49
50     /* if memory was allocated, insert node at top of stack */
51     if ( newPtr ) {
52         newPtr->data = info;
53         newPtr->nextPtr = *topPtr;
54         *topPtr = newPtr;
55     } /* end if */
56     else {
57         printf( "%d not inserted. No memory available.\n", info );
58     } /* end else */

```

```
59
60 } /* end function push */
61
62 /* Remove a node from the stack top */
63 char pop( StackNodePtr *topPtr )
64 {
65     StackNodePtr tempPtr; /* temporary node pointer */
66     int popValue;         /* value of popped node */
67
68     tempPtr = *topPtr;
69     popValue = ( *topPtr )->data;
70     *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
71     free( tempPtr ); /* free memory */
72
73     return popValue; /* return value of popped node */
74
75 } /* end function pop */
76
77 /* Is the stack empty? */
78 int isEmpty( StackNodePtr topPtr )
79 {
80     return !topPtr; /* return NULL if stack is empty */
81
82 } /* end function isEmpty */
```

```
Enter a line of text:
this is a line of text
```

```
The line is reverse is:
txet fo enil a si siht
```

12.11 Write a program that uses a stack to determine if a string is a palindrome (i.e., the string is spelled identically backward and forward). The program should ignore spaces and punctuation.

ANS:

```

1  /* Exercise 12.11 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <ctype.h>
5
6  #define YES 1
7  #define NO 0
8
9  /* stackNode structure definition */
10 struct stackNode {
11     char data; /* node data */
12     struct stackNode *nextPtr; /* pointer to next node */
13 }; /* end struct stackNode */
14
15 typedef struct stackNode STACKNODE;
16 typedef STACKNODE *STACKNODEPTR;
17
18 /* function prototypes */
19 void push( STACKNODEPTR *topPtr, char info );
20 char pop( STACKNODEPTR *topPtr );
21 int isEmpty( STACKNODEPTR topPtr );
22
23 int main()
24 {
25     STACKNODEPTR stackPtr = NULL; /* points to the stack top */
26     char c; /* current character from text */
27     char line[ 50 ]; /* text from user */
28     char condensedLine[ 50 ]; /* text with only letters */
29     int i = 0; /* length of condensed line */
30     int j = 0; /* length of line */
31     int palindrome = YES; /* result of palindrome test */
32
33     printf( "Enter a line of text:\n" );
34
35     /* read each letter with getchar and add to line */
36     while ( ( c = getchar() ) != '\n' ) {
37         line[ j++ ] = c;
38
39         /* remove all spaces and punctuation */
40         if ( isalpha( c ) ) {
41             condensedLine[ i++ ] = tolower( c );
42             push( &stackPtr, tolower( c ) );
43         } /* end if */
44     } /* end while */
45
46     line[ j ] = '\0';
47
48     /* loop through condensedLine */
49     for ( j = 0; j < i; j++ ) {
50
51         /* if condensedLine does not equal stack */
52         if ( condensedLine[ j ] != pop( &stackPtr ) ) {
53             palindrome = NO;
54             break; /* exit loop */
55         } /* end if */
56     } /* end for */
57
58 } /* end main */

```



```

59
60     /* if text is a palindrome */
61     if ( palindrome ) {
62         printf( "\\\"%s\\\" is a palindrome\\n", line );
63     } /* end if */
64     else {
65         printf( "\\\"%s\\\" is not a palindrome\\n", line );
66     } /* end else */
67
68     return 0; /* indicate successful termination */
69
70 } /* end main */
71
72 /* Insert a node at the stack top */
73 void push( STACKNODEPTR *topPtr, char info )
74 {
75     STACKNODEPTR newPtr; /* temporary node pointer */
76
77     /* dynamically allocate memory */
78     newPtr = malloc( sizeof( STACKNODE ) );
79
80     /* if memory was allocated, insert node at top of stack */
81     if ( newPtr ) {
82         newPtr->data = info;
83         newPtr->nextPtr = *topPtr;
84         *topPtr = newPtr;
85     } /* end if */
86     else {
87         printf( "%d not inserted. No memory available.\\n", info );
88     } /* end else */
89
90 } /* end function push */
91
92 /* Remove a node from the stack top */
93 char pop( STACKNODEPTR *topPtr )
94 {
95     STACKNODEPTR tempPtr; /* temporary node pointer */
96     int popValue; /* value of popped node */
97
98     tempPtr = *topPtr;
99     popValue = ( *topPtr )->data;
100    *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
101    free( tempPtr ); /* free memory */
102
103    return popValue; /* return value of popped node */
104
105 } /* end function pop */
106
107 /* Is the stack empty? */
108 int isEmpty( STACKNODEPTR topPtr )
109 {
110     return !topPtr; /* return NULL if stack is empty */
111
112 } /* end function isEmpty */

```

```

Enter a line of text:
able was i ere i saw elba
"able was i ere i saw elba" is a palindrome

```

```
Enter a line of text:  
this is not a palindrome  
"this is not a palindrome" is not a palindrome
```

12.12 Stacks are used by compilers to help in the process of evaluating expressions and generating machine language code. In this and the next exercise, we investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Humans generally write expressions like $3 + 4$ and $7 / 9$ in which the operator (+ or / here) is written between its operands—this is called *infix notation*. Computers “prefer” *postfix notation* in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as $3\ 4\ +$ and $7\ 9\ /$, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation, and then evaluate the postfix version of the expression. Each of these algorithms requires only a single left-to-right pass of the expression. Each algorithm uses a stack in support of its operation, and in each the stack is used for a different purpose.

In this exercise, you will write a version of the infix-to-postfix conversion algorithm. In the next exercise, you will write a version of the postfix expression evaluation algorithm.

Write a program that converts an ordinary infix arithmetic expression (assume a valid expression is entered) with single digit integers such as

$$(6 + 2) * 5 - 8 / 4$$

to a postfix expression. The postfix version of the preceding infix expression is

$$6\ 2\ +\ 5\ *\ 8\ 4\ /\ -$$

The program should read the expression into character array `infix`, and use modified versions of the stack functions implemented in this chapter to help create the postfix expression in character array `postfix`. The algorithm for creating a postfix expression is as follows:

- 1) Push a left parenthesis '(' onto the stack.
- 2) Append a right parenthesis ')' to the end of `infix`.
- 3) While the stack is not empty, read `infix` from left to right and do the following:
 - If the current character in `infix` is a digit, copy it to the next element of `postfix`.
 - If the current character in `infix` is a left parenthesis, push it onto the stack.
 - If the current character in `infix` is an operator,
 - Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and insert the popped operators in `postfix`.
 - Push the current character in `infix` onto the stack.
 - If the current character in `infix` is a right parenthesis
 - Pop operators from the top of the stack and insert them in `postfix` until a left parenthesis is at the top of the stack.
 - Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

- + addition
- subtraction
- * multiplication
- / division
- ^ exponentiation
- % remainder

The stack should be maintained with the following declarations:

```
struct stackNode {
    char data;
    struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
```

The program should consist of `main` and eight other functions with the following function headers:

```
void convertToPostfix( char infix[], char postfix[] )
```

Convert the infix expression to postfix notation.

```
int isOperator( char c )
```

Determine if C is an operator.

```
int precedence( char operator1, char operator2 )
```

Determine if the precedence of operator1 is less than, equal to, or greater than the precedence of operator2. The function returns -1, 0 and 1, respectively.

```
void push( StackNodePtr *topPtr, char value )
```

Push a value on the stack.

```
char pop( StackNodePtr *topPtr )
```

Pop a value off the stack.

```
char stackTop( StackNodePtr topPtr )
```

Return the top value of the stack without popping the stack.

```
int isEmpty( StackNodePtr topPtr )
```

Determine if the stack is empty.

```
void printStack( StackNodePtr topPtr )
```

Print the stack.

ANS:

```

1  /* Exercise 12.12 Solution */
2  /* Infix to postfix conversion */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <ctype.h>
6  #include <string.h>
7
8  #define MAXCOLS 100
9
10 /* stackNode structure definition */
11 typedef struct stackNode {
12     char data;           /* node data */
13     struct stackNode *nextPtr; /* pointer to next node */
14 } STACKNODE; /* end struct stackNode */
15
16 typedef STACKNODE *STACKNODEPTR;
17
18 /* function prototypes */
19 void convertToPostfix( char inFix[], char postFix[] );
20 int isOperator( char c );
21 int precedence( char operator1, char operator2 );
22 void push( STACKNODEPTR *topPtr, char info );
23 char pop( STACKNODEPTR *topPtr );
24 char stackTop( STACKNODEPTR topPtr );
25 int isEmpty( STACKNODEPTR topPtr );
26 void printStack( STACKNODEPTR currentPtr );
27
28 int main()
29 {
30     char c;           /* current character from expression */
31     char inFix[ MAXCOLS ]; /* expression in infix notation */

```

```

32 char postfix[ MAXCOLS ]; /* expression in postfix notation */
33 int pos = 0; /* indexing variable */
34
35 printf( "Enter the infix expression.\n" );
36
37 /* read each character with getchar */
38 while ( ( c = getchar() ) != '\n' ) {
39
40     /* remove any spaces */
41     if ( c != ' ' ) {
42         infix[ pos++ ] = c;
43     } /* end if */
44
45 } /* end while */
46
47 infix[ pos ] = '\0';
48
49 /* print infix expression, convert to postfix and print */
50 printf( "%s\n%s\n", "The original infix expression is:", infix );
51 convertToPostfix( infix, postfix );
52 printf( "The expression in postfix notation is:\n%s\n", postfix );
53
54 return 0; /* indicate successful termination */
55
56 } /* end main */
57
58 /* convert infix expression to postfix notation */
59 void convertToPostfix( char infix[], char postfix[] )
60 {
61     STACKNODEPTR stackPtr = NULL; /* points to the stack top */
62     int i; /* loop counter */
63     int j; /* indexing variable */
64     int higher; /* operator flag */
65     char popValue; /* value of popped node */
66
67     /* push left parenthesis onto the stack */
68     push( &stackPtr, '(' );
69     printStack( stackPtr );
70
71     /* add a right parenthesis to infix */
72     strcat( infix, " )" );
73
74     /* convert the infix expression to postfix */
75     for ( i = 0, j = 0; stackTop( stackPtr ); i++ ) {
76
77         /* if current character is a digit */
78         if ( isdigit( infix[ i ] ) ) {
79             postfix[ j++ ] = infix[ i ];
80         } /* end if */
81
82         /* if character is left parenthesis, push on stack */
83         else if ( infix[ i ] == '(' ) {
84             push( &stackPtr, '(' );
85             printStack( stackPtr );
86         } /* end else if */
87
88         /* if character is an operator */
89         else if ( isOperator( infix[ i ] ) ) {
90             higher = 1; /* used to store value of precedence test */
91

```

```

92     /* loop while current operator does not have
93     the highest precedence */
94     while ( higher ) {
95
96         /* if the top of the stack is an operator */
97         if ( isOperator( stackTop( stackPtr ) ) ) {
98
99             /* compare precedence of operators */
100            if ( precedence( stackTop( stackPtr ), inFix[ i ] ) ) {
101                postFix[ j++ ] = pop( &stackPtr );
102                printStack( stackPtr );
103            } /* end if */
104            else {
105                higher = 0; /* reset flag */
106            } /* end else */
107
108        } /* end if */
109        else {
110            higher = 0; /* reset flag */
111        } /* end else */
112
113    } /* end while */
114
115    push( &stackPtr, inFix[ i ] );
116    printStack( stackPtr );
117 } /* end else if */
118
119 /* if character is a right parenthesis */
120 else if ( inFix[ i ] == ')' ) {
121
122     /* pop stack until popped value is a left parenthesis */
123     while ( ( popValue = pop( &stackPtr ) ) != '(' ) {
124         printStack( stackPtr );
125         postFix[ j++ ] = popValue;
126     } /* end while */
127
128     printStack( stackPtr );
129 } /* end else if */
130
131 } /* end for */
132
133 postFix[ j ] = '\0';
134 } /* end function convertToPostfix */
135
136 /* check if c is an operator */
137 int isOperator( char c )
138 {
139
140     /* if c is an operator return true */
141     if ( c == '+' || c == '-' || c == '*' || c == '/' || c == '^' ) {
142         return 1;
143     }
144     /* end if */
145     else { /* return false */
146         return 0;
147     }
148     /* end else */
149
150 } /* end function isOperator */
151

```

```

152 /* if the precedence of operator1 is >= operator2,
153     return 1 ( true ), else return 0 ( false ) */
154 int precedence( char operator1, char operator2 )
155 {
156
157     /* compare precedence of operator1 and operator2 */
158     if ( operator1 == '^' ) {
159         return 1;
160     } /* end if */
161     else if ( operator2 == '^' ) {
162         return 0;
163     } /* end else if */
164     else if ( operator1 == '*' || operator1 == '/' ) {
165         return 1;
166     } /* end else if */
167     else if ( operator1 == '+' || operator1 == '-' ) {
168
169         /* if operator2 is * or / than return true */
170         if ( operator2 == '*' || operator2 == '/' ) {
171             return 0;
172         } /* end if */
173         else {
174             return 1;
175         } /* end else */
176
177     } /* end else if */
178
179     return 0; /* default */
180 } /* end function precedence */
181
182
183 /* Insert a node at the stack top */
184 void push( STACKNODEPTR *topPtr, char info )
185 {
186     STACKNODEPTR newPtr; /* temporary node pointer */
187
188     /* dynamically allocate memory */
189     newPtr = malloc( sizeof( STACKNODE ) );
190
191     /* if memory was allocated, insert node at top of stack */
192     if ( newPtr ) {
193         newPtr->data = info;
194         newPtr->nextPtr = *topPtr;
195         *topPtr = newPtr;
196     } /* end if */
197     else {
198         printf( "%c not inserted. No memory available.\n", info );
199     } /* end else */
200 } /* end function push */
201
202
203 /* Remove a node from the stack top */
204 char pop( STACKNODEPTR *topPtr )
205 {
206     STACKNODEPTR tempPtr; /* temporary node pointer */
207     char popValue; /* value of popped node */
208
209     tempPtr = *topPtr;
210     popValue = ( *topPtr )->data;
211     *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
212     free( tempPtr ); /* free memory */

```

```
213
214     return popValue; /* return value of popped node */
215
216 } /* end function pop */
217
218 /* View the top element of the stack */
219 char stackTop( STACKNODEPTR topPtr )
220 {
221
222     /* if the stack is not empty */
223     if ( !isEmpty( topPtr ) ) {
224         return topPtr->data;
225     } /* end if */
226     else {
227         return 0;
228     } /* end else */
229
230 } /* end function stackTop */
231
232 /* Is the stack empty? */
233 int isEmpty( STACKNODEPTR topPtr )
234 {
235     return !topPtr; /* return NULL if stack is empty */
236
237 } /* end function isEmpty */
238
239 /* Print the stack */
240 void printStack( STACKNODEPTR currentPtr )
241 {
242
243     /* if the stack is empty */
244     if ( currentPtr == NULL ) {
245         printf( "The stack is empty.\n\n" );
246     } /* end if */
247     else { /* print stack */
248
249         /* loop through stack */
250         while ( currentPtr != NULL ) {
251             printf( "%c  ", currentPtr->data );
252             currentPtr = currentPtr->nextPtr;
253         } /* end while */
254
255         printf( "NULL\n" );
256     } /* end else */
257
258 } /* end function printStack */
```


Enter the infix expression.

1+(2*3-(4/5^6)*7)*8

The original infix expression is:

1+(2*3-(4/5^6)*7)*8

```
( NULL
+ ( NULL
( + ( NULL
* ( + ( NULL
( + ( NULL
- ( + ( NULL
( - ( + ( NULL
/ ( - ( + ( NULL
^ / ( - ( + ( NULL
/ ( - ( + ( NULL
( - ( + ( NULL
- ( + ( NULL
* - ( + ( NULL
- ( + ( NULL
( + ( NULL
+ ( NULL
* + ( NULL
+ ( NULL
( NULL
```

The stack is empty.

The expression in postfix notation is:

123*456^/7*-8*+

12.13 Write a program that evaluates a postfix expression (assume it is valid) such as

6 2 + 5 * 8 4 / -

The program should read a postfix expression consisting of digits and operators into a character array. Using modified versions of the stack functions implemented earlier in this chapter, the program should scan the expression and evaluate it. The algorithm is as follows:

- 1) Append the null character ('\0') to the end of the postfix expression. When the null character is encountered, no further processing is necessary.
- 2) While '\0' has not been encountered, read the expression from left to right.
 - If the current character is a digit,
 - Push its integer value onto the stack (the integer value of a digit character is its value in the computer's character set minus the value of '0' in the computer's character set).
 - Otherwise, if the current character is an *operator*,
 - Pop the two top elements of the stack into variables x and y.
 - Calculate $y \text{ operator } x$.
 - Push the result of the calculation onto the stack.
- 3) When the null character is encountered in the expression, pop the top value of the stack. This is the result of the postfix expression.

[*Note:* In 2) above, if the operator is '/', the top of the stack is 2, and the next element in the stack is 8, then pop 2 into x, pop 8 into y, evaluate $8 / 2$, and push the result, 4, back on the stack. This note also applies to operator '-'.] The arithmetic operations allowed in an expression are:

- + addition
- subtraction
- * multiplication
- / division
- ^ exponentiation
- % remainder]

The stack should be maintained with the following declarations:

```
struct stackNode {
    int data;
    struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
```

The program should consist of main and six other functions with the following function headers:

```
int evaluatePostfixExpression( char *expr )
    Evaluate the postfix expression.

int calculate( int op1, int op2, char operator )
    Evaluate the expression op1 operator op2.

void push( StackNodePtr *topPtr, int value )
    Push a value on the stack.

int pop( StackNodePtr *topPtr )
    Pop a value off the stack.

int isEmpty( StackNodePtr topPtr )
    Determine if the stack is empty.

void printStack( StackNodePtr topPtr )
```

Print the stack.

ANS:

```

1  /* Exercise 12.13 Solution */
2  /* Using a stack to evaluate an expression in postfix notation */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <ctype.h>
7  #include <math.h>
8
9  /* StackNode structure definition */
10 struct StackNode {
11     int data;           /* node data */
12     struct StackNode *nextPtr; /* pointer to next node */
13 }; /* end struct StackNode */
14
15 typedef struct StackNode StackNode;
16 typedef StackNode *StackNodePtr;
17
18 /* function prototypes */
19 int evaluatePostfixExpression( char *expr );
20 int calculate( int op1, int op2, char operator );
21 void push( StackNodePtr *topPtr, int info );
22 int pop( StackNodePtr *topPtr );
23 int isEmpty( StackNodePtr topPtr );
24 void printStack( StackNodePtr currentPtr );
25
26 int main()
27 {
28     char expression[ 100 ]; /* postfix expression */
29     char c;                 /* current character from expression */
30     int answer;             /* expression answer */
31     int i = 0;              /* indexing variable */
32
33     printf( "Enter a postfix expression:\n" );
34
35     /* read each character with getchar */
36     while ( ( c = getchar() ) != '\n' ) {
37
38         /* remove any spaces */
39         if ( c != ' ' ) {
40             expression[ i++ ] = c;
41         } /* end if */
42
43     } /* end while */
44
45     expression[ i ] = '\0';
46
47     /* calculate answer and print result */
48     answer = evaluatePostfixExpression( expression );
49     printf( "The value of the expression is: %d\n", answer );
50
51     return 0; /* indicate successful termination */
52 } /* end main */
53
54 /* evaluate the postfix expression */
55 int evaluatePostfixExpression( char *expr )
56 {
57     int i;                  /* loop counter */

```

```

59     int popVal1;           /* right value of current operation */
60     int popVal2;           /* left value of current operation */
61     StackNodePtr stackPtr = NULL; /* points to the stack top */
62     char c;                /* current character */
63
64     /* loop through expression */
65     for ( i = 0; ( c = expr[ i ] ) != '\0'; i++ ) {
66
67         /* if character is a digit, push it on stack */
68         if ( isdigit( c ) ) {
69             push( &stackPtr, c - '0' );
70             printStack( stackPtr );
71         } /* end if */
72         else { /* calculate current operation */
73             popVal2 = pop( &stackPtr );
74             printStack( stackPtr );
75             popVal1 = pop( &stackPtr );
76             printStack( stackPtr );
77
78             /* calculate answer and push on stack */
79             push( &stackPtr, calculate( popVal1, popVal2, c ) );
80             printStack( stackPtr );
81         } /* end else */
82     } /* end for */
83
84     return pop( &stackPtr ); /* return final answer */
85 } /* end function evaluatePostfixExpression */
86
87 /* evaluate the expression op1 operator op2 */
88 int calculate( int op1, int op2, char operator )
89 {
90     /* use correct operator to calculate answer */
91     switch( operator ) {
92
93         case '+': /* addition */
94             return op1 + op2;
95
96         case '-': /* subtraction */
97             return op1 - op2;
98
99         case '*': /* multiplication */
100            return op1 * op2;
101
102         case '/': /* division */
103            return op1 / op2;
104
105         case '^': /* exponentiation */
106            return pow( op1, op2 );
107     } /* end switch */
108     return 0; /* default */
109 } /* end function calculate */
110
111 /* Insert a node at the stack top */
112 void push( StackNodePtr *topPtr, int info )
113 {
114     StackNodePtr newPtr; /* temporary node pointer */

```

```
120
121     /* dynamically allocate memory */
122     newPtr = malloc( sizeof( StackNode ) );
123
124     /* if memory was allocated, insert node at top of stack */
125     if ( newPtr ) {
126         newPtr->data = info;
127         newPtr->nextPtr = *topPtr;
128         *topPtr = newPtr;
129     } /* end if */
130     else {
131         printf( "%d not inserted. No memory available.\n", info );
132     } /* end else */
133
134 } /* end function push */
135
136 /* Remove a node from the stack top */
137 int pop( StackNodePtr *topPtr )
138 {
139     StackNodePtr tempPtr; /* temporary node pointer */
140     int popValue;         /* value of popped node */
141
142     tempPtr = *topPtr;
143     popValue = ( *topPtr )->data;
144     *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
145     free( tempPtr ); /* free memory */
146
147     return popValue; /* return value of popped node */
148 } /* end function pop */
149
150
151 /* Is the stack empty? */
152 int isEmpty( StackNodePtr topPtr )
153 {
154     return !topPtr; /* return NULL if stack is empty */
155 } /* end function isEmpty */
156
157
158 /* Print the stack */
159 void printStack( StackNodePtr currentPtr )
160 {
161
162     /* loop through stack */
163     while ( currentPtr != NULL ) {
164         printf( "%d ", currentPtr->data );
165         currentPtr = currentPtr->nextPtr;
166     } /* end while */
167
168     printf( "NULL\n" );
169 } /* end function printStack */
```

```
Enter a postfix expression:
```

```
123*456^/7*-8*+
```

```
1 NULL
```

```
2 1 NULL
```

```
3 2 1 NULL
```

```
2 1 NULL
```

```
1 NULL
```

```
6 1 NULL
```

```
4 6 1 NULL
```

```
5 4 6 1 NULL
```

```
6 5 4 6 1 NULL
```

```
5 4 6 1 NULL
```

```
4 6 1 NULL
```

```
15625 4 6 1 NULL
```

```
4 6 1 NULL
```

```
6 1 NULL
```

```
0 6 1 NULL
```

```
7 0 6 1 NULL
```

```
0 6 1 NULL
```

```
6 1 NULL
```

```
0 6 1 NULL
```

```
6 1 NULL
```

```
1 NULL
```

```
6 1 NULL
```

```
8 6 1 NULL
```

```
6 1 NULL
```

```
1 NULL
```

```
48 1 NULL
```

```
1 NULL
```

```
NULL
```

```
49 NULL
```

```
The value of the expression is: 49
```

12.14 Modify the postfix evaluator program of Exercise 12.13 so that it can process integer operands larger than 9.

12.15 (*Supermarket Simulation*) Write a program that simulates a check-out line at a supermarket. The line is a queue. Customers arrive in random integer intervals of 1 to 4 minutes. Also, each customer is serviced in random integer intervals of 1 to 4 minutes. Obviously, the rates need to be balanced. If the average arrival rate is larger than the average service rate, the queue will grow infinitely. Even with balanced rates, randomness can still cause long lines. Run the supermarket simulation for a 12-hour day (720 minutes) using the following algorithm:

- Choose a random integer between 1 and 4 to determine the minute at which the first customer arrives.
- 2) At the first customer's arrival time:
 - Determine customer's service time (random integer from 1 to 4);
 - Begin servicing the customer;
 - Schedule arrival time of next customer (random integer 1 to 4 added to the current time).
- 3) For each minute of the day:
 - If the next customer arrives,
 - Say so;
 - Enqueue the customer;
 - Schedule the arrival time of the next customer;
 - If service was completed for the last customer;
 - Say so;
 - Dequeue next customer to be serviced;
 - Determine customer's service completion time
(random integer from 1 to 4 added to the current time).

Now run your simulation for 720 minutes and answer each of the following:

- a) What is the maximum number of customers in the queue at any time?
- b) What is the longest wait any one customer experienced?
- c) What happens if the arrival interval is changed from 1 to 4 minutes to 1 to 3 minutes?

12.16 Modify the program of Fig. 12.19 to allow the binary tree to contain duplicate values.

ANS:

```

1  /* Exercise 12.16 Solution */
2  /* This is a modification of figure 12.19 */
3  /* Only function insertNode has been modified */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  /* TreeNode structure definition */
9  struct TreeNode {
10     struct TreeNode *leftPtr; /* pointer to left subtree */
11     int data; /* node data */
12     struct TreeNode *rightPtr; /* pointer to right subtree */
13 }; /* end struct TreeNode */
14
15 typedef struct TreeNode TreeNode;
16 typedef TreeNode *TreeNodePtr;
17
18 /* function prototypes */
19 void insertNode( TreeNodePtr *treePtr, int value );
20 void inOrder( TreeNodePtr treePtr );
21 void preOrder( TreeNodePtr treePtr );
22 void postOrder( TreeNodePtr treePtr );
23
24 int main()
25 {
26     int i; /* loop counter */
27     int item; /* random value to insert in tree */
28     TreeNodePtr rootPtr = NULL; /* points to the tree root */
29
30     srand( time( NULL ) ); /* randomize */
31     printf( "The numbers being placed in the tree are:\n" );
32
33     /* insert random values between 1 and 15 in the tree */
34     for ( i = 1; i <= 10; i++ ) {
35         item = rand() % 15;
36         printf( "%3d", item );
37         insertNode( &rootPtr, item );
38     } /* end for */
39
40     /* traverse the tree preorder */
41     printf( "\n\nThe preorder traversal is:\n" );
42     preOrder( rootPtr );
43
44     /* traverse the tree inorder */
45     printf( "\n\nThe inorder traversal is:\n" );
46     inOrder( rootPtr );
47
48     /* traverse the tree postorder */
49     printf( "\n\nThe postorder traversal is:\n" );
50     postOrder( rootPtr );
51
52     return 0; /* indicate successful termination */
53
54 } /* end main */
55
56 /* insert a node into the tree */
57 void insertNode( TreeNodePtr *treePtr, int value )
58 {

```



```
59
60  /* if treePtr is NULL */
61  if ( !*treePtr ) {
62
63      /* dynamically allocate memory */
64      *treePtr = malloc( sizeof( TreeNode ) );
65
66      /* if memory was allocated, insert node */
67      if ( *treePtr ) {
68          ( *treePtr )->data = value;
69          ( *treePtr )->leftPtr = NULL;
70          ( *treePtr )->rightPtr = NULL;
71      } /* end if */
72      else {
73          printf( "%d not inserted. No memory available.\n", value );
74      } /* end else */
75
76      return;
77  } /* end if */
78  else { /* recursively call insertNode */
79
80      /* insert node in left subtree */
81      if ( value <= ( *treePtr )->data ) {
82          insertNode( &(amp; ( *treePtr )->leftPtr ), value );
83      } /* end if */
84      else { /* insert node in right subtree */
85          insertNode( &(amp; ( *treePtr )->rightPtr ), value );
86      } /* end else */
87
88      } /* end else */
89
90  } /* end function insertNode */
91
92  /* traverse the tree inorder */
93  void inOrder( TreeNodePtr treePtr )
94  {
95
96      /* traverse left subtree, print node, traverse right subtree */
97      if ( treePtr ) {
98          inOrder( treePtr->leftPtr );
99          printf( "%3d", treePtr->data );
100         inOrder( treePtr->rightPtr );
101     } /* end if */
102 } /* end function inOrder */
103
104 /* traverse the tree preorder */
105 void preOrder( TreeNodePtr treePtr )
106 {
107
108     /* print node, traverse left subtree, traverse right subtree */
109     if ( treePtr ) {
110         printf( "%3d", treePtr->data );
111         preOrder( treePtr->leftPtr );
112         preOrder( treePtr->rightPtr );
113     } /* end if */
114 } /* end function preOrder */
115
116 } /* end function preOrder */
117
```

```
118 /* traverse the tree postorder */
119 void postOrder( TreeNodePtr treePtr )
120 {
121
122     /* traverse left subtree, traverse right subtree, print node */
123     if ( treePtr ) {
124         postOrder( treePtr->leftPtr );
125         postOrder( treePtr->rightPtr );
126         printf( "%3d", treePtr->data );
127     } /* end if */
128
129 } /* end function postOrder */
```

The numbers being placed in the tree are:

14 3 7 7 3 12 7 11 1 2

The preorder traversal is:

14 3 3 1 2 7 7 7 12 11

The inorder traversal is:

1 2 3 3 7 7 7 11 12 14

The postorder traversal is:

2 1 3 7 7 11 12 7 3 14

12.17 Write a program based on the program of Fig. 12.19 that inputs a line of text, tokenizes the sentence into separate words, inserts the words in a binary search tree, and prints the inorder, preorder, and postorder traversals of the tree.

[Hint: Read the line of text into an array. Use strtok to tokenize the text. When a token is found, create a new node for the tree, assign the pointer returned by strtok to member string of the new node, and insert the node in the tree.]

ANS:

```

1  /* Exercise 12.17 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  /* TreeNode structure definition */
7  struct TreeNode {
8      struct TreeNode *leftPtr; /* pointer to left subtree */
9      char *token; /* node data */
10     struct TreeNode *rightPtr; /* pointer to right subtree */
11 }; /* end struct TreeNode */
12
13 typedef struct TreeNode TreeNode;
14 typedef TreeNode *TreeNodePtr;
15
16 /* function prototypes */
17 void insertNode( TreeNodePtr *treePtr, char *tokenPtr );
18 void inOrder( TreeNodePtr treePtr );
19 void preOrder( TreeNodePtr treePtr );
20 void postOrder( TreeNodePtr treePtr );
21
22 int main()
23 {
24     TreeNodePtr rootPtr = NULL; /* points to the tree root */
25     char sentence[ 80 ]; /* text from user */
26     char *tokenPtr; /* pointer to current token */
27
28     /* prompt user and read a sentence */
29     printf( "Enter a sentence:\n" );
30     gets( sentence );
31
32     /* tokenize the sentence */
33     tokenPtr = strtok( sentence, " " );
34
35     /* insert the tokens in the tree */
36     while ( tokenPtr ) {
37         insertNode( &rootPtr, tokenPtr );
38         tokenPtr = strtok( NULL, " " );
39     } /* end while */
40
41     /* traverse the tree preorder */
42     printf( "\nThe preorder traversal is:\n" );
43     preOrder( rootPtr );
44
45     /* traverse the tree inorder */
46     printf( "\n\nThe inorder traversal is:\n" );
47     inOrder( rootPtr );
48
49     /* traverse the tree postorder */
50     printf( "\n\nThe postorder traversal is:\n" );
51     postOrder( rootPtr );
52
53     return 0; /* indicate successful termination */
54 } /* end main */

```

```
56
57 /* insert a node into the tree */
58 void insertNode( TreeNodePtr *treePtr, char *tokenPtr )
59 {
60
61     /* if treePtr is NULL */
62     if ( !*treePtr ) {
63
64         /* dynamically allocate memory */
65         *treePtr = malloc( sizeof( TreeNode ) );
66
67         /* if memory was allocated, insert node */
68         if ( *treePtr ) {
69             ( *treePtr )->token = tokenPtr;
70             ( *treePtr )->leftPtr = NULL;
71             ( *treePtr )->rightPtr = NULL;
72         } /* end if */
73         else {
74             printf( "\"%s\" not inserted. No memory available.\n",
75                 tokenPtr );
76         } /* end else */
77
78         return;
79     } /* end if */
80     else { /* recursively call insertNode */
81
82         /* insert node in left subtree */
83         if ( strcmp( tokenPtr, ( *treePtr )->token ) <= 0 ) {
84             insertNode( &( ( *treePtr )->leftPtr ), tokenPtr );
85         } /* end if */
86         else { /* insert node in right subtree */
87             insertNode( &( ( *treePtr )->rightPtr ), tokenPtr );
88         } /* end else */
89
90     } /* end else */
91
92 } /* end function insertNode */
93
94 /* traverse the tree inorder */
95 void inOrder( TreeNodePtr treePtr )
96 {
97
98     /* traverse left subtree, print node, traverse right subtree */
99     if ( treePtr ) {
100         inOrder( treePtr->leftPtr );
101         printf( "%s ", treePtr->token );
102         inOrder( treePtr->rightPtr );
103     } /* end if */
104
105 } /* end function inOrder */
106
107 /* traverse the tree preorder */
108 void preOrder( TreeNodePtr treePtr )
109 {
110
111     /* print node, traverse left subtree, traverse right subtree */
112     if ( treePtr ) {
113         printf( "%s ", treePtr->token );
114         preOrder( treePtr->leftPtr );
115         preOrder( treePtr->rightPtr );
116     } /* end if */
```

```

117 } /* end function preOrder */
118 } /* end function preOrder */
119
120 /* traverse the tree postorder */
121 void postOrder( TreeNodePtr treePtr )
122 {
123
124     /* traverse left subtree, traverse right subtree, print node */
125     if ( treePtr ) {
126         postOrder( treePtr->leftPtr );
127         postOrder( treePtr->rightPtr );
128         printf( "%s ", treePtr->token );
129     } /* end if */
130
131 } /* end function postOrder */

```

```

Enter a sentence:
this program inserts strings of different lengths in a tree

The preorder traversal is:
this program inserts different a in of lengths strings tree

The inorder traversal is:
a different in inserts lengths of program strings this tree

The postorder traversal is:
a in different lengths of inserts strings program tree this

```

12.18 In this chapter, we saw that duplicate elimination is straightforward when creating a binary search tree. Describe how you would perform duplicate elimination using only a single subscripted array. Compare the performance of array-based duplicate elimination with the performance of binary-search-tree-based duplicate elimination.

ANS: Using a single subscripted array, it is necessary to compare each value to be inserted in the array with all the array elements until a match is found or until it is determined that there is not a duplicate value in the array. If there is not a duplicate, the value can be inserted in the array. On average, half the array elements must be searched when the value is a duplicate and all the array elements must be searched when the value is not a duplicate. The binary search tree only compares the value to be inserted with the values in its path down the tree. If a leaf node is reached, and the value does not match the value in the leaf node, the value can be inserted. Otherwise the value can be discarded.

12.19 Write a function `depth` that receives a binary tree and determines how many levels it has.

12.20 (*Recursively Print a List Backwards*) Write a function `printListBackwards` that recursively outputs the items in a list in reverse order. Use your function in a test program that creates a sorted list of integers and prints the list in reverse order.

ANS:

```

1  /* Exercise 12.20 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* ListNode structure definition */
6  struct ListNode {
7      int data; /* node data */
8      struct ListNode *nextPtr; /* pointer to next node */
9  }; /* end struct ListNode */
10
11 typedef struct ListNode ListNode;
12 typedef ListNode *ListNodePtr;
13
14 /* function prototype */
15 void printList( ListNodePtr currentPtr );

```

```

16 void printListBackwards( ListNodePtr currentPtr );
17 void insertItem( ListNodePtr *sPtr, int value );
18
19 int main()
20 {
21     ListNodePtr startPtr = NULL; /* list pointer */
22     int item; /* loop counter */
23
24     /* insert integers into list */
25     for ( item = 1; item < 11; item++ ) {
26         insertItem( &startPtr, item );
27     } /* end for */
28
29     printList( startPtr );
30     printf( "\n" );
31     printListBackwards( startPtr );
32
33     return 0; /* indicate successful termination */
34
35 } /* end main */
36
37 /* Insert a new value into the list in sorted order */
38 void insertItem( ListNodePtr *sPtr, int value )
39 {
40     ListNodePtr newPtr; /* new node */
41     ListNodePtr previousPtr; /* previous node */
42     ListNodePtr currentPtr; /* current node */
43
44     /* dynamically allocate memory */
45     newPtr = malloc( sizeof( ListNode ) );
46
47     /* if newPtr does not equal NULL */
48     if ( newPtr ) {
49         newPtr->data = value;
50         newPtr->nextPtr = NULL;
51
52         previousPtr = NULL;
53         currentPtr = *sPtr; /* set currentPtr to start of list */
54
55         /* loop to find correct location in list */
56         while ( currentPtr != NULL && value > currentPtr->data ) {
57             previousPtr = currentPtr;
58             currentPtr = currentPtr->nextPtr;
59         } /* end while */
60
61         /* insert at beginning of list */
62         if ( previousPtr == NULL ) {
63             newPtr->nextPtr = *sPtr;
64             *sPtr = newPtr;
65         } /* end if */
66         else { /* insert node between previousPtr and currentPtr */
67             previousPtr->nextPtr = newPtr;
68             newPtr->nextPtr = currentPtr;
69         } /* end else */
70
71     } /* end if */
72     else {
73         printf( "%c not inserted. No memory available.\n", value );
74     } /* end else */
75
76 } /* end function insertItem */

```

```
77
78 /* Print the list */
79 void printList( ListNodePtr currentPtr )
80 {
81
82     /* if list is empty */
83     if ( !currentPtr ) {
84         printf( "List is empty.\n\n" );
85     } /* end if */
86     else {
87
88         /* loop while currentPtr does not equal NULL */
89         while ( currentPtr ) {
90             printf( "%d ", currentPtr->data );
91             currentPtr = currentPtr->nextPtr;
92         } /* end while */
93
94         printf( "\n\n" );
95     } /* end else */
96 } /* end function printList */
97
98
99 /* Print the list recursively backwards */
100 void printListBackwards( ListNodePtr currentPtr )
101 {
102
103     /* if at end of list */
104     if ( currentPtr == NULL ) {
105         printf( "The list reversed is:\n" );
106     } /* end if */
107     else { /* recursive call */
108         printListBackwards( currentPtr->nextPtr );
109         printf( "%d ", currentPtr->data );
110     } /* end else */
111 }
112 } /* end function printListBackwards */
```

```
the list is:
1 2 3 4 5 6 7 8 9 10

The list reversed is:
10 9 8 7 6 5 4 3 2 1
```

12.21 (*Recursively Search a List*) Write a function `searchList` that recursively searches a linked list for a specified value. The function should return a pointer to the value if it is found; otherwise, `NULL` should be returned. Use your function in a test program that creates a list of integers. The program should prompt the user for a value to locate in the list.

ANS:

```

1  /* Exercise 12.21 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* ListNode structure definition */
6  struct ListNode {
7      int data;           /* node data */
8      struct ListNode *nextPtr; /* pointer to next node */
9  }; /* end struct ListNode */
10
11 typedef struct ListNode ListNode;
12 typedef ListNode *ListNodePtr;
13
14 /* function prototypes */
15 void insertItem( ListNodePtr *sPtr, int value );
16 void printList( ListNodePtr currentPtr );
17 void instructions( void );
18 ListNodePtr searchList( ListNodePtr currentPtr, const int key );
19
20 int main()
21 {
22     ListNodePtr startPtr = NULL; /* list pointer */
23     ListNodePtr searchResultPtr; /* pointer to search result */
24     int choice;                 /* user's menu choice */
25     int item;                   /* value to insert into list */
26     int searchKey;              /* value to search for in list */
27
28     instructions(); /* display the menu */
29     printf( "? " );
30     scanf( "%d", &choice );
31
32     /* while user does not choose 3 */
33     while ( choice != 3 ) {
34
35         /* determine user's choice */
36         switch ( choice ) {
37
38             /* insert an integer into the list */
39             case 1:
40
41                 /* prompt user and read integer */
42                 printf( "Enter an integer: " );
43                 scanf( "\n%d", &item );
44
45                 /* insert integer and print list */
46                 insertItem( &startPtr, item );
47                 printList( startPtr );
48                 break; /* exit switch */
49
50             /* search for given integer */
51             case 2:
52
53                 /* prompt user and read integer */
54                 printf( "Enter integer to recursively search for: " );
55                 scanf( "%d", &searchKey );
56

```



```

57     searchResultPtr = searchList( startPtr, searchKey );
58
59     /* if searchKey not found */
60     if ( searchResultPtr == NULL ) {
61         printf( "%d is not in the list.\n\n", searchKey );
62     } /* end if */
63     else { /* if searchKey was found */
64         printf( "%d is in the list.\n\n",
65             searchResultPtr->data );
66     } /* end else */
67
68     break; /* exit switch */
69
70     /* default case */
71     default:
72         printf( "Invalid choice.\n\n" );
73         instructions();
74         break; /* exit switch */
75 } /* end switch */
76
77 printf( "? " );
78 scanf( "%d", &choice ); /* get next choice */
79 } /* end while */
80
81 printf( "End of run.\n" );
82
83 return 0; /* indicate successful termination */
84
85 } /* end main */
86
87 /* Print the instructions */
88 void instructions( void )
89 {
90     printf( "Enter your choice:\n"
91         " 1 to insertItem an element into the list.\n"
92         " 2 to recursively search list for an element.\n"
93         " 3 to end.\n" );
94 } /* end function instructions */
95
96 /* Insert a new value into the list in sorted order */
97 void insertItem( ListNodePtr *sPtr, int value )
98 {
99     ListNodePtr newPtr; /* new node */
100    ListNodePtr previousPtr; /* previous node */
101    ListNodePtr currentPtr; /* current node */
102
103    /* dynamically allocate memory */
104    newPtr = malloc( sizeof( ListNode ) );
105
106    /* if newPtr does not equal NULL */
107    if ( newPtr ) {
108        newPtr->data = value;
109        newPtr->nextPtr = NULL;
110
111        previousPtr = NULL;
112        currentPtr = *sPtr; /* set currentPtr to start of list */
113
114        /* loop to find correct location in list */
115        while ( currentPtr != NULL && value > currentPtr->data ) {
116            previousPtr = currentPtr;
117            currentPtr = currentPtr->nextPtr;
118        } /* end while */

```

```
119
120     /* insert at beginning of list */
121     if ( previousPtr == NULL ) {
122         newPtr->nextPtr = *sPtr;
123         *sPtr = newPtr;
124     } /* end if */
125     else { /* insert node between previousPtr and currentPtr */
126         previousPtr->nextPtr = newPtr;
127         newPtr->nextPtr = currentPtr;
128     } /* end else */
129
130 } /* end if */
131 else {
132     printf( "%c not inserted. No memory available.\n", value );
133 } /* end else */
134
135 } /* end function insertItem */
136
137 /* Print the list */
138 void printList( ListNodePtr currentPtr )
139 {
140
141     /* if list is empty */
142     if ( !currentPtr ) {
143         printf( "List is empty.\n\n" );
144     } /* end if */
145     else {
146
147         /* loop while currentPtr does not equal NULL */
148         while ( currentPtr ) {
149             printf( "%d --> ", currentPtr->data );
150             currentPtr = currentPtr->nextPtr;
151         } /* end while */
152
153         printf( "NULL\n\n" );
154     } /* end else */
155
156 } /* end function printList */
157
158 /* search for key in list */
159 ListNodePtr searchList( ListNodePtr currentPtr, const int key )
160 {
161
162     /* if currentPtr is at end of list */
163     if ( currentPtr == NULL ) {
164         return NULL; /* key not found */
165     } /* end if */
166     else if ( currentPtr->data == key ) {
167         return currentPtr; /* key found */
168     } /* end else if */
169     else {
170         searchList( currentPtr->nextPtr, key ); /* keep searching */
171     } /* end else */
172
173 } /* end function ListNodePtr */
```

```

Enter your choice:
  1 to insertItem an element into the list.
  2 to recursively search list for an element
  3 to end.
? 1
Enter an integer: 7
The list is:
7 --> NULL

? 1
Enter an integer: 99
The list is:
7 --> 99 --> NULL

? 1
Enter an integer: 56
The list is:
7 --> 56 --> 99 --> NULL

? 1
Enter an integer: 73
The list is:
7 --> 56 --> 73 --> 99 --> NULL

? 2
Enter integer to recursively search for: 7
7 is in the list.

? 2
Enter integer to recursively search for: 55
55 is not in the list.

? 2
Enter integer to recursively search for: 99
99 is in the list.

? 3
End of run.

```

12.22 (*Binary Tree Delete*) In this exercise, we discuss deleting items from binary search trees. The deletion algorithm is not as straightforward as the insertion algorithm. There are three cases that are encountered when deleting an item—the item is contained in a leaf node (i.e., it has no children), the item is contained in a node that has one child, or the item is contained in a node that has two children.

If the item to be deleted is contained in a leaf node, the node is deleted and the pointer in the parent node is set to NULL.

If the item to be deleted is contained in a node with one child, the pointer in the parent node is set to point to the child node and the node containing the data item is deleted. This causes the child node to take the place of the deleted node in the tree.

The last case is the most difficult. When a node with two children is deleted, another node must take its place. However, the pointer in the parent node cannot simply be assigned to point to one of the children of the node to be deleted. In most cases, the resulting binary search tree would not adhere to the following characteristic of binary search trees: *The values in any left subtree are less than the value in the parent node, and the values in any right subtree are greater than the value in the parent node.*

Which node is used as a *replacement node* to maintain this characteristic? Either the node containing the largest value in the tree less than the value in the node being deleted, or the node containing the smallest value in the tree greater than the value in the node being deleted. Let us consider the node with the smaller value. In a binary search tree, the largest value less than a parent's value is located in the left subtree of the parent node and is guaranteed to be contained in the rightmost node of the subtree. This node is located by walking down the left subtree to the right until the pointer to the right child of the current node is NULL. We are now pointing to the replacement node which is either a leaf node or a node with one child to its left. If the replacement node is a leaf node, the steps to perform the deletion are as follows:

- 1) Store the pointer to the node to be deleted in a temporary pointer variable (this pointer is used to delete the dynamically allocated memory).
- 2) Set the pointer in the parent of the node being deleted to point to the replacement node.
- 3) Set the pointer in the parent of the replacement node to null.
- 4) Set the pointer to the right subtree in the replacement node to point to the right subtree of the node to be deleted.
- 5) Delete the node to which the temporary pointer variable points.

The deletion steps for a replacement node with a left child are similar to those for a replacement node with no children, but the algorithm also must move the child to the replacement node's position. If the replacement node is a node with a left child, the steps to perform the deletion are as follows:

- 1) Store the pointer to the node to be deleted in a temporary pointer variable.
- 2) Set the pointer in the parent of the node being deleted to point to the replacement node.
- 3) Set the pointer in the parent of the replacement node to point to the left child of the replacement node.
- 4) Set the pointer to the right subtree in the replacement node to point to the right subtree of the node to be deleted.
- 5) Delete the node to which the temporary pointer variable points.

Write function `deleteNode` which takes as its arguments a pointer to the root node of the tree and the value to be deleted. The function should locate in the tree the node containing the value to be deleted and use the algorithms discussed here to delete the node. If the value is not found in the tree, the function should print a message that indicates whether or not the value is deleted. Modify the program of Fig. 12.19 to use this function. After deleting an item, call the `inOrder`, `preOrder` and `postOrder` traversal functions to confirm that the delete operation was performed correctly.

12.23 (*Binary Tree Search*) Write function `binaryTreeSearch` that attempts to locate a specified value in a binary search tree. The function should take as arguments a pointer to the root node of the binary tree and a search key to be located. If the node containing the search key is found, the function should return a pointer to that node; otherwise, the function should return a `NULL` pointer.

ANS:

```

1  /* Exercise 12.23 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* TreeNode structure definition */
7  struct TreeNode {
8      struct TreeNode *leftPtr; /* pointer to left subtree */
9      int data;                 /* node data */
10     struct TreeNode *rightPtr; /* pointer to right subtree */
11 }; /* end struct TreeNode */
12
13 typedef struct TreeNode TreeNode;
14 typedef TreeNode *TreeNodePtr;
15
16 /* function prototypes */
17 void insertNode( TreeNodePtr *treePtr, int value );
18 TreeNodePtr binaryTreeSearch( TreeNodePtr treePtr, const int key );
19
20 int main()
21 {
22     int i;                      /* loop counter */
23     int item;                   /* random value to insert in tree */
24     int searchKey;             /* value to search for */
25     TreeNodePtr rootPtr = NULL; /* points to the tree root */
26     TreeNodePtr searchResultPtr; /* pointer to search result */
27
28     srand( time( NULL ) ); /* randomize */
29     printf( "The numbers being placed in the tree are:\n" );
30
31     /* insert random values between 1 and 20 in the tree */
32     for ( i = 1; i <= 10; i++ ) {
33         item = 1 + rand() % 20;
34         printf( "%3d", item );
35         insertNode( &rootPtr, item );
36     } /* end for */
37
38     /* prompt user and read integer search key */
39     printf( "\n\nEnter an integer to search for: " );
40     scanf( "%d", &searchKey );
41
42     searchResultPtr = binaryTreeSearch( rootPtr, searchKey );
43
44     /* if searchKey not found */
45     if ( searchResultPtr == NULL ) {
46         printf( "\n%d was not found in the tree.\n\n", searchKey );
47     } /* end if */
48     else { /* if key found */
49         printf( "\n%d was found in the tree.\n\n",
50             searchResultPtr->data );
51     } /* end else */
52
53     return 0; /* indicate successful termination */
54
55 } /* end main */

```

```

56
57 /* insert a node into the tree */
58 void insertNode( TreeNodePtr *treePtr, int value )
59 {
60
61     /* if treePtr is NULL */
62     if ( *treePtr == NULL ) {
63
64         /* dynamically allocate memory */
65         *treePtr = malloc( sizeof( TreeNode ) );
66
67         /* if memory was allocated, insert node */
68         if ( *treePtr != NULL ) {
69             ( *treePtr )->data = value;
70             ( *treePtr )->leftPtr = NULL;
71             ( *treePtr )->rightPtr = NULL;
72         } /* end if */
73         else {
74             printf( "%d not inserted. No memory available.\n", value );
75         } /* end else */
76
77     } /* end if */
78     else { /* recursively call insertNode */
79
80         /* insert node in left subtree */
81         if ( value < ( *treePtr )->data ) {
82             insertNode( &( ( *treePtr )->leftPtr ), value );
83         } /* end if */
84         else {
85
86             /* insert node in right subtree */
87             if ( value > ( *treePtr )->data ) {
88                 insertNode( &( ( *treePtr )->rightPtr ), value );
89             } /* end if */
90             else { /* duplicate value */
91                 printf( "dup" );
92             } /* end else */
93
94         } /* end else */
95
96     } /* end else */
97
98 } /* end function insertNode */
99
100 /* search for key in tree */
101 TreeNodePtr binaryTreeSearch( TreeNodePtr treePtr, const int key )
102 {
103
104     /* traverse the tree inOrder */
105     if ( treePtr == NULL ) {
106         return NULL; /* key not found */
107     } /* end if */
108     else if ( treePtr->data == key ) {
109         return treePtr; /* key found */
110     } /* end else if */
111     else if ( key < treePtr->data ) {
112         binaryTreeSearch( treePtr->leftPtr, key ); /* search left */
113     } /* end else if */
114     else if ( key > treePtr->data ) {
115         binaryTreeSearch( treePtr->rightPtr, key ); /* search right */
116     } /* end else if */

```

```

117
118 } /* end function binaryTreeSearch */

```

The numbers being placed in the tree are:

```
18 9 7 2 13 2dup 10 1 19 2dup
```

Enter an integer to search for: 8

8 was not found in the tree.

12.24 (*Level Order Binary Tree Traversal*) The program of Fig. 12.19 illustrated three recursive methods of traversing a binary tree—inorder traversal, preorder traversal, and postorder traversal. This exercise presents the *level order traversal* of a binary tree in which the node values are printed level-by-level starting at the root node level. The nodes on each level are printed from left to right. The level order traversal is not a recursive algorithm. It uses the queue data structure to control the output of the nodes. The algorithm is as follows:

- 1) Insert the root node in the queue
- 2) While there are nodes left in the queue,
 - Get the next node in the queue
 - Print the node's value
 - If the pointer to the left child of the node is not null
 - Insert the left child node in the queue
 - If the pointer to the right child of the node is not null
 - Insert the right child node in the queue.

Write function `levelOrder` to perform a level order traversal of a binary tree. The function should take as an argument a pointer to the root node of the binary tree. Modify the program of Fig. 12.19 to use this function. Compare the output from this function to the outputs of the other traversal algorithms to see that it worked correctly. [*Note:* You will also need to modify and incorporate the queue processing functions of Fig. 12.13 in this program.]

ANS:

```

1  /* Exercise 12.24 solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* TreeNode structure definition */
7  struct TreeNode {
8      struct TreeNode *leftPtr; /* pointer to left subtree */
9      int data; /* node data */
10     struct TreeNode *rightPtr; /* pointer to right subtree */
11 }; /* end struct TreeNode */
12
13 typedef struct TreeNode TreeNode;
14 typedef TreeNode *TreeNodePtr;
15
16 /* tree function prototypes */
17 void insertNode( TreeNodePtr *treePtr, int value );
18 void levelOrderTraversal( TreeNodePtr treePtr );
19
20 /* QueueNode structure definition */
21 struct QueueNode {
22     TreeNodePtr data; /* node data */
23     struct QueueNode *nextPtr; /* pointer to next node */
24 }; /* end struct QueueNode */
25
26 typedef struct QueueNode QueueNode;
27 typedef QueueNode *QueueNodePtr;
28

```

```

29  /* queue function prototypes */
30  int isEmpty( QueueNodePtr headPtr );
31  TreeNodePtr dequeue( QueueNodePtr *headPtr, QueueNodePtr * tailPtr );
32  void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr,
33              TreeNodePtr node );
34
35  int main()
36  {
37      int i;                /* loop counter */
38      int item;            /* random value to insert in tree */
39      TreeNodePtr rootPtr = NULL; /* points to the tree root */
40
41      srand( time( NULL ) ); /* randomize */
42      printf( "The values being inserted in the tree are:\n" );
43
44      /* insert random values between 1 and 15 in the tree */
45      for ( i = 1; i <= 15; i++ ) {
46          item = 1 + rand() % 20;
47          printf( " %d", item );
48          insertNode( &rootPtr, item );
49      } /* end for */
50
51      /* traverse the tree level order */
52      printf( "\n\nThe level order traversal is:\n" );
53      levelOrderTraversal( rootPtr );
54      printf( "\n" );
55
56      return 0; /* indicate successful termination */
57  } /* end main */
58
59
60  /* Level order traversal of a binary tree */
61  void levelOrderTraversal( TreeNodePtr ptr )
62  {
63      QueueNodePtr head = NULL; /* points to queue head */
64      QueueNodePtr tail = NULL; /* points to queue tail */
65      TreeNodePtr node;        /* current tree node */
66
67      /* if tree is not empty */
68      if ( ptr != NULL ) {
69          enqueue( &head, &tail, ptr ); /* enqueue root node */
70
71          /* while queue is not empty */
72          while ( !isEmpty( head ) ) {
73
74              /* dequeue next node and print data */
75              node = dequeue( &head, &tail );
76              printf( "%d ", node->data );
77
78              /* insert left child node in the queue */
79              if ( node->leftPtr != NULL ) {
80                  enqueue( &head, &tail, node->leftPtr );
81              } /* end if */
82
83              /* insert right child node in the queue */
84              if ( node->rightPtr != NULL ) {
85                  enqueue( &head, &tail, node->rightPtr );
86              } /* end if */
87
88          } /* end while */
89
90      } /* end if */

```



```

91
92 } /* end function levelOrderTraversal */
93
94 /* insert a node into the tree */
95 void insertNode( TreeNodePtr *treePtr, int value )
96 {
97
98     /* if treePtr is NULL */
99     if ( *treePtr == NULL ) {
100
101         /* dynamically allocate memory */
102         *treePtr = malloc( sizeof( TreeNode ) );
103
104         /* if memory was allocated, insert node */
105         if ( *treePtr != NULL ) {
106             ( *treePtr )->data = value;
107             ( *treePtr )->leftPtr = NULL;
108             ( *treePtr )->rightPtr = NULL;
109         } /* end if */
110         else {
111             printf( "%d not inserted. No memory available.\n", value );
112         } /* end else */
113
114     } /* end if */
115     else { /* recursively call insertNode */
116
117         /* insert node in left subtree */
118         if ( value < ( *treePtr )->data ) {
119             insertNode( &( ( *treePtr )->leftPtr ), value );
120         } /* end if */
121         else {
122
123             /* insert node in right subtree */
124             if ( value > ( *treePtr )->data ) {
125                 insertNode( &( ( *treePtr )->rightPtr ), value );
126             } /* end if */
127             else { /* duplicate value */
128                 printf( "dup" );
129             } /* end else */
130
131         } /* end else */
132
133     } /* end else */
134 } /* end function insertNode */
135
136 /* enqueue node */
137 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, TreeNodePtr node )
138 {
139     QueueNodePtr newPtr; /* temporary node pointer */
140
141     /* dynamically allocate memory */
142     newPtr = malloc( sizeof( QueueNode ) );
143
144     /* if newPtr does not equal NULL */
145     if ( newPtr != NULL ) {
146         newPtr->data = node;
147         newPtr->nextPtr = NULL;
148
149         /* if queue is empty, insert at head */
150         if ( isEmpty( *headPtr ) ) {

```

```

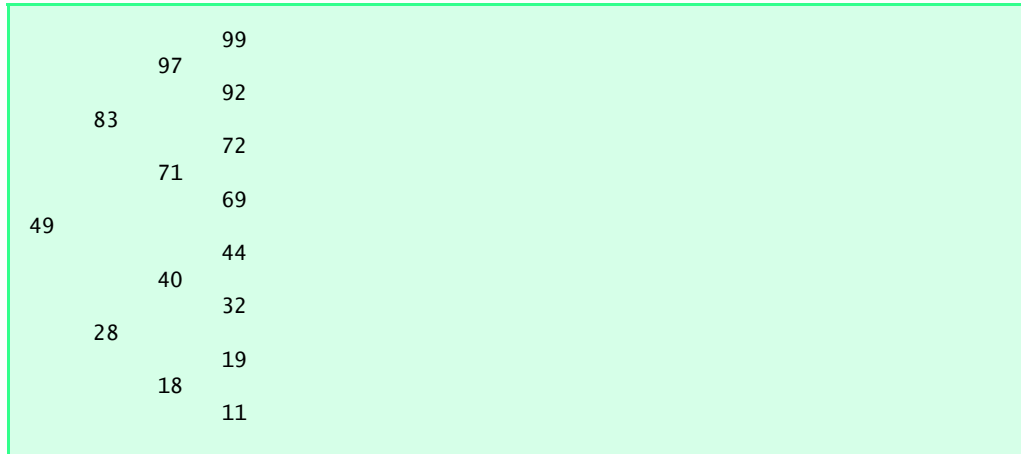
152     *headPtr = newPtr;
153 } /* end if */
154 else { /* insert at tail */
155     ( *tailPtr )->nextPtr = newPtr;
156 } /* end else */
157
158     *tailPtr = newPtr;
159 } /* end if */
160 else {
161     printf( "Node not inserted\n" );
162 } /* end else */
163
164 } /* end function enqueue */
165
166 /* dequeue node from queue */
167 TreeNodePtr dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr )
168 {
169     TreeNodePtr node; /* dequeued node */
170     QueueNodePtr tempPtr; /* temporary node pointer */
171
172     /* dequeue node and reset queue headPtr */
173     node = ( *headPtr )->data;
174     tempPtr = *headPtr;
175     *headPtr = ( *headPtr )->nextPtr;
176
177     /* if queue is empty */
178     if ( *headPtr == NULL ) {
179         *tailPtr = NULL;
180     } /* end if */
181
182     free( tempPtr ); /* free memory */
183
184     return node; /* return dequeued node */
185
186 } /* end function dequeue */
187
188 /* is queue empty? */
189 int isEmpty( QueueNodePtr headPtr )
190 {
191     return headPtr == NULL; /* return NULL is queue is empty */
192
193 } /* end function isEmpty */

```

The values being inserted in the tree are:
5 10 7 5dup 11 9 15 1 7dup 20 6 20dup 4 16 4dup

The level order traversal is:
5 1 10 4 7 11 6 9 15 20 16

12.25 (*Printing Trees*) Write a recursive function `outputTree` to display a binary tree on the screen. The function should output the tree row-by-row with the top of the tree at the left of the screen and the bottom of the tree toward the right of the screen. Each row is output vertically. For example, the binary tree illustrated in Fig. 12.22 is output as follows:



Note the rightmost leaf node appears at the top of the output in the rightmost column, and the root node appears at the left of the output. Each column of output starts five spaces to the right of the previous column. Function `outputTree` should receive as arguments a pointer to the root node of the tree and an integer `totalSpaces` representing the number of spaces preceding the value to be output (this variable should start at zero so the root node is output at the left of the screen). The function uses a modified inorder traversal to output the tree—it starts at the rightmost node in the tree and works back to the left. The algorithm is as follows:

- While the pointer to the current node is not null
- Recursively call `outputTree` with the right subtree of the current node and `totalSpaces + 5`
- Use a for statement to count from 1 to `totalSpaces` and output spaces
- Output the value in the current node
- Set the pointer to the current node to point to the left subtree of the current node
- Increment `totalSpaces` by 5.

ANS:

```

1  /* Exercise 12.25 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* TreeNode structure definition */
7  struct TreeNode {
8      struct TreeNode *leftPtr; /* pointer to left subtree */
9      int data; /* node data */
10     struct TreeNode *rightPtr; /* pointer to right subtree */
11 }; /* end struct TreeNode */
12
13 typedef struct TreeNode TreeNode;
14 typedef TreeNode *TreeNodePtr;
15
16 /* function prototypes */
17 void insertNode( TreeNodePtr *treePtr, int value );
18 void outputTree( TreeNodePtr treePtr, int spaces );
19
20 int main()
21 {
22     int i; /* loop counter */
  
```

```

23     int item;                /* random value to be inserted */
24     int totalSpaces = 0;     /* spaces preceding output */
25     TreeNodePtr rootPtr = NULL; /* points to the tree root */
26
27     srand( time( NULL ) ); /* randomize */
28     printf( "The numbers being placed in the tree are:\n" );
29
30     /* insert random values between 1 and 10 in the tree */
31     for ( i = 1; i <= 10; i++ ) {
32         item = rand() % 15;
33         printf( "%3d", item );
34         insertNode( &rootPtr, item );
35     } /* end for */
36
37     printf( "\n\n" );
38     outputTree( rootPtr, totalSpaces ); /* display tree */
39
40     return 0; /* indicate successful termination */
41
42 } /* end main */
43
44 /* insert a node into the tree */
45 void insertNode( TreeNodePtr *treePtr, int value )
46 {
47
48     /* if treePtr is NULL */
49     if ( *treePtr == NULL ) {
50
51         /* dynamically allocate memory */
52         *treePtr = malloc( sizeof( TreeNode ) );
53
54         /* if memory was allocated, insert node */
55         if ( *treePtr != NULL ) {
56             ( *treePtr )->data = value;
57             ( *treePtr )->leftPtr = NULL;
58             ( *treePtr )->rightPtr = NULL;
59         } /* end if */
60         else {
61             printf( "%d not inserted. No memory available.\n", value );
62         } /* end else */
63
64     } /* end if */
65     else { /* recursively call insertNode */
66
67         /* insert node in left subtree */
68         if ( value < ( *treePtr )->data ) {
69             insertNode( &( ( *treePtr )->leftPtr ), value );
70         } /* end if */
71         else {
72
73             /* insert node in right subtree */
74             if ( value > ( *treePtr )->data ) {
75                 insertNode( &( ( *treePtr )->rightPtr ), value );
76             } /* end if */
77             else { /* duplicate value */
78                 printf( "dup" );
79             } /* end else */
80
81         } /* end else */
82
83     } /* end else */

```

```

84
85 } /* end function insertNode */
86
87 /* display the tree */
88 void outputTree( TreeNodePtr treePtr, int spaces )
89 {
90     int loop; /* loop counter */
91
92     /* while not the end of tree */
93     while ( treePtr != NULL ) {
94
95         /* recursive call with right subtree */
96         outputTree( treePtr->rightPtr, spaces + 5 );
97
98         /* loop and output spaces */
99         for ( loop = 1; loop <= spaces; loop++ ) {
100             printf( " " );
101         } /* end for */
102
103         printf( "%d\n", treePtr->data );
104
105         /* set pointer to left subtree and make recursive call */
106         outputTree( treePtr->leftPtr, spaces + 5 );
107         treePtr = NULL;
108     } /* end while */
109 } /* end function outputTree */

```

The numbers being placed in the tree are:

```

1  5  6 14 11  2  9  9dup 14dup  8
      14
        11
          9
            8
              6
                5
                  2
                    1

```

SPECIAL SECTION: BUILDING YOUR OWN COMPILER

In Exercise 7.18 and Exercise 7.19, we introduced Simpletron Machine Language (SML) and created the Simpletron computer simulator to execute programs written in SML. In this section, we build a compiler that converts programs written in a high-level programming language to SML. This section “ties” together the entire programming process. We will write programs in this new high-level language, compile the programs on the compiler we build, and run the programs on the simulator we built in Exercise 7.19.

12.26 (*The Simple Language*) Before we begin building the compiler, we discuss a simple, yet powerful, high-level language similar to early versions of the popular language BASIC. We call the language *Simple*. Every *Simple statement* consists of a *line number* and a *Simple instruction*. Line numbers must appear in ascending order. Each instruction begins with one of the following *Simple commands*: *rem*, *input*, *let*, *print*, *goto*, *if...goto* or *end* (see Fig. 12.23). All commands except *end* can be used repeatedly. *Simple* evaluates only integer expressions using the +, -, *, and / operators. These operators have the same precedence as in C. Parentheses can be used to change the order of evaluation of an expression.

Command	Example statement	Description
rem	50 rem this is a remark	Any text following the command <code>rem</code> is for documentation purposes only and is ignored by the compiler.
input	30 input x	Display a question mark to prompt the user to enter an integer. Read that integer from the keyboard and store the integer in <code>x</code> .
let	80 let u = 4 * (j - 56)	Assign <code>u</code> the value of $4 * (j - 56)$. Note that an arbitrarily complex expression can appear to the right of the equal sign.
print	10 print w	Display the value of <code>w</code> .
goto	70 goto 45	Transfer program control to line 45.
if...goto	35 if i == z goto 80	Compare <code>i</code> and <code>z</code> for equality and transfer program control to line 80 if the condition is true; otherwise, continue execution with the next statement.
end	99 end	Terminate program execution.

Fig. 12.1 Simple commands.

Our Simple compiler recognizes only lowercase letters. All characters in a Simple file should be lowercase (uppercase letters result in a syntax error unless they appear in a `rem` statement in which case they are ignored). A *variable name* is a single letter. Simple does not allow descriptive variable names, so variables should be explained in remarks to indicate their use in the program. Simple uses only integer variables. Simple does not have variable declarations—merely mentioning a variable name in a program causes the variable to be declared and initialized to zero automatically. The syntax of Simple does not allow string manipulation (reading a string, writing a string, comparing strings, etc.). If a string is encountered in a Simple program (after a command other than `rem`), the compiler generates a syntax error. Our compiler will assume that Simple programs are entered correctly. Exercise 12.29 asks the student to modify the compiler to perform syntax error checking.

Simple uses the conditional `if...goto` statement and the unconditional `goto` statement to alter the flow of control during program execution. If the condition in the `if...goto` statement is true, control is transferred to a specific line of the program. The following relational and equality operators are valid in an `if...goto` statement: `<`, `>`, `<=`, `>=`, `==` or `!=`. The precedence of these operators is the same as in C.

Let us now consider several Simple programs that demonstrate Simple's features. The first program (Fig. 12.24) reads two integers from the keyboard, stores the values in variables `a` and `b`, and computes and prints their sum (stored in variable `c`).

```

1 10 rem  determine and print the sum of two integers
2 15 rem
3 20 rem  input the two integers
4 30 input a
5 40 input b
6 45 rem
7 50 rem  add integers and store result in c
8 60 let c = a + b
9 65 rem
10 70 rem  print the result
11 80 print c
12 90 rem  terminate program execution
13 99 end

```

Figure 12.25 determines and prints the larger of two integers. The integers are input from the keyboard and stored in `s` and `t`. The `if...goto` statement tests the condition `s >= t`. If the condition is true, control is transferred to line 90 and `s` is output; otherwise, `t` is output and control is transferred to the end statement in line 99 where the program terminates.

```

1 10 rem  determine the larger of two integers
2 20 input s
3 30 input t
4 32 rem
5 35 rem  test if s >= t
6 40 if s >= t goto 90
7 45 rem
8 50 rem  t is greater than s, so print t
9 60 print t
10 70 goto 99
11 75 rem
12 80 rem  s is greater than or equal to t, so print s
13 90 print s
14 99 end

```

Simple does not provide a repetition structure (such as C's `for`, `while` or `do...while`). However, Simple can simulate each of C's repetition structures using the `if...goto` and `goto` statements. Figure 12.26 uses a sentinel-controlled loop to calculate the squares of several integers. Each integer is input from the keyboard and stored in variable `j`. If the value entered is the sentinel -9999, control is transferred to line 99 where the program terminates. Otherwise, `k` is assigned the square of `j`, `k` is output to the screen and control is passed to line 20 where the next integer is input.

```

1 10 rem  calculate the squares of several integers
2 20 input j
3 23 rem
4 25 rem  test for sentinel value
5 30 if j == -9999 goto 99
6 33 rem
7 35 rem  calculate square of j and assign result to k
8 40 let k = j * j
9 50 print k
10 53 rem
11 55 rem  loop to get next j
12 60 goto 20
13 99 end

```

Using the sample programs of Fig. 12.24, Fig. 12.25 and Fig. 12.26 as your guide, write a Simple program to accomplish each of the following:

- a) Input three integers, determine their average and print the result.

ANS:

```

1 5 rem      Exercise 12.26 Part A Solution
2 6 rem
3 10 input x
4 15 input y
5 10 input z
6 21 rem    calculate average a
7 25 let a = ( x + y + z ) / 3
8 26 rem
9 30 print a
10 99 end

```

b) Use a sentinel-controlled loop to input 10 integers and compute and print their sum.

ANS:

```
1 5 rem      Exercise 12.26 Part B Solution
2 6 rem
3 10 input n
4 12 rem    set up sentinel loop
5 15 if n == 9999 goto 40
6 16 rem
7 17 rem    add n to the sum of s
8 20 let s = s + n
9 21 rem
10 22 rem   loop to get next n
11 25 goto 10
12 36 rem   print sum s
13 40 print s
14 99 end
```

c) Use a counter-controlled loop to input seven integers, some positive and some negative, and compute and print their average.

ANS:

```
1 5 rem      exercise 12.26 Part C Solution
2 6 rem
3 10 input m
4 11 rem    increment counter by 1
5 12 rem    c is automatically initiated to 0
6 13 rem    when created
7 15 let c = c + 1
8 16 rem
9 17 rem    calculate sum s
10 20 let s = s + m
11 22 rem   loop to get next m
12 23 rem   if c is not yet 7
13 25 if c <= 7 goto 10
14 26 rem
15 27 rem   compute average a
16 30 let a = s / 7
17 31 rem
18 35 print a
19 99 end
```


- d) Input a series of integers and determine and print the largest. The first integer input indicates how many numbers should be processed.

ANS:

```

1 5 rem      Exercise 12.26 Part D Solution
2 6 rem
3 7 rem      Enter the number of integers
4 8 rem      to be processed
5 10 input n
6 23 rem     begin entering numbers t
7 25 input t
8 26 rem     check if t is larger than l
9 27 rem     l's initial value is zero
10 30 rem    if t <= 1 goto 50
11 31 rem    t must be larger than 1
12 32 rem    so assign t as largest
13 35 let l = t
14 49 rem    decrement n
15 50 let n = n - 1
16 59 rem    test for loop exit condition
17 60 if n == 0 goto 80
18 69 rem    loop to get next t
19 70 goto 25
20 79 rem    print largest value
21 80 print l
22 99 end

```

- e) Input 10 integers and print the smallest.

ANS:

```

1 1 rem      Exercise 12.26 Part E Solution
2 2 rem
3 3 rem      set counter c equal to 1
4 5 let c = 1
5 6 rem      input integer m
6 7 rem      assign first entry to
7 8 rem      the smallest value s
8 9 input m
9 10 let s = m
10 11 rem    enter main loop
11 13 goto 20
12 14 rem    main loop
13 15 input m
14 18 rem    determine if m is smaller
15 19 rem    than current s
16 20 if m < s goto 50
17 29 rem    increment counter
18 30 let c = c + 1
19 34 rem    exit when c becomes 11
20 35 if c == 11 goto 60
21 39 rem    loop for next m
22 40 goto 15
23 48 rem    assign m to s as
24 49 rem    smallest value
25 50 let s = m
26 51 rem    loop to counter increment
27 55 goto 30
28 59 rem    print smallest value
29 60 print s
30 99 end

```

f) Calculate and print the sum of the even integers from 2 to 30.

ANS:

```
1 5 rem      Exercise 12.26 Part F Solution
2 6 rem
3 7 rem
4 9 rem      initialize i to 2
5 10 let i = 2
6 14 rem     store sum in s
7 15 let s = s + 1
8 19 rem     increment i by 2
9 20 let i = i + 2
10 28 rem    set loop terminating
11 29 rem    condition at 32
12 30 if < 32 goto 15
13 39 rem    print sum
14 40 print s
15 99 end
```

g) Calculate and print the product of the odd integers from 1 to 9.

ANS:

```
1 5 rem      Exercise 12.26 Part G Solution
2 6 rem
3 7 rem
4 9 rem      initialize k to 1
5 10 let k = 1
6 11 rem     initialize p to 1
7 13 let p = 1
8 14 rem     store product in p
9 15 let p = p * k
10 19 rem    increment k by 1
11 20 let k = k + 2
12 28 rem    set loop terminating
13 29 rem    condition at 10
14 30 if k < 10 goto 15
15 39 rem    print product
16 40 print p
17 99 end
```

12.27 (*Building A Compiler; Prerequisite: Complete Exercise 7.18, Exercise 7.19, Exercise 12.12, Exercise 12.13 and Exercise 12.26*) Now that the Simple language has been presented (Exercise 12.26), we discuss how to build our Simple compiler. First, we consider the process by which a Simple program is converted to SML and executed by the Simpletron simulator (see Fig. 12.27). A file containing a Simple program is read by the compiler and converted to SML code. The SML code is output to a file on disk, in which SML instructions appear one per line. The SML file is then loaded into the Simpletron simulator, and the results are sent to a file on disk and to the screen. Note that the Simpletron program developed in Exercise 7.19 took its input from the keyboard. It must be modified to read from a file so it can run the programs produced by our compiler.

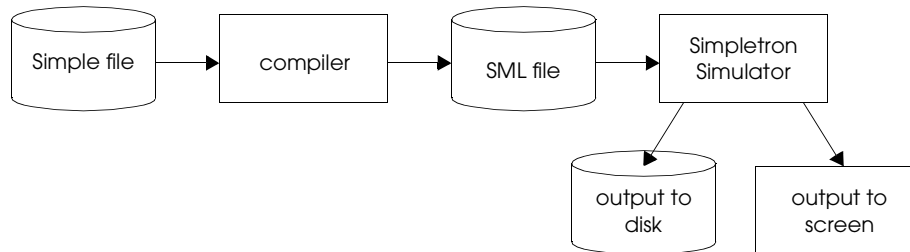


Fig. 12.2 Writing, compiling and executing a Simple language program.

The compiler performs two *passes* of the Simple program to convert it to SML. The first pass constructs a *symbol table* in which every *line number*, *variable name* and *constant* of the Simple program is stored with its type and corresponding location in the final SML code (the symbol table is discussed in detail below). The first pass also produces the corresponding SML instruction(s) for each Simple statement. As we will see, if the Simple program contains statements that transfer control to a line later in the program, the first pass results in an SML program containing some incomplete instructions. The second pass of the compiler locates and completes the unfinished instructions, and outputs the SML program to a file.

First Pass

The compiler begins by reading one statement of the Simple program into memory. The line must be separated into its individual *tokens* (i.e., “pieces” of a statement) for processing and compilation (standard library function `strtok` can be used to facilitate this task). Recall that every statement begins with a line number followed by a command. As the compiler breaks a statement into tokens, if the token is a line number, a variable, or a constant, it is placed in the symbol table. A line number is placed in the symbol table only if it is the first token in a statement. The `symbolTable` is an array of `tableEntry` structures representing each symbol in the program. There is no restriction on the number of symbols that can appear in the program. Therefore, the `symbolTable` for a particular program could be large. Make the `symbolTable` a 100-element array for now. You can increase or decrease its size once the program is working.

The `tableEntry` structure definition is as follows:

```

struct tableEntry {
    int symbol;
    char type;    /* 'C', 'L' or 'V' */
    int location; /* 00 to 99 */
};
  
```

Each `tableEntry` structure contains three members. Member `symbol` is an integer containing the ASCII representation of a variable (remember that variable names are single characters), a line number, or a constant. Member `type` is one of the following characters indicating the symbol’s type: ‘C’ for constant, ‘L’ for line number, or ‘V’ for variable. Member `location` contains the Simpletron memory location (00 to 99) to which the symbol refers. Simpletron memory is an array of 100 integers in which SML instructions and data are stored. For a line number, the location is the element in the Simpletron memory array at which the SML instructions for the Simple statement begin. For a variable or constant, the location is the element in the Simpletron memory array in which the variable or constant is stored. Variables and constants are allocated from the end of Simpletron’s memory backwards. The first variable or constant is stored in location at 99, the next in location at 98, etc.

The symbol table plays an integral part in converting Simple programs to SML. We learned in Chapter 7 that an SML instruction is a four-digit integer that comprises two parts—the *operation code* and the *operand*. The operation code is determined by commands in Simple. For example, the simple command `input` corresponds to SML operation code 10 (read), and the Simple command `print` corresponds to SML operation code 11 (write). The operand is a memory location containing the data on which the operation code performs its task (e.g., operation code 10 reads a value from the keyboard and stores it in the memory location specified by the operand). The compiler searches `symbolTable` to determine the Simpletron memory location for each symbol so

the corresponding location can be used to complete the SML instructions.

The compilation of each Simple statement is based on its command. For example, after the line number in a `rem` statement is inserted in the symbol table, the remainder of the statement is ignored by the compiler, because a remark is for documentation purposes only. The `input`, `print`, `goto` and `end` statements correspond to the SML *read*, *write*, *branch* (to a specific location) and *halt* instructions. Statements containing these Simple commands are converted directly to SML [Note: That a `goto` statement may contain an unresolved reference if the specified line number refers to a statement further into the Simple program file; this is sometimes called a forward reference.]

When a `goto` statement is compiled with an unresolved reference, the SML instruction must be *flagged* to indicate that the second pass of the compiler must complete the instruction. The flags are stored in 100-element array `flags` of type `int` in which each element is initialized to `-1`. If the memory location to which a line number in the Simple program refers is not yet known (i.e., it is not in the symbol table), the line number is stored in array `flags` in the element with the same subscript as the incomplete instruction. The operand of the incomplete instruction is set to `00` temporarily. For example, an unconditional branch instruction (making a forward reference) is left as `+4000` until the second pass of the compiler. The second pass of the compiler will be described shortly.

Compilation of `if...goto` and `let` statements is more complicated than other statements—they are the only statements that produce more than one SML instruction. For an `if...goto` statement, the compiler produces code to test the condition and to branch to another line if necessary. The result of the branch could be an unresolved reference. Each of the relational and equality operators can be simulated using SML's *branch zero* and *branch negative* instructions (or possibly a combination of both).

For a `let` statement, the compiler produces code to evaluate an arbitrarily complex arithmetic expression consisting of integer variables and/or constants. Expressions should separate each operand and operator with spaces. Exercise 12.12 and Exercise 12.13 presented the infix-to-postfix conversion algorithm and the postfix evaluation algorithm used by compilers to evaluate expressions. Before proceeding with your compiler, you should complete each of these exercises. When a compiler encounters an expression, it converts the expression from infix notation to postfix notation, then evaluates the postfix expression.

How is it that the compiler produces the machine language to evaluate an expression containing variables? The postfix evaluation algorithm contains a “hook” that allows our compiler to generate SML instructions rather than actually evaluating the expression. To enable this “hook” in the compiler, the postfix evaluation algorithm must be modified to search the symbol table for each symbol it encounters (and possibly insert it), determine the symbol's corresponding memory location, and *push the memory location on the stack instead of the symbol*. When an operator is encountered in the postfix expression, the two memory locations at the top of the stack are popped and machine language for effecting the operation is produced using the memory locations as operands. The result of each subexpression is stored in a temporary location in memory and pushed back onto the stack so the evaluation of the postfix expression can continue. When postfix evaluation is complete, the memory location containing the result is the only location left on the stack. This is popped and SML instructions are generated to assign the result to the variable at the left of the `let` statement.

Second Pass

The second pass of the compiler performs two tasks: resolve any unresolved references and output the SML code to a file. Resolution of references occurs as follows:

- 1) Search the `flags` array for an unresolved reference (i.e., an element with a value other than `-1`).
- 2) Locate the structure in array `symbolTable` containing the symbol stored in the `flags` array (be sure that the type of the symbol is 'L' for line number).
- 3) Insert the memory location from structure member `location` into the instruction with the unresolved reference (remember that an instruction containing an unresolved reference has operand `00`).
- 4) Repeat steps 1, 2 and 3 until the end of the `flags` array is reached.

After the resolution process is complete, the entire array containing the SML code is output to a disk file with one SML instruction per line. This file can be read by the Simpletron for execution (after the simulator is modified to read its input from a file).

A Complete Example

The following example illustrates a complete conversion of a Simple program to SML as it will be performed by the Simple compiler. Consider a Simple program that inputs an integer and sums the values from 1 to that integer. The program and the SML instructions produced by the first pass are illustrated in Fig. 12.3. The symbol table constructed by the first pass is shown in Fig. 12.29

Simple program	SML location and instruction	Description
5 rem sum 1 to x	<i>none</i>	rem ignored
10 input x	00 +1099	read x into location 99
15 rem check y == x	<i>none</i>	rem ignored
20 if y == x goto 60	01 +2098	load y (98) into accumulator
	02 +3199	sub x (99) from accumulator
	03 +4200	<i>branch zero to unresolved location</i>
25 rem increment y	<i>none</i>	rem ignored
30 let y = y + 1	04 +2098	load y into accumulator
	05 +3097	add 1 (97) to accumulator
	06 +2196	store in temporary location 96
	07 +2096	load from temporary location 96
	08 +2198	store accumulator in y
35 rem add y to total	<i>none</i>	rem ignored
40 let t = t + y	09 +2095	load t (95) into accumulator
	10 +3098	add y to accumulator
	11 +2194	store in temporary location 94
	12 +2094	load from temporary location 94
	13 +2195	store accumulator in t
45 rem loop y	<i>none</i>	rem ignored
50 goto 20	14 +4001	branch to location 01
55 rem output result	<i>none</i>	rem ignored
60 print t	15 +1195	output t to screen
99 end	16 +4300	terminate execution

Fig. 12.3 SML instructions produced after the compiler's first pass.

Symbol	Type	Location
5	L	00
10	L	00
'x'	V	99
15	L	01
20	L	01
'y'	V	98
25	L	04
30	L	04
1	C	97
35	L	09
40	L	09

Fig. 12.4 Symbol table for program of Fig. 12.3

Symbol	Type	Location
't'	V	95
45	L	14
50	L	14
55	L	15
60	L	15
99	L	16

Fig. 12.4 Symbol table for program of Fig. 12.3

Most Simple statements convert directly to single SML instructions. The exceptions in this program are remarks, the `if...goto` statement in line 20, and the `let` statements. Remarks do not translate into machine language. However, the line number for a remark is placed in the symbol table in case the line number is referenced in a `goto` statement or an `if...goto` statement. Line 20 of the program specifies that if the condition `y == x` is true, program control is transferred to line 60. Because line 60 appears later in the program, the first pass of the compiler has not as yet placed 60 in the symbol table (line numbers are placed in the symbol table only when they appear as the first token in a statement). Therefore, it is not possible at this time to determine the operand of the SML *branch zero* instruction at location 03 in the array of SML instructions. The compiler places 60 in location 03 of the `flags` array to indicate that the second pass completes this instruction.

We must keep track of the next instruction location in the SML array because there is not a one-to-one correspondence between Simple statements and SML instructions. For example, the `if...goto` statement of line 20 compiles into three SML instructions. Each time an instruction is produced, we must increment the *instruction counter* to the next location in the SML array. Note that the size of Simpletron's memory could present a problem for Simple programs with many statements, variables and constants. It is conceivable that the compiler will run out of memory. To test for this case, your program should contain a *data counter* to keep track of the location at which the next variable or constant will be stored in the SML array. If the value of the instruction counter is larger than the value of the data counter, the SML array is full. In this case, the compilation process should terminate and the compiler should print an error message indicating that it ran out of memory during compilation.

Step-by-Step View of the Compilation Process

Let us now walk through the compilation process for the Simple program in Fig. 12.3. The compiler reads the first line of the program

```
5 rem sum 1 to x
```

into memory. The first token in the statement (the line number) is determined using `strtok` (see Chapter 8 for a discussion of C's string manipulation functions). The token returned by `strtok` is converted to an integer using `atoi`, so the symbol 5 can be located in the symbol table. If the symbol is not found, it is inserted in the symbol table. Since we are at the beginning of the program and this is the first line, no symbols are in the table yet. So, 5 is inserted into the symbol table as type L (line number) and assigned the first location in SML array (00). Although this line is a remark, a space in the symbol table is allocated for the line number (in case it is referenced by a `goto` or an `if...goto`). No SML instruction is generated for a `rem` statement, so the instruction counter is not incremented.

The statement

```
10 input x
```

is tokenized next. The line number 10 is placed in the symbol table as type L and assigned the first location in the SML array (00 because a remark began the program, so the instruction counter is currently 00). The command `input` indicates that the next token is a variable (only a variable can appear in an `input` statement). Because `input` corresponds directly to an SML operation code, the compiler simply has to determine the location of `x` in the SML array. Symbol `x` is not found in the symbol table. So, it is inserted into the symbol table as the ASCII representation of `x`, given type V, and assigned location 99 in the SML array (data storage begins at 99 and is allocated backwards). SML code can now be generated for this statement. Operation code 10 (the SML read operation code) is multiplied by 100, and the location of `x` (as determined in the symbol table) is added to complete the instruction. The instruction is then stored in the SML array at location 00. The instruction counter is incremented by 1 because a single SML instruction was produced.

The statement

```
15 rem    check y == x
```

is tokenized next. The symbol table is searched for line number 15 (which is not found). The line number is inserted as type L and assigned the next location in the array, 01 (remember that `rem` statements do not produce code, so the instruction counter is not incremented).

The statement

```
20 if y == x goto 60
```

is tokenized next. Line number 20 is inserted in the symbol table and given type L with the next location in the SML array 01. The command `if` indicates that a condition is to be evaluated. The variable `y` is not found in the symbol table, so it is inserted and given the type V and the SML location 98. Next, SML instructions are generated to evaluate the condition. Since there is no direct equivalent in SML for the `if...goto`, it must be simulated by performing a calculation using `x` and `y` and branching based on the result. If `y` is equal to `x`, the result of subtracting `x` from `y` is zero, so the *branch zero* instruction can be used with the result of the calculation to simulate the `if...goto` statement. The first step requires that `y` be loaded (from SML location 98) into the accumulator. This produces the instruction 01 +2098. Next, `x` is subtracted from the accumulator. This produces the instruction 02 +3199. The value in the accumulator may be zero, positive or negative. Since the operator is `==`, we want to *branch zero*. First, the symbol table is searched for the branch location (60 in this case), which is not found. So, 60 is placed in the `flags` array at location 03, and the instruction 03 +4200 is generated (we cannot add the branch location because we have not assigned a location to line 60 in the SML array yet). The instruction counter is incremented to 04.

The compiler proceeds to the statement

```
25 rem    increment y
```

The line number 25 is inserted in the symbol table as type L and assigned SML location 04. The instruction counter is not incremented.

When the statement

```
30 let y = y + 1
```

is tokenized, the line number 30 is inserted in the symbol table as type L and assigned SML location 04. Command `let` indicates that the line is an assignment statement. First, all the symbols on the line are inserted in the symbol table (if they are not already there). The integer 1 is added to the symbol table as type C and assigned SML location 97. Next, the right side of the assignment is converted from infix to postfix notation. Then the postfix expression (`y 1 +`) is evaluated. Symbol `y` is located in the symbol table and its corresponding memory location is pushed onto the stack. Symbol 1 is also located in the symbol table, and its corresponding memory location is pushed onto the stack. When the operator `+` is encountered, the postfix evaluator pops the stack into the right operand of the operator and pops the stack again into the left operand of the operator, then produces the SML instructions

```
04 +2098  (load y)
05 +3097  (add 1)
```

The result of the expression is stored in a temporary location in memory (96) with instruction

```
06 +2196  (store temporary)
```

and the temporary location is pushed on the stack. Now that the expression has been evaluated, the result must be stored in `y` (i.e., the variable on the left side of `=`). So, the temporary location is loaded into the accumulator and the accumulator is stored in `y` with the instructions

```
07 +2096  (load temporary)
08 +2198  (store y)
```

The reader will immediately notice that SML instructions appear to be redundant. We will discuss this issue shortly.

When the statement

```
35 rem    add y to total
```

is tokenized, line number 35 is inserted in the symbol table as type L and assigned location 09.

The statement

```
40 let t = t + y
```

is similar to line 30. The variable `t` is inserted in the symbol table as type V and assigned SML location 95. The instructions follow the same logic and format as line 30, and the instructions 09 +2095, 10 +3098, 11 +2194, 12 +2094, and 13 +2195 are generated. Note that the result of `t + y` is assigned to temporary location 94 before being assigned to `t` (95). Once again, the reader will note that the instructions in memory locations 11 and 12 appear to be redundant. Again, we will discuss this shortly.

The statement

```
45 rem    loop y
```

is a remark, so line 45 is added to the symbol table as type L and assigned SML location 14.

The statement

```
50 goto 20
```

transfers control to line 20. Line number 50 is inserted in the symbol table as type L and assigned SML location 14. The equivalent of `goto` in SML is the *unconditional branch* (40) instruction that transfers control to a specific SML location. The compiler searches the symbol table for line 20 and finds that it corresponds to SML location 01. The operation code (40) is multiplied by 100 and location 01 is added to it to produce the instruction 14 +4001.

The statement

```
55 rem    output result
```

is a remark, so line 55 is inserted in the symbol table as type L and assigned SML location 15.

The statement

```
60 print t
```

is an output statement. Line number 60 is inserted in the symbol table as type L and assigned SML location 15. The equivalent of `print` in SML is operation code 11 (*write*). The location of `t` is determined from the symbol table and added to the result of the operation code multiplied by 100.

The statement

```
99 end
```

is the final line of the program. Line number 99 is stored in the symbol table as type L and assigned SML location 16. The end command produces the SML instruction +4300 (43 is *halt* in SML) which is written as the final instruction in the SML memory array.

This completes the first pass of the compiler. We now consider the second pass. The `flags` array is searched for values other than `-1`. Location 03 contains 60, so the compiler knows that instruction 03 is incomplete. The compiler completes the instruction by searching the symbol table for 60, determining its location and adding the location to the incomplete instruction. In this case, the search determines that line 60 corresponds to SML location 15, so the completed instruction 03 +4215 is produced replacing 03 +4200. The Simple program has now been compiled successfully.

To build the compiler, you will have to perform each of the following tasks:

- Modify the Simpletron simulator program you wrote in Exercise 7.19 to take its input from a file specified by the user (see Chapter 11). Also, the simulator should output its results to a disk file in the same format as the screen output.
- Modify the infix-to-postfix evaluation algorithm of Exercise 12.12 to process multi-digit integer operands and single-letter variable-name operands. [*Hint*: Standard library function `strtok` can be used to locate each constant and variable in an expression, and constants can be converted from strings to integers using standard library function `atoi`.] [*Note*: The data representation of the postfix expression must be altered to support variable names and integer constants.]
- Modify the postfix evaluation algorithm to process multi-digit integer operands and variable name operands. Also, the algorithm should now implement the previously discussed “hook” so that SML instructions are produced rather than directly evaluating the expression. [*Hint*: Standard library function `strtok` can be used to locate each constant and variable in an expression, and constants can be converted from strings to integers using standard library function `atoi`.] [*Note*: The data representation of the postfix expression must be altered to support variable names and integer constants.]
- Build the compiler. Incorporate parts (b) and (c) for evaluating expressions in `let` statements. Your program should contain a function that performs the first pass of the compiler and a function that performs the second pass of the compiler. Both functions can call other functions to accomplish their tasks.

12.28 (*Optimizing the Simple Compiler*) When a program is compiled and converted into SML, a set of instructions is generated. Certain combinations of instructions often repeat themselves, usually in triplets called *productions*. A production normally consists of three instructions such as *load*, *add* and *store*. For example, Fig. 12.30 illustrates five of the SML instructions that were produced in the compilation of the program in Fig. 12.3. The first three instructions are the production that adds 1 to *y*. Note that instructions 06 and 07 store the accumulator value in temporary location 96, then load the value back into the accumulator so instruction 08 can store the value in location 98. Often a production is followed by a load instruction for the same location that was just stored. This code can be *optimized* by eliminating the store instruction and the subsequent load instruction that operate on the same memory location. This optimization would enable the Simpletron to execute the program faster because there are fewer instructions in this version. Figure 12.31 illustrates the optimized SML for the program of Fig. 12.3. Note that there are four fewer instructions in the optimized code—a memory-space savings of 25%.

04	+2098	(load)
05	+3097	(add)
06	+2196	(store)
07	+2096	(load)
08	+2198	(store)

Modify the compiler to provide an option for optimizing the Simpletron Machine Language code it produces. Manually compare the non-optimized code with the optimized code, and calculate the percentage reduction.

Simple program	SML location and instruction	Description
5 rem sum 1 to x	<i>none</i>	rem ignored
10 input x	00 +1099	read x into location 99
15 rem check y == x	<i>none</i>	rem ignored
20 if y == x goto 60	01 +2098	load y (98) into accumulator
	02 +3199	sub x (99) from accumulator
	03 +4211	branch to location 11 if zero
25 rem increment y	<i>none</i>	rem ignored
30 let y = y + 1	04 +2098	load y into accumulator
	05 +3097	add 1 (97) to accumulator
	06 +2198	store accumulator in y (98)
35 rem add y to total	<i>none</i>	rem ignored
40 let t = t + y	07 +2096	load t from location (96)
	08 +3098	add y (98) accumulator
	09 +2196	store accumulator in t (96)
45 rem loop y	<i>none</i>	rem ignored
5 rem sum 1 to x	<i>none</i>	rem ignored
10 input x	00 +1099	read x into location 99
15 rem check y == x	<i>none</i>	rem ignored
20 if y == x goto 60	01 +2098	load y (98) into accumulator
	02 +3199	sub x (99) from accumulator
	03 +4211	branch to location 11 if zero
25 rem increment y	<i>none</i>	rem ignored
30 let y = y + 1	04 +2098	load y into accumulator
	05 +3097	add 1 (97) to accumulator

Fig. 12.5 Optimized code for the program of Fig. 12.28.

Simple program	SML location and instruction	Description
	06 +2198	store accumulator in y (98)
35 rem add y to total	none	rem ignored
40 let t = t + y	07 +2096	load t from location (96)
	08 +3098	add y (98) accumulator
	09 +2196	store accumulator in t (96)
45 rem loop y	none	rem ignored
50 goto 20	10 +4001	branch to location 01
55 rem output result	none	rem ignored
60 print t	11 +1196	output t (96) to screen
99 end	12 +4300	terminate execution

Fig. 12.5 Optimized code for the program of Fig. 12.28.

12.29 (*Modifications to the Simple Compiler*) Perform the following modifications to the Simple compiler. Some of these modifications may also require modifications to the Simpletron Simulator program written in Exercise 7.19.

- Allow the modulus operator (%) to be used in `let` statements. Simpletron Machine Language must be modified to include a modulus instruction.
- Allow exponentiation in a `let` statement using `^` as the exponentiation operator. Simpletron Machine Language must be modified to include an exponentiation instruction.
- Allow the compiler to recognize uppercase and lowercase letters in Simple statements (e.g., 'A' is equivalent to 'a'). No modifications to the Simpletron Simulator are required.
- Allow `input` statements to read values for multiple variables such as `input x, y`. No modifications to the Simpletron Simulator are required.
- Allow the compiler to output multiple values in a single `print` statement such as `print a, b, c`. No modifications to the Simpletron Simulator are required.
- Add syntax checking capabilities to the compiler so error messages are output when syntax errors are encountered in a Simple program. No modifications to the Simpletron Simulator are required.
- Allow arrays of integers. No modifications to the Simpletron Simulator are required.
- Allow subroutines specified by the Simple commands `gosub` and `return`. Command `gosub` passes program control to a subroutine and command `return` passes control back to the statement after the `gosub`. This is similar to a function call in C. The same subroutine can be called from many `gosubs` distributed throughout a program. No modifications to the Simpletron Simulator are required.
- Allow repetition structures of the form

```
for x = 2 to 10 step 2
  rem Simple statements
next
```

- This `for` statement loops from 2 to 10 with an increment of 2. The `next` line marks the end of the body of the `for` line. No modifications to the Simpletron Simulator are required.
- Allow repetition structures of the form

```
for x = 2 to 10
  rem Simple statements
next
```

- This `for` statement loops from 2 to 10 with a default increment of 1. No modifications to the Simpletron Simulator are required.
- Allow the compiler to process string input and output. This requires the Simpletron Simulator to be modified to process and store string values. [*Hint*: Each Simpletron word can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the ASCII decimal equivalent of a character.] Add a machine language instruction that will print a string beginning at a certain Simpletron memory location. The first half of the word at that location is

a count of the number of characters in the string (i.e., the length of the string). Each succeeding half word contains one ASCII character expressed as two decimal digits. The machine language instruction checks the length and prints the string by translating each two-digit number into its equivalent character.

- n) Allow the compiler to process floating-point values in addition to integers. The Simpletron Simulator must also be modified to process floating-point values.

12.30 (*A Simple Interpreter*) An interpreter is a program that reads a high-level language program statement, determines the operation to be performed by the statement, and executes the operation immediately. The program is not converted into machine language first. Interpreters execute slowly because each statement encountered in the program must first be deciphered. If statements are contained in a loop, the statements are deciphered each time they are encountered in the loop. Early versions of the BASIC programming language were implemented as interpreters.

Write an interpreter for the Simple language discussed in Exercise 12.26. The program should use the infix-to-postfix converter developed in Exercise 12.12 and the postfix evaluator developed in Exercise 12.13 to evaluate expressions in a `let` statement. The same restrictions placed on the Simple language in Exercise 12.26 should be adhered to in this program. Test the interpreter with the Simple programs written in Exercise 12.26. Compare the results of running these programs in the interpreter with the results of compiling the Simple programs and running them in the Simpletron simulator built in Exercise 7.19.

13

The Preprocessor: Solutions

SOLUTIONS

13.4 Write a program that defines a macro with one argument to compute the volume of a sphere. The program should compute the volume for spheres of radius 1 to 10 and print the results in tabular format. The formula for the volume of a sphere is

$$\left(\frac{4.0}{3} \right) * \pi * r^3$$

where π is 3.14159.

ANS:

```
1  /* Exercise 13.4 Solution: sphere volume macro */
2  #include <stdio.h>
3
4  #define PI 3.14159 /* constant representing Pi */
5
6  /* define preprocessor directive sphere volume */
7  #define SPHEREVOLUME( r ) ( 4.0 / 3.0 * PI * ( r ) * ( r ) * ( r ) )
8
9  int main()
10 {
11     int i; /* loop counter */
12
13     /* print header */
14     printf( "%10s%10s\n", "Radius", "Volume" );
15
16     /* use sphere volume macro */
17     for ( i = 1; i <= 10; i++ ) {
18         printf( "%10d%10.3f\n", i, SPHEREVOLUME( i ) );
19     } /* end for */
20
21     return 0; /* indicate successful termination */
22
23 }
```

Radius	Volume
1	4.189
2	33.510
3	113.097
4	268.082
5	523.598
6	904.778
7	1436.754
8	2144.659
9	3053.625
10	4188.787

13.5 Write a program that produces the following output:

The sum of x and y is 13

The program should define macro SUM with two arguments, x and y, and use SUM to produce the output.

ANS:

```
1  /* Exercise 13.5 Solution */
2  #include <stdio.h>
3
4  /* macro to add two value */
5  #define SUM( x, y ) ( ( x ) + ( y ) )
6
7  int main()
8  {
9
10     /* display sum of x and y using macro SUM */
11     printf( "The sum of x and y is %d\n", SUM( 6, 7 ) );
12
13     return 0; /* indicate successful termination */
14
15 }
```

13.6 Write a program that defines and uses macro `MINIMUM2` to determine the smallest of two numeric values. Input the values from the keyboard.

ANS:

```
1  /* Exercise 13.6 Solution */
2  #include <stdio.h>
3
4  /* macro to determine smallest of two values */
5  #define MINIMUM2( x, y ) ( ( x ) < ( y ) ? ( x ) : ( y ) )
6
7  int main()
8  {
9      int a;    /* first integer */
10     int b;    /* second integer */
11     double c; /* first double */
12     double d; /* second double */
13
14     /* prompt user and read two integers */
15     printf( "Enter two integers: " );
16     scanf( "%d%d", &a, &b );
17
18     /* use macro MINIMUM to determine and display
19        smallest user entered integer */
20     printf( "The minimum of %d and %d is %d\n\n", a, b,
21            MINIMUM2( a,b ) );
22
23     /* prompt user and read two doubles */
24     printf( "Enter two doubles: " );
25     scanf( "%lf%lf", &c, &d );
26
27     /* use macro MINIMUM to determine and display
28        smallest user entered double */
29     printf( "The minimum of %.2f and %.2f is %.2f\n\n",
30            c, d, MINIMUM2( c,d ) );
31
32     return 0; /* indicate successful termination */
33 }
34 }
```

```
Enter two integers: 4 9
The minimum of 4 and 9 is 4

Enter two doubles: 45.7 13.2
The minimum of 45.70 and 13.20 is 13.20
```

13.7 Write a program that defines and uses macro `MINIMUM3` to determine the smallest of three numeric values. Macro `MINIMUM3` should use macro `MINIMUM2` defined in Exercise 13.6 to determine the smallest number. Input the values from the keyboard.

ANS:

```

1  /* Exercise 13.7 Solution */
2  #include <stdio.h>
3
4  /* macro to determine smallest of two values */
5  #define MINIMUM2( x, y ) ( ( x ) < ( y ) ? ( x ) : ( y ) )
6
7  /* macro that uses MINIMUM2 to determine smallest of three values */
8  #define MINIMUM3( u, v, w ) ( MINIMUM2( w, MINIMUM2( u, v ) ) )
9
10 int main()
11 {
12     int a;    /* first integer */
13     int b;    /* second integer */
14     int c;    /* third integer */
15     double d; /* first double */
16     double e; /* second double */
17     double f; /* third double */
18
19     /* prompt user and read three integers */
20     printf( "Enter three integers: " );
21     scanf( "%d%d%d", &a, &b, &c );
22
23     /* use macro MINIMUM3 to determine smallest
24        of three user input integers */
25     printf( "The minimum of %d, %d, and %d is %d\n\n",
26           a, b, c, MINIMUM3( a, b, c ) );
27
28     /* prompt user and read three doubles */
29     printf( "Enter three doubles: " );
30     scanf( "%lf%lf%lf", &d, &e, &f );
31
32     /* use macro MINIMUM3 to determine smallest
33        of three user input doubles */
34     printf( "The minimum of %.2f, %.2f, and %.2f is %.2f\n\n",
35           d, e, f, MINIMUM3( d, e, f ) );
36
37     return 0; /* indicate successful termination */
38
39 } /* end main */

```

```

Enter three integers: 7 2 10
The minimum of 7, 2, and 10 is 2

Enter three doubles: 4.9 93.2 1.3
The minimum of 4.90, 93.20, and 1.30 is 1.30

```

13.8 Write a program that defines and uses macro PRINT to print a string value.

ANS:

```

1  /* Exercise 13.8 Solution */
2  #include <stdio.h>
3
4  /* macro that prints its argument */
5  #define PRINT( string ) printf( "%s", ( string ) )
6
7  int main()
8  {
9      char text[ 20 ]; /* array to hold user input string */
10
11     /* prompt user and read string */
12     PRINT( "Enter a string: " );
13     scanf( "%s", text );
14
15     /* use macro to output string entered by user */
16     PRINT( "The string entered was: " );
17     PRINT( text );
18     PRINT( "\n" );
19
20     return 0; /* indicate successful termination */
21
22 } /* end main */

```

```

Enter a string: Hello
The string entered was: Hello

```

13.9 Write a program that defines and uses macro PRINTARRAY to print an array of integers. The macro should receive the array and the number of elements in the array as arguments.

ANS:

```

1  /* Exercise 13.9 Solution */
2  #include <stdio.h>
3
4  /* macro that prints an array of values */
5  #define PRINTARRAY( a, n ) for ( i = 0; i < ( n ); i++ ) \
6                          printf( "%d ", a[ i ] )
7
8  int main()
9  {
10     int i; /* defines i for use in PRINTARRAY */
11
12     /* initialize array to be printed */
13     int b[ 10 ] = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
14
15     printf( "The array values are:\n" );
16     PRINTARRAY( b, 10 ); /* print the array */
17
18     return 0; /* indicate successful termination */
19
20 } /* end main */

```

```

The array values are:
2 4 6 8 10 12 14 16 18 20

```


13.10 Write a program that defines and uses macro SUMARRAY to sum the values in a numeric array. The macro should receive the array and the number of elements in the array as arguments.

ANS:

```
1  /* Exercise 13.10 Solution */
2  #include <stdio.h>
3
4  /* macro that adds values of a numeric array */
5  #define SUMARRAY( a, n ) for ( i = 0; i < ( n ); i++ ) \
6                          sum += a[ i ]
7
8  int main()
9  {
10     int i;          /* loop counter */
11     int sum = 0;    /* sum of array elements */
12
13     /* initialize array whose values will be added */
14     int b[ 10 ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
15
16     /* use macro SUMARRAY to add elements of array */
17     SUMARRAY( b, 10 );
18     printf( "The sum of the elements of array b is %d\n", sum );
19
20     return 0; /* indicate successful termination */
21
22 } /* end main */
```

```
The sum of the elements of array b is 55
```

14

Other C Topics: Solutions

SOLUTIONS

14.2 Write a program that calculates the product of a series of integers that are passed to function `product` using a variable-length argument list. Test your function with several calls, each with a different number of arguments.

ANS:

```
1  /* Exercise 14.2 Solution */
2  #include <stdio.h>
3  #include <stdarg.h>
4
5  /* function with variable length argument list */
6  int sum( int i, ... );
7
8  int main()
9  {
10     int a = 1; /* values to sum */
11     int b = 2;
12     int c = 3;
13     int d = 4;
14     int e = 5;
15
16     /* display integer values */
17     printf( "%s%d, %s%d, %s%d, %s%d, %s%d\n", "a = ", a, "b = ",
18            b, "c = ", c, "d = ", d, "e = ", e );
19
20     /* call sum with different number of arguments in each call */
21     printf( "%s%d\n%s%d\n%s%d\n%s%d\n", "The sum of a and b is: ",
22            sum( 2, a, b ), "The sum of a, b, and c is: ", sum( 3, a, b, c ),
23            "The sum of a, b, c, and d is: ", sum( 4, a, b, c, d ),
24            "The sum of a, b, c, d, and e is: ", sum( 5, a, b, c, d, e ) );
25
26     return 0; /* indicate successful termination */
27 }
28 /* end main */
29
```

```

30 /* sums integers passed as arguments */
31 int sum( int i, ... )
32 {
33     int total = 0; /* sum of integers */
34     int j;        /* loop counter */
35     va_list ap;   /* variable length argument list */
36
37     va_start( ap, i ); /* invoke macro to access arguments */
38
39     /* calculate total */
40     for ( j = 1; j <= i; j++ ) {
41         total += va_arg( ap, int );
42     } /* end for */
43
44     va_end( ap ); /* perform termination housekeeping */
45
46     return total; /* return sum of arguments */
47
48 } /* end function sum */

```

```

a = 1, b = 2, c = 3, d = 4, e = 5
The sum of a and b is: 3
The sum of a, b, and c is: 6
The sum of a, b, c, and d is: 10
The sum of a, b, c, d, and e is: 15

```

14.3 Write a program that prints the command-line arguments of the program.

ANS:

```

1 /* Exercise 14.3 Solution */
2 #include <stdio.h>
3
4 int main( int argc, char *argv[] )
5 {
6     int i; /* loop counter */
7
8     printf( "The command line arguments are:\n" );
9
10    /* display arguments given to program at command line */
11    for ( i = 0; i < argc; i++ ) {
12        printf( "%s ", argv[ i ] );
13    } /* end for */
14
15    return 0; /* indicate successful termination */
16
17 } /* end main */

```

```

The command line arguments are:
C:\P14_3.exe arg1 arg2 arg3

```

14.4 Write a program that sorts an array of integers into ascending order or descending order. The program should use command-line arguments to pass either argument `-a` for ascending order or `-d` for descending order. [Note: This is the standard format for passing options to a program in UNIX.]

ANS:

The DOS command line `p14_4.exe -a < p14_4.dat` produces the first output shown below, and the DOS command line `p14_4.exe -d < p14_4.dat` produces the second output shown below. The data file `p14_4.dat` contains the values 8, 2, 1, 7, 5, 4, 9, 11, 19, and 13.

```

1  /* Exercise 14.4 Solution */
2  #include <stdio.h>
3
4  int main( int argc, char *argv[] )
5  {
6      int a[ 100 ]; /* array of integers from user */
7      int count;   /* count of integers entered */
8      int temp;    /* temporary integer for swapping */
9      int i;       /* loop counter */
10     int j;       /* loop counter */
11     int order;   /* sort in ascending or descending order */
12
13     /* tell user if improper arguments were passed */
14     if ( argc != 2 ) {
15         printf( "Usage: p14_4 -option\n" );
16     } /* end if */
17     else {
18
19         /* prompt user for integers to be sorted */
20         printf( "Enter up to 100 integers ( EOF to end input ): " );
21
22         /* store integers until 100 elements or EOF entered */
23         for ( count = 0; !feof( stdin ) && count < 100; count++ ) {
24             scanf( "%d", &a[ count ] );
25         } /* end for */
26
27         /* set order based on command-line argument */
28         order = ( argv[ 1 ][ 1 ] == 'd' ) ? 0 : 1;
29
30         /* loop through array and swap elements as needed */
31         for ( i = 1; i < count - 1; i++ ) {
32
33             for ( j = 0; j < count - 1; j++ ) {
34
35                 /* swap in ascending order if that option specified */
36                 if ( order == 1 ) {
37
38                     if ( a[ i ] < a[ j ] ) {
39                         temp = a[ i ];
40                         a[ i ] = a[ j ];
41                         a[ j ] = temp;
42                     } /* end if */
43
44                 } /* end if */
45                 else { /* swap in descending order */
46
47                     if ( a[ i ] > a[ j ] ) {
48                         temp = a[ i ];
49                         a[ i ] = a[ j ];
50                         a[ j ] = temp;
51                     } /* end if */

```

```

52
53         } /* end else */
54     } /* end for */
55 } /* end for */
56
57 } /* end for */
58
59 printf( "\n\nThe sorted array is:\n" );
60
61 /* display sorted array */
62 for ( i = 0; i < count - 1; i++ ) {
63     printf( "%d ", a[ i ] );
64 } /* end for */
65
66 printf( "\n" );
67 } /* end else */
68
69 return 0; /* indicate successful termination */
70
71 } /* end main */

```

```

The sorted array is:
1 2 4 5 7 8 9 11 13 19

```

```

The sorted array is:
19 13 11 9 8 7 5 4 2 1

```

14.5 Write a program that places a space between each character in a file. The program should first write the contents of the file being modified into a temporary file with spaces between each character, then copy the file back to the original file. This operation should overwrite the original contents of the file.

ANS:

```

1  /* Exercise 14.5 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      FILE *filePtr;          /* pointer to file being modified */
7      FILE *tempFilePtr;     /* temporary file pointer */
8      int c;                 /* current character */
9      char fileName[ 30 ];   /* name of file to be modified */
10
11     /* prompt user and read file name */
12     printf( "This program inserts spaces between each character\n"
13            "of a file. Enter a file to be modified: " );
14     scanf( "%s", fileName );
15
16     /* exit program if file cannot be opened */
17     if ( ( filePtr = fopen( fileName, "r+" ) ) != NULL ) {
18
19         /* exit program if temporary file cannot be opened */
20         if ( ( tempFilePtr = tmpfile() ) != NULL ) {
21             printf( "\nThe file before modification is:\n" );
22
23             /* read each character from file */
24             while ( ( c = getc( filePtr ) ) != EOF ) {
25                 putchar( c );

```

```

26         putchar( c, tempFilePtr ); /* put character in temp file */
27
28         /* write a space to temp file */
29         if ( c != '\n' ) {
30             putchar( ' ', tempFilePtr );
31         } /* end if */
32
33     } /* end while */
34
35     rewind( tempFilePtr ); /* rewind both file pointers */
36     rewind( filePtr );
37     printf( "\n\nThe file after modification is:\n" );
38
39     /* read each character from temp file */
40     while ( ( c = getc( tempFilePtr ) ) != EOF ) {
41         putchar( c );
42         putchar( c, filePtr ); /* rewrite character to file */
43     } /* end while */
44
45     } /* end if */
46     else {
47         printf( "Unable to open temporary file\n" );
48     } /* end else */
49
50     } /* end if */
51     else {
52         printf( "Unable to open %s\n", fileName );
53     } /* end else */
54
55     return 0; /* indicate successful termination */
56
57 } /* end main */

```

This program inserts spaces between each character of a file. Enter a file to be modified: test.dat

The file before modification is:
This is a test file for
exercise 14.5.

The file after modification is:
T h i s i s a t e s t f i l e f o r
e x e r c i s e 1 4 . 5 .

14.6 Read the manuals for your compiler to determine what signals are supported by the signal handling library (`signal.h`). Write a program that contains signal handlers for the standard signals `SIGABRT` and `SIGINT`. The program should test the trapping of these signals by calling function `abort` to generate a signal of type `SIGABRT` and by typing `<ctrl>c` to generate a signal of type `SIGINT`.

14.7 Write a program that dynamically allocates an array of integers. The size of the array should be input from the keyboard. The elements of the array should be assigned values input from the keyboard. Print the values of the array. Next, reallocate the memory for the array to 1/2 of the current number of elements. Print the values remaining in the array to confirm that they match the first half of the values in the original array.

ANS:

```
1  /* Exercise 14.7 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      int count; /* number of elements in array */
8      int i;     /* loop counter */
9      int *array; /* pointer to the array */
10
11     /* prompt user and read integer size of array */
12     printf( "This program dynamically allocates an array of integers.\n"
13            "Enter the number of elements in the array: " );
14     scanf( "%d", &count );
15
16     /* dynamically allocate memory */
17     array = calloc( count, sizeof( int ) );
18
19     /* initialize elements of array with user-entered data */
20     for ( i = 0; i < count; i++ ) {
21         printf( "Enter an integer: " );
22         scanf( "%d", &array[ i ] );
23     } /* end for */
24
25     printf( "\nThe elements of the array are:\n" );
26
27     /* display the original array */
28     for ( i = 0; i < count; i++ ) {
29         printf( "%d ", array[ i ] );
30     } /* end for */
31
32     /* reallocate to half the original size */
33     realloc( array, count / 2 * sizeof( int ) );
34
35     printf( "\n\nThe elements of the array after reallocation are:\n" );
36
37     /* display array after cut in half */
38     for ( i = 0; i < count / 2; i++ ) {
39         printf( "%d ", array[ i ] );
40     } /* end for */
41
42     return 0; /* indicate successful termination */
43
44 } /* end main */
```

```
This program dynamically allocates an array of integers.
```

```
Enter the number of elements in the array: 10
```

```
Enter an integer: 1
```

```
Enter an integer: 2
```

```
Enter an integer: 3
```

```
Enter an integer: 4
```

```
Enter an integer: 5
```

```
Enter an integer: 6
```

```
Enter an integer: 7
```

```
Enter an integer: 8
```

```
Enter an integer: 9
```

```
Enter an integer: 10
```

```
The elements of the array are:
```

```
1 2 3 4 5 6 7 8 9 10
```

```
The elements of the array after reallocation are:
```

```
1 2 3 4 5
```


14.8 Write a program that takes two command-line arguments that are file names, reads the characters from the first file one at a time and writes the characters in reverse order to the second file.

ANS:

```

1  /* Exercise 14.8 Solution */
2  #include <stdio.h>
3
4  /* function prototype */
5  void reverseFile( FILE *inPtr, FILE *outPtr );
6
7  int main( int argc, int *argv[] )
8  {
9      FILE *inFilePtr; /* input file pointer */
10     FILE *outFilePtr; /* output file pointer */
11
12     /* tell user if invalid arguments */
13     if ( argc != 3 ) {
14         printf( "Usage: copy infile outfile\n" );
15     } /* end if */
16     else {
17
18         /* exit program if input file cannot be opened */
19         if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {
20
21             /* exit program if output file cannot be opened */
22             if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
23                 reverseFile( inFilePtr, outFilePtr );
24             } /* end if */
25             else {
26                 printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
27             } /* end else */
28
29         } /* end if */
30         else {
31             printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
32         } /* end else */
33
34     } /* end else */
35
36     return 0; /* indicate successful termination */
37
38 } /* end main */
39
40 /* function that writes characters in reverse order */
41 void reverseFile( FILE *inPtr, FILE *outPtr )
42 {
43     int c; /* current character */
44
45     /* if not end of file */
46     if ( ( c = fgetc( inPtr ) ) != EOF ) {
47         reverseFile( inPtr, outPtr );
48     } /* end if */
49
50     fputc( c, outPtr ); /* write character to output file */
51 } /* end function reverseFile */

```

```
.stnemugra enil dnammoc eht rof ecnetnes tset a si siht
```

14.9 Write a program that uses `goto` statements to simulate a nested looping structure that prints a square of asterisks as follows:

```
*****  
*   *  
*   *  
*   *  
*   *  
*****
```

The program should use only the following three `printf` statements:

```
printf( "*" );  
printf( "  " );  
printf( "\n" );
```

ANS:

```
1  /* Exercise 14.9 Solution */
2  #include <stdio.h>
3
4  int main()
5  {
6      int size;      /* length of square sides */
7      int row = 0;   /* number of rows */
8      int col;      /* number of columns */
9
10     /* obtain length of side of square from user */
11     printf( "Enter the side length of the square: " );
12     scanf( "%d", &size );
13
14     start: /* label */
15         ++row;
16         printf( "\n" );
17
18         /* if all rows have been made end program */
19         if ( row > size ) {
20             goto end;
21         } /* end if */
22
23         col = 1; /* set column variable to first character of line */
24
25         innerLoop: /* label */
26
27             /* if all columns have been displayed return to top of loop */
28             if ( col > size ) {
29                 goto start;
30             } /* end if */
31
32             /* display stars and spaces in appropriate positions */
33             if ( row == 1 || row == size || col == 1 || col == size ) {
34                 printf( "*" );
35             } /* end if */
36             else {
37                 printf( " " );
38             } /* end else */
39
40             ++col; /* increment column */
41             goto innerLoop; /* continue displaying columns */
42
43         end: /* label */
44
45         return 0; /* indicate successful termination */
46
47     } /* end main */
```

15

C++ as a “Better C”: Solutions

SOLUTIONS

15.5 Write a C++ program that uses an inline function `circleArea` to prompt the user for the radius of a circle and to calculate and print the area of that circle.

ANS:

```
1 // Exercise 15.5 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6 using std::cin;
7
8 double pi = 3.14159; // global variable
9
10 inline double circleArea( double r ) { return pi * r * r; }
11
12 int main()
13 {
14     double radius;
15
16     cout << "Enter the radius of the circle: ";
17     cin >> radius;
18     cout << "The area of the circle is " << circleArea( radius ) << endl;
19
20     return 0;
21 }
```

```
Enter the radius of the circle: 10
The area of the circle is 314.159
```

15.6 Write a complete C++ program with the two alternate functions specified below, of which each simply triples the variable `count` defined in `main`. Then compare and contrast the two approaches. These two functions are

- Function `tripleCallByValue` that passes a copy of `count` call-by-value, triples the copy and returns the new value.
- Function `tripleByReference` that passes `count` with true call-by-reference via a reference parameter and triples the original copy of `count` through its alias (i.e., the reference parameter).

ANS:

```

1 // Exercise 15.6 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6 using std::cin;
7
8 int tripleCallByValue( int );
9 void tripleByReference( int & );
10
11 int main()
12 {
13     int value, &valueRef = value;
14
15     cout << "Enter an integer: ";
16     cin >> value;
17
18     cout << "\nValue before call to tripleCallByValue() is: "
19           << value << "\nValue returned from tripleCallByValue() is: "
20           << tripleCallByValue( value )
21           << "\nValue (in main) after tripleCallByValue() is: " << value
22           << "\n\nValue before call to tripleByReference() is: "
23           << value << '\n';
24
25     tripleByReference( valueRef );
26
27     cout << "Value (in main) after call to tripleByReference() is: "
28           << value << endl;
29
30     return 0;
31 }
32
33 int tripleCallByValue( int valueCopy )
34 {
35     return valueCopy *= 3;
36 }
37
38 void tripleByReference( int &aliasRef )
39 {
40     aliasRef *= 3;
41 }

```

```

Enter an integer: 8

Value before call to tripleCallByValue() is: 8
Value returned from tripleCallByValue() is: 24
Value (in main) after tripleCallByValue() is: 8

Value before call to tripleByReference() is: 8
Value (in main) after call to tripleByReference() is: 24

```

15.7 What is the purpose of the unary scope resolution operator?

ANS: The unary scope resolution operator is used to access a global variable. In particular, the unary scope resolution operator is useful when a global variable needs to be accessed and a local variable has the same name.

15.8 Write a program that uses a function template called `min` to determine the smaller of two arguments. Test the program using integer, character and floating-point number pairs.

ANS:

```
1 // Exercise 15.8 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6
7 template < class T >
8 void minimum( T value1, T value2 ) // find the smallest value
9 {
10     if ( value1 > value2 )
11         cout << value2 << " is smaller than " << value1;
12     else
13         cout << value1 << " is smaller than " << value2;
14
15     cout << endl;
16 }
17
18 int main()
19 {
20     minimum( 7, 54 ); // integers
21     minimum( 4.35, 8.46 ); // doubles
22     minimum( 'g', 'T' ); // characters
23
24     return 0;
25 }
```

```
7 is smaller than 54
4.35 is smaller than 8.46
T is smaller than g
```

15.9 Write a program that uses a function template called `max` to determine the largest of three arguments. Test the program using integer, character and floating-point number pairs.

ANS:

```
1 // Exercise 15.9 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6
7 template < class T >
8 void max( T value1, T value2, T value3 ) // find the largest value
9 {
10     if ( value1 > value2 && value1 > value3 )
11         cout << value1 << " is greater than " << value2
12             << " and " << value3;
13     else if ( value2 > value1 && value2 > value3 )
14         cout << value2 << " is greater than " << value1
15             << " and " << value3;
16     else
17         cout << value3 << " is greater than " << value1
18             << " and " << value2;
19
20     cout << endl;
21 }
22
23 int main()
24 {
25     max( 7, 5, 2 );           // integers
26     max( 9.35, 8.461, 94.3 ); // doubles
27     max( '!', 'T', '$' );    // characters
28
29     return 0;
30 }
```

```
7 is greater than 5 and 2
94.3 is greater than 9.35 and 8.461
T is greater than ! and $
```

15.10 Determine whether the following program segments contain errors. For each error, explain how it can be corrected. [*Note:* For a particular program segment, it is possible that no errors are present in the segment.]

a)

```
template < class A >
int sum( int num1, int num2, int num3 )
{ return num1 + num2 + num3; }
```

ANS: The function return type and parameter types should be A.

b)

```
void printResults( int x, int y )
{
    cout << "The sum is " << x + y << '\n';
    return x + y;
}
```

ANS: The function specifies a void return type and attempts to return a value. Two possible solutions: (1) change void to int or (2) remove the line return x + y;.

c)

```
template < A >
A product( A num1, A num2, A num3 )
{
    return num1 * num2 * num3;
}
```

ANS: The keyword class is needed in the template declaration template <class A>.

d)

```
double cube( int );
int cube( int );
```

ANS: The signatures are not different. Overloaded functions must have different signatures meaning that the name and parameter list must be different. If only return types differ, the compiler generates an error message.

16

C++ Classes and Data Abstraction: Solutions

SOLUTIONS

16.3 What is the purpose of the scope resolution operator?

ANS: The scope resolution operator is used to specify the class to which a function belongs. It also resolves the ambiguity caused by multiple classes having member functions of the same name.

16.4 Provide a constructor that is capable of using the current time from the `time` function—declared in the C Standard Library header `ctime`—to initialize an object of the `Time` class.

ANS:

```
1 // p16_4.H
2 #ifndef p16_4_H
3 #define p16_4_H
4
5 class Time {
6 public:
7     Time();
8     void setHour( int );
9     void setMinute( int );
10    void setSecond( int );
11    int getHour( void ) const;
12    int getMinute( void ) const;
13    int getSecond( void ) const;
14    void printStandard( void ) const;
15 private:
16    int hour;
17    int minute;
18    int second;
19 };
20
21 #endif
```

```
22 // p16_4.cpp
23 // member function definitions for p16_4.cpp
24 #include <iostream.h>
25
26 using std::cout;
27
```

```

28 #include <ctime>
29 #include "p16_4.h"
30
31 Time::Time()
32 {
33     long int totalTime;           // time in seconds since 1970
34     int currentYear = 1994 - 1970; // current year
35     double totalYear;           // current time in years
36     double totalDay;           // days since beginning of year
37     double day;                // current time in days
38     long double divisor;       // conversion divisor
39     int timeShift = 7;         // time returned by time() is
40                               // given as the number of seconds
41                               // elapsed since 1/1/70 GMT.
42                               // Depending on the time zone
43                               // you are in, you must shift
44                               // the time by a certain
45                               // number of hours. For this
46                               // problem, 7 hours is the
47                               // current shift for EST.
48
49     totalTime = time( NULL );
50     divisor = ( 60.0 * 60.0 * 24.0 * 365.0 );
51     totalYear = totalTime / divisor - currentYear;
52     totalDay = 365 * totalYear; // leap years ignored
53     day = totalDay - ( int ) totalDay;
54
55     setHour( day * 24 + timeShift );
56     setMinute( ( day * 24 - ( int )( day * 24 ) ) * 60 );
57     setSecond( ( minute * 60 - ( int )( minute * 60 ) ) * 60 );
58 }
59
60 void Time::setHour( int h ) { hour = ( h >= 0 && h < 24 ) ? h : 0; }
61
62 void Time::setMinute( int m ) { minute = ( m >= 0 && m < 60 ) ? m : 0; }
63
64 void Time::setSecond( int s ) { second = ( s >= 0 && s < 60 ) ? s : 0; }
65
66 int Time::getHour() const { return hour; }
67
68 int Time::getMinute() const { return minute; }
69
70 int Time::getSecond() const { return second; }
71
72 void Time::printStandard() const
73 {
74     cout << ( ( hour % 12 == 0 ) ? 12 : hour % 12 ) << ":"
75     << ( minute < 10 ? "0" : "" ) << minute << ":"
76     << ( second < 10 ? "0" : "" ) << second
77     << ( hour < 12 ? " AM" : " PM" );
78 }

```

```

79 // driver for p16_4.cpp
80 #include "p16_4.h"
81
82 int main( void )
83 {
84     Time t;
85
86     t.printStandard();
87
88     return 0;
89 }

```

12:15:00 PM

16.5 Create a class called `Complex` for performing arithmetic with complex numbers. Write a driver program to test your class. Complex numbers have the form

$$\text{realPart} + \text{imaginaryPart} * i$$

where i is

$$\sqrt{-1}$$

Use `double` variables to represent the private data of the class. Provide a constructor function that enables an object of this class to be initialized when it is declared. The constructor should contain default values in case no initializers are provided. Provide public member functions for each of the following:

- Addition of two `Complex` numbers: The real parts are added together and the imaginary parts are added together.
- Subtraction of two `Complex` numbers: The real part of the right operand is subtracted from the real part of the left operand and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.
- Printing `Complex` numbers in the form (a, b) where a is the real part and b is the imaginary part.

ANS:

```

1 // 16.5.H
2 #ifndef p16_5_H
3 #define p16_5_H
4
5 class Complex {
6 public:
7     Complex( double = 0.0, double = 0.0 ); // default constructor
8     void addition( const Complex & );
9     void subtraction( const Complex & );
10    void printComplex( void ) const;
11    void initialize( const double, const double );
12 private:
13    double realPart;
14    double imaginaryPart;
15 };
16
17 #endif

```

```

18 // p16_5M.cpp
19 // member function definitions for p16_5.cpp
20 #include <iostream>
21
22 using std::cout;
23
24 #include "p16_5.h"
25
26 Complex::Complex( double real, double imaginary )
27 {
28     initialize( real, imaginary );
29 }
30
31 void Complex::addition( const Complex &a )
32 {
33     realPart += a.realPart;
34     imaginaryPart += a.imaginaryPart;
35 }
36
37 void Complex::subtraction( const Complex &s )
38 {
39     realPart -= s.realPart;
40     imaginaryPart -= s.imaginaryPart;
41 }
42
43 void Complex::printComplex( void ) const
44 {
45     cout << "( " << realPart << ", " << imaginaryPart << " )";
46 }

```

```
47
48 void Complex::initialize( const double rp, const double ip )
49 {
50     realPart = rp;
51     imaginaryPart = ip;
52 }

53 // driver for p16_5.cpp
54 #include <iostream>
55
56 using std::cout;
57 using std::endl;
58
59 #include "p16_5.h"
60
61 int main( void )
62 {
63     Complex b( 1, 7 ), c( 9, 2 );
64
65     b.printComplex();
66     cout << " + ";
67     c.printComplex();
68     cout << " = ";
69     b.addition( c );
70     b.printComplex();
71
72     cout << "\n";
73     b.initialize( 10,1 ); // reset realPart and imaginaryPart
74     c.initialize( 11,5 );
75     b.printComplex();
76     cout << " - ";
77     c.printComplex();
78     cout << " = ";
79     b.subtraction( c );
80     b.printComplex();
81     cout << endl;
82
83     return 0;
84 }
```

```
( 1, 7 ) + ( 9, 2 ) = ( 10, 9 )
( 10, 1 ) - ( 11, 5 ) = ( -1, -4 )
```

16.6 Create a class called `Rational` for performing arithmetic with fractions. Write a driver program to test your class.

Use integer variables to represent the `private` data of the class—the numerator and the denominator. Provide a constructor function that enables an object of this class to be initialized when it is declared. The constructor should contain default values in case no initializers are provided and should store the fraction in reduced form (i.e., the fraction

$$\frac{2}{4}$$

would be stored in the object as 1 in the numerator and 2 in the denominator). Provide `public` member functions for each of the following:

- Addition of two `Rational` numbers. The result should be stored in reduced form.
- Subtraction of two `Rational` numbers. The result should be stored in reduced form.
- Multiplication of two `Rational` numbers. The result should be stored in reduced form.
- Division of two `Rational` numbers. The result should be stored in reduced form.
- Printing `Rational` numbers in the form `a/b` where `a` is the numerator and `b` is the denominator.
- Printing `Rational` numbers in floating-point format.

ANS:

```

1 // P16_6.H
2 #ifndef P16_6_H
3 #define P16_6_H
4
5 class RationalNumber {
6 public:
7     RationalNumber( int = 0, int = 1 ); // default constructor
8     RationalNumber addition( const RationalNumber& );
9     RationalNumber subtraction( const RationalNumber& );
10    RationalNumber multiplication( const RationalNumber& );
11    RationalNumber division( RationalNumber& );
12    void printRational( void ) const;
13    void printRationalF( void ) const;
14 private:
15     int numerator;
16     int denominator;
17     void reduction( void );
18 };
19
20 #endif

```

```

21 // P16_6M.cpp
22 // member function definitions for p16_6.cpp
23 #include <iostream>
24
25 using std::cout;
26
27 #include "p16_6.h"
28
29 RationalNumber::RationalNumber( int n, int d )
30 {
31     numerator = n;
32     denominator = d;
33 }
34
35 RationalNumber RationalNumber::addition( const RationalNumber &a )
36 {
37     RationalNumber t;
38
39     t.numerator = a.numerator * denominator + a.denominator * numerator;
40
41     t.denominator = a.denominator * denominator;
42     t.reduction();
43
44     return t;
45 }

```

```
46
47 RationalNumber RationalNumber::subtraction( const RationalNumber &s )
48 {
49     RationalNumber t;
50
51     t.numerator = s.denominator * numerator - denominator * s.numerator;
52     t.denominator = s.denominator * denominator;
53     t.reduction();
54
55     return t;
56 }
57
58 RationalNumber RationalNumber::multiplication( const RationalNumber &m )
59 {
60     RationalNumber t;
61
62     t.numerator = m.numerator * numerator;
63     t.denominator = m.denominator * denominator;
64     t.reduction();
65
66     return t;
67 }
68
69 RationalNumber RationalNumber::division( RationalNumber &v )
70 {
71     RationalNumber t;
72
73     t.numerator = v.denominator * numerator;
74     t.denominator = denominator * v.numerator;
75     t.reduction();
76
77     return t;
78 }
79
80 void RationalNumber::printRational( void ) const
81 {
82     if ( denominator == 0 )
83         cout << "\nDIVIDE BY ZERO ERROR!!!\n";
84     else if ( numerator == 0 )
85         cout << 0;
86     else
87         cout << numerator << "/" << denominator;
88 }
89
90 void RationalNumber::printRationalF( void ) const
91 { cout << ( double ) numerator / denominator; }
92
93 void RationalNumber::reduction( void )
94 {
95     int largest;
96
97     largest = numerator > denominator ? numerator : denominator;
98
99     int gcd = 0; // greatest common divisor
100    for ( int loop = 2; loop <= largest; loop++ )
101        if ( numerator % loop == 0 && denominator % loop == 0 )
102            gcd = loop;
103
104    if ( gcd != 0 ) {
105        numerator /= gcd;
106        denominator /= gcd;
107    }
108 }
```

```
109 // driver for P16_6.cpp
110 #include <iostream>
111
112 using std::cout;
113
114 #include "p16_6.h"
115
116 int main( void )
117 {
118     RationalNumber c( 1,3 ), d( 7,8 ), x;
119
120     c.printRational();
121     cout << " + ";
122     d.printRational();
123     x = c.addition( d );
124     cout << " = ";
125     x.printRational();
126     cout << "\n";
127     x.printRational();
128     cout << " = ";
129     x.printRationalF();
130     cout << "\n\n";
131
132     c.printRational();
133     cout << " - ";
134     d.printRational();
135     x = c.subtraction( d );
136     cout << " = ";
137     x.printRational();
138     cout << "\n";
139     x.printRational();
140     cout << " = ";
141     x.printRationalF();
142     cout << "\n\n";
143
144     c.printRational();
145     cout << " x ";
146     d.printRational();
147     x = c.multiplication( d );
148     cout << " = ";
149     x.printRational();
150     cout << "\n";
151     x.printRational();
152     cout << " = ";
153     x.printRationalF();
154     cout << "\n\n";
155
156     c.printRational();
157     cout << " / ";
158     d.printRational();
159     x = c.division( d );
160     cout << " = ";
161     x.printRational();
162     cout << "\n";
163     x.printRational();
164     cout << " = ";
165     x.printRationalF();
166     cout << "\n";
167
168     return 0;
169 }
```


$$\begin{aligned} 1/3 + 7/8 &= 29/24 \\ 29/24 &= 1.20833 \end{aligned}$$

$$\begin{aligned} 1/3 - 7/8 &= -13/24 \\ -13/24 &= -0.541667 \end{aligned}$$

$$\begin{aligned} 1/3 \times 7/8 &= 7/24 \\ 7/24 &= 0.291667 \end{aligned}$$

$$\begin{aligned} 1/3 / 7/8 &= 8/21 \\ 8/21 &= 0.380952 \end{aligned}$$

16.7 Create a class `Rectangle`. The class has attributes `length` and `width`, each of which defaults to 1. It has member functions that calculate the perimeter and the area of the rectangle. It has *set* and *get* functions for both `length` and `width`. The *set* functions should verify that `length` and `width` are each floating-point numbers larger than 0.0 and less than 20.0.

ANS:

```
1 // P16_7.H
2 #ifndef P16_7_H
3 #define P16_7_H
4
5 class Rectangle {
6 public:
7     Rectangle( double = 1.0, double = 1.0 );
8     double perimeter( void );
9     double area( void );
10    void setWidth( double w );
11    void setLength( double l );
12    double getWidth( void );
13    double getLength( void );
14 private:
15    double length;
16    double width;
17 };
18
19 #endif

```

```
20 // P16_7M.cpp
21 // member function definitions for p16_7.cpp
22
23 #include "p16_7.h"
24
25 Rectangle::Rectangle( double w, double l )
26 {
27     setWidth(w);
28     setLength(l);
29 }
30
31 double Rectangle::perimeter( void )
32 {
33     return 2 * ( width + length );
34 }
35
36 double Rectangle::area( void )
37 {
38     return width * length;
39 }
40
41 void Rectangle::setWidth( double w )
42 {
43     width = w > 0 && w < 20.0 ? w : 1.0;
44 }
45
46 void Rectangle::setLength( double l )
47 {
48     length = l > 0 && l < 20.0 ? l : 1.0;
49 }
50
51 double Rectangle::getWidth( void ) { return width; }
52
53 double Rectangle::getLength( void ) { return length; }
```

```
54 // driver for p16_7.cpp
55 #include <iostream>
56
57 using std::cout;
58 using std::endl;
59 using std::ios;
60
61 #include <iomanip>
62
63 using std::setprecision;
64 using std::setiosflags;
65
66 #include "p16_7.h"
67
68 int main()
69 {
70     Rectangle a, b( 4.0, 5.0 ), c( 67.0, 888.0 );
71
72     cout << setiosflags( ios::fixed | ios::showpoint );
73     cout << setprecision( 1 );
74
75     // output Rectangle a
76     cout << "a: length = " << a.getLength()
77         << "; width = " << a.getWidth()
78         << "; perimeter = " << a.perimeter() << "; area = "
16.8     << a.area() << '\n';
79
80     // output Rectangle b
81     cout << "b: length = " << b.getLength()
82         << "; width = " << b.getWidth()
83         << "; perimeter = " << b.perimeter() << "; area = "
84         << b.area() << '\n';
85
86     // output Rectangle c; bad values attempted
87     cout << "c: length = " << c.getLength()
88         << "; width = " << c.getWidth()
89         << "; perimeter = " << c.perimeter() << "; area = "
90         << c.area() << endl;
91
92     return 0;
93 }
```

```
a: length = 1.0; width = 1.0; perimeter = 4.0; area = 1.0
b: length = 5.0; width = 4.0; perimeter = 18.0; area = 20.0
c: length = 1.0; width = 1.0; perimeter = 4.0; area = 1.0
```

16.8 Create a more sophisticated `Rectangle` class than the one you created in Exercise 16.7. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a `set` function that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single x or y coordinate larger than 20.0. The `set` function also verifies that the supplied coordinates do, in fact, specify a rectangle. Member functions calculate the `length`, `width`, `perimeter` and `area`. The `length` is the larger of the two dimensions. Include a predicate function `square` that determines if the rectangle is a square.

ANS:

```

1 // P16_8.H
2 #ifndef P16_8_H
3 #define P16_8_H
4
5 class Rectangle {
6 public:
7     Rectangle( double *, double *, double *, double * );
8     void setCoord( double *, double *, double *, double * );
9     void perimeter( void );
10    void area( void );
11    void square( void );
12 private:
13    double point1[ 2 ];
14    double point2[ 2 ];
15    double point3[ 2 ];
16    double point4[ 2 ];
17 };
18
19 #endif

20 // P16_8M.cpp
21 // member function definitions for p16_8.cpp
22 #include <iostream>
23
24 using std::cout;
25 using std::ios;
26
27 #include <iomanip>
28
29 using std::setprecision;
30 using std::setiosflags;
31 using std::resetiosflags;
32
33 #include <cmath>
34
35 #include "p16_8.h"
36
37 Rectangle::Rectangle( double *a, double *b, double *c, double *d )
38     { setCoord( a, b, c, d ); }
39
40 void Rectangle::setCoord( double *p1, double *p2, double *p3, double *p4 )
41 {
42     // Arrangement of points
43     // p4.....p3
44     // .          .
45     // .          .
46     // p1.....p2
47
48     const int x = 0, y = 1; // added for clarity
49
50     // validate all points
51     point1[ x ] = ( p1[ x ] > 20.0 || p1[ x ] < 0.0 )? 0.0 : p1[ x ];
52     point1[ y ] = ( p1[ y ] > 20.0 || p1[ y ] < 0.0 )? 0.0 : p1[ y ];
53     point2[ x ] = ( p2[ x ] > 20.0 || p2[ x ] < 0.0 )? 0.0 : p2[ x ];
54     point2[ y ] = ( p2[ y ] > 20.0 || p2[ y ] < 0.0 )? 0.0 : p2[ y ];
55     point3[ x ] = ( p3[ x ] > 20.0 || p3[ x ] < 0.0 )? 0.0 : p3[ x ];
56     point3[ y ] = ( p3[ y ] > 20.0 || p3[ y ] < 0.0 )? 0.0 : p3[ y ];
57     point4[ x ] = ( p4[ x ] > 20.0 || p4[ x ] < 0.0 )? 0.0 : p4[ x ];

```

```

58     point4[ y ] = ( p4[ y ] > 20.0 || p4[ y ] < 0.0 )? 0.0 : p4[ y ];
59
60     // verify that points form a rectangle
61     if ( p1[ y ] == p2[ y ] && p1[ x ] ==
62         p4[ x ] && p2[ x ] == p3[ x ] && p3[ y ] == p4[ y ] )
63     {
64
65         perimeter();
66         area();
67         square();
68     }
69     else
70         cout << "Coordinates do not form a rectangle!\n";
71 }
72
73 void Rectangle::perimeter( void )
74 {
75     double l = fabs( point4[ 1 ] - point1[ 1 ] ),
76           w = fabs( point2[ 0 ] - point1[ 0 ] );
77
78     cout << setiosflags( ios::fixed | ios::showpoint )
79           << "length = " << setprecision( 1 ) << ( l > w ? l : w )
80           << '\t' << "width = " << ( l > w ? w : l )
81           << "\nThe perimeter is: " << 2 * ( w + l ) << '\n'
82           << resetiosflags( ios::fixed | ios::showpoint );
83 }
84
85 void Rectangle::area( void )
86 {
87     double l = fabs( point4[ 1 ] - point1[ 1 ] ),
88           w = fabs( point2[ 0 ] - point1[ 0 ] );
89
90     cout << setiosflags( ios::fixed | ios::showpoint )
91           << "The area is: " << setprecision( 1 ) << w * l
92           << resetiosflags( ios::fixed | ios::showpoint )
93           << "\n\n";
94 }
95
96 void Rectangle::square( void )
97 {
98     const int x = 0, y = 1;    // added for clarity
99
100    if ( fabs( point4[ y ] - point1[ y ] ) ==
101        fabs( point2[ x ] - point1[ x ] ) )
102        cout << "The rectangle is a square.\n\n";
103 }

```

```

104 // driver for p16_8.cpp
105
106 #include "p16_8.h"
107
108 int main()
109 {
110     double w[ 2 ] = { 1.0, 1.0 }, x[ 2 ] = { 5.0, 1.0 },
111           y[ 2 ] = { 5.0, 3.0 }, z[ 2 ] = { 1.0, 3.0 },
112           j[ 2 ] = { 0.0, 0.0 }, k[ 2 ] = { 1.0, 0.0 },
113           m[ 2 ] = { 1.0, 1.0 }, n[ 2 ] = { 0.0, 1.0 },
114           v[ 2 ] = { 99.0, -2.3 };
115     Rectangle a( z, y, x, w ), b( j, k, m, n ),
116              c( w, x, m, n ), d( v, x, y, z );
117
118     return 0;
119 }

```

```
length = 4.0    width = 2.0  
The perimeter is: 12.0  
The area is: 8.0
```

```
length = 1.0    width = 1.0  
The perimeter is: 4.0  
The area is: 1.0
```

```
The rectangle is a square.
```

```
Coordinates do not form a rectangle!  
Coordinates do not form a rectangle!
```

16.9 Modify the `Rectangle` class of Exercise 16.8 to include a `draw` function that displays the rectangle inside a 25-by-25 box enclosing the portion of the first quadrant in which the rectangle resides. Include a `setFillCharacter` function to specify the character out of which the body of the rectangle will be drawn. Include a `setPerimeterCharacter` function to specify the character that will be used to draw the border of the rectangle. If you feel ambitious, you might include functions to scale the size of the rectangle, rotate it and move it around within the designated portion of the first quadrant.

ANS:

```

1 // P16_9.H
2 #ifndef P16_9_H
3 #define P16_9_H
4
5 class Rectangle {
6 public:
7     Rectangle( double *, double *, double *, double *, char, char );
8     void setCoord( double *, double *, double *, double * );
9     void perimeter( void );
10    void area( void );
11    void draw( void );
12    void square( void );
13    void setFillCharacter( char c ) { fillChar = c; }
14    void setPerimeterCharacter( char c ) { periChar = c; }
15    bool isValid( void ) { return valid; }
16    void setValid( bool v ) { valid = v; }
17 private:
18    double point1[ 2 ];
19    double point2[ 2 ];
20    double point3[ 2 ];
21    double point4[ 2 ];
22    char fillChar;
23    char periChar;
24    bool valid;
25 };
26
27 #endif

```

```

28 // P16_9M.cpp
29 // member function definitions for p16_9.cpp
30 #include <iostream>
31
32 using std::cout;
33 using std::ios;
34
35 #include <iomanip>
36
37 using std::setprecision;
38 using std::setiosflags;
39 using std::resetiosflags;
40
41 #include <cmath>
42
43 #include "p16_9.h"
44
45 Rectangle::Rectangle( double *a, double *b, double *c, double *d,
46                     char x, char y )
47 {
48     setCoord( a, b, c, d );
49     setFillCharacter( x );
50     setPerimeterCharacter( y );
51 }
52
53 void Rectangle::setCoord( double *p1, double *p2,
54                         double *p3, double *p4 )
55 {
56     // Arrangement of points
57     // p4.....p3

```

```

58 // .      .
59 // .      .
60 // p1.....p2
61
62 const int x = 0, y = 1; // added for clarity
63
64 // validate all points
65 point1[ x ] = ( p1[ x ] > 20.0 || p1[ x ] < 0.0 )? 0.0 : p1[ x ];
66 point1[ y ] = ( p1[ y ] > 20.0 || p1[ y ] < 0.0 )? 0.0 : p1[ y ];
67 point2[ x ] = ( p2[ x ] > 20.0 || p2[ x ] < 0.0 )? 0.0 : p2[ x ];
68 point2[ y ] = ( p2[ y ] > 20.0 || p2[ y ] < 0.0 )? 0.0 : p2[ y ];
69 point3[ x ] = ( p3[ x ] > 20.0 || p3[ x ] < 0.0 )? 0.0 : p3[ x ];
70 point3[ y ] = ( p3[ y ] > 20.0 || p3[ y ] < 0.0 )? 0.0 : p3[ y ];
71 point4[ x ] = ( p4[ x ] > 20.0 || p4[ x ] < 0.0 )? 0.0 : p4[ x ];
72 point4[ y ] = ( p4[ y ] > 20.0 || p4[ y ] < 0.0 )? 0.0 : p4[ y ];
73
74 // verify that points form a rectangle
75 if ( point1[ y ] == point2[ y ] && point1[ x ] == point4[ x ] &&
76     point2[ x ] == point3[ x ] && point3[ y ] == point4[ y ] ) {
77
78     perimeter();
79     area();
80     square();
81     setValid( true ); // valid set of points
82 }
83 else {
84     cout << "Coordinates do not form a rectangle!\n";
85     setValid( false ); // invalid set of points
86 }
87 }
88
89 void Rectangle::perimeter( void )
90 {
91     double l = fabs( point4[ 1 ] - point1[ 1 ] ),
92            w = fabs( point2[ 0 ] - point1[ 0 ] );
93
94     cout << setiosflags( ios::fixed | ios::showpoint )
95           << "length = " << setprecision( 1 ) << ( l > w ? l : w )
96           << "\twidth = " << ( l > w ? w : l )
97           << "\nThe perimeter is: " << 2 * ( w + l ) << '\n'
98           << resetiosflags( ios::fixed | ios::showpoint );
99 }
100
101 void Rectangle::area( void )
102 {
103     double l = fabs( point4[ 1 ] - point1[ 1 ] ),
104            w = fabs( point2[ 0 ] - point1[ 0 ] );
105
106     cout << setiosflags( ios::fixed | ios::showpoint )
107           << "The area is: " << setprecision( 1 ) << w * l
108           << resetiosflags( ios::fixed | ios::showpoint ) << "\n\n";
109 }
110
111 void Rectangle::square( void )
112 {
113     const int x = 0, y = 1; // added for clarity
114
115     if ( fabs( point4[ y ] - point1[ y ] ) ==
116         fabs( point2[ x ] - point1[ x ] ) )
117
118         cout << "The rectangle is a square.\n\n";
119 }
120
121 void Rectangle::draw( void )
122 {
123     for ( double y = 25.0; y >= 0.0; --y ) {
124         for ( double x = 0.0; x <= 25.0; ++x ) {
125             if ( ( point1[ 0 ] == x && point1[ 1 ] == y ) ||
126                 ( point4[ 0 ] == x && point4[ 1 ] == y ) ) {

```



```
127
128         // print horizontal perimeter of rectangle
129         while ( x <= point2[ 0 ] ) {
130             cout << periChar;
131             ++x;
132         }
133
134         // print remainder of quadrant
135         cout << '.';
136     }
137     // prints vertical perimeter of rectangle
138     else if ( ( ( x <= point4[ 0 ] && x >= point1[ 0 ] ) ) &&
139             point4[ 1 ] >= y && point1[ 1 ] <= y ) {
140         cout << periChar;
141
142         // fill inside of rectangle
143         for ( x++; x < point2[ 0 ]; ) {
144             cout << fillChar;
145             ++x;
146         }
147
148         cout << periChar;
149     }
150     else
151         cout << '.'; // print quadrant background
152 }
153
154 cout << '\n';
155 }
156 }
```

```
157 // driver for p16_9.cpp
158
159 #include "p16_9.h"
160
161 int main()
162 {
163     double xy1[ 2 ] = { 12.0, 12.0 }, xy2[ 2 ] = { 18.0, 12.0 },
164           xy3[ 2 ] = { 18.0, 20.0 }, xy4[ 2 ] = { 12.0, 20.0 };
165     Rectangle a( xy1, xy2, xy3, xy4, '?', '*' );
166
167     if ( a.isValid() )
168         a.draw();
169
170     return 0;
171 }
```


16.11 Create a class `TicTacToe` that will enable you to write a complete program to play the game of tic-tac-toe. The class contains as private data a 3-by-3 double array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square; place a 2 wherever the second player moves. Each move must be to an empty square. After each move, determine if the game has been won or if the game is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players automatically. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play three-dimensional tic-tac-toe on a 4-by-4-by-4 board (Caution: This is an extremely challenging project that could take many weeks of effort!).

ANS:

```

1 // p16_11.H
2 #ifndef P16_11_H
3 #define P16_11_H
4
5 class TicTacToe {
6 private:
7     enum Status { WIN, DRAW, CONTINUE };
8     int board[ 3 ][ 3 ];
9 public:
10    TicTacToe();
11    void makeMove( void );
12    void printBoard( void );
13    bool validMove( int, int );
14    bool xoMove( int );
15    Status gameStatus( void );
16 };
17
18 #endif

```

```

19 // P16_11M.cpp
20 // member function definitions for p16_9.cpp
21 #include <iostream>
22
23 using std::cout;
24 using std::cin;
25
26 #include <iomanip>
27
28 using std::setw;
29
30 #include "p16_11.h"
31
32 TicTacToe::TicTacToe()
33 {
34     for ( int j = 0; j < 3; ++j ) // initialize board
35         for ( int k = 0; k < 3; ++k )
36             board[ j ][ k ] = ' ';
37 }
38
39 bool TicTacToe::validMove( int r, int c )
40 {
41     return r >= 0 && r < 3 && c >= 0 && c < 3 && board[ r ][ c ] == ' ';
42 }
43
44 // must specify that type Status is part of the TicTacToe class.
45 // See Chapter 21 for a discussion of namespaces.
46 TicTacToe::Status TicTacToe::gameStatus( void )
47 {
48     int a;
49
50     // check for a win on diagonals
51     if ( board[ 0 ][ 0 ] != ' ' && board[ 0 ][ 0 ] == board[ 1 ][ 1 ] &&
52         board[ 0 ][ 0 ] == board[ 2 ][ 2 ] )
53         return WIN;

```

```

54     else if ( board[ 2 ][ 0 ] != ' ' && board[ 2 ][ 0 ] ==
55     board[ 1 ][ 1 ] && board[ 2 ][ 0 ] == board[ 0 ][ 2 ] )
56         return WIN;
57
58     // check for win in rows
59     for ( a = 0; a < 3; ++a )
60         if ( board[ a ][ 0 ] != ' ' && board[ a ][ 0 ] ==
61     board[ a ][ 1 ] && board[ a ][ 0 ] == board[ a ][ 2 ] )
62             return WIN;
63
64     // check for win in columns
65     for ( a = 0; a < 3; ++a )
66         if ( board[ 0 ][ a ] != ' ' && board[ 0 ][ a ] ==
67     board[ 1 ][ a ] && board[ 0 ][ a ] == board[ 2 ][ a ] )
68             return WIN;
69
70     // check for a completed game
71     for ( int r = 0; r < 3; ++r )
72         for ( int c = 0; c < 3; ++c )
73             if ( board[ r ][ c ] == ' ' )
74                 return CONTINUE; // game is not finished
75
76     return DRAW; // game is a draw
77 }
78
79 void TicTacToe::printBoard( void )
80 {
81     cout << "  0   1   2\n\n";
82
83     for ( int r = 0; r < 3; ++r ) {
84         cout << r;
85
86         for ( int c = 0; c < 3; ++c ) {
87             cout << setw( 3 ) << static_cast< char >( board[ r ][ c ] );
88
89             if ( c != 2 )
90                 cout << " |";
91         }
92
93         if ( r != 2 )
94             cout << "\n  ---|---|---"
95             << "\n  |   |   |   \n";
96     }
97     cout << "\n\n";
98 }
99
100 void TicTacToe::makeMove( void )
101 {
102     printBoard();
103
104     while ( true ) {
105         if ( xoMove( 'X' ) )
106             break;
107         else if ( xoMove( 'O' ) )
108             break;
109     }
110 }
111
112 bool TicTacToe::xoMove( int symbol )
113 {
114     int x, y;
115
116     do {
117         cout << "Player " << static_cast< char >( symbol )
118         << " enter move: ";
119         cin >> x >> y;
120         cout << '\n';
121     } while ( !validMove( x, y ) );

```

```
123     board[ x ][ y ] = symbol;
124     printBoard();
125     Status xoStatus = gameStatus();
126
127     if ( xoStatus == WIN ) {
128         cout << "Player " << static_cast< char >( symbol ) << " wins!\n";
129         return true;
130     }
131     else if ( xoStatus == DRAW ) {
132         cout << "Game is a draw.\n";
133         return true;
134     }
135     else // CONTINUE
136         return false;
137 }
138 }
```

```
139 // driver for p16_11.cpp
140 #include "p16_11.h"
141
142 int main()
143 {
144     TicTacToe g;
145     g.makeMove();
146
147     return 0;
148 }
```

```

  0   1   2
0  |   |   |
  --|---|---
1  |   |   |
  --|---|---
2  |   |   |

```

Player X enter move: 0 0

```

  0   1   2
0 X |   |   |
  --|---|---
1  |   |   |
  --|---|---
2  |   |   |

```

...

Player O enter move: 0 2

```

  0   1   2
0 X |   |  O
  --|---|---
1 X |  O |   |
  --|---|---
2  |   |   |

```

Player X enter move: 2 0

```

  0   1   2
0 X |   |  O
  --|---|---
1 X |  O |   |
  --|---|---
2 X |   |   |

```

Player X wins!

17

C++ Classes: Part II: Solutions

SOLUTIONS

17.3 Compare and contrast dynamic memory allocation using the C++'s `new` and `delete` operators, with dynamic memory allocation using the C Standard Library functions `malloc` and `free`.

ANS: In C, dynamic memory allocation requires function calls to `malloc` and `free`. Also, `malloc` must be told the exact number of bytes to allocate (normally this is accomplished with the `sizeof` operator), then it returns a `void` pointer. C++ uses operators `new` and `delete`. The `new` operator automatically determines the number of bytes to allocate and returns a pointer to the appropriate type. The `delete` operator guarantees a call to the destructor for the object(s) begin deleted.

17.4 Explain the notion of friendship in C++. Explain the negative aspects of friendship as described in the text.

ANS: Functions that are declared as `friends` of a class have access to that class's `private` and `protected` members. Some people in the object-oriented programming community prefer not to use `friend` functions because they break the encapsulation of a class -- i.e., they allow direct access to a class's implementation details that are supposed to be hidden.

17.5 Can a correct `Time` class definition include both of the following constructors? If not, explain why not.

```
Time( int h = 0, int m = 0, int s = 0 );  
Time();
```

ANS: No, because there is ambiguity between the two constructors. When a call is made to the default constructor, the compiler cannot determine which one to use because they both can be called with no arguments.

17.6 What happens when a return type, even `void`, is specified for a constructor or destructor?

ANS: A compiler syntax error occurs. No return types can be specified for constructors.

17.7 Create a `Date` class with the following capabilities:

a) Output the date in multiple formats such as

```
DDD YYYY  
MM/DD/YY  
June 14, 1992
```

b) Use overloaded constructors to create `Date` objects initialized with dates of the formats in part (a).

c) Create a `Date` constructor that reads the system date using the standard library functions of the `<ctime>` header and sets the `Date` members.

In Chapter 18, we will be able to create operators for testing the equality of two dates and for comparing dates to determine if one date is prior to, or after, another.

ANS:

```

1 // P17_07.H
2 #ifndef p17_07_H
3 #define p17_07_H
4
5 #include <ctime>
6 #include <cstring>
7
8 class Date {
9 public:
10     Date();
11     Date( int, int );
12     Date( int, int, int );
13     Date( char *, int, int );
14     void setMonth( int );
15     void setDay( int );
16     void setYear( int );
17     void printDateSlash( void ) const;
18     void printDateMonth( void ) const;
19     void printDateDay( void ) const;
20     const char *monthName( void ) const;
21     bool leapYear( void ) const;
22     int daysOfMonth( void ) const;
23     void convert1( int );
24     int convert2( void ) const;
25     void convert3( const char * const );
26     const char *monthList( int ) const;
27     int days( int ) const;
28 private:
29     int day;
30     int month;
31     int year;
32 };
33
34 #endif

```

```

1 // P17_07M.cpp
2 // member function definitions for p17_07.cpp
3 #include <iostream>
4
5 using std::cout;
6
7 #include <ctime>
8
9 #include "p17_07.h"
10
11 // Date constructor
12 Date::Date()
13 {
14     long int totalTime;
15     double totalYear;
16     long double divisor;
17
18     totalTime = time( NULL ); // time in seconds since 1970
19     divisor = ( 60.0 * 60.0 * 24.0 * 365.25 ); //number of seconds in a year
20     totalYear = totalTime / divisor + 1970;
21     year = ( int ) totalYear;
22     totalYear -= year;
23     day = ( int ) ( 365 * totalYear );
24     month = 1;
25
26     while ( day - days( month + 1 ) > 0 )
27         day -= days( month++ );
28 }
29

```

```

30 // Date constructor that uses day of year and year
31 Date::Date( int ddd, int yyyy )
32 {
33     setYear( yyyy );
34     convert1( ddd ); // convert to month and day
35 }
36
37 // Date constructor that uses month, day and year
38 Date::Date( int mm, int dd, int yy )
39 {
40     setYear( yy + 1900 );
41     setMonth( mm );
42     setDay( dd );
43 }
44
45 // Date constructor that uses month name, day and year
46 Date::Date( char *mPtr, int dd, int yyyy )
47 {
48     setYear( yyyy );
49     convert3( mPtr );
50     setDay( dd );
51 }
52
53 // Set the day
54 void Date::setDay( int d )
55     { day = d >= 1 && d <= daysOfMonth() ? d : 1; }
56
57 // Set the month
58 void Date::setMonth( int m ) { month = m >= 1 && m <= 12 ? m : 1; }
59
60 // Set the year
61 void Date::setYear( int y ) { year = y >= 1900 && y <= 1999 ? y : 1900; }
62
63 // Print Date in the form: mm/dd/yyyy
64 void Date::printDateSlash( void ) const
65     { cout << month << '/' << day << '/' << year << '\n'; }
66
67 // Print Date in the form: monthname dd, yyyy
68 void Date::printDateMonth( void ) const
69     { cout << monthName() << ' ' << day << ", " << year << '\n'; }
70
71 // Print Date in the form: ddd yyyy
72 void Date::printDateDay( void ) const
73     { cout << convert2() << ' ' << year << '\n'; }
74
75 // Return the month name
76 const char *Date::monthName( void ) const { return monthList( month - 1 ); }
77
78 // Return the number of days in the month
79 int Date::daysOfMonth( void ) const
80     { return leapYear() && month == 2 ? 29 : days( month ); }
81
82 // Test for a leap year
83 bool Date::leapYear( void ) const
84 {
85     if ( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) )
86         return true;
87     else
88         return false;
89 }
90
91 // Convert ddd to mm and dd
92 void Date::convert1( int ddd ) // convert to mm / dd / yyyy
93 {
94     int dayTotal = 0;
95
96     if ( ddd < 1 || ddd > 366 ) // check for invalid day
97         ddd = 1;

```

```

98
99     setMonth( 1 );
100
101     int m = 1;
102
103     for ( ; m < 13 && ( dayTotal + daysOfMonth() ) < ddd; ++m ) {
104         dayTotal += daysOfMonth();
105         setMonth( m + 1 );
106     }
107
108     setDay( ddd - dayTotal );
109     setMonth( m );
110 }
111
112 // Convert mm and dd to ddd
113 int Date::convert2( void ) const    // convert to a ddd yyyy format
114 {
115     int ddd = 0;
116
117     for ( int m = 1; m < month; ++m )
118         ddd += days( m );
119
120     ddd += day;
121     return ddd;
122 }
123
124 // Convert from month name to month number
125 void Date::convert3( const char * const mPtr )    // convert to mm / dd / yyyy
126 {
127     bool flag = false;
128
129     for ( int subscript = 0; subscript < 12; ++subscript )
130         if ( !strcmp( mPtr, monthList( subscript ) ) ) {
131             setMonth( subscript + 1 );
132             flag = true; // set flag
133             break;     // stop checking for month
134         }
135
136     if ( !flag )
137         setMonth( 1 ); // invalid month default is january
138 }
139
140 // Return the name of the month
141 const char *Date::monthList( int mm ) const
142 {
143     char *months[] = { "January", "February", "March", "April", "May",
144                       "June", "July", "August", "September", "October",
145                       "November", "December" };
146
147     return months[ mm ];
148 }
149
150 // Return the days in the month
151 int Date::days( int m ) const
152 {
153     const int monthDays[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
154
155     return monthDays[ m - 1 ];
156 }

```

```
157 // driver for p17_07.cpp
158 #include <iostream>
159
160 using std::cout;
161 using std::endl;
162
163 #include "p17_07.h"
164
165 int main()
166 {
167     Date d1( 7, 4, 98 ), d2( 86, 1999 ),
168         d3, d4( "September", 1, 1998 );
169
170     d1.printDateSlash(); // format m / dd / yy
171     d2.printDateSlash();
172     d3.printDateSlash();
173     d4.printDateSlash();
174     cout << '\n';
175
176     d1.printDateDay(); // format ddd yyyy
177     d2.printDateDay();
178     d3.printDateDay();
179     d4.printDateDay();
180     cout << '\n';
181
182     d1.printDateMonth(); // format "month" d, yyyy
183     d2.printDateMonth();
184     d3.printDateMonth();
185     d4.printDateMonth();
186     cout << endl;
187
188     return 0;
189 }
```

```
7/4/1998
3/27/1999
7/26/2000
9/1/1998
```

```
185 1998
86 1999
207 2000
244 1998
```

```
July 4, 1998
March 27, 1999
July 26, 2000
September 1, 1998
```

17.8 Create a `SavingsAccount` class. Use a static data member to contain the `annualInterestRate` for each of the savers. Each member of the class contains a private data member `savingsBalance` indicating the amount the saver currently has on deposit. Provide a `calculateMonthlyInterest` member function that calculates the monthly interest by multiplying the balance by `annualInterestRate` divided by 12; this interest should be added to `savingsBalance`. Provide a static member function `modifyInterestRate` that sets the static `annualInterestRate` to a new value. Write a driver program to test class `SavingsAccount`. Instantiate two different `savingsAccount` objects, `saver1` and `saver2`, with balances of \$2000.00 and \$3000.00, respectively. Set `annualInterestRate` to 3%, then calculate the monthly interest and print the new balances for each of the savers. Then set the `annualInterestRate` to 4% and calculate the next month's interest and print the new balances for each of the savers.

ANS:

```

1 // P17_08.H
2 #ifndef P17_08_H
3 #define P17_08_H
4
5 class SavingsAccount {
6 public:
7     SavingsAccount( double b ) { savingsBalance = b >= 0 ? b : 0; }
8     void calculateMonthlyInterest( void );
9     static void modifyInterestRate( double );
10    void printBalance( void ) const;
11 private:
12    double savingsBalance;
13    static double annualInterestRate;
14 };
15
16 #endif

```

```

17 // P17.08M.cpp
18 // Member function definitions for p17_08.cpp
19 #include "p17_08.h"
20 #include <iostream>
21
22 using std::cout;
23 using std::ios;
24
25 #include <iomanip>
26
27 using std::setprecision;
28 using std::setiosflags;
29 using std::resetiosflags;
30
31 // initialize static data member
32 double SavingsAccount::annualInterestRate = 0.0;
33
34 void SavingsAccount::calculateMonthlyInterest( void )
35     { savingsBalance += savingsBalance * ( annualInterestRate / 12.0 ); }
36
37 void SavingsAccount::modifyInterestRate( double i )
38     { annualInterestRate = ( i >= 0 && i <= 1.0 ) ? i : 0.03; }
39
40 void SavingsAccount::printBalance( void ) const
41 {
42     cout << setiosflags( ios::fixed | ios::showpoint )
43         << '$' << setprecision( 2 ) << savingsBalance
44         << resetiosflags( ios::fixed | ios::showpoint );
45 }

```

```

46 // driver for p17_08.cpp
47 #include <iostream>
48
49 using std::cout;
50 using std::endl;
51
52 #include <iomanip>
53
54 using std::setw;
55
56 #include "p17_08.h"
57
58 int main()
59 {
60     SavingsAccount saver1( 2000.0 ), saver2( 3000.0 );
61
62     SavingsAccount::modifyInterestRate( .03 );
63
64     cout << "\nOutput monthly balances for one year at .03"
65          << "\nBalances: Saver 1 ";
66     saver1.printBalance();
67     cout << "\tSaver 2 ";
68     saver2.printBalance();
69
70     for ( int month = 1; month <= 12; ++month ) {
71         saver1.calculateMonthlyInterest();
72         saver2.calculateMonthlyInterest();
73
74         cout << "\nMonth" << setw( 3 ) << month << ": Saver 1 ";
75         saver1.printBalance();
76         cout << "\tSaver 2 ";
77         saver2.printBalance();
78     }
79
80     SavingsAccount::modifyInterestRate( .04 );
81     saver1.calculateMonthlyInterest();
82     saver2.calculateMonthlyInterest();
83     cout << "\nAfter setting interest rate to .04"
84          << "\nBalances: Saver 1 ";
85     saver1.printBalance();
86     cout << "\tSaver 2 ";
87     saver2.printBalance();
88     cout << endl;
89     return 0;
90 }

```

```

Output monthly balances for one year at .03
Balances: Saver 1 $2000.00    Saver 2 $3000.00
Month 1: Saver 1 $2005.00    Saver 2 $3007.50
Month 2: Saver 1 $2010.01    Saver 2 $3015.02
Month 3: Saver 1 $2015.04    Saver 2 $3022.56
Month 4: Saver 1 $2020.08    Saver 2 $3030.11
Month 5: Saver 1 $2025.13    Saver 2 $3037.69
Month 6: Saver 1 $2030.19    Saver 2 $3045.28
Month 7: Saver 1 $2035.26    Saver 2 $3052.90
Month 8: Saver 1 $2040.35    Saver 2 $3060.53
Month 9: Saver 1 $2045.45    Saver 2 $3068.18
Month 10: Saver 1 $2050.57    Saver 2 $3075.85
Month 11: Saver 1 $2055.69    Saver 2 $3083.54
Month 12: Saver 1 $2060.83    Saver 2 $3091.25
After setting interest rate to .04
Balances: Saver 1 $2067.70    Saver 2 $3101.55

```

17.9 It would be perfectly reasonable for the `Time` class of Fig. 17.8 to represent the time internally as the number of seconds since midnight rather than the three integer values `hour`, `minute` and `second`. Clients could use the same `public` methods and get the same results. Modify the `Time` class of Fig. 17.8 to implement the `Time` as the number of seconds since midnight and show that there is no visible change in functionality to the clients of the class.

18

C++ Operator Overloading: Solutions

SOLUTIONS

18.6 Give as many examples as you can of operator overloading implicit in C++. Give a reasonable example of a situation in which you might want to overload an operator explicitly in C++.

ANS: In C, the operators +, -, *, and & are overloaded. The context of these operators determines how they are used. It can be argued that the arithmetic operators are all overloaded, because they can be used to perform operations on more than one type of data. In C++, the same operators as in C are overloaded, as well as << and >>.

18.7 The C++ operators that cannot be overloaded are _____, _____, _____, _____, and _____.

ANS: sizeof, ., ?:, .*, and ::.

18.8 (*Project*) C++ is an evolving language, and new languages are always being developed. What additional operators would you recommend adding to C++ or to a future language like C++ that would support both procedural programming and object-oriented programming? Write a careful justification. You might consider sending your suggestions to the ANSI C++ Committee or the newsgroup comp.std.c++.

18.9 Overload the subscript operator to return the largest element of a collection, the second largest, the third largest, etc.

18.10 Consider class `Complex` shown in Fig. 18.5. The class enables operations on so-called *complex numbers*. These are numbers of the form $\text{realPart} + \text{imaginaryPart} * i$ where i has the value:

$$\sqrt{-1}$$

- Modify the class to enable input and output of complex numbers through the overloaded >> and << operators, respectively (you should remove the `print` function from the class).
- Overload the multiplication operator to enable multiplication of two complex numbers as in algebra.
- Overload the == and != operators to allow comparisons of complex numbers.

```
1 // Fig. 18.5: complex1.h
2 // Definition of class Complex
3 #ifndef COMPLEX1_H
4 #define COMPLEX1_H
5
6 class Complex {
7 public:
```

Fig. 18.5 A complex number class—`complex1.h`.


```

8     Complex( double = 0.0, double = 0.0 );           // constructor
9     Complex operator+( const Complex & ) const;    // addition
10    Complex operator-( const Complex & ) const;    // subtraction
11    const Complex &operator=( const Complex & );   // assignment
12    void print() const;                             // output
13 private:
14     double real;           // real part
15     double imaginary;     // imaginary part
16 };
17
18 #endif

```

Fig. 18.5 A complex number class—complex1.h.

```

19 // Fig. 18.5: complex1.cpp
20 // Member function definitions for class Complex
21 #include <iostream>
22
23 using std::cout;
24
25 #include "complex1.h"
26
27 // Constructor
28 Complex::Complex( double r, double i )
29     : real( r ), imaginary( i ) { }
30
31 // Overloaded addition operator
32 Complex Complex::operator+( const Complex &operand2 ) const
33 {
34     return Complex( real + operand2.real,
35                    imaginary + operand2.imaginary );
36 }
37
38 // Overloaded subtraction operator
39 Complex Complex::operator-( const Complex &operand2 ) const
40 {
41     return Complex( real - operand2.real,
42                    imaginary - operand2.imaginary );
43 }
44
45 // Overloaded = operator
46 const Complex& Complex::operator=( const Complex &right )
47 {
48     real = right.real;
49     imaginary = right.imaginary;
50     return *this; // enables cascading
51 }
52
53 // Display a Complex object in the form: (a, b)
54 void Complex::print() const
55     { cout << '(' << real << ", " << imaginary << ')'; }

```

Fig. 18.5 A complex number class—complex1.cpp.

```

56 // Fig. 18.5: fig18_05.cpp
57 // Driver for class Complex
58 #include <iostream>
59
60 using std::cout;
61 using std::endl;
62
63 #include "complex1.h"
64
65 int main()
66 {
67     Complex x, y( 4.3, 8.2 ), z( 3.3, 1.1 );
68
69     cout << "x: ";
70     x.print();
71     cout << "\ny: ";
72     y.print();
73     cout << "\nz: ";
74     z.print();
75
76     x = y + z;
77     cout << "\n\nx = y + z:\n";
78     x.print();
79     cout << " = ";
80     y.print();
81     cout << " + ";
82     z.print();
83
84     x = y - z;
85     cout << "\n\nx = y - z:\n";
86     x.print();
87     cout << " = ";
88     y.print();
89     cout << " - ";
90     z.print();
91     cout << endl;
92
93     return 0;
94 }

```

```

x: (0, 0)
y: (4.3, 8.2)
z: (3.3, 1.1)

x = y + z:
(7.6, 9.3) = (4.3, 8.2) + (3.3, 1.1)

x = y - z:
(1, 7.1) = (4.3, 8.2) - (3.3, 1.1)

```

Fig. 18.5 A complex number class—fig18_05.cpp.

ANS:

```

1 // P18_10.H
2 #ifndef P18_10_H
3 #define P18_10_H
4 #include <iostream>
5
6 using std::ostream;
7 using std::istream;
8

```

```

9  class Complex {
10     friend ostream &operator<<( ostream &, const Complex & );
11     friend istream &operator>>( istream &, Complex & );
12 public:
13     Complex( double = 0.0, double = 0.0 );    // constructor
14     Complex operator+( const Complex& ) const; // addition
15     Complex operator-( const Complex& ) const; // subtraction
16     Complex operator*( const Complex& ) const; // multiplication
17     Complex& operator=( const Complex& );    // assignment
18     bool operator==( const Complex& ) const;
19     bool operator!=( const Complex& ) const;
20 private:
21     double real;        // real part
22     double imaginary;  // imaginary part
23 };
24
25 #endif

```

```

26 // P18_10M.cpp
27 // member function definitions for p18_10.cpp
28 #include "p18_10.h"
29 #include <iostream>
30
31 using std::ostream;
32 using std::istream;
33
34 // Constructor
35 Complex::Complex( double r, double i )
36 {
37     real = r;
38     imaginary = i;
39 }
40
41 // Overloaded addition operator
42 Complex Complex::operator+( const Complex &operand2 ) const
43 {
44     Complex sum;
45
46     sum.real = real + operand2.real;
47     sum.imaginary = imaginary + operand2.imaginary;
48     return sum;
49 }
50
51 // Overloaded subtraction operator
52 Complex Complex::operator-( const Complex &operand2 ) const
53 {
54     Complex diff;
55
56     diff.real = real - operand2.real;
57     diff.imaginary = imaginary - operand2.imaginary;
58     return diff;
59 }
60
61 // Overloaded multiplication operator
62 Complex Complex::operator*( const Complex &operand2 ) const
63 {
64     Complex times;
65
66     times.real = real * operand2.real + imaginary * operand2.imaginary;
67     times.imaginary = real * operand2.imaginary + imaginary * operand2.real;
68     return times;
69 }
70
71 // Overloaded = operator
72 Complex& Complex::operator=( const Complex &right )
73 {

```

```

74     real = right.real;
75     imaginary = right.imaginary;
76     return *this;    // enables concatenation
77 }
78
79 bool Complex::operator==( const Complex &right ) const
80 { return right.real == real && right.imaginary == imaginary ? true : false; }
81
82 bool Complex::operator!=( const Complex &right ) const
83 { return !( *this == right ); }
84
85 ostream& operator<<( ostream &output, const Complex &complex )
86 {
87     output << complex.real << " + " << complex.imaginary << 'i';
88     return output;
89 }
90
91 istream& operator>>( istream &input, Complex &complex )
92 {
93     input >> complex.real;
94     input.ignore( 3 );    // skip spaces and +
95     input >> complex.imaginary;
96     input.ignore( 2 );
97
98     return input;
99 }

```

```

100 // driver for p18_10.cpp
101 #include <iostream>
102
103 using std::cout;
104 using std::cin;
105
106 #include "p18_10.h"
107
108 int main()
109 {
110     Complex x, y( 4.3, 8.2 ), z( 3.3, 1.1 ), k;
111
112     cout << "Enter a complex number in the form: a + bi\n? ";
113     cin >> k;
114
115     cout << "x: " << x << "\ny: " << y << "\nz: " << z << "\nk: "
116         << k << '\n';
117
118     x = y + z;
119     cout << "\nx = y + z:\n" << x << " = " << y << " + " << z << '\n';
120
121     x = y - z;
122     cout << "\nx = y - z:\n" << x << " = " << y << " - " << z << '\n';
123
124     x = y * z;
125     cout << "\nx = y * z:\n" << x << " = " << y << " * " << z << "\n\n";
126
127     if ( x != k )
128         cout << x << " != " << k << '\n';
129
130     cout << '\n';
131
132     x = k;
133
134     if ( x == k )
135         cout << x << " == " << k << '\n';
136
137     return 0;
138 }

```

```

Enter a complex number in the form: a + bi
? 22 + 8i
x: 0 + 0i
y: 4.3 + 8.2i
z: 3.3 + 1.1i
k: 22 + 8i

x = y + z:
7.6 + 9.3i = 4.3 + 8.2i + 3.3 + 1.1i

x = y - z:
1 + 7.1i = 4.3 + 8.2i - 3.3 + 1.1i

x = y * z:
23.21 + 31.79i = 4.3 + 8.2i * 3.3 + 1.1i

23.21 + 31.79i != 22 + 8i

22 + 8i == 22 + 8i

```

18.11 The program of Fig. 18.3 contains the comment

```

// Overloaded stream-insertion operator (cannot be
// a member function if we would like to invoke it with
// cout << somePhoneNumber;)

```

Actually, it cannot be a member function of class ostream, but it can be a member function of class PhoneNumber if we were willing to invoke it in either of the following ways:

```
somePhoneNumber.operator<<( cout );
```

or

```
somePhoneNumber << cout;
```

Rewrite the program of Fig. 18.3 with the overloaded stream-insertion operator<< as a member function and try the two preceding statements in the program to prove that they work.

19

C++ Inheritance: Solutions

SOLUTIONS

19.2 Consider the class `Bicycle`. Given your knowledge of some common components of bicycles, show a class hierarchy in which the class `Bicycle` inherits from other classes, which, in turn, inherit from yet other classes. Discuss the instantiation of various objects of class `Bicycle`. Discuss inheritance from class `Bicycle` for other closely related derived classes.

ANS: Possible classes are displayed in bold.

`Bicycle` composed of:

`HandleBars`

`Seat`

`Frame`

`Wheels` composed of:

`Tires`

`Rims` composed of:

`Spokes`

`Pedals`

`Chain` composed of:

`Links`

`Brakes` composed of:

`Wires`

`Brickbats`

`Breadlines`

Classes that can be derived from `Bicycle` are `Unicycle`, `Tricycle`, `Tandem Bicycle`, etc.

19.3 Briefly define each of the following terms: inheritance, multiple inheritance, base class and derived class.

ANS:

inheritance: The process by which a class incorporates the attributes and behaviors of a previously defined class.

multiple inheritance: The process by which a class incorporates the attributes and behaviors of two or more previously defined classes.

base class: A class from which other classes inherit attributes and behaviors.

derived class: A class that has inherited attributes and behaviors from one or more base classes.

19.4 Discuss why converting a base-class pointer to a derived-class pointer is considered dangerous by the compiler.

ANS: The pointer must “point” to the object of the derived class, before being dereferenced. When the compiler looks at an object through a derived-class pointer, it expects to see all the pieces of the derived class. However, if the base-class pointer originally pointed to a base-class object, the additional pieces added by the derived class do not exist.

19.5 (True/False) A derived class is often called a subclass because it represents a subset of its base class (i.e., a derived class is generally smaller than its base class).

ANS: False. Derived classes are often larger than their base classes, because they need specific features in addition to those inherited from the base class. The term subclass means that the derived class is a more specific version of its base class. For example, a cat is a specific type of animal.

19.6 (True/False) A derived-class object is also an object of that derived class's base class.

ANS: True.

19.7 Some programmers prefer not to use `protected` access because it breaks the encapsulation of the base class. Discuss the relative merits of using `protected` access vs. insisting on using `private` access in base classes.

ANS: Inherited `private` data is hidden in the derived class and is accessible only through the `public` or `protected` member functions of the base class. Using `protected` access enables the derived class to manipulate the `protected` members without using the base class access functions. If the base class members are `private`, the `public` or `protected` member functions of the base class must be used to access `private` members. This can result in additional function calls—which can decrease performance.

19.8 Many programs written with inheritance could be solved with composition instead, and vice versa. Discuss the relative merits of these approaches in the context of the `Point`, `Circle`, `Cylinder` class hierarchy in this chapter. Rewrite the program of Fig. 19.10 (and the supporting classes) to use composition rather than inheritance. After you do this, reassess the relative merits of the two approaches both for the `Point`, `Circle`, `Cylinder` problem and for object-oriented programs in general.

ANS:

```

1 // P19_08.H
2 #ifndef P19_08_H
3 #define P19_08_H
4
5 #include <iostream>
6 using std::ostream;
7
8 class Point {
9     friend ostream &operator<<( ostream &, const Point & );
10 public:
11     Point( double a = 0, double b = 0 ) { setPoint( a, b ); }
12     void setPoint( double, double );
13     void print( void ) const;
14     double getX( void ) const { return x; }
15     double getY( void ) const { return y; }
16 private:
17     double x, y;
18 };
19
20 #endif

```

```

21 // P19_08PM.cpp
22 // Member functions for class Point
23 #include <iostream>
24
25 using std::cout;
26 using std::ostream;
27
28 #include "p19_08.h"
29
30 void Point::setPoint( double a, double b )
31 {
32     x = a;
33     y = b;
34 }
35
36 ostream &operator<<( ostream &output, const Point &p )
37 {
38     p.print();
39     return output;
40 }

```

```

41
42 void Point::print( void ) const
43     { cout << '[' << getX() << ", " << getY() << ']' ; }

```

```

44 // P19_08C.H
45 #ifndef P19_08C_H
46 #define P19_08C_H
47 #include "P19_08.h"
48
49 class Circle {
50     friend ostream &operator<<( ostream &, const Circle & );
51 public:
52     Circle( double = 0.0, double = 0.0, double = 0.0 );
53     void setRadius( double r ) { radius = r; }
54     double getRadius( void ) const { return radius; }
55     double area( void ) const;
56     void print( void ) const;
57 private:
58     double radius;
59     Point pointObject;
60 };
61
62 #endif
63

```

```

64 // P19_08CM.cpp
65 // Member function definitions for class Circle
66 #include <iostream>
67
68 using std::cout;
69 using std::ios;
70
71 #include <iomanip>
72
73 using std::setprecision;
74 using std::setiosflags;
75 using std::resetiosflags;
76
77 #include "P19_08c.h"
78
79 Circle::Circle( double r, double a, double b ) : pointObject( a, b )
80     { setRadius( r ); }
81
82 double Circle::area( void ) const
83     { return 3.14159 * getRadius() * getRadius(); }
84
85 ostream &operator<<( ostream &output, const Circle &c )
86 {
87     c.print();
88     return output;
89 }
90
91 void Circle::print( void ) const
92 {
93     cout << "Center = ";
94     pointObject.print();
95     cout << "; Radius = " << setiosflags( ios::fixed | ios::showpoint )
96         << setprecision( 2 ) << getRadius()
97         << resetiosflags( ios::fixed | ios::showpoint );
98 }

```

```
99 // P19_08CY.H
100 #ifndef P19_08CY_H
101 #define P19_08CY_H
102 #include "P19_08.h"
103 #include "P19_08c.h"
104
105 class Cylinder {
106     friend ostream& operator<<(ostream&, const Cylinder&);
107 public:
108     Cylinder(double = 0.0, double = 0.0, double = 0.0, double = 0.0);
109     void setHeight(double h) { height = h; }
110     double getHeight(void) const { return height; }
111     void print(void) const;
112     double area(void) const;
113     double volume(void) const;
114 private:
115     double height;
116     Circle circleObject;
117 };
118
119 #endif
```

```
120 // P19_08CYM.cpp
121 // Member function definitions for class Cylinder.
122 #include <iostream>
123
124 using std::cout;
125 using std::ostream;
126
127 #include "p19_08cy.h"
128
129 Cylinder::Cylinder( double h, double r, double x, double y )
130     : circleObject( r, x, y ) { height = h; }
131
132 double Cylinder::area( void ) const
133     { return 2 * circleObject.area() + 2 * 3.14159 *
134       circleObject.getRadius() * getHeight(); }
135
136 ostream& operator<<( ostream &output, const Cylinder& c )
137 {
138     c.print();
139     return output;
140 }
141
142 double Cylinder::volume( void ) const
143     { return circleObject.area() * getHeight(); }
144
145 void Cylinder::print( void ) const
146 {
147     circleObject.print();
148     cout << "; Height = " << getHeight() << '\n';
149 }
```

```

150 // P19_08.cpp
151 #include <iostream>
152
153 using std::cout;
154 using std::endl;
155
156 #include "P19_08.h"
157 #include "P19_08c.h"
158 #include "P19_08cy.h"
159
160 int main()
161 {
162     Point p( 1.1, 8.5 );
163     Circle c( 2.0, 6.4, 9.8 );
164     Cylinder cyl( 5.7, 2.5, 1.2, 2.3 );
165
166     cout << "Point: " << p << "\nCircle: " << c
167         << "\nCylinder: " << cyl << endl;
168
169     return 0;
170 }

```

```

Point: [1.1, 8.5]
Circle: Center = [6.4, 9.8]; Radius = 2.00
Cylinder: Center = [1.2, 2.3]; Radius = 2.50; Height = 5.7

```

19.9 In the chapter, we stated, “When a base-class member is inappropriate for a derived class, that member can be overridden in the derived class with an appropriate implementation.” If this is done, does the derived-class-is-a-base-class-object relationship still hold? Explain your answer.

ANS: No. The “is a” relationship assumes that everything belongs to the base class object belongs to the derived class object and also assumes that all functionality of the base class is present in the derived class object.

19.10 Study the inheritance hierarchy of Fig. 19.2. For each class, indicate some common attributes and behaviors consistent with the hierarchy. Add some other classes (UndergraduateStudent, GraduateStudent, Freshman, Sophomore, Junior, Senior, etc.) to enrich the hierarchy.

ANS:

```

CommunityMember
  Employee
    Staff
      Maintenance
      Janitorial
    Faculty
      Administrator
      Professor
      TenuredProfessor
  Student
    Graduate
      MasterCandidate
      DoctoralCandidate
    Undergraduate
      Freshman
      Sophomore
      Junior
      Senior

```

19.11 Write an inheritance hierarchy for class `Quadrilateral`, `Trapezoid`, `Parallelogram`, `Rectangle` and `Square`. Use `Quadrilateral` as the base class of the hierarchy. Make the hierarchy as deep (i.e., as many levels) as possible. The private data of `Quadrilateral` should be the (x, y) coordinate pairs for the four endpoints of the `Quadrilateral`. Write a driver program that instantiates and displays objects of each of these classes.

ANS:

```

1 // P19_11.H
2 #ifndef P19_11_H
3 #define P19_11_H
4
5 #include <iostream>
6 using std::ostream;
7
8 class Point {
9     friend ostream &operator<<( ostream&, const Point& );
10 public:
11     Point( double = 0, double = 0 );
12     void setPoint( double, double );
13     void print( void ) const;
14     double getX( void ) const { return x; }
15     double getY( void ) const { return y; }
16 private:
17     double x, y;
18 };
19
20 #endif

```

```

21 // P19_11PM.cpp
22 // member function definitions for class Point
23 #include <iostream>
24
25 using std::cout;
26 using std::ios;
27 using std::ostream;
28
29 #include <iomanip>
30
31 using std::setprecision;
32 using std::setiosflags;
33 using std::resetiosflags;
34
35 #include "p19_11.h"
36
37 Point::Point( double a, double b ) { setPoint( a, b ); }
38
39 void Point::setPoint( double a, double b )
40 {
41     x = a;
42     y = b;
43 }
44
45 ostream &operator<<( ostream &output, const Point &p )
46 {
47     output << "The point is: ";
48     p.print();
49     return output;
50 }
51
52 void Point::print( void ) const
53 {
54     cout << setiosflags( ios::fixed | ios::showpoint )
55          << '[' << setprecision( 2 ) << getX()
56          << ", " << setprecision( 2 ) << getY() << "]\n"
57          << resetiosflags( ios::fixed | ios::showpoint );
58 }

```

```

59 // P19_11Q.H
60 #ifndef P19_11Q_H
61 #define P19_11Q_H
62 #include "p19_11.h"
63
64 #include <iostream>
65 using std::ostream;
66
67 class Quadrilateral {
68     friend ostream &operator<<( ostream&, Quadrilateral& );
69 public:
70     Quadrilateral( double = 0, double = 0, double = 0, double = 0, double = 0,
71                 double = 0, double = 0, double = 0 );
72     void print( void ) const;
73 protected:
74     Point p1;
75     Point p2;
76     Point p3;
77     Point p4;
78 };
79
80 #endif

```

```

81 // P19_11QM.cpp
82 // member functions for class Quadrilateral
83 #include "p19_11q.h"
84
85 #include <iostream>
86 using std::cout;
87 using std::ostream;
88
89 Quadrilateral::Quadrilateral( double x1, double y1, double x2, double y2,
90                             double x3, double y3, double x4, double y4 )
91     : p1( x1, y1 ), p2( x2, y2 ), p3( x3, y3 ), p4( x4, y4 ) { }
92
93 ostream &operator<<( ostream& output, Quadrilateral& q )
94 {
95     output << "Coordinates of Quadrilateral are:\n";
96     q.print();
97     output << '\n';
98     return output;
99 }
100
101 void Quadrilateral::print( void ) const
102 {
103     cout << '(' << p1.getX()
104          << ", " << p1.getY() << " ) , (" << p2.getX() << ", " << p2.getY()
105          << " ) , (" << p3.getX() << ", " << p3.getY() << " ) , ("
106          << p4.getX() << ", " << p4.getY() << ")\n";
107 }

```

```
108 // P19_11T.H
109 #ifndef P19_11T_H
110 #define P19_11T_H
111 #include "p19_11q.h"
112
113 #include <iostream>
114 using std::ostream;
115
116 class Trapazoid : public Quadrilateral {
117     friend ostream& operator<<( ostream&, Trapazoid& );
118 public:
119     Trapazoid( double = 0, double = 0, double = 0, double = 0, double = 0,
120             double = 0, double = 0, double = 0, double = 0 );
121     void print( void ) const;
122     void setHeight( double h ) { height = h; }
123     double getHeight( void ) const { return height; }
124 private:
125     double height;
126 };
127
128 #endif
```

```
129 // P19_11TM.cpp
130 // member function definitions for class Trapazoid
131 #include "p19_11t.h"
132
133 #include <iostream>
134
135 using std::cout;
136 using std::ostream;
137
138 Trapazoid::Trapazoid( double h, double x1, double y1, double x2, double y2,
139                     double x3, double y3, double x4, double y4 )
140     : Quadrilateral( x1, y1, x2, y2, x3, y3, x4, y4 )
141 { setHeight( h ); }
142
143 ostream& operator<<( ostream& out, Trapazoid& t )
144 {
145     out << "The Coordinates of the Trapazoid are:\n";
146     t.print();
147     return out;
148 }
149
150 void Trapazoid::print( void ) const
151 {
152     Quadrilateral::print();
153     cout << "Height is : " << getHeight() << "\n\n";
154 }
```

```

155 // P19_11PA.H
156 #ifndef P19_11PA_H
157 #define P19_11PA_H
158 #include "p19_11q.h"
159
160 #include <iostream>
161 using std::ostream;
162
163 class Parallelogram : public Quadrilateral {
164     friend ostream& operator<<( ostream&, Parallelogram& );
165 public:
166     Parallelogram( double = 0, double = 0, double = 0, double = 0,
167                 double = 0, double = 0, double = 0, double = 0 );
168     void print( void ) const;
169 private:
170     // no private data members
171 };
172
173 #endif

```

```

174 // P19_11PAM.cpp
175 #include "p19_11q.h"
176 #include "p19_11pa.h"
177
178 #include <iostream>
179 using std::ostream;
180
181 Parallelogram::Parallelogram( double x1, double y1, double x2, double y2,
182                             double x3, double y3, double x4, double y4 )
183     : Quadrilateral( x1, y1, x2, y2, x3, y3, x4, y4 ) { }
184
185 ostream& operator<<( ostream& out, Parallelogram& pa )
186 {
187     out << "The coordinates of the Parallelogram are:\n";
188     pa.print();
189     return out;
190 }
191
192 void Parallelogram::print( void ) const
193     { Quadrilateral::print(); }

```

```

194 // P19_11R.H
195 #ifndef P19_11R_H
196 #define P19_11R_H
197 #include "p19_11pa.h"
198
199 #include <iostream>
200 using std::ostream;
201
202 class Rectangle : public Parallelogram {
203     friend ostream& operator<<( ostream&, Rectangle& );
204 public:
205     Rectangle( double = 0, double = 0, double = 0, double = 0,
206             double = 0, double = 0, double = 0, double = 0 );
207     void print( void ) const;
208 private:
209     // no private data members
210 };
211
212 #endif

```

```

213 // P19_11RM.cpp
214 #include "p19_11r.h"
215 #include "p19_11pa.h"
216
217 #include <iostream>
218 using std::ostream;
219
220 Rectangle::Rectangle( double x1, double y1, double x2, double y2,
221                     double x3, double y3, double x4, double y4 )
222     : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }
223
224 ostream& operator<<( ostream& out, Rectangle& r )
225 {
226     out << "\nThe coordinates of the Rectangle are:\n";
227     r.print();
228     return out;
229 }
230
231 void Rectangle::print( void ) const
232     { Parallelogram::print(); }

```

```

233 // P19_11RH.H
234 #ifndef P19_11RH_H
235 #define P19_11RH_H
236 #include "p19_11pa.h"
237
238 #include <iostream>
239 using std::ostream;
240
241 class Rhombus : public Parallelogram {
242     friend ostream& operator<<(ostream&, Rhombus&);
243 public:
244     Rhombus( double = 0, double = 0, double = 0, double = 0, double = 0,
245            double = 0, double = 0, double = 0 );
246     void print( void ) const { Parallelogram::print(); }
247 private:
248     // no private data members
249 };
250
251 #endif

```

```

252 //P19_11HM.cpp
253 #include "p19_11rh.h"
254 #include "p19_11pa.h"
255
256 #include <iostream>
257 using std::ostream;
258
259 Rhombus::Rhombus( double x1, double y1, double x2, double y2,
260                 double x3, double y3, double x4, double y4 )
261     : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }
262
263 ostream& operator<<( ostream& out, Rhombus& r )
264 {
265     out << "\nThe coordinates of the Rhombus are:\n";
266     r.print();
267     return out;
268 }

```

```

269 // P19_11S.H
270 #ifndef P19_11S_H
271 #define P19_11S_H
272 #include "p19_11pa.h"
273
274 #include <iostream>
275 using std::ostream;
276
277 class Square : public Parallelogram {
278     friend ostream& operator<<( ostream&, Square& );
279 public:
280     Square( double = 0, double = 0, double = 0, double = 0,
281           double = 0, double = 0, double = 0, double = 0 );
282     void print( void ) const { Parallelogram::print(); }
283 private:
284     // no private data members
285 };
286
287 #endif

```

```

288 // P19_11SM.cpp
289 #include "p19_11s.h"
290 #include "p19_11pa.h"
291
292 #include <iostream>
293 using std::ostream;
294
295 Square::Square( double x1, double y1, double x2, double y2,
296               double x3, double y3, double x4, double y4 )
297     : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }
298
299 ostream& operator<<( ostream& out, Square& s )
300 {
301     out << "\nThe coordinates of the Square are:\n";
302     s.print();
303     return out;
304 }

```

```

305 // P19_11.cpp
306 #include "p19_11.h"
307 #include "p19_11q.h"
308 #include "p19_11t.h"
309 #include "p19_11pa.h"
310 #include "p19_11rh.h"
311 #include "p19_11r.h"
312 #include "p19_11s.h"
313
314 #include <iostream>
315 using std::cout;
316 using std::endl;
317
318 int main()
319 {
320     // NOTE: All coordinates are assumed to form the proper shapes
321
322     // A quadrilateral is a four-sided polygon
323     Quadrilateral q( 1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4 );
324     // A trapazoid is a quadrilateral having two and only two parallel sides
325     Trapazoid t( 5.0, 0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0 );
326     // A parallelogram is a quadrilateral whose opposite sides are parallel
327     Parallelogram p( 5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0 );
328     // A rhombus is an equilateral parallelogram
329     Rhombus rh( 0.0, 0.0, 5.0, 0.0, 8.5, 3.5, 3.5, 3.5 );
330     // A rectangle is an equiangular parallelogram

```



```

331 Rectangle r( 17.0, 14.0, 30.0, 14.0, 30.0, 28.0, 17.0, 28.0 );
332 // A square is an equiangular and equilateral parallelogram
333 Square s( 4.0, 0.0, 8.0, 0.0, 8.0, 4.0, 4.0, 4.0 );
334
335 cout << q << t << p << rh << r << s << endl;
336
337 return 0;
338 }

```

```

Coordinates of Quadrilateral are:
(1.1, 1.2) , (6.6, 2.8) , (6.2, 9.9) , (2.2, 7.4)

The Coordinates of the Trapazoid are:
(0, 0) , (10, 0) , (8, 5) , (3.3, 5)
Height is : 5

The coordinates of the Parallelogram are:
(5, 5) , (11, 5) , (12, 20) , (6, 20)

The coordinates of the Rhombus are:
(0, 0) , (5, 0) , (8.5, 3.5) , (3.5, 3.5)

The coordinates of the Rectangle are:
(17, 14) , (30, 14) , (30, 28) , (17, 28)

The coordinates of the Square are:
(4, 0) , (8, 0) , (8, 4) , (4, 4)

```

19.12 Write down all the shapes you can think of—both two-dimensional and three-dimensional—and form those shapes into a shape hierarchy. Your hierarchy should have base class `Shape` from which class `TwoDimensionalShape` and class `ThreeDimensionalShape` are derived. Once you have developed the hierarchy, define each of the classes in the hierarchy. We will use this hierarchy in the exercises of Chapter 20 to process all shapes as objects of base-class `Shape`. This is a technique called polymorphism.

ANS:

```

Shape
  TwoDimensionalShape
    Quadrilateral
      Parallelogram
        Rectangle
          Square
        Rhombus
      Ellipse
        Circle
      Triangle
        RightTriangle
        EquilateralTriangle
        IsocelesTriangle
      Parabola
      Line
      Hyperbola
  ThreeDimensionalShape
    Ellipsoid
      Sphere
    Prism
    Cylinder
    Cone
    Cube
    Tetrahedron
    Hyperboloid
      OneSheetedHyperboloid
      TwoSheetedHyperboloid
    Plane

```

20

C++ Virtual Functions and Polymorphism: Solutions

SOLUTIONS

- 20.2** What are `virtual` functions? Describe a circumstance in which `virtual` functions would be appropriate.
ANS: Virtual functions are functions with the same function prototype that are defined throughout a class hierarchy. At least the base class occurrence of the function is preceded by the keyword `virtual`. Virtual functions are used to enable generic processing of an entire class hierarchy of objects through a base class pointer. For example, in a shape hierarchy, all shapes can be drawn. If all shapes are derived from a base class `Shape` which contains a `virtual draw` function, then generic processing of the hierarchy can be performed by calling every shape's `draw` generically through a base class `Shape` pointer.
- 20.3** Given that constructors cannot be `virtual`, describe a scheme for how you might achieve a similar effect.
ANS: Create a `virtual` function called `initialize` that the constructor invokes.
- 20.4** How is it that polymorphism enables you to program “in the general” rather than “in the specific.” Discuss the key advantages of programming “in the general.”
ANS: Polymorphism enables the programmer to concentrate on the processing of common operations that are applied to all data types in the system without going into the individual details of each data type. The general processing capabilities are separated from the internal details of each type.
- 20.5** Discuss the problems of programming with `switch` logic. Explain why polymorphism is an effective alternative to using `switch` logic.
ANS: The main problem with programming using the `switch` structure is extensibility and maintainability of the program. A program containing many `switch` structures is difficult to modify. Many, but not necessarily all, `switch` structures will need to add or remove cases for a specified type. [Note: `switch` logic includes `if/else` structures which are more flexible than the `switch` structure.]
- 20.6** Distinguish between static binding and dynamic binding. Explain the use of `virtual` functions and the *vtable* in dynamic binding.
ANS: Static binding is performed at compile-time when a function is called via a specific object or via a pointer to an object. Dynamic binding is performed at run-time when a `virtual` function is called via a base class pointer to a derived class object (the object can be of any derived class). The `virtual` functions table (*vtable*) is used at run-time to enable the proper function to be called for the object to which the base class pointer “points”. Each class containing `virtual` functions has its own *vtable* that specifies where the `virtual` functions for that class are located. Every object of a class with `virtual` functions contains a hidden pointer to the class's *vtable*. When a `virtual` function is called via a base class pointer, the hidden pointer is dereferenced to locate the *vtable*, then the *vtable* is searched for the proper function call.
- 20.7** Distinguish between inheriting interface and inheriting implementation. How do inheritance hierarchies designed for inheriting interface differ from those designed for inheriting implementation?

ANS: When a class inherits implementation, it inherits previously defined functionality from another class. When a class inherits interface, it inherits the definition of what the interface to the new class type should be. The implementation is then provided by the programmer defining the new class type. Inheritance hierarchies designed for inheriting implementation are used to reduce the amount of new code that is being written. Such hierarchies are used to facilitate software reusability. Inheritance hierarchies designed for inheriting interface are used to write programs that perform generic processing of many class types. Such hierarchies are commonly used to facilitate software extensibility (i.e., new types can be added to the hierarchy without changing the generic processing capabilities of the program).

20.8 Distinguish between `virtual` functions and pure `virtual` functions.

ANS: A `virtual` function must have a definition in the class in which it is declared. A pure `virtual` function does not provide a definition. Classes derived directly from the abstract base class must provide definitions for the inherited pure `virtual` functions in order to avoid becoming an abstract base class.

20.9 (True/False) All `virtual` functions in an abstract base class must be declared as pure `virtual` functions.

ANS: False.

20.10 Suggest one or more levels of abstract base classes for the Shape hierarchy discussed in this chapter (the first level is Shape and the second level consists of the classes `TwoDimensionalShape` and `ThreeDimensionalShape`).

20.11 How does polymorphism promote extensibility?

ANS: Polymorphism makes programs more extensible by making all function calls generic. When a new class type with the appropriate `virtual` functions is added to the hierarchy, no changes need to be made to the generic function calls.

20.12 You have been asked to develop a flight simulator that will have elaborate graphical outputs. Explain why polymorphic programming would be especially effective for a problem of this nature.

20.13 Develop a basic graphics package. Use the Shape class inheritance hierarchy from Chapter 19. Limit yourself to two-dimensional shapes such as squares, rectangles, triangles and circles. Interact with the user. Let the user specify the position, size, shape and fill characters to be used in drawing each shape. The user can specify many items of the same shape. As you create each shape, place a `Shape *` pointer to each new Shape object into an array. Each class has its own `draw` member function. Write a polymorphic screen manager that walks through the array (preferably using an iterator) sending `draw` messages to each object in the array to form a screen image. Redraw the screen image each time the user specifies an additional shape.

20.14 In Exercise 19.12, you developed a Shape class hierarchy and defined the classes in the hierarchy. Modify the hierarchy so that class Shape is an abstract base class containing the interface to the hierarchy. Derive TwoDimensionalShape and ThreeDimensionalShape from class Shape—these classes should also be abstract. Use a virtual print function to output the type and dimensions of each class. Also include virtual area and volume functions so these calculations can be performed for objects of each concrete class in the hierarchy. Write a driver program that tests the Shape class hierarchy.

ANS:

```

1 // SHAPE.H
2 // Definition of base-class Shape
3 #ifndef SHAPE_H
4 #define SHAPE_H
5
6 #include <iostream>
7 using std::ostream;
8
9 class Shape {
10     friend ostream & operator<<( ostream &, Shape & );
11 public:
12     Shape( double = 0, double = 0 );
13     double getCenterX() const;
14     double getCenterY() const;
15     virtual void print() const = 0;
16 protected:
17     double xCenter;
18     double yCenter;
19 };
20
21 #endif

```

```

22 // SHAPE.CPP
23 // Member and friend definitions for Shape
24 #include "shape.h"
25
26 Shape::Shape( double x, double y )
27 {
28     xCenter = x;
29     yCenter = y;
30 }
31
32 double Shape::getCenterX() const { return xCenter; }
33
34 double Shape::getCenterY() const { return yCenter; }
35
36 ostream & operator<<( ostream &out, Shape &s )
37 {
38     s.print();
39     return out;
40 }

```

```

41 // TWODIM.H
42 // Definition of class TwoDimensionalShape
43 #ifndef TWODIM_H
44 #define TWODIM_H
45
46 #include "shape.h"
47
48 class TwoDimensionalShape : public Shape {
49 public:
50     TwoDimensionalShape( double x, double y ) : Shape( x, y ) { }
51     virtual double area() const = 0;
52 };
53
54 #endif

```

```
55 // THREEDIM.H
56 // Definition of class ThreeDimensionalShape
57 #ifndef THREEDIM_H
58 #define THREEDIM_H
59
60 #include "shape.h"
61
62 class ThreeDimensionalShape : public Shape {
63 public:
64     ThreeDimensionalShape( double x, double y ) : Shape( x, y ) { }
65     virtual double area() const = 0;
66     virtual double volume() const = 0;
67 };
68
69 #endif

70 // CIRCLE.H
71 // Definition of class Circle
72 #ifndef CIRCLE_H
73 #define CIRCLE_H
74
75 #include "twodim.h"
76
77 class Circle : public TwoDimensionalShape {
78 public:
79     Circle( double = 0, double = 0, double = 0 );
80     double getRadius() const;
81     double area() const;
82     void print() const;
83 private:
84     double radius;
85 };
86
87 #endif

88 // CIRCLE.CPP
89 // Member function definitions for Circle
90 #include "circle.h"
91
92 #include <iostream>
93 using std::cout;
94
95 Circle::Circle( double r, double x, double y )
96     : TwoDimensionalShape( x, y ) { radius = r > 0 ? r : 0; }
97
98 double Circle::getRadius() const { return radius; }
99
100 double Circle::area() const { return 3.14159 * radius * radius; }
101
102 void Circle::print() const
103 {
104     cout << "Circle with radius " << radius << "; center at ("
105         << xCenter << ", " << yCenter << ");\narea of " << area() << '\n';
106 }
```

```
107 // SQUARE.H
108 // Definition of class Square
109 #ifndef SQUARE_H
110 #define SQUARE_H
111
112 #include "twodim.h"
113
114 class Square : public TwoDimensionalShape {
115 public:
116     Square( double = 0, double = 0, double = 0 );
117     double getSideLength() const;
118     double area() const;
119     void print() const;
120 private:
121     double sideLength;
122 };
123
124 #endif
```

```
125 // SQUARE.CPP
126 // Member function definitions for Square
127 #include "square.h"
128
129 #include <iostream>
130 using std::cout;
131
132 Square::Square( double s, double x, double y )
133     : TwoDimensionalShape( x, y ) { sideLength = s > 0 ? s : 0; }
134
135 double Square::getSideLength() const { return sideLength; }
136
137 double Square::area() const { return sideLength * sideLength; }
138
139 void Square::print() const
140 {
141     cout << "Square with side length " << sideLength << "; center at ("
142         << xCenter << ", " << yCenter << ");\narea of " << area() << '\n';
143 }
```

```
144 // CUBE.H
145 // Definition of class Cube
146 #ifndef CUBE_H
147 #define CUBE_H
148
149 #include "threedim.h"
150
151 class Cube : public ThreeDimensionalShape {
152 public:
153     Cube( double = 0, double = 0, double = 0 );
154     double area() const;
155     double volume() const;
156     double getSideLength() const;
157     void print() const;
158 private:
159     double sideLength;
160 };
161
162 #endif
```

```

163 // CUBE.CPP
164 // Member function definitions for Cube
165 #include "cube.h"
166
167 #include <iostream>
168 using std::cout;
169
170 Cube::Cube( double s, double x, double y )
171     : ThreeDimensionalShape( x, y ) { sideLength = s > 0 ? s : 0; }
172
173 double Cube::area() const { return 6 * sideLength * sideLength; }
174
175 double Cube::volume() const
176     { return sideLength * sideLength * sideLength; }
177
178 double Cube::getSideLength() const { return sideLength; }
179
180 void Cube::print() const
181 {
182     cout << "Cube with side length " << sideLength << "; center at ("
183         << xCenter << ", " << yCenter << ");\narea of "
184         << area() << "; volume of " << volume() << '\n';
185 }

```

```

186 // SPHERE.H
187 // Definition of class Sphere
188 #ifndef SPHERE_H
189 #define SPHERE_H
190
191 #include "threedim.h"
192
193 class Sphere : public ThreeDimensionalShape {
194 public:
195     Sphere( double = 0, double = 0, double = 0 );
196     double area() const;
197     double volume() const;
198     double getRadius() const;
199     void print() const;
200 private:
201     double radius;
202 };
203
204 #endif

```

```

205 // SQUARE.CPP
206 // Member function definitions for Square
207 #include "square.h"
208
209 #include <iostream>
210 using std::cout;
211
212 Square::Square( double s, double x, double y )
213     : TwoDimensionalShape( x, y ) { sideLength = s > 0 ? s : 0; }
214
215 double Square::getSideLength() const { return sideLength; }
216
217 double Square::area() const { return sideLength * sideLength; }
218
219 void Square::print() const
220 {
221     cout << "Square with side length " << sideLength << "; center at ("
222         << xCenter << ", " << yCenter << ");\narea of " << area() << '\n';
223 }

```

```
224 // Exercise 20.14 solution
225 // Driver to test Shape hierarchy
226 #include <iostream>
227
228 using std::cout;
229 using std::endl;
230 using std::cin;
231 using std::ios;
232
233 #include "circle.h"
234 #include "square.h"
235 #include "sphere.h"
236 #include "cube.h"
237
238 int main()
239 {
240     Circle cir( 3.5, 6, 9 );
241     Square sqr( 12, 2, 2 );
242     Sphere sph( 5, 1.5, 4.5 );
243     Cube cub( 2.2 );
244     Shape *ptr[ 4 ] = { &cir, &sqr, &sph, &cub };
245
246     for ( int x = 0; x < 4; ++x )
247         cout << *( ptr[ x ] ) << '\n';
248
249     return 0;
250 }
```

Circle with radius 3.5; center at (6, 9);
area of 38.4845

Square with side length 12; center at (2, 2);
area of 144

Sphere with radius 5; center at (1.5, 4.5);
area of 314.159; volume of 523.598

Cube with side length 2.2; center at (0, 0);
area of 29.04; volume of 10.648

21

C++ Stream Input/Output: Solutions

EXERCISES

21.6 Write a statement for each of the following:

a) Print integer 40000 left-justified in a 15-digit field.

ANS: `cout << setiosflags(ios::left) << setw(15) << 40000 << '\n';`

b) Read a string into character array variable state.

ANS: `cin >> state;`

c) Print 200 with and without a sign.

ANS:

```
cout << setiosflags( ios::showpos ) << 200 << setw( 4 )  
    << resetiosflags( ios::showpos ) << 200 << '\n';
```

d) Print the decimal value 100 in hexadecimal form preceded by 0x.

ANS: `cout << setiosflags(ios::showbase) << hex << 100 << '\n';`

e) Read characters into array s until the character 'p' is encountered up to a limit of 10 characters (including the terminating null character). Extract the delimiter from the input stream and discard it.

ANS: `cin.getline(s, 10, 'p');`

f) Print 1.234 in a 9-digit field with preceding zeros.

ANS:

```
cout << setiosflags( ios::fixed | ios::showpoint ) << setw( 9 )  
    << setfill( '0' ) << setiosflags( ios::internal ) << 1.234 << '\n';
```

g) Read a string of the form "characters" from the standard input. Store the string in character array s. Eliminate the quotation marks from the input stream. Read a maximum of 50 characters (including the terminating null character).

21.7 Write a program to test inputting integer values in decimal, octal and hexadecimal format. Output each integer read by the program in all three formats. Test the program with the following input data: 10, 010, 0x10.

ANS:

```

1 // Exercise 21.7 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6 using std::cin;
7 using std::ios;
8
9 #include <iomanip>
10
11 using std::setiosflags;
12 using std::hex;
13 using std::oct;
14 using std::dec;
15
16 int main()
17 {
18     int integer;
19
20     cout << "Enter an integer: ";
21     cin >> integer;
22
23     cout << setiosflags( ios::showbase ) << "As a decimal number " << dec
24         << integer << "\nAs an octal number " << oct << integer
25         << "\nAs a hexadecimal number " << hex << integer << endl;
26
27     cout << "\nEnter an integer in octal format\n";
28     cin >> setiosflags( ios::showbase ) >> oct >> integer;
29
30     cout << setiosflags( ios::showbase ) << "As a decimal number " << dec
31         << integer << "\nAs an octal number " << oct << integer
32         << "\nAs a hexadecimal number " << hex << integer << endl;
33
34     cout << "\nEnter an integer in hexadecimal format\n";
35     cin >> setiosflags( ios::showbase ) >> hex >> integer;
36
37     cout << setiosflags( ios::showbase ) << "As a decimal number " << dec
38         << integer << "\nAs an octal number " << oct << integer
39         << "\nAs a hexadecimal number " << hex << integer << endl;
40
41     return 0;
42 }
```

```

Enter an integer: 10
As a decimal number 10
As an octal number 012
As a hexadecimal number 0xa

Enter an integer in octal format
010
As a decimal number 8
As an octal number 010
As a hexadecimal number 0x8

Enter an integer in hexadecimal format
0x10
As a decimal number 16
As an octal number 020
As a hexadecimal number 0x10
```

21.8 Write a program that prints pointer values using casts to all the integer data types. Which ones print strange values? Which ones cause errors?

ANS:

```

1 // Exercise 21.8 Solution
2 #include <iostream>
3
4 using std::cout;
5
6 int main()
7 {
8     char *string = "test";
9
10    cout << "Value of string is           : "
11         << string << '\n';
12
13    cout << "Value of static_cast<void *>( string ) is : "
14         << static_cast<void *>( string ) << '\n';
15
16        // The Following generate errors.
17        // reinterpret_cast will allow this type of casting
18
19    /* cout << "Value of static_cast<char>(string) is : "
20         << static_cast<char>( string ) << '\n';
21
22    cout << "Value of static_cast<int>(string) is : "
23         << static_cast<int>( string ) << '\n';
24
25    cout << "Value of static_cast<long>(string) is : "
26         << static_cast<long>( string ) << '\n';
27
28    cout << "Value of static_cast<short>(string) is : "
29         << static_cast<short>( string ) << '\n';
30
31    cout << "Value of static_cast<unsigned>(string) is : "
32         << static_cast<unsigned>( string )
33    */
34
35    return 0;
36 }
```

```

Value of string is           : test
Value of static_cast<void *>( string ) is : 0041A178
```

21.9 Write a program to test the results of printing the integer value 12345 and the floating-point value 1.2345 in various-size fields. What happens when the values are printed in fields containing fewer digits than the values?

ANS:

```
1 // Exercise 21.9 Solution
2 #include <iostream>
3
4 using std::cout;
5
6 #include <iomanip>
7
8 using std::setw;
9
10 int main()
11 {
12     int x = 12345;
13     double y = 1.2345;
14
15     for ( int loop = 0; loop <= 10; ++loop )
16         cout << x << " printed in a field of size " << loop << " is "
17             << setw( loop ) << x << '\n' << y << " printed in a field "
18             << "of size " << loop << " is " << setw( loop ) << y << '\n';
19
20     return 0;
21 }
```

```
12345 printed in a field of size 0 is 12345
1.2345 printed in a field of size 0 is 1.2345
12345 printed in a field of size 1 is 12345
1.2345 printed in a field of size 1 is 1.2345
12345 printed in a field of size 2 is 12345
1.2345 printed in a field of size 2 is 1.2345
12345 printed in a field of size 3 is 12345
1.2345 printed in a field of size 3 is 1.2345
12345 printed in a field of size 4 is 12345
1.2345 printed in a field of size 4 is 1.2345
12345 printed in a field of size 5 is 12345
1.2345 printed in a field of size 5 is 1.2345
12345 printed in a field of size 6 is 12345
1.2345 printed in a field of size 6 is 1.2345
12345 printed in a field of size 7 is 12345
1.2345 printed in a field of size 7 is 1.2345
12345 printed in a field of size 8 is 12345
1.2345 printed in a field of size 8 is 1.2345
12345 printed in a field of size 9 is 12345
1.2345 printed in a field of size 9 is 1.2345
12345 printed in a field of size 10 is 12345
1.2345 printed in a field of size 10 is 1.2345
```

21.10 Write a program that prints the value 100.453627 rounded to the nearest digit, tenth, hundredth, thousandth and ten thousandth.

ANS:

```
1 // Exercise 21.10 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6 using std::ios;
7
8 #include <iomanip>
9
10 using std::setw;
11 using std::setprecision;
12 using std::setiosflags;
13
14 int main()
15 {
16     double x = 100.453627;
17
18     cout << setiosflags( ios::fixed );
19     for ( int loop = 0; loop <= 5; ++loop )
20         cout << setprecision( loop ) << "Rounded to " << loop
21             << " digit(s) is " << x << endl;
22
23     return 0;
24 }
```

```
Rounded to 0 digit(s) is 100
Rounded to 1 digit(s) is 100.5
Rounded to 2 digit(s) is 100.45
Rounded to 3 digit(s) is 100.454
Rounded to 4 digit(s) is 100.4536
Rounded to 5 digit(s) is 100.45363
```

21.11 Write a program that inputs a string from the keyboard and determines the length of the string. Print the string using twice the length as the field width.

ANS:

```
1 // Exercise 21.11 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6 using std::cin;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 #include <cstring>
13
14 const int SIZE = 80;
15
16 int main()
17 {
18     char string[ SIZE ];
19     int stringLength;
20
21     cout << "Enter a string: ";
22     cin >> string;
23
24     stringLength = strlen( string );
25
26     cout << "the length of the string is " << strlen( string ) << endl;
27
28     // print string using twice the length as field with
29     cout << setw( 2 * stringLength ) << string << endl;
30
31     return 0;
32 }
```

```
Enter a string: castle
the length of the string is 6
  castle
```

21.12 Write a program that converts integer Fahrenheit temperatures from 0 to 212 degrees to floating-point Celsius temperatures with 3 digits of precision. Use the formula

$$\text{celsius} = 5.0 / 9.0 * (\text{fahrenheit} - 32);$$

to perform the calculation. The output should be printed in two right-justified columns and the Celsius temperatures should be preceded by a sign for both positive and negative values.

ANS:

```

1 // Exercise 21.12 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::ios;
6
7 #include <iomanip>
8
9 using std::setw;
10 using std::setprecision;
11 using std::setiosflags;
12 using std::resetiosflags;
13
14 int main()
15 {
16     double celsius;
17
18     cout << setw( 20 ) << "Fahrenheit " << setw( 20 ) << "Celsius\n"
19         << setiosflags( ios::fixed | ios::showpoint );
20
21     for ( int fahrenheit = 0; fahrenheit <= 212; ++fahrenheit ) {
22         celsius = 5.0 / 9.0 * ( fahrenheit - 32 );
23         cout << setw( 15 ) << resetiosflags( ios::showpos ) << fahrenheit
24             << setw( 23 ) << setprecision( 3 ) << setiosflags( ios::showpos )
25             << celsius << '\n';
26     }
27
28     return 0;
29 }

```

Fahrenheit	Celsius
0	-17.778
1	-17.222
2	-16.667
3	-16.111
4	-15.556
5	-15.000
...	
206	+96.667
207	+97.222
208	+97.778
209	+98.333
210	+98.889
211	+99.444
212	+100.000

21.13 In some programming languages, strings are entered surrounded by either single or double quotation marks. Write a program that reads the three strings `suzy`, `"suzy"` and `'suzy'`. Are the single and double quotes ignored or read as part of the string?

ANS:

```
1 // Exercise 21.13 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::cin;
6
7 const int SIZE = 80;
8
9 int main()
10 {
11     char string[ SIZE ];
12
13     for ( int k = 0; k < 3; ++k ) {
14         cout << "Enter a string: ";
15         cin >> string;
16         cout << "String is " << string << '\n';
17     }
18
19     return 0;
20 }
```

```
Enter a string: "vaccum"
String is "vaccum"
Enter a string: 'grape'
String is 'grape'
Enter a string: water
String is water
```

21.14 In Fig. 18.3, the stream-extraction and -insertion operators were overloaded for input and output of objects of the `PhoneNumber` class. Rewrite the stream-extraction operator to perform the following error checking on input. The `operator>>` function will need to be entirely recoded.

- Input the entire phone number into an array. Test that the proper number of characters has been entered. There should be a total of 14 characters read for a phone number of the form (800) 555-1212. Use the stream member function `clear` to set `ios::failbit` for improper input.
- The area code and exchange do not begin with 0 or 1. Test the first digit of the area code and exchange portions of the phone number to be sure that neither begins with 0 or 1. Use stream member function `clear` to set `ios::failbit` for improper input.
- The middle digit of an area code used to always be 0 or 1 (although this has changed recently). Test the middle digit for a value of 0 or 1. Use the stream member function `clear` to set `ios::failbit` for improper input. If none of the above operations results in `ios::failbit` being set for improper input, copy the three parts of the telephone number into the `areaCode`, `exchange` and `line` members of the `PhoneNumber` object. In the main program, if `ios::failbit` has been set on the input, have the program print an error message and end rather than print the phone number.

ANS:

```

1 // P21_14.H
2 #ifndef P21_14_H
3 #define P21_14_H
4
5 #include <iostream>
6
7 using std::cout;
8 using std::endl;
9 using std::cin;
10 using std::ios;
11 using std::ostream;
12 using std::istream;
13 using std::cerr;
14
15 #include <string>
16 using std::string;
17
18 #include <cstdlib>
19
20 class PhoneNumber {
21     friend ostream& operator<<( ostream&, PhoneNumber& );
22     friend istream& operator>>( istream&, PhoneNumber& );
23 public:
24     PhoneNumber();
25 private:
26     char phone[ 15 ];
27     char areaCode[ 4 ];
28     char exchange[ 4 ];
29     char line[ 5 ];
30 };
31
32 #endif

```

```

33 // P21_14M.cpp
34 // member function definition definition for p21_14.cpp
35 #include "p21_14.h"
36
37 PhoneNumber::PhoneNumber()
38 {
39     phone[ 0 ] = '\0';
40     areaCode[ 0 ] = '\0';
41     exchange[ 0 ] = '\0';
42     line[ 0 ] = '\0';
43 }
44
45 ostream &operator<<( ostream &output, PhoneNumber &number )
46 {

```

```

47     output << "(" << number.areaCode << ")" " << number.exchange
48         << "-" << number.line << '\n';
49
50     return output;
51 }
52
53 istream &operator>>( istream &input, PhoneNumber &number )
54 {
55     cin.getline( number.phone, 15 );
56
57     if ( strlen( number.phone ) != 14 )
58         cin.clear( ios::failbit );
59
60     if ( number.phone[ 1 ] == '0' || number.phone[ 6 ] == '0' ||
61         number.phone[ 1 ] == '1' || number.phone[ 6 ] == '1' )
62         cin.clear( ios::failbit );
63
64     if ( number.phone[ 2 ] != '0' && number.phone[ 2 ] != '1' )
65         cin.clear( ios::failbit );
66
67     if ( !cin.fail() ) {
68         int loop = 0;
69         for ( ; loop <= 2; ++loop ) {
70             number.areaCode[ loop ] = number.phone[ loop + 1 ];
71             number.exchange[ loop ] = number.phone[ loop + 6 ];
72         }
73
74         number.areaCode[ loop ] = number.exchange[ loop ] = '\0';
75
76         for ( loop = 0; loop <= 3; ++loop )
77             number.line[ loop ] = number.phone[ loop + 10 ];
78
79         number.line[ loop ] = '\0';
80     }
81     else {
82         cerr << "Invalid phone number entered.\n";
83         exit( 1 );
84     }
85
86     return input;
87 }

```

```

88 // driver for p21_14.cpp
89 #include "p21_14.h"
90
91 int main()
92 {
93     PhoneNumber telephone;
94
95     cout << "Enter a phone number in the form (123) 456-7890:\n";
96     cin >> telephone;
97
98     cout << "The phone number entered was: " << telephone << endl;
99
100    cout << "Now enter an invalid phone number:\n";
101    cin >> telephone;
102
103    return 0;
104 }

```

```
Enter a phone number in the form (123) 456-7890:  
(800) 987-4567  
The phone number entered was: (800) 987-4567
```

```
Now enter an invalid phone number:  
abcdefghijkl
```

```
Invalid phone number entered.
```

21.15 Write a program that accomplishes each of the following:

- Create the user-defined class `Point` that contains the private integer data members `xCoordinate` and `yCoordinate` and declares stream-insertion and stream-extraction overloaded operator functions as friends of the class.
- Define the stream-insertion and stream-extraction operator functions. The stream-extraction operator function should determine if the data entered are valid data, and if not, it should set the `ios::failbit` to indicate improper input. The stream-insertion operator should not be able to display the point after an input error occurred.
- Write a `main` function that tests input and output of user-defined class `Point` using the overloaded stream-extraction and stream-insertion operators.

ANS:

```

1 // P21_15.H
2 #ifndef P21_15_H
3 #define P21_15_H
4 #include <iostream.h>
5
6 class Point {
7     friend ostream &operator<<( ostream&, Point& );
8     friend istream &operator>>( istream&, Point& );
9 private:
10     int xCoordinate;
11     int yCoordinate;
12 };
13
14 #endif

```

```

15 // P21_15M.cpp
16 // member function definitions for p21_15.cpp
17 #include "p21_15.h"
18
19 ostream& operator<<( ostream& out, Point& p )
20 {
21     if ( !cin.fail() )
22         cout << "(" << p.xCoordinate << ", " << p.yCoordinate << ")" << '\n';
23     else
24         cout << "\nInvalid data\n";
25
26     return out;
27 }
28
29 istream& operator>>( istream& i, Point& p )
30 {
31     if ( cin.peek() != '(' )
32         cin.clear( ios::failbit );
33     else
34         i.ignore(); // skip (
35
36     cin >> p.xCoordinate;
37
38     if ( cin.peek() != ',' )
39         cin.clear( ios::failbit );
40     else {
41         i.ignore(); // skip ,
42
43         if ( cin.peek() == ' ' )
44             i.ignore(); // skip space
45         else
46             cin.clear( ios::failbit );
47     }
48
49     cin >> p.yCoordinate;
50
51     if ( cin.peek() == ')' )
52         i.ignore(); // skip )
53     else
54         cin.clear( ios::failbit );

```

```

55
56     return i;
57 }

58 // driver for p21_15.cpp
59 #include "p21_15.h"
60
61 int main()
62 {
63     Point pt;
64
65     cout << "Enter a point in the form (x, y):\n";
66     cin >> pt;
67
68     cout << "Point entered was: " << pt << endl;
69     return 0;
70 }

```

```

Enter a point in the form (x, y):
(7, 8)
Point entered was: (7, 8)

```

21.16 Write a program that accomplishes each of the following:

- Create the user-defined class `Complex` that contains the private integer data members `real` and `imaginary` and declares stream-insertion and stream-extraction overloaded operator functions as friends of the class.
- Define the stream-insertion and -extraction operator functions. The stream-extraction operator function should determine if the data entered are valid, and if not, it should set `ios::failbit` to indicate improper input. The input should be of the form

3 + 8i

- The values can be negative or positive, and it is possible that one of the two values is not provided. If a value is not provided, the appropriate data member should be set to 0. The stream-insertion operator should not be able to display the point if an input error occurred. The output format should be identical to the input format shown above. For negative imaginary values, a minus sign should be printed rather than a plus sign.
- Write a `main` function that tests input and output of user-defined class `Complex` using the overloaded stream-extraction and stream-insertion operators.

ANS:

```

1 // P21_16.H
2 #ifndef P21_16_H
3 #define P21_16_H
4 #include <iostream>
5
6 using std::ostream;
7 using std::istream;
8
9 class Complex {
10     friend ostream &operator<<( ostream&, Complex& );
11     friend istream &operator>>( istream&, Complex& );
12 public:
13     Complex( void );           // constructor
14 private:
15     int real;
16     int imaginary;
17 };
18
19 #endif

```

```
20 // P21_16M.cpp
21 // member function definitions for p21_16.cpp
22 #include <iostream>
23
24 using std::cout;
25 using std::cin;
26 using std::ios;
27 using std::ostream;
28 using std::istream;
29
30 #include <iomanip>
31
32 using std::setiosflags;
33 using std::resetiosflags;
34
35 #include "p21_16.h"
36
37 Complex::Complex( void )
38 {
39     real = 0;
40     imaginary = 0;
41 }
42
43 ostream &operator<<( ostream &output, Complex &c )
44 {
45     if ( !cin.fail() )
46         output << c.real
47             << setiosflags( ios::showpos )
48             << c.imaginary << "i\n"
49             << resetiosflags( ios::showpos );
50     else
51         output << "Invalid Data Entered" << '\n';
52
53     return output;
54 }
55
56 istream &operator>>( istream &input, Complex &c )
57 {
58     int number, multiplier;
59     char temp;
60
61     input >> number;
62
63     if ( cin.peek() == ' ' ) {           // case a + bi
64         c.real = number;
65         cin >> temp;
66
67         multiplier = ( temp == '+' ) ? 1 : -1;
68
69         if ( cin.peek() != ' ' )
70             cin.clear( ios::failbit ); // set bad bit
71         else {
72             if ( cin.peek() == ' ' ) {
73                 input >> c.imaginary;
74                 c.imaginary *= multiplier;
75
76                 cin >> temp;
77                 if ( cin.peek() != '\n' )
78                     cin.clear( ios::failbit ); // set bad bit
79             }
80             else
81                 cin.clear( ios::failbit ); // set bad bit
82         }
83     }
84 }
85 else if ( cin.peek() == 'i' ) {       // case bi
86     cin >> temp;
87 }
```

```
88     if ( cin.peek() == '\n' ) {
89         c.real = 0;
90         c.imaginary = number;
91     }
92     else
93         cin.clear( ios::failbit );    // set bad bit
94 }
95
96 else if ( cin.peek() == '\n' ) {    // case a
97     c.real = number;
98     c.imaginary = 0;
99 }
100 else
101     cin.clear( ios::failbit );    // set bad bit
102
103 return input;
104 }
```

```
105 // driver for p21_16.cpp
106 #include <iostream>
107
108 using std::cout;
109 using std::cin;
110 using std::endl;
111
112 #include "p21_16.h"
113
114 int main()
115 {
116     Complex complex;
117
118     cout << "Input a complex number in the form A + Bi:\n";
119     cin >> complex;
120
121     cout << "Complex number entered was:\n" << complex << endl;
122     return 0;
123 }
```

```
Input a complex number in the form A + Bi:
7 - 7777i
Complex number entered was:
7-7777i
```


21.17 Write a program that uses a `for` structure to print a table of ASCII values for the characters in the ASCII character set from 33 to 126. The program should print the decimal value, octal value, hexadecimal value and character value for each character. Use the stream manipulators `dec`, `oct` and `hex` to print the integer values.

ANS:

```

1 // Exercise 21.17 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl; ;
6 using std::ios;
7
8 #include <iomanip>
9
10 using std::setw;
11 using std::setiosflags;
12 using std::dec;
13 using std::oct;
14 using std::hex;
15
16 int main()
17 {
18     cout << setw( 7 ) << "Decimal" << setw( 9 ) << "Octal " << setw( 15 )
19         << "Hexadecimal " << setw( 13 ) << "Character"
20         << setiosflags( ios::showbase ) << '\n';
21
22     for ( int loop = 33; loop <= 126; ++loop )
23         cout << setw( 7 ) << dec << loop << setw( 9 ) << oct << loop
24             << setw( 15 ) << hex << loop << setw(13)
25             << static_cast<char>( loop ) << endl;
26
27     return 0;
28 }

```

Decimal	Octal	Hexadecimal	Character
33	041	0x21	!
34	042	0x22	"
35	043	0x23	#
36	044	0x24	\$
37	045	0x25	%
38	046	0x26	&
39	047	0x27	'
40	050	0x28	(
41	051	0x29)
42	052	0x2a	*
43	053	0x2b	+
...			
120	0170	0x78	x
121	0171	0x79	y
122	0172	0x7a	z
123	0173	0x7b	{
124	0174	0x7c	
125	0175	0x7d	}
126	0176	0x7e	~

21.18 Write a program to show that the `getline` and three-argument `get` `istream` member functions each end the input string with a string-terminating null character. Also, show that `get` leaves the delimiter character on the input stream while `getline` extracts the delimiter character and discards it. What happens to the unread characters in the stream?

ANS:

```

1 // Exercise 21.18 Solution
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6 using std::cin;
7 using std::ios;
8
9 #include <cctype>
10
11 const int SIZE = 80;
12
13 int main()
14 {
15     char array[ SIZE ], array2[ SIZE ], c;
16
17     cout << "Enter a sentence to test getline() and get():\n";
18     cin.getline( array, SIZE, '*' );
19     cout << array << '\n';
20
21     cin >> c; // read next character in input
22     cout << "The next character in the input is: " << c << '\n';
23
24     cin.get( array2, SIZE, '*' );
25     cout << array2 << '\n';
26
27     cin >> c; // read next character in input
28     cout << "The next character in the input is: " << c << '\n';
29
30     return 0;
31 }
```

```

Enter a sentence to test getline() and get():
wishing*on*a*star
wishing
The next character in the input is: o
n
The next character in the input is: *
```

21.19 Write a program that creates the user-defined manipulator `skipwhite` to skip leading whitespace characters in the input stream. The manipulator should use the `isspace` function from the `<cctype>` library to test if the character is a whitespace character. Each character should be input using the `istream` member function `get`. When a non-whitespace character is encountered, the `skipwhite` manipulator finishes its job by placing the character back on the input stream and returning an `istream` reference.

Test the manipulator by creating a `main` function in which the `ios::skipws` flag is unset so that the stream-extraction operator does not automatically skip whitespace. Then test the manipulator on the input stream by entering a character preceded by whitespace as input. Print the character that was input to confirm that a whitespace character was not input.

22

C++ Templates: Solutions

SOLUTIONS

22.3 Use a nontype parameter `numberOfElements` and a type parameter `elementType` to help create a template for the `Array` class we developed in Chapter 18, “Operator Overloading.” This template will enable `Array` objects to be instantiated with a specified number of elements of a specified element type at compile time.

ANS:

```
1  #ifndef ARRAY1_H
2  #define ARRAY1_H
3
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8  using std::cin;
9
10 #include <cstdlib>
11 #include <cassert>
12
13 template < class elementType, int numberOfElements >
14 class Array {
15 public:
16     Array(); // default constructor
17     ~Array(); // destructor
18     int getSize() const; // return size
19     bool operator==( const Array & ) const; // compare equal
20     bool operator!=( const Array & ) const; // compare !equal
21     elementType &operator[]( int ); // subscript operator
22     static int getArrayCount(); // Return count of
23     // arrays instantiated.
24     void inputArray(); // input the array elements
25     void outputArray() const; // output the array elements
26 private:
27     elementType ptr[ numberOfElements ]; // pointer to first element of array
28     int size; // size of the array
29     static int arrayCount; // # of Arrays instantiated
30 };
```

```
31
32 // Initialize static data member at file scope
33 template < class elementType, int numberOfElements >
34 int Array< elementType, numberOfElements >::arrayCount = 0; // no objects yet
35
36 // Default constructor for class Array
37 template < class elementType, int numberOfElements >
38 Array< elementType, numberOfElements >::Array()
39 {
40     ++arrayCount; // count one more object
41     size = numberOfElements;
42
43     for ( int i = 0; i < size; ++i )
44         ptr[ i ] = 0; // initialize array
45 }
46
47 // Destructor for class Array
48 template < class elementType, int numberOfElements >
49 Array< elementType, numberOfElements >::~~Array() { --arrayCount; }
50
51 // Get the size of the array
52 template < class elementType, int numberOfElements >
53 int Array< elementType, numberOfElements >::getSize() const { return size; }
54
55 // Determine if two arrays are equal and
56 // return true or false.
57 template < class elementType, int numberOfElements >
58 bool Array< elementType, numberOfElements >::
59     operator==( const Array &right ) const
60 {
61     if ( size != right.size )
62         return false; // arrays of different sizes
63
64     for ( int i = 0; i < size; ++i )
65         if ( ptr[ i ] != right.ptr[ i ] )
66             return false; // arrays are not equal
67
68     return true; // arrays are equal
69 }
70
71 // Determine if two arrays are not equal and
72 // return true or false.
73 template < class elementType, int numberOfElements >
74 bool Array< elementType, numberOfElements >::
75     operator!=( const Array &right ) const
76 {
77     if ( size != right.size )
78         return true; // arrays of different sizes
79
80     for ( int i = 0; i < size; ++i )
81         if ( ptr[ i ] != right.ptr[ i ] )
82             return true; // arrays are not equal
83
84     return false; // arrays are equal
85 }
86
87 // Overloaded subscript operator
88 template < class elementType, int numberOfElements >
89 elementType &Array< elementType, numberOfElements >::
90     operator[]( int subscript )
91 {
```

```

92 // check for subscript out of range error
93 assert( 0 <= subscript && subscript < size );
94
95 return ptr[ subscript ]; // reference return creates lvalue
96 }
97
98 // Return the number of Array objects instantiated
99 template < class elementType, int numberOfElements >
100 int Array< elementType, numberOfElements >::getArrayCount()
101 { return arrayCount; }
102
103 // Input values for entire array.
104 template < class elementType, int numberOfElements >
105 void Array< elementType, numberOfElements >::inputArray()
106 {
107     for ( int i = 0; i < size; ++i )
108         cin >> ptr[ i ];
109 }
110
111 // Output the array values
112 template < class elementType, int numberOfElements >
113 void Array< elementType, numberOfElements >::outputArray() const
114 {
115     int i = 0;
116     for ( ; i < size; ++i ) {
117         cout << ptr[ i ] << ' ';
118
119         if ( ( i + 1 ) % 10 == 0 )
120             cout << '\n';
121     }
122
123     if ( i % 10 != 0 )
124         cout << '\n';
125 }
126
127 #endif

```

```

128 // Exercise 22.3 solution
129 #include <iostream>
130
131 using std::cout;
132
133 #include "arraytmp.h"
134
135 int main()
136 {
137     Array< int, 5 > intArray;
138
139     cout << "Enter " << intArray.getSize() << " integer values:\n";
140     intArray.inputArray();
141
142     cout << "\nThe values in intArray are:\n";
143     intArray.outputArray();
144
145     Array< float, 5 > floatArray;
146
147     cout << "\nEnter " << floatArray.getSize()
148         << " floating point values:\n";
149     floatArray.inputArray();
150

```

```

151     cout << "\nThe values in the doubleArray are:\n";
152     floatArray.outputArray();
153
154     return 0;
155 }

```

```

Enter 5 integer values:
99 98 97 96 95

```

```

The values in intArray are:
99 98 97 96 95

```

```

Enter 5 floating point values:
1.12 1.13 1.14 1.22 9.11

```

```

The values in the doubleArray are:
1.12 1.13 1.14 1.22 9.11

```

22.4 Write a program with class template `Array`. The template can instantiate an `Array` of any element type. Override the template with a specific definition for an `Array` of `float` elements (`class Array<float>`). The driver should demonstrate the instantiation of an `Array` of `int` through the template and should show that an attempt to instantiate an `Array` of `float` uses the definition provided in `class Array<float>`.

22.5 Which is more like a stencil—a class template or a template class? Explain your answer.

ANS: A class template can be viewed as a stencil from which a template class can be created. A template class can be viewed as a stencil from which objects of that class can be created. So, in a way, both can be viewed as stencils.

22.6 What performance problem can result from using class templates?

ANS: There can be a tremendous proliferation of code in the program due to many copies of code generated by the compiler.

22.7 Why is it appropriate to call a class template a parameterized type?

ANS: When creating template classes from a class template, it is necessary to provide a type (or possibly several types) to complete the definition of the new type being declared. For example, when creating an “array of integers” from an `Array` class template, the type `int` is provided to the class template to complete the definition of an array of integers.

22.8 Explain why you might use the statement

```
Array< Employee > workerList( 100 );
```

in a C++ program.

ANS: Declares an `Array` object to store `Employee` objects and passes 100 to the constructor.

22.9 Review your answer to Exercise 22.8. Now, why might you use the statement

```
Array< Employee > workerList;
```

in a C++ program?

ANS: Declares an `Array` object to store an `Employee`. The default constructor is called.

22.10 Explain the use of the following notation in a C++ program:

```
template< class T > Array< T >::Array( int s )
```

ANS: This notation is used to begin the definition of the `Array(int)` constructor for the class template `Array`.

22.11 Why might you typically use a nontype parameter with a class template for a container such as an array or stack?

ANS: To specify at compile time the size of the container class object being declared.

22.12 Describe how to provide a class for a specific type to override the class template for that type.

22.13 Describe the relationship between class templates and inheritance.

22.14 Suppose a class template has the header

```
template< class T1 > class C1
```

Describe the friendship relationships established by placing each of the following friendship declarations inside this class template header. Identifiers beginning with “f” are functions, identifiers beginning with “C” are classes and identifiers beginning with “T” can represent any type (i.e., built-in types or class types).

a) `friend void f1();`

ANS: Function `f1` is a friend of all template classes instantiated from class template `C1`.

b) `friend void f2(C1< T1 > &);`

ANS: Function `f2` for a specific type of `T1` is a friend of the template class of type `T1`. For example, if `T1` is of type `int` the function with the prototype

```
void f2( C1< int > & );
```

is a friend of the class `C1< int >`.

c) `friend void C2::f4();`

ANS: Function `f4` of class `C2` is a friend of all template classes instantiated from class template `C1`.

d) `friend void C3< T1 >::f5(C1< T1 > &);`

ANS: Function `f5` of class `C3` for a specific type of `T1` is a friend of the template class of type `T1`. For example, if `T1` is `int`, the function with the prototype

```
void C3< int >::f5( C1< int > & );
```

is a friend of the class `C1< int >`.

e) `friend class C5;`

ANS: Makes every member function of class `C5` a friend of all template classes instantiated from the class template `C1`.

f) `friend class C6< T1 >;`

ANS: For a specific type `T1`, makes every member function of `C6< T1 >` a friend of class `C1< T1 >`. For example, if `T1` is `int`, every member function of class `C6< int >` is a friend of `C1< int >`.

22.15 Suppose class template `Employee` has a `static` data member `count`. Suppose three template classes are instantiated from the class template. How many copies of the `static` data member will exist? How will the use of each be constrained (if at all)?

ANS: For `static` members of a class template, each template class instantiated receives its own copy of all the `static` members. Then all objects instantiated for a given template class can access that particular template class' `static` members

23

C++ Exception Handling: Solution

SOLUTIONS

23.20 Under what circumstances would the programmer not provide a parameter name when defining the type of the object that will be caught by a handler?

ANS: If there is no information in the object that is required in the handler, a parameter name is not required in the handler.

23.21 A program contains the statement

```
throw;
```

Where would you normally expect to find such a statement? What if that statement appeared in a different part of the program?

ANS: The statement would be found in an exception handler to rethrow an exception. If any `throw` expression occurs outside a `try` block, the function `unexpected` is called.

23.22 Under what circumstances would you use the following statement?

```
catch(...) { throw; }
```

ANS: The preceding statement is used to `catch` any exception and rethrow it for handling by an exception handler in a function within the call stack.

23.23 Compare and contrast exception handling with the various other error-processing schemes discussed in the text.

ANS: Exception handling enables the programmer to build more robust classes with built-in error processing capabilities. Once created, such classes allow clients of classes to concentrate on using the classes rather than defining what should happen if an error occurs while using the class. Exception handling offers the possibility that an error can be processed and that the program can continue execution. Other forms of error checking such as `assert`, exit the program immediately without any further processing.

23.24 List the advantages of exception handling over conventional means of error processing.

23.25 Use inheritance to create a base exception class and various derived exception classes. Then show that a catch handler specifying the base class can catch derived-class exceptions.

ANS:

```
1 // Exercise 23.25 Solution
2 #include <iostream>
3
4 using std::cout;
5
6 #include <cstdlib>
7 #include <ctime>
8
9 class BaseException {
10 public:
11     BaseException( char *mPtr ) : message( mPtr ) {}
12     void print() const { cout << message << '\n'; }
13 private:
14     char *message;
15 };
16
17 class DerivedException : public BaseException {
18 public:
19     DerivedException( char *mPtr ) : BaseException( mPtr ) {}
20 };
21
22 class DerivedException2 : public DerivedException {
23 public:
24     DerivedException2( char *mPtr ) : DerivedException( mPtr ) {}
25 };
26
27 int main()
28 {
29     srand( time( 0 ) );
30
31     try {
32         throw ( rand() % 2 ? DerivedException( "DerivedException" ) :
33                 DerivedException2( "DerivedException2" ) );
34     }
35     catch ( BaseException &b ) {
36         b.print();
37     }
38
39     return 0;
40 }
```

DerivedException

23.26 Write a program designed to generate and handle a memory exhaustion error. Your program should loop on a request to create dynamic storage through operator `new`.

ANS:

```
1 // Exercise 23.26 solution
2 #include <iostream>
3
4 using std::cout;
5 using std::cerr;
6
7 #include <new>
8 using std::bad_alloc;
9
10 #include <cstdlib>
11
12 int main()
13 {
14     long double *ptr[ 10 ];
15
16     try {
17         for ( int i = 0; i < 10; ++i ) {
18             ptr[ i ] = new long double[ 1000000 ];
19             cout << "Allocated 1000000 long doubles in ptr[ "
20                 << i << " ]\n";
21         }
22     }
23     catch ( bad_alloc ex ) {
24         cerr << "Memory Allocation Failed.\n";
25         exit( EXIT_FAILURE );
26     }
27
28     return 0;
29 }
```

```
Allocated 1000000 long doubles in ptr[ 0 ]
Allocated 1000000 long doubles in ptr[ 1 ]
Allocated 1000000 long doubles in ptr[ 2 ]
Allocated 1000000 long doubles in ptr[ 3 ]
Allocated 1000000 long doubles in ptr[ 4 ]
Allocated 1000000 long doubles in ptr[ 5 ]
Allocated 1000000 long doubles in ptr[ 6 ]
Allocated 1000000 long doubles in ptr[ 7 ]
Allocated 1000000 long doubles in ptr[ 8 ]
Memory Allocation Failed.
```


24

Introduction to Java Applications and Applets: Solutions

SOLUTIONS

24.8 Fill in the blanks in each of the following:

a) _____ are used to document a program and improve its readability.

ANS: Comments.

b) An input dialog capable of receiving input from the user is displayed with method _____ of class _____.

ANS: `showInputDialog`, `JOptionPane`.

24.9 Write Java statements that accomplish each of the following:

a) Display the message "Enter two numbers" using class `JOptionPane`.

ANS: `JOptionPane.showMessageDialog(null, "Enter two numbers");`

b) Assign the product of variables `b` and `c` to variable `a`.

ANS: `a = b * c;`

c) State that a program performs a sample payroll calculation (i.e., use text that helps to document a program).

ANS: `// This program performs a simple payroll calculation.`

24.10 What displays in the message dialog when each of the following Java statements is performed? Assume `x = 2` and `y = 3`.

a) `JOptionPane.showMessageDialog(null, "x = " + x);`

ANS: `x = 2`

b) `JOptionPane.showMessageDialog(null, "The value of x + x is " + (x + x));`

ANS: The value of `x + x` is 4

c) `JOptionPane.showMessageDialog(null, "x =");`

ANS: `x =`

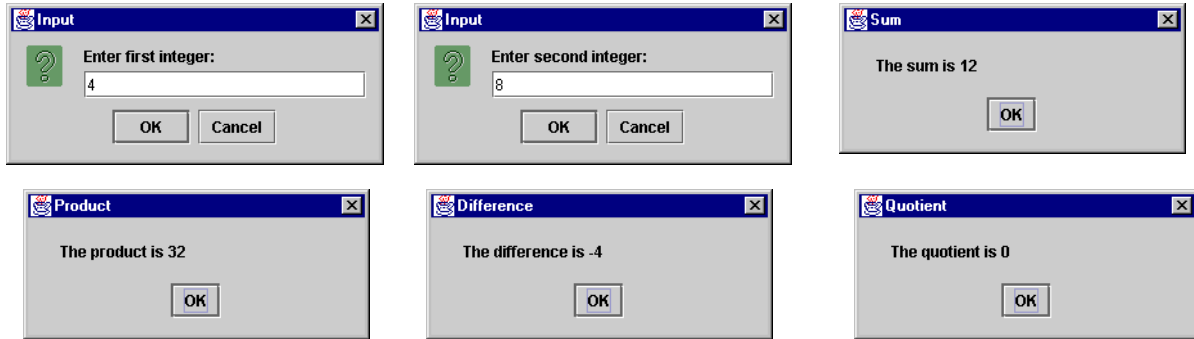
d) `JOptionPane.showMessageDialog(null, (x + y) + " = " + (y + x));`

ANS: `5 = 5`

24.11 Write an application that asks the user to enter two numbers, obtains the two numbers from the user and prints the sum, product, difference and quotient of the two numbers. Use the techniques shown in Fig. 24.6.

ANS:

```
1 // Exercise 24.11 Solution
2 // Calculate.java
3 // Program prints the sum, product, difference and quotient of
4 // the two numbers.
5
6 import javax.swing.JOptionPane;
7
8 public class Calculate {
9     public static void main( String args[] )
10    {
11        String firstNumber,    // first string entered by user
12            secondNumber;    // second string entered by user
13        int number1,          // first number
14            number2;          // second number
15        int sum;
16        int product;
17        int difference;
18        int quotient;
19
20        // read first number from user as a string
21        firstNumber =
22            JOptionPane.showInputDialog( "Enter first integer:" );
23
24        // read second number from user as a string
25        secondNumber =
26            JOptionPane.showInputDialog( "Enter second integer:" );
27
28        // convert numbers from type String to type int
29        number1 = Integer.parseInt( firstNumber );
30        number2 = Integer.parseInt( secondNumber );
31
32        // calculate
33        sum      = number1 + number2;
34        product  = number1 * number2;
35        difference = number1 - number2;
36        quotient = number1 / number2;
37
38        // Display results
39        JOptionPane.showMessageDialog(
40            null, "The sum is " + sum, "Sum",
41            JOptionPane.PLAIN_MESSAGE );
42        JOptionPane.showMessageDialog(
43            null, "The product is " + product, "Product",
44            JOptionPane.PLAIN_MESSAGE );
45        JOptionPane.showMessageDialog(
46            null, "The difference is " + difference, "Difference",
47            JOptionPane.PLAIN_MESSAGE );
48        JOptionPane.showMessageDialog(
49            null, "The quotient is " + quotient, "Quotient",
50            JOptionPane.PLAIN_MESSAGE );
51
52        System.exit( 0 );
53    }
54 }
```



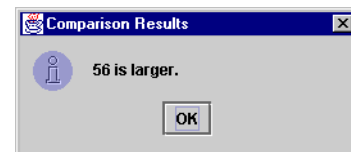
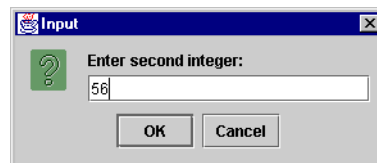
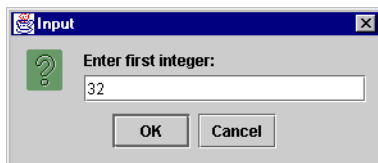
24.12 Write an application that asks the user to enter two integers, obtains the numbers from the user and displays the larger number followed by the words “is larger” in an information message dialog. If the numbers are equal, print the message “These numbers are equal.” Use the techniques shown in Fig. 24.6.

ANS:

```

1 // Exercise 24.12 Solution
2 // Larger.java
3 // Program determines the larger of two numbers
4 import javax.swing.JOptionPane;
5
6 public class Larger {
7     public static void main( String args[] )
8     {
9         String firstNumber,    // first string entered by user
10            secondNumber;      // second string entered by user
11         int number1,          // first number to compare
12            number2;           // second number to compare
13
14         // read first number from user as a string
15         firstNumber =
16             JOptionPane.showInputDialog( "Enter first integer:" );
17
18         // read second number from user as a string
19         secondNumber =
20             JOptionPane.showInputDialog( "Enter second integer:" );
21
22         // convert numbers from type String to type int
23         number1 = Integer.parseInt( firstNumber );
24         number2 = Integer.parseInt( secondNumber );
25
26         String result;        // a string containing the output
27         if ( number1 > number2 )
28             result = number1 + " is larger.";
29         else if ( number1 < number2 )
30             result = number2 + " is larger.";
31         else
32             result = "These numbers are equal.";
33
34         // Display results
35         JOptionPane.showMessageDialog(
36             null, result, "Comparison Results",
37             JOptionPane.INFORMATION_MESSAGE );
38
39         System.exit( 0 );
40     }
41 }

```

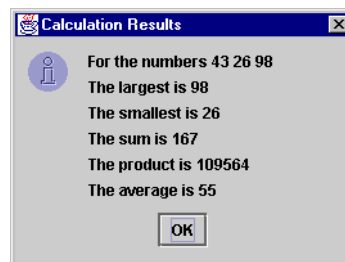
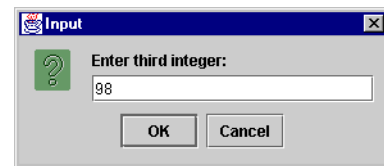
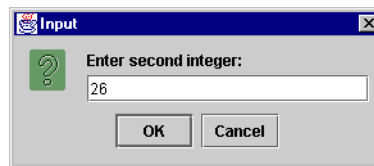
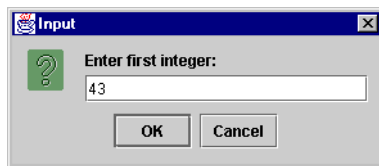


24.13 Write an application that inputs three integers from the user and displays the sum, average, product, smallest and largest of these numbers in an information message dialog. Use the GUI techniques shown in Fig. 24.6. Note: The average calculation in this exercise should result in an integer representation of the average. So, if the sum of the values is 7, the average will be 2 not 2.3333...

ANS:

```
1 // Exercise 24.13 Solution
2 // Calculate.java
3 // Program make simple calculations on three integers.
4 import javax.swing.JOptionPane;
5
6 public class Calculate {
7     public static void main( String args[] )
8     {
9         String firstNumber,    // first string entered by user
10            secondNumber,     // second string entered by user
11            thirdNumber;      // third string entered by user
12        String result;
13        int number1,          // first number
14            number2,          // second number
15            number3;          // third number
16
17        int sum;
18        int largest;
19        int smallest;
20        int product;
21        int average;
22
23        // read first number from user as a string
24        firstNumber =
25            JOptionPane.showInputDialog( "Enter first integer:" );
26
27        // read second number from user as a string
28        secondNumber =
29            JOptionPane.showInputDialog( "Enter second integer:" );
30
31        // read third number from user as a string
32        thirdNumber =
33            JOptionPane.showInputDialog( "Enter third integer:" );
34
35        // convert numbers from type String to type int
36        number1 = Integer.parseInt( firstNumber );
37        number2 = Integer.parseInt( secondNumber );
38        number3 = Integer.parseInt( thirdNumber );
39
40        if ( number1 > number2 ) {
41            largest = number1;
42            smallest = number2;
43        }
44        else {
45            largest = number2;
46            smallest = number1;
47        }
48
49        if ( number3 > largest )
50            largest = number3;
51
52        if ( number3 < smallest )
53            smallest = number3;
54
55        sum = number1 + number2 + number3;
56        product = number1 * number2 * number3;
```

```
57     average = sum / 3;
58
59     result = "For the numbers " + number1 + " "
60             + number2 + " "
61             + number3 + "\n" +
62             "The largest is " + largest + "\n" +
63             "The smallest is " + smallest + "\n" +
64             "The sum is " + sum + "\n" +
65             "The product is " + product + "\n" +
66             "The average is " + average + "\n";
67
68     // Display results
69     JOptionPane.showMessageDialog(
70         null, result, "Calculation Results",
71         JOptionPane.INFORMATION_MESSAGE );
72
73     System.exit( 0 );
74 }
75 }
```



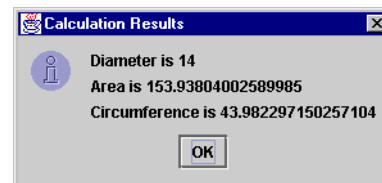
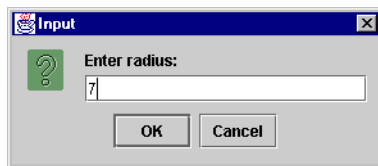
24.14 Write an application that inputs from the user the radius of a circle and prints the circle's diameter, circumference and area. Use the constant value 3.14159 for π . Use the GUI techniques shown in Fig. 24.6. [Note: You may also use the predefined constant `Math.PI` for the value of π . This constant is more precise than the value 3.14159. Class `Math` is defined in the `java.lang` package, so you do not need to import it.] Use the following formulas (r is the radius): $diameter = 2r$, $circumference = 2\pi r$, $area = \pi r^2$.

ANS:

```

1 // Exercise 24.14 Solution
2 // Circle.java
3 // Program calculate the area, circumference, and diameter for a circle
4 import javax.swing.JOptionPane;
5
6 public class Circle {
7     public static void main( String args[] )
8     {
9         String input,        // string entered by user
10        result;             // output display string
11        int radius;         // radius of circle
12
13        // read from user as a string
14        input =
15            JOptionPane.showInputDialog( "Enter radius:" );
16
17        // convert number from type String to type int
18        radius = Integer.parseInt( input );
19
20        result = "Diameter is " + ( 2 * radius ) +
21            "\nArea is " + ( Math.PI * radius * radius ) +
22            "\nCircumference is " + ( 2 * Math.PI * radius );
23
24        // Display results
25        JOptionPane.showMessageDialog(
26            null, result, "Calculation Results",
27            JOptionPane.INFORMATION_MESSAGE );
28
29        System.exit( 0 );
30    }
31 }

```



24.15 Write an application that displays in the command window a box, an oval, an arrow and a diamond using asterisks (*) as follows:

```

*****      ***      *      *
*      *      *      *      ****      * *
*      *      *      *      *****      * *
*      *      *      *      *      *      * *
*      *      *      *      *      *      * *
*      *      *      *      *      *      * *
*      *      *      *      *      *      * *
*****      ***      *      *

```

ANS:

```

1 // Exercise 24.15 Solution
2 // Shapes.java
3 // Program draws four shapes to the command window.
4
5 public class Shapes {
6     public static void main( String args[] )
7     {
8         System.out.println( "*****      ***      *      * " );
9         System.out.println( "*      *      *      *      ****      * * " );
10        System.out.println( "*      * *      *      *****      * * " );
11        System.out.println( "*      * *      *      *      *      * * " );
12        System.out.println( "*      * *      *      *      *      * * " );
13        System.out.println( "*      * *      *      *      *      * * " );
14        System.out.println( "*      * *      *      *      *      * * " );
15        System.out.println( "*      * *      *      *      *      * * " );
16        System.out.println( "*****      ***      *      * " );
17    }
18 }

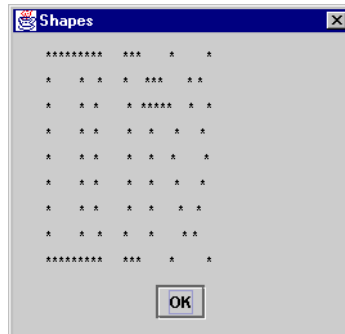
```

- 24.16** Modify the program you created in Exercise 24.15 to display the shapes in a `JOptionPane.PLAIN_MESSAGE` dialog.
ANS:

```

1 // Exercise 24.16 Solution
2 // Shapes.java
3 // Program draws four shapes in a Plain Message Dialog
4 import javax.swing.JOptionPane;
5
6 public class Shapes {
7     public static void main( String args[] )
8     {
9         String shapeString;
10        shapeString =
11            "*****      ***      *      *      " + "\n"
12            + "*      * *      *      ***      * *      " + "\n"
13            + "*      * *      *      *      *      *      " + "\n"
14            + "*      * *      *      *      *      *      " + "\n"
15            + "*      * *      *      *      *      *      " + "\n"
16            + "*      * *      *      *      *      *      " + "\n"
17            + "*      * *      *      *      *      *      " + "\n"
18            + "*      * *      *      *      *      *      " + "\n"
19            + "*****      ***      *      *      " + "\n" ;
20
21        JOptionPane.showMessageDialog(
22            null, shapeString, "Shapes",
23            JOptionPane.PLAIN_MESSAGE );
24
25        System.exit( 0 );
26    }
27 }

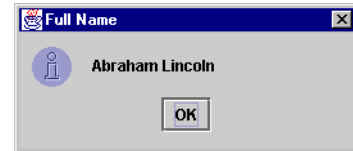
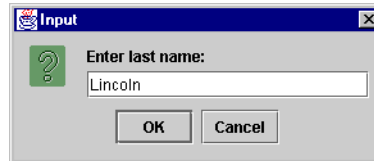
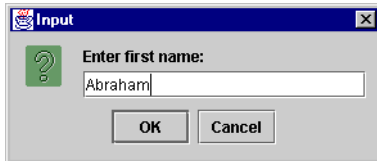
```



24.17 Write a program that reads a first name and a last name from the user as two separate inputs and concatenates the first name and last name separated by a space. Display in a message dialog the concatenated name.

ANS:

```
1 // Exercise 24.17 Solution
2 // Name.java
3 // Program puts a first name and a last name together
4 // as input by the user
5 import javax.swing.JOptionPane;
6
7 public class Name {
8     public static void main( String args[] )
9     {
10        String firstName,    // first string entered by user
11           lastName;        // last string entered by user
12
13        // read first name from user
14        firstName = JOptionPane.showInputDialog( "Enter first name: " );
15
16        // read last name from user
17        lastName = JOptionPane.showInputDialog( "Enter last name: " );
18
19        // Display results
20        JOptionPane.showMessageDialog(
21            null, firstName + " " + lastName, "Full Name",
22            JOptionPane.INFORMATION_MESSAGE );
23
24        System.exit( 0 );
25    }
26 }
```



25

Beyond C & C++: Operators, Methods & Arrays in Java: Solutions

SOLUTIONS

25.5 Answer each of the following questions:

a) What does it mean to choose numbers “at random?”

ANS: Every number has an equal chance of being chosen at any time.

b) Why is the `Math.random` method useful for simulating games of chance?

ANS: Because it produces a series of random numbers.

c) Why is it often necessary to scale and/or shift the values produced by `Math.random`?

ANS: To produce random numbers in a specific range.

d) Why is computerized simulation of real-world situations a useful technique?

ANS: It enables more accurate predictions of random events such as cars arriving at toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at a specified time.

25.6 Write statements that assign random integers to the variable *n* in the following ranges:

a) $1 \leq n \leq 2$

ANS: `n = (int) (1 + Math.random() * 2);`

b) $1 \leq n \leq 100$

ANS: `n = (int) (1 + Math.random() * 100);`

c) $0 \leq n \leq 9$

ANS: `n = (int) (Math.random() * 10);`

d) $1000 \leq n \leq 1112$

ANS: `n = (int) (1000 + Math.random() * 113);`

e) $-1 \leq n \leq 1$

ANS: `n = (int) (-1 + Math.random() * 3);`

f) $-3 \leq n \leq 1$

ANS: `n = (int) (-3 + Math.random() * 15);`

25.7 For each of the following sets of integers, write a single statement that will print a number at random from the set.

a) 2, 4, 6, 8, 10.

ANS: `System.out.print((int) ((1 + Math.random() * 5) * 2));`

b) 3, 5, 7, 9, 11.

ANS: `System.out.print((int) ((1 + Math.random() * 5) * 2 + 1));`

c) 6, 10, 14, 18, 22.

ANS: `System.out.print((int) ((Math.random() * 5) * 4 + 6));`

25.8 Define a method `hypotenuse` that calculates the length of the hypotenuse of a right triangle when the other two sides are given. The method should take two arguments of type `double` and return the hypotenuse as a `double`. Incorporate this method into an applet that reads integer values for `side1` and `side2` from `JTextField`s and performs the calculation with the `hypotenuse` method. Determine the length of the hypotenuse for each of the following triangles. [Note: Register for event handling on only the second `JTextField`. The user should interact with the program by typing numbers in both `JTextField`s and pressing *Enter* in the second `JTextField`.]

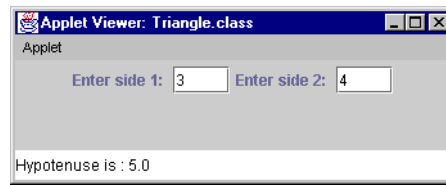
Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

ANS:

```

1 // Exercise 25.8 Solution
2 // Triangle.java
3 // Program calculates the hypotenuse of
4 // a right triangle.
5 import javax.swing.*;
6 import java.awt.event.*;
7 import java.awt.*;
8
9 public class Triangle extends JApplet
10 implements ActionListener {
11     JTextField sideInput, side2Input;
12     JLabel sidePrompt, sidePrompt2;
13
14     public void init()
15     {
16         sideInput = new JTextField( 4 );
17         side2Input = new JTextField( 4 );
18         side2Input.addActionListener( this );
19         sidePrompt = new JLabel( "Enter side 1: " );
20         sidePrompt2 = new JLabel( "Enter side 2: " );
21         Container c = getContentPane();
22         c.setLayout( new FlowLayout() );
23         c.add( sidePrompt );
24         c.add( sideInput );
25         c.add( sidePrompt2 );
26         c.add( side2Input );
27     }
28
29     public void actionPerformed( ActionEvent e )
30     {
31         double side1, side2;
32
33         side1 = Double.parseDouble( side2Input.getText() );
34         side2 = Double.parseDouble( sideInput.getText() );
35
36         double h = hypotenuse( side1, side2 );
37         showStatus( "Hypotenuse is : " + h );
38     }
39
40     public double hypotenuse( double s1, double s2 )
41     {
42         double hypotSquared = Math.pow( s1, 2 ) + Math.pow( s2, 2 );
43
44         return Math.sqrt( hypotSquared );
45     }
46 }

```



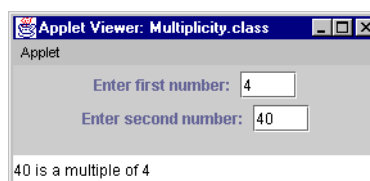
25.9 Write a method `multiple` that determines for a pair of integers whether the second integer is a multiple of the first. The method should take two integer arguments and return `true` if the second is a multiple of the first and `false` otherwise. Incorporate this method into an applet that inputs a series of pairs of integers (one pair at a time using `JTextField`s). [*Note:* Register for event handling on only the second `JTextField`. The user should interact with the program by typing numbers in both `JTextField`s and pressing *Enter* in the second `JTextField`.]

ANS:

```

1 // Exercise 25.9 Solution
2 // Multiplicity.java
3 // Determines if the second number entered
4 // is a multiple of the first.
5 import javax.swing.*;
6 import java.awt.event.*;
7 import java.awt.*;
8
9 public class Multiplicity extends JApplet implements ActionListener {
10     JTextField input, input2;
11     JLabel prompt, prompt2;
12
13     public void init()
14     {
15         input = new JTextField( 4 );
16         input2 = new JTextField( 4 );
17         input2.addActionListener( this );
18         prompt = new JLabel( "Enter first number: " );
19         prompt2 = new JLabel( "Enter second number: " );
20         Container c = getContentPane();
21         c.setLayout( new FlowLayout() );
22         c.add( prompt );
23         c.add( input );
24         c.add( prompt2 );
25         c.add( input2 );
26     }
27
28     public void actionPerformed((ActionEvent e) )
29     {
30         int first, second;
31
32         first = Integer.parseInt( input.getText() );
33         second = Integer.parseInt( input2.getText() );
34
35         if ( multiple( first, second ) == true )
36             showStatus( second + " is a multiple of " +
37                         first );
38         else
39             showStatus( second + " is not a multiple of " +
40                         first );
41     }
42
43     public boolean multiple( int one, int two )
44     {
45         if ( ( two % one == 0 ) && two != 0 )
46             return true;
47
48         return false;
49     }
50 }

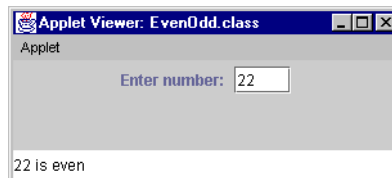
```



25.10 Write an applet that inputs integers (one at a time) and passes them one at a time to method `isEven`, which uses the modulus operator to determine if an integer is even. The method should take an integer argument and return `true` if the integer is even and `false` otherwise. Use an input dialog to obtain the data from the user.

ANS:

```
1 // Exercise 25.10 Solution
2 // EvenOdd.java
3 // Determines if a number is odd or even
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class EvenOdd extends JApplet implements ActionListener {
9     JTextField input;
10    JLabel prompt;
11
12    public void init()
13    {
14        input = new JTextField( 4 );
15        input.addActionListener( this );
16        prompt = new JLabel( "Enter number: " );
17        Container c = getContentPane();
18        c.setLayout( new FlowLayout() );
19        c.add( prompt );
20        c.add( input );
21    }
22
23    public void actionPerformed((ActionEvent e) )
24    {
25        int number = Integer.parseInt( input.getText() );
26        String result = "";
27
28        if ( isEven( number ) == true )
29            result = number + " is even";
30        else
31            result = number + " is odd ";
32
33        showStatus( result );
34    }
35
36    public boolean isEven( int num )
37    {
38        if ( num % 2 == 0 )
39            return true;
40
41        return false;
42    }
43 }
```



25.11 Write a method `squareOfAsterisks` that displays a solid square of asterisks whose side is specified in integer parameter `side`. For example, if `side` is 4, the method displays

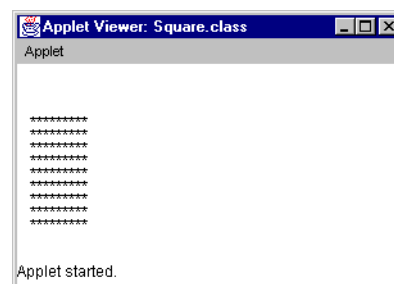
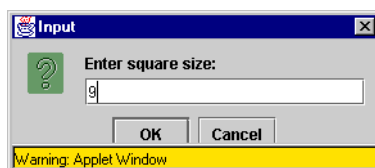
```
****
****
****
****
```

Incorporate this method into an applet that reads an integer value for `side` from the user at the keyboard and performs the drawing with the `squareOfAsterisks` method. Note that this method should be called from the applet's `paint` method and should be passed the `Graphics` object from `paint`.

ANS:

```

1 // Exercise 25.11 Solution
2 // Square.java
3 // Program draws a square of asterisks
4 import javax.swing.*;
5 import java.awt.*;
6
7 public class Square extends JApplet {
8     int size;
9
10    public void init()
11    {
12        String input = JOptionPane.showInputDialog(
13            "Enter square size:" );
14
15        size = Integer.parseInt( input );
16    }
17
18    public void squareOfAsterisks( Graphics g )
19    {
20        int y = 50, x = 5;
21
22        for ( int a = 1; a <= size * size; a++ ) {
23            g.drawString( "*", x += 5, y );
24
25            if ( a % size == 0 ) {
26                y += 10;
27                x = 5;
28            }
29        }
30    }
31
32    public void paint( Graphics g )
33    {
34        squareOfAsterisks( g );
35    }
36 }
```



25.12 Implement the following integer methods:

- a) Method `celsius` returns the Celsius equivalent of a Fahrenheit temperature using the calculation

$$C = 5.0 / 9.0 * (F - 32);$$

- b) Method `fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature.

$$F = 9.0 / 5.0 * C + 32;$$

- c) Use these methods to write an applet that enables the user to enter either a Fahrenheit temperature and display the Celsius equivalent or enter a Celsius temperature and display the Fahrenheit equivalent.

[*Note:* This applet will require that two `TextField` objects that have registered action events. When `actionPerformed` is invoked, the `ActionEvent` parameter has method `getSource()` to determine the GUI component with which the user interacted. Your `actionPerformed` method should contain an `if/else` structure of the following form:

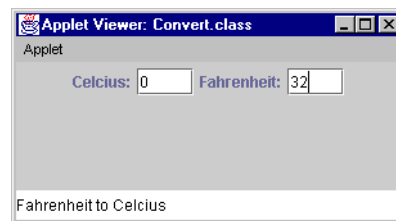
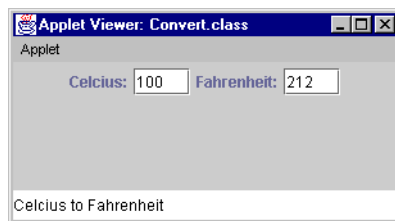
```
if ( e.getSource() == input1 ) {
    // process input1 interaction here
}
else { // e.getSource() == input2
    // process input2 interaction here
}
```

where `input1` and `input2` are `TextField` references.]

ANS:

```
1 // Exercise 25.12 Solution
2 // Convert.java
3 // Program converts Fahrenheit to Celcius
4 // and vice versa.
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class Convert extends JApplet implements ActionListener {
10     JTextField cInput, fInput;
11     JLabel cLabel, fLabel;
12
13     public void init()
14     {
15         cInput = new JTextField( 4 );
16         fInput = new JTextField( 4 );
17         cInput.addActionListener( this );
18         fInput.addActionListener( this );
19         cLabel = new JLabel( "Celcius:" );
20         fLabel = new JLabel( "Fahrenheit:" );
21         Container c = getContentPane();
22         c.setLayout( new FlowLayout() );
23         c.add( cLabel );
24         c.add( cInput );
25         c.add( fLabel );
26         c.add( fInput );
27     }
28
29     public void actionPerformed( ActionEvent e )
30     {
31         if ( e.getSource() == cInput ) {
32             int c = Integer.parseInt( cInput.getText() );
33
34             fInput.setText( String.valueOf( celsius( c ) ) );
35             showStatus( "Celcius to Fahrenheit" );
36         }
37         else if ( e.getSource() == fInput ) {
38             int f = Integer.parseInt( fInput.getText() );
39
40             cInput.setText( String.valueOf( fahrenheit( f ) ) );
```

```
41     showStatus( "Fahrenheit to Celcius" );
42     }
43 }
44
45 public int celcius( int cTemp )
46 {
47     return ( ( int ) ( 9.0 / 5.0 * cTemp + 32 ) );
48 }
49
50 public int fahrenheit( int fTemp )
51 {
52     return ( ( int ) ( 5.0 / 9.0 * ( fTemp - 32 ) ) );
53 }
54 }
```

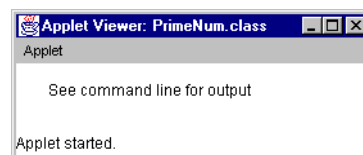


25.13 An integer is said to be *prime* if it is divisible only by 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

- a) Write a method that determines if a number is prime.
- b) Use this method in an applet that determines and prints all the prime numbers between 1 and 10,000. How many of these 10,000 numbers do you really have to test before being sure that you have found all the primes? Display the results in a JTextArea that has scrolling functionality.
- c) Initially you might think that $n/2$ is the upper limit for which you must test to see if a number is prime, but you need only go as high as the square root of n . Why? Rewrite the program and run it both ways. Estimate the performance improvement.

ANS:

```
1 // Exercise 25.13 n/2 limit
2 // PrimeNum.java
3 // Program calculates prime numbers
4 import javax.swing.*;
5 import java.awt.*;
6
7 public class PrimeNum extends JApplet {
8
9     public void start()
10    {
11        int number, count = 0;
12
13        System.out.println( "Prime numbers between 1 and 1000 are: " );
14
15        for ( int m = 1; m <= 10000; m++ )
16            if ( prime( m ) == true ) {
17                ++count;
18                System.out.println( m );
19            }
20    }
21
22    public void paint( Graphics g )
23    {
24        g.drawString( "See command line for output", 25, 25 );
25    }
26
27    public boolean prime( int n )
28    {
29        for ( int v = 2; v <= n / 2; v++ )
30            if ( n % v == 0 )
31                return false;
32
33        return true;
34    }
35 }
```




```

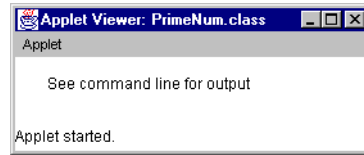
Prime numbers between 1 and 1000 are:
1
2
3
5
7
11
...
9931
9941
9949
9967
9973

```

```

1 // Exercise 25.13 sqrt(n) limit
2 // PrimeNum2.java
3 // Program calculates prime numbers
4 import javax.swing.*;
5 import java.awt.*;
6
7 public class PrimeNum2 extends JApplet {
8
9     public void start()
10    {
11        int number, count = 0;
12
13        System.out.println( "Prime numbers between 1 and 1000 are: " );
14
15        for ( int m = 1; m <= 10000; m++ )
16            if ( prime( m ) == true ) {
17                ++count;
18                System.out.println( m );
19            }
20    }
21
22    public void paint( Graphics g )
23    {
24        g.drawString( "See command line for output", 25, 25 );
25    }
26
27    public boolean prime( int n )
28    {
29        for ( int v = 2; v <= ( int ) Math.sqrt( n ); v++ )
30            if ( n % v == 0 )
31                return false;
32
33        return true;
34    }
35 }

```



Prime numbers between 1 and 1000 are:

1
2
3
5
7
11

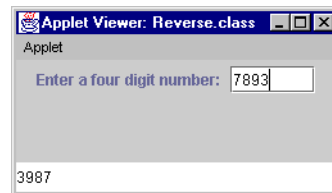
...
9931
9941
9949
9967
9973

25.14 Write a method that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the method should return 1367. Incorporate the method into an applet that reads a value from the user. Display the result of the method in the status bar.

ANS:

```
1 // Exercise 25.14 Solution
2 // Reverse.java
3 // Program takes a four digit number
4 // and prints out its digits reversed
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class Reverse extends JApplet implements ActionListener {
10     JTextField input;
11     JLabel prompt;
12     int number;
13
14     public void init()
15     {
16         input = new JTextField( 6 );
17         input.addActionListener( this );
18         prompt = new JLabel( "Enter a four digit number: " );
19         Container c = getContentPane();
20         c.setLayout( new FlowLayout() );
21         c.add( prompt );
22         c.add( input );
23     }
24
25     public void actionPerformed((ActionEvent e) )
26     {
27         number = Integer.parseInt( input.getText() );
28         reverseDigits();
29     }
30
31     public void reverseDigits()
32     {
33         int digit1 = 0, digit2 = 0, digit3 = 0,
34             digit4 = 0, factor = 1000, value = 0;
35
36         while ( factor >= 1 ) {
37             int temp = number / factor;
38
39             switch ( factor ) {
40                 case 1000:
41                     digit4 = temp;
42                     break;
43                 case 100:
44                     digit3 = temp * 10;
45                     break;
46                 case 10:
47                     digit2 = temp * 100;
48                     break;
49                 case 1:
50                     digit1 = temp * 1000;
51                     break;
52             }
53
54             number %= factor;
55             factor /= 10;
56         }
57
58         if ( digit1 == 0 ) // special case when last digit initially is 0
59             showStatus( String.valueOf( 0 ) + String.valueOf( digit2 / 100 )
60                 + String.valueOf( digit3 / 10 ) +
61                 String.valueOf( digit4 ) );
```

```
62     else
63         showStatus( String.valueOf(digit1 + digit2 + digit3 + digit4) );
64     }
65 }
```



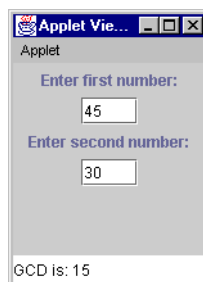
25.15 The *greatest common divisor (GCD)* of two integers is the largest integer that evenly divides each of the two numbers. Write a method `gcd` that returns the greatest common divisor of two integers. Incorporate the method into an applet that reads two values from the user. Display the result of the method in the status bar.

ANS:

```

1 // Exercise 25.15 Solution
2 // Divisor.java
3 // Program finds the greatest
4 // common divisor of two numbers.
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class Divisor extends JApplet implements ActionListener {
10     JTextField input1, input2;
11     JLabel label1, label2;
12
13     public void init()
14     {
15         input1 = new JTextField( 4 );
16         input2 = new JTextField( 4 );
17         input2.addActionListener( this );
18         label1 = new JLabel( "Enter first number:" );
19         label2 = new JLabel( "Enter second number:" );
20         Container c = getContentPane();
21         c.setLayout( new FlowLayout() );
22         c.add( label1 );
23         c.add( input1 );
24         c.add( label2 );
25         c.add( input2 );
26     }
27
28     public void actionPerformed((ActionEvent e)
29     {
30         int num1, num2;
31
32         num1 = Integer.parseInt( input1.getText() );
33         num2 = Integer.parseInt( input2.getText() );
34
35         showStatus( "GCD is: " + gcd( num1, num2 ) );
36     }
37
38     public int gcd( int x, int y )
39     {
40         int greatest = 1;
41
42         for ( int z = 2; z <= ( ( x < y ) ? x : y ); z++ )
43             if ( ( x % z == 0 ) && ( y % z == 0 ) )
44                 greatest = z;
45
46         return greatest;
47     }
48 }

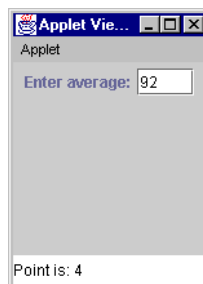
```



25.16 Write a method `qualityPoints` that inputs a student's average and returns 4 if a student's average is 90–100, 3 if the average is 80–89, 2 if the average is 70–79, 1 if the average is 60–69 and 0 if the average is lower than 60. Incorporate the method into an applet that reads a value from the user. Display the result of the method in the status bar.

ANS:

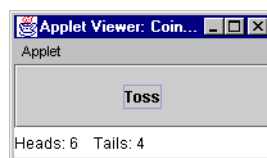
```
1 // Exercise 25.16 Solution
2 // Average.java
3 // Program displays a number
4 // representing the student's average
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class Average extends JApplet
10 implements ActionListener {
11     JTextField input;
12     JLabel prompt;
13
14     public void init()
15     {
16         input = new JTextField( 4 );
17         input.addActionListener( this );
18         prompt = new JLabel( "Enter average:" );
19         Container c = getContentPane();
20         c.setLayout( new FlowLayout() );
21         c.add( prompt );
22         c.add( input );
23     }
24
25     public void actionPerformed((ActionEvent e) )
26     {
27         int number = Integer.parseInt( input.getText() );
28
29         if ( number >= 0 && number <= 100 )
30             showStatus( "Point is: " + qualityPoints( number ) );
31         else
32             showStatus( "Invalid input." );
33     }
34
35     public int qualityPoints( int grade )
36     {
37         if ( grade >= 90 )
38             return 4;
39         else if ( grade >= 80 )
40             return 3;
41         else if ( grade >= 70 )
42             return 2;
43         else if ( grade >= 60 )
44             return 1;
45         else
46             return 0;
47     }
48 }
```



25.17 Write an applet that simulates coin tossing. Let the program toss the coin each time the user presses the “Toss” button. Count the number of times each side of the coin appears. Display the results. The program should call a separate method `flip` that takes no arguments and returns `false` for tails and `true` for heads. [*Note*: If the program realistically simulates the coin tossing, each side of the coin should appear approximately half the time.]

ANS:

```
1 // Exercise 25.17 Solution
2 // Coin.java
3 // Program simulates tossing a coin.
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class Coin extends JApplet
8     implements ActionListener {
9     int heads, tails;
10    JButton b;
11
12    public void init()
13    {
14        b = new JButton( "Toss" );
15        b.addActionListener( this );
16        getContentPane().add( b );
17    }
18
19    public void actionPerformed((ActionEvent e )
20    {
21        if ( flip() == true )
22            ++heads;
23        else
24            ++tails;
25
26        showStatus( "Heads: " + heads + "    Tails: " + tails );
27    }
28
29    public boolean flip()
30    {
31        if ( ( int ) ( Math.random() * 2 ) == 1 )
32            return true;
33        else
34            return false;
35    }
36 }
```



25.18 Computers are playing an increasing role in education. Write a program that will help an elementary school student learn multiplication. Use `Math.random` to produce two positive one-digit integers. It should then display a question in the status bar such as

How much is 6 times 7?

The student then types the answer into a `JTextField`. Your program checks the student's answer. If it is correct, draw the string "Very good!" on the applet, then ask another multiplication question. If the answer is wrong, draw the string "No. Please try again." on the applet, then let the student try the same question again repeatedly until the student finally gets it right. A separate method should be used to generate each new question. This method should be called once when the applet begins execution and each time the user answers the question correctly. All drawing on the applet should be performed by the `paint` method.

ANS:

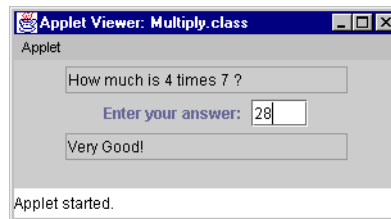
```

1 // Exercise 25.18 Solution
2 // Multiply.java
3 // Program generates single digit multiplication
4 // problems.
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 // Note: Applets have a problem rendering Paint graphics when
10 // Swing components are attached to the UI. Consequently, this
11 // solution does not use paint
12
13 public class Multiply extends JApplet implements ActionListener {
14     JTextField question;
15     JTextField input;
16     JTextField response;
17     JLabel prompt;
18     int answer;
19     String questionString;
20
21     public void init()
22     {
23         input = new JTextField( 4 );
24         input.addActionListener( this );
25         prompt = new JLabel( "Enter your answer: " );
26
27         response = new JTextField( 20 );
28         response.setEditable( false );
29
30         question = new JTextField( 20 );
31         question.setEditable( false );
32
33         Container c = getContentPane();
34         c.setLayout( new FlowLayout() );
35         c.add( question );
36         c.add( prompt );
37         c.add( input );
38         c.add( response );
39
40         createQuestion();
41     }
42
43     public void start()
44     {
45         question.setText( questionString );
46     }
47
48     public void actionPerformed( ActionEvent e )
49     {
50         int guess = Integer.parseInt( input.getText() );
51
52         input.setText( "" );
53
54         if ( guess != answer )
55             response.setText( "No. Please try again." );

```



```
56     else {
57         response.setText( "Very Good!" );
58         createQuestion();
59     }
60
61     question.setText(questionString);
62 }
63
64 // Create a new question, and
65 // the corresponding answer
66 public void createQuestion()
67 {
68     int digit1 = getNumber();
69     int digit2 = getNumber();
70
71     answer = digit1 * digit2;
72     questionString = "How much is " + digit1 + " times " + digit2 + " ?";
73 }
74
75 public int getNumber()
76 {
77     return ( ( int ) ( Math.random() * 10 ) );
78 }
79 }
```



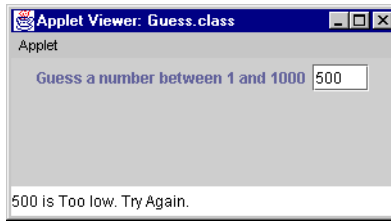
25.19 Write an applet that plays the “guess the number” game as follows: Your program chooses the number to be guessed by selecting a random integer in the range 1 to 1000. The applet displays the prompt `Guess a number between 1 and 1000` next to a `JTextField`. The player types a first guess into the `JTextField` and presses the *Enter* key. If the player's guess is incorrect, your program should display `Too high. Try again.` or `Too low. Try again.` in the status bar to help the player “zero in” on the correct answer and should clear the `JTextField` so the user can enter the next guess. When the user enters the correct answer, display `Congratulations. You guessed the number!` in the status bar and clear the `JTextField` so the user can play again. [Note: The guessing technique employed in this problem is similar to a *binary search*.]

ANS:

```

1 // Exercise 25.19 Solution
2 // Guess.java
3 // Program plays guess the number.
4 // problems.
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class Guess extends JApplet implements ActionListener {
10     JTextField input;
11     JLabel prompt;
12     int answer;
13
14     public void init()
15     {
16         input = new JTextField( 4 );
17         input.addActionListener( this );
18         prompt = new JLabel( "Guess a number between 1 and 1000" );
19         Container c = getContentPane();
20         c.setLayout( new FlowLayout() );
21         c.add( prompt );
22         c.add( input );
23         answer = getNumber();
24     }
25
26     public void actionPerformed((ActionEvent e)
27     {
28         int userGuess = Integer.parseInt( input.getText() );
29
30         checkUserGuess( userGuess );
31         input.setText( "" );
32     }
33
34     public int getNumber()
35     {
36         return ( ( int ) ( 1 + Math.random() * 1000 ) );
37     }
38
39     public void checkUserGuess( int userGuess )
40     {
41         if ( userGuess < answer )
42             showStatus( userGuess + " is Too low. Try Again." );
43         else if ( userGuess > answer )
44             showStatus( userGuess + " is Too High. Try Again." );
45         else {
46             showStatus( "Congratulations. You guessed the number!" );
47             input.setText( "" );
48
49             // new search
50             answer = getNumber();
51         }
52     }
53 }

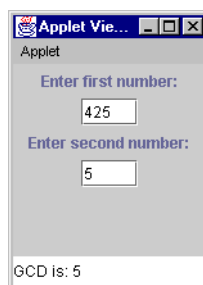
```



25.20 The greatest common divisor of integers x and y is the largest integer that evenly divides both x and y . Write a recursive method `gcd` that returns the greatest common divisor of x and y . The `gcd` of x and y is defined recursively as follows: If y is equal to 0, then `gcd(x, y)` is x ; otherwise, `gcd(x, y)` is `gcd(y, x % y)`, where `%` is the modulus operator. Use this method to replace the one you wrote in the applet of Exercise 25.15.

ANS:

```
1 // Exercise 25.20 Solution
2 // Divisor.java
3 // Program recursively finds the greatest
4 // common divisor of two numbers.
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class Divisor extends JApplet implements ActionListener {
10     JTextField input1, input2;
11     JLabel label1, label2;
12
13     public void init()
14     {
15         input1 = new JTextField( 4 );
16         input2 = new JTextField( 4 );
17         input2.addActionListener( this );
18         label1 = new JLabel( "Enter first number:" );
19         label2 = new JLabel( "Enter second number:" );
20         Container c = getContentPane();
21         c.setLayout( new FlowLayout() );
22         c.add( label1 );
23         c.add( input1 );
24         c.add( label2 );
25         c.add( input2 );
26     }
27
28     public void actionPerformed((ActionEvent e) )
29     {
30         int num1, num2;
31
32         num1 = Integer.parseInt( input1.getText() );
33         num2 = Integer.parseInt( input2.getText() );
34         showStatus( "GCD is: " + gcd( num1, num2 ) );
35     }
36
37     public int gcd( int x, int y )
38     {
39         if ( y == 0 )
40             return x;
41         else
42             return gcd( y, x % y );
43     }
44 }
```



25.21 Modify the craps program of Fig. 25.13 to allow wagering. Initialize variable `bankBalance` to 1000 dollars. Prompt the player to enter a wager. Check that `wager` is less than or equal to `bankBalance`, and if not, have the user reenter `wager` until a valid `wager` is entered. After a correct `wager` is entered, run one game of craps. If the player wins, increase `bankBalance` by `wager` and print the new `bankBalance`. If the player loses, decrease `bankBalance` by `wager`, print the new `bankBalance`, check if `bankBalance` has become zero, and if so, print the message "Sorry. You busted!" As the game progresses, print various messages to create some "chatter," such as "Oh, you're going for broke, huh?" or "Aw c'mon, take a chance!" or "You're up big. Now's the time to cash in your chips!". Implement the "chatter" as a separate method that randomly chooses the string to display.

ANS:

```

1 // Exercise 25.21 Solution
2 // Craps.java
3 // Program plays Craps
4 import java.awt.*;
5 import javax.swing.*;
6 import java.awt.event.*;
7
8 public class Craps extends JApplet implements ActionListener {
9     // constant variables for status of game
10    final int WON = 0, LOST = 1, CONTINUE = 2;
11
12    // other variables used in program
13    boolean firstRoll = true; // true if first roll
14    int dieSum; // sum of the dice
15    int myPoint; // point if no win/loss on first roll
16    int gameStatus = CONTINUE; // WON, LOST, CONTINUE
17    int bankBalance, wager;
18
19    // graphical user interface components
20    JLabel die1Label, die2Label, sumLabel, pointLabel, betLabel;
21    JTextField firstDie, secondDie, sum, point, better, chatter;
22    JButton roll;
23
24    // setup graphical user interface components
25    public void init()
26    {
27        Container c = getContentPane();
28        c.setLayout( new FlowLayout() );
29
30        bankBalance = 1000;
31        betLabel = new JLabel( "bet:" );
32        better = new JTextField( "100", 10 );
33        die1Label = new JLabel( "Die 1" );
34        firstDie = new JTextField( 10 );
35        firstDie.setEditable( false );
36        die2Label = new JLabel( "Die 2" );
37        secondDie = new JTextField( 10 );
38        secondDie.setEditable( false );
39        sumLabel = new JLabel( "Sum is" );
40        sum = new JTextField( 10 );
41        sum.setEditable( false );
42        roll = new JButton( "Roll Dice" );
43        roll.addActionListener( this );
44        pointLabel = new JLabel( "Point is" );
45        point = new JTextField( 10 );
46        point.setEditable( false );
47
48        chatter = new JTextField( 25 );
49        chatter.setEditable( false );
50        c.add( die1Label );
51        c.add( firstDie );
52        c.add( die2Label );
53        c.add( secondDie );
54        c.add( sumLabel );
55        c.add( sum );
56        c.add( pointLabel );

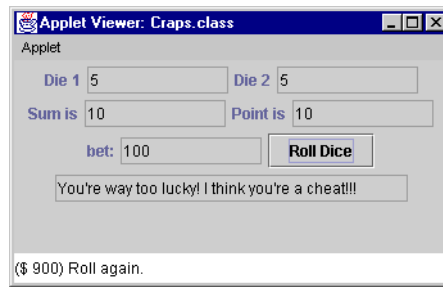
```

```

57     c.add( point );
58     c.add( betLabel );
59     c.add( better );
60     c.add( roll );
61     c.add( chatter );
62 }
63
64 // process one roll of the dice
65 public void play()
66 {
67     if ( firstRoll ) {                // first roll of the dice
68         dieSum = rollDice();
69
70         switch ( dieSum ) {
71             case 7: case 11:          // win on first roll
72                 gameStatus = WON;
73                 point.setText( "" ); // clear point text field
74                 firstRoll = true;    // allow new game to start
75                 break;
76             case 2: case 3: case 12: // lose on first roll
77                 gameStatus = LOST;
78                 point.setText( "" ); // clear point text field
79                 firstRoll = true;    // allow new game to start
80                 break;
81             default:                  // remember point
82                 gameStatus = CONTINUE;
83                 myPoint = dieSum;
84                 point.setText( Integer.toString( myPoint ) );
85                 firstRoll = false;
86                 break;
87         }
88     }
89     else {
90         dieSum = rollDice();
91
92         if ( dieSum == myPoint )      // win by making point
93             gameStatus = WON;
94         else if ( dieSum == 7 )      // lose by rolling 7
95             gameStatus = LOST;
96     }
97
98     if ( gameStatus == CONTINUE )
99         showStatus( "$ " + bankBalance + " Roll again." );
100    else {
101
102        if ( gameStatus == WON ) {
103            bankBalance += wager;
104            showStatus( "$ " + bankBalance + " Player wins. " +
105                "Click Roll Dice to play again." );
106        }
107        else {
108            bankBalance -= wager;
109            checkBalance();
110            showStatus( "$ " + bankBalance + " Player loses. " +
111                "Click Roll Dice to play again." );
112        }
113
114        better.setEditable( true );
115        firstRoll = true;
116    }
117 }
118
119 void checkBalance()
120 {
121     if ( bankBalance == 0 ) {
122         System.out.println( "Sorry. You busted!" );
123         System.exit( 0 );
124     }
125 }

```

```
126 // call method play when button is clicked
127 public void actionPerformed( ActionEvent e )
128 {
129     int w = Integer.parseInt( better.getText() );
130
131     if ( w > bankBalance || w < 0 )
132         showStatus( "( $" + bankBalance + " ) " +
133                     "Enter a valid wager!" );
134     else {
135         wager = w;
136         better.setEditable( false );
137         play();
138     }
139
140     chatter.setText( chatter() );
141 }
142
143 // roll the dice
144 int rollDice()
145 {
146     int die1, die2, workSum;
147
148     die1 = 1 + ( int ) ( Math.random() * 6 );
149     die2 = 1 + ( int ) ( Math.random() * 6 );
150     workSum = die1 + die2;
151
152     firstDie.setText( Integer.toString( die1 ) );
153     secondDie.setText( Integer.toString( die2 ) );
154     sum.setText( Integer.toString( workSum ) );
155
156     return workSum;
157 }
158
159 public String chatter()
160 {
161     String s = null;
162
163     switch ( ( int ) ( Math.random() * 5 ) ) {
164         case 0:
165             s = "Oh, you're going for broke huh?";
166             break;
167         case 1:
168             s = "Aw cmon, take a chance!";
169             break;
170         case 2:
171             s = "You're up big. Now's the " +
172                 "time to cash in your chips!";
173             break;
174         case 3:
175             s = "You're way too lucky! I think you're " +
176                 "a cheat!!!";
177             break;
178         case 4:
179             s = "I'm betting all my money on you.";
180             break;
181     }
182
183     return s;
184 }
185 }
186 }
```



25.22 Write a program to simulate the rolling of two dice. The program should use `Math.random` to roll the first die and should use `Math.random` again to roll the second die. The sum of the two values should then be calculated. [Note: Since each die can show an integer value from 1 to 6, the sum of the values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 being the least frequent sums. Figure 25.24 shows the 36 possible combinations of the two dice. Your program should roll the dice 36,000 times. Use a single-subscripted array to tally the numbers of times each possible sum appears. Print the results in a tabular format. Also, determine if the totals are reasonable (i.e., there are six ways to roll a 7, so approximately one sixth of all the rolls should be 7).

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Fig. 25.24 The 36 possible outcomes of rolling two dice.

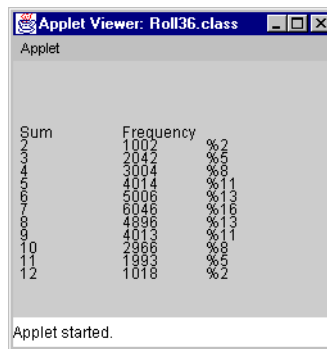
ANS:

```

1 // Exercise 25.22 Solution
2 // Roll36.java
3 // Program simulates rolling two
4 // six-sided die 36,000 times
5 // NOTE: this program could take a
6 // few seconds before displaying the data
7 import javax.swing.*;
8 import java.awt.*;
9
10 public class Roll36 extends JApplet {
11     int total[];
12
13     public void init()
14     {
15         total = new int[ 13 ];
16
17         for ( int i = 0; i < total.length; i++ )
18             total[ i ] = 0;
19
20         roll2Dice();
21     }
22
23     public void roll2Dice()
24     {
25         int face1, face2;
26
27         for ( int x = 1; x <= 36000; x++ ) {
28             face1 = ( int ) ( 1 + Math.random() * 6 );
29             face2 = ( int ) ( 1 + Math.random() * 6 );
30
31             total[ face1 + face2 ]++;
32         }
33     }
34
35     public void paint( Graphics g )
36     {
37         super.paint( g );
38         int y = 60;
39

```

```
40 g.drawString( "Sum", 5, 60 );
41 g.drawString( "Frequency", 85, 60 );
42
43 // ignore subscripts 0 and 1
44 for ( int k = 2; k < total.length; k++ ) {
45     g.drawString( String.valueOf( k ), 5, y += 10 );
46     g.drawString( String.valueOf( total[ k ] ), 85, y );
47
48     double percent = ( double ) total[ k ] / 360.0;
49     g.drawString( "%" + ( int ) percent, 150, y );
50 }
51 }
52 }
```



26

Java Object-Based Programming: Solutions

SOLUTIONS

26.2 Create a class called `Rational` for performing arithmetic with fractions. Write a driver program to test your class.

Use integer variables to represent the `private` instance variables of the class—the `numerator` and the `denominator`. Provide a constructor method that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form (i.e., the fraction

2/4

would be stored in the object as 1 in the `numerator` and 2 in the `denominator`). Provide a no-argument constructor that sets default values in case no initializers are provided. Provide `public` methods for each of the following:

- Addition of two `Rational` numbers. The result of the addition should be stored in reduced form.
- Subtraction of two `Rational` numbers. The result of the subtraction should be stored in reduced form.
- Multiplication of two `Rational` numbers. The result of the multiplication should be stored in reduced form.
- Division of two `Rational` numbers. The result of the division should be stored in reduced form.
- Printing `Rational` numbers in the form `a/b`, where `a` is the `numerator` and `b` is the `denominator`.
- Printing `Rational` numbers in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

ANS:

```
1 // Exercise 26.2 Solution
2 // Rational.java
3 // Definition of class Rational
4
5 public class Rational {
6     private int numerator;
7     private int denominator;
8
9     // Initialize numerator to 0 and denominator to 1
10    public Rational() { this( 0, 1 ); }
11
12    // Initialize numerator part to n and denominator part to 1
13    public Rational( int n ) { this( n, 1 ); }
14
15    // Initialize numerator part to n and denominator part to d
16    public Rational( int n, int d )
17    {
```

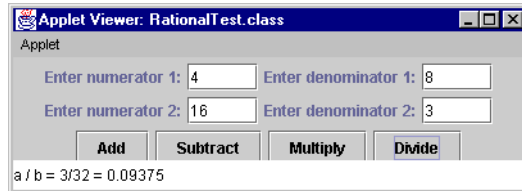
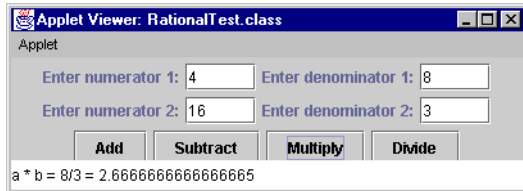
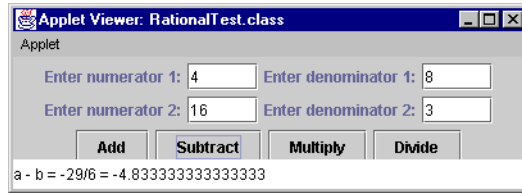
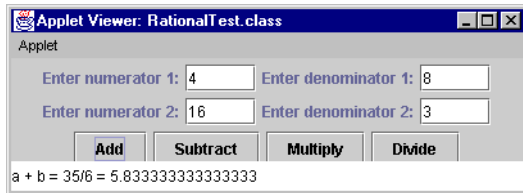
```
18     numerator = n;
19     denominator = d;
20     reduce();
21 }
22
23 // Add two Rational numbers
24 public Rational sum( Rational right )
25 {
26     int cd = denominator * right.denominator;
27     int numer = numerator * right.denominator +
28             right.numerator * denominator;
29
30     return new Rational( numer, cd );
31 }
32
33 // Subtract two Rational numbers
34 public Rational subtract( Rational right )
35 {
36     int cd = denominator * right.denominator;
37     int numer = numerator * right.denominator -
38             right.numerator * denominator;
39
40     return new Rational( numer, cd );
41 }
42
43 // Multiply two Rational numbers
44 public Rational multiply( Rational right )
45 {
46     return new Rational( numerator * right.numerator,
47                         denominator * right.denominator );
48 }
49
50 // Divide two Rational numbers
51 public Rational divide( Rational right )
52 {
53     return new Rational( numerator * right.denominator,
54                         denominator * right.numerator );
55 }
56
57 // Reduce the fraction
58 private void reduce()
59 {
60     int gcd = 0;
61     int smaller = Math.min( numerator, denominator );
62
63     for ( int x = 2; x <= smaller; x++ )
64         if ( numerator % x == 0 && denominator % x == 0 )
65             gcd = x;
66
67     if ( gcd != 0 ) {
68         numerator /= gcd;
69         denominator /= gcd;
70     }
71 }
72
73 // Return String representation of a Rational number
74 public String toString()
75     { return numerator + "/" + denominator; }
76
77 // Return floating-point String representation of
78 // a Rational number
79 public String toFloatString()
80 {
81     return Double.toString(
82         ( double ) numerator / denominator );
83 }
84 }
```

```
85 // Exercise 26.2: RationalTest.java
86 // Test the Rational number class
87 import java.awt.*;
88 import javax.swing.*;
89 import java.awt.event.*;
90
91 public class RationalTest extends JApplet implements ActionListener {
92     private Rational a, b;
93     private JLabel nlabel1, nlabel2, dlabel1, dlabel2;
94     private JTextField numer1, numer2, denom1, denom2;
95     private JButton addit, subtract, multiply, divide;
96
97     public void init()
98     {
99         nlabel1 = new JLabel( "Enter numerator 1:" );
100        nlabel2 = new JLabel( "Enter numerator 2:" );
101        dlabel1 = new JLabel( "Enter denominator 1:" );
102        dlabel2 = new JLabel( "Enter denominator 2:" );
103
104        numer1 = new JTextField( 5 );
105        numer2 = new JTextField( 5 );
106        denom1 = new JTextField( 5 );
107        denom2 = new JTextField( 5 );
108
109        addit = new JButton( "Add" );
110        subtract = new JButton( "Subtract" );
111        multiply = new JButton( "Multiply" );
112        divide = new JButton( "Divide" );
113
114        addit.addActionListener( this );
115        subtract.addActionListener( this );
116        multiply.addActionListener( this );
117        divide.addActionListener( this );
118
119        Container c = getContentPane();
120        c.setLayout(new FlowLayout());
121
122        c.add( nlabel1 );
123        c.add( numer1 );
124        c.add( dlabel1 );
125        c.add( denom1 );
126        c.add( nlabel2 );
127        c.add( numer2 );
128        c.add( dlabel2 );
129        c.add( denom2 );
130        c.add( addit );
131        c.add( subtract );
132        c.add( multiply );
133        c.add( divide );
134    }
135
136    public void actionPerformed((ActionEvent e)
137    {
138        Rational r;
139        a = new Rational( Integer.parseInt( numer1.getText() ),
140                        Integer.parseInt( denom1.getText() ) );
141        b = new Rational( Integer.parseInt( numer2.getText() ),
142                        Integer.parseInt( denom2.getText() ) );
143
144        if ( e.getSource() == addit ) {
145            r = a.sum( b );
146            showStatus( "a + b = " + r + " = " + r.toFloatString() );
147        }
148        else if ( e.getSource() == subtract ) {
149            r = a.subtract( b );
150            showStatus( "a - b = " + r + " = " + r.toFloatString() );
151        }
152        else if ( e.getSource() == multiply ) {
```

```

153     r = a.multiply( b );
154     showStatus( "a * b = " + r + " = " + r.toFloatString() );
155 }
156 else if ( e.getSource() == divide ) {
157     r = a.divide( b );
158     showStatus( "a / b = " + r + " = " + r.toFloatString() );
159 }
160 }
161 }

```



26.3 Modify the `Time2` class of Fig. 26.3 to include the `tick` method that increments the time stored in a `Time2` object by one second. Also provide method `incrementMinute` to increment the minute and method `incrementHour` to increment the hour. The `Time2` object should always remain in a consistent state. Write a driver program that tests the `tick` method, the `incrementMinute` method and the `incrementHour` method to ensure that they work correctly. Be sure to test the following cases:

- a) Incrementing into the next minute.
- b) Incrementing into the next hour.
- c) Incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

ANS:

```

1 // Exercise 26.3 Solution
2 // Time3.java
3 // Time3 class definition
4 public class Time3 {
5     private int hour;    // 0 - 23
6     private int minute; // 0 - 59
7     private int second; // 0 - 59
8
9     // Time constructor initializes each instance variable
10    // to zero. Ensures that Time3 object starts in a
11    // consistent state.
12    public Time3() { setTime( 0, 0, 0 ); }
13
14    // Time3 constructor: hour supplied, minute and second
15    // defaulted to 0.
16    public Time3( int h ) { setTime( h, 0, 0 ); }
17
18    // Time3 constructor: hour and minute supplied, second
19    // defaulted to 0.
20    public Time3( int h, int m ) { setTime( h, m, 0 ); }
21
22    // Time3 constructor: hour, minute and second supplied.
23    public Time3( int h, int m, int s ) { setTime( h, m, s ); }
24
25    // Set Methods
26    // Set a new Time3 value using military time. Perform
27    // validity checks on the data. Set invalid values
28    // to zero.
29    public void setTime( int h, int m, int s )
30    {
31        setHour( h );    // set the hour
32        setMinute( m ); // set the minute
33        setSecond( s ); // set the second
34    }
35
36    // set the hour
37    public void setHour( int h )
38        { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }
39
40    // set the minute
41    public void setMinute( int m )
42        { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
43
44    // set the second
45    public void setSecond( int s )
46        { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }
47
48    // Get Methods
49    // get the hour
50    public int getHour() { return hour; }
51
52    // get the minute
53    public int getMinute() { return minute; }
54
55    // get the second
56    public int getSecond() { return second; }
57

```



```

58 // Convert to String in military-time format
59 public String toMilitaryString()
60 {
61     return ( hour < 10 ? "0" : "" ) + hour +
62           ( minute < 10 ? "0" : "" ) + minute;
63 }
64
65 // Convert to String in standard-time format
66 public String toString()
67 {
68     return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
69           ":" + ( minute < 10 ? "0" : "" ) + minute +
70           ":" + ( second < 10 ? "0" : "" ) + second +
71           ( hour < 12 ? " AM" : " PM" );
72 }
73
74 // Tick the time by one second
75 public void tick()
76 {
77     setSecond( second + 1 );
78
79     if ( second == 0 )
80         incrementMinute();
81 }
82
83 // Increment the minute
84 public void incrementMinute()
85 {
86     setMinute( minute + 1 );
87
88     if ( minute == 0 )
89         incrementHour();
90 }
91
92 // Increment the hour
93 public void incrementHour()
94 {
95     setHour( hour + 1 );
96 }
97 }

```

```

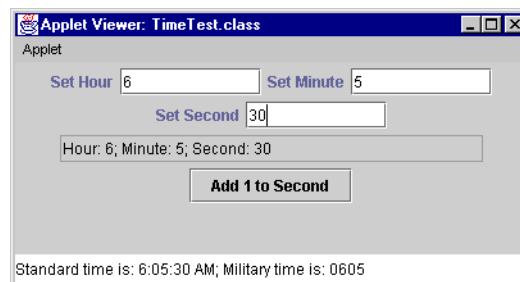
98 // Exercise 26.3 Solution
99 // TimeTest.java
100 // Demonstrating the Time class set and get methods
101 import java.awt.*;
102 import javax.swing.*;
103 import java.awt.event.*;
104
105 public class TimeTest extends JApplet implements ActionListener {
106     private Time3 t;
107     private JLabel hrLabel, minLabel, secLabel;
108     private JTextField hrField, minField, secField, display;
109     private JButton tickButton;
110
111     public void init()
112     {
113         t = new Time3();
114
115         hrLabel = new JLabel( "Set Hour" );
116         hrField = new JTextField( 10 );
117         hrField.addActionListener( this );
118         minLabel = new JLabel( "Set Minute" );
119         minField = new JTextField( 10 );
120         minField.addActionListener( this );
121         secLabel = new JLabel( "Set Second" );
122         secField = new JTextField( 10 );

```

```

123     secField.addActionListener( this );
124     display = new JTextField( 30 );
125     display.setEditable( false );
126     tickButton = new JButton( "Add 1 to Second" );
127     tickButton.addActionListener( this );
128
129     Container c = getContentPane();
130     c.setLayout( new FlowLayout() );
131     c.add( hrLabel );
132     c.add( hrField );
133     c.add( minLabel );
134     c.add( minField );
135     c.add( secLabel );
136     c.add( secField );
137     c.add( display );
138     c.add( tickButton );
139     updateDisplay();
140 }
141
142 public void actionPerformed((ActionEvent e)
143 {
144     if ( e.getSource() == tickButton )
145         t.tick();
146     else if ( e.getSource() == hrField ) {
147         t.setHour( Integer.parseInt( e.getActionCommand().toString() ) );
148         hrField.setText( "" );
149     }
150     else if ( e.getSource() == minField ) {
151         t.setMinute( Integer.parseInt( e.getActionCommand().toString() ) );
152         minField.setText( "" );
153     }
154     else if ( e.getSource() == secField ) {
155         t.setSecond( Integer.parseInt( e.getActionCommand().toString() ) );
156         secField.setText( "" );
157     }
158
159     updateDisplay();
160 }
161
162 public void updateDisplay()
163 {
164     display.setText( "Hour: " + t.getHour() +
165                    "; Minute: " + t.getMinute() +
166                    "; Second: " + t.getSecond() );
167     showStatus( "Standard time is: " + t.toString()+
168               "; Military time is: " + t.toMilitaryString() );
169 }
170 }

```



26.4 Create a class `Rectangle`. The class has attributes `length` and `width`, each of which defaults to 1. It has methods that calculate the perimeter and the area of the rectangle. It has *set* and *get* methods for both `length` and `width`. The *set* methods should verify that `length` and `width` are each floating-point numbers larger than 0.0 and less than 20.0.

ANS:

```

1 // Exercise 26.4 Solution
2 // MyRectangle.java
3 // Definition of class MyRectangle
4
5 public class MyRectangle {
6     private double length, width;
7
8     public MyRectangle() { this( 1.0, 1.0 ); }
9
10    public MyRectangle( double l, double w )
11    {
12        setLength( l );
13        setWidth( w );
14    }
15
16    public void setLength( double len )
17    { length = ( len >= 0.0 && len <= 20.0 ? len : 1.0 ); }
18
19    public void setWidth( double w )
20    { width = ( w >= 0 && w <= 20.0 ? w : 1.0 ); }
21
22    public double getLength() { return length; }
23
24    public double getWidth() { return width; }
25
26    public double perimeter() { return 2 * length + 2 * width; }
27
28    public double area() { return length * width; }
29
30    public String toString () {
31        return ("Length: " + length + "\n" +
32            " Width: " + width + "\n" +
33            " Perimeter: " + perimeter() + "\n" +
34            " Area: " + area() );
35    }
36 }

```

```

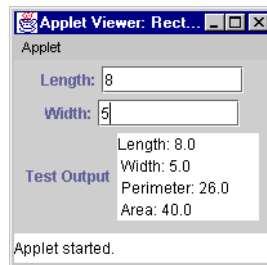
37 // Exercise 26.4 Solution
38 // Definition of class RectangleTest
39 import java.awt.*;
40 import javax.swing.*;
41 import java.awt.event.*;
42
43 public class RectangleTest extends JApplet implements ActionListener {
44     private JLabel prompt1, prompt2;
45     private JTextField input1, input2;
46     private JLabel outputLabel;
47     private JTextArea output;
48     private MyRectangle r;
49
50     public void init()
51     {
52         prompt1 = new JLabel( "Length:" );
53         prompt2 = new JLabel( "Width:" );
54         input1 = new JTextField( 10 );
55         input2 = new JTextField( 10 );
56         input2.addActionListener( this );
57
58         outputLabel = new JLabel( "Test Output" );
59         output = new JTextArea( 4, 10 );

```

```

60
61     Container c = getContentPane();
62     c.setLayout( new FlowLayout() );
63     c.add( prompt1 );
64     c.add( input1 );
65     c.add( prompt2 );
66     c.add( input2 );
67     c.add( outputLabel);
68     c.add( output);
69     r = new MyRectangle();
70 }
71
72 public void actionPerformed((ActionEvent e)
73 {
74     double d1, d2;
75
76     d1 = Double.parseDouble( input1.getText() );
77     d2 = Double.parseDouble( input2.getText() );
78
79     r.setLength( d1 );
80     r.setWidth( d2 );
81
82     output.setText( r.toString() );
83 }
84 }

```



26.5 Create a more sophisticated `Rectangle` class than the one you created in Exercise 26.4. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a `set` method that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single x - or y -coordinate larger than 20.0. The `set` method also verifies that the supplied coordinates do, in fact, specify a rectangle. Provide methods to calculate the length, width, perimeter and area. The length is the larger of the two dimensions. Include a predicate method `isSquare` which determines if the rectangle is a square.

26.6 Modify the `Rectangle` class of Exercise 26.5 to include a `draw` method that displays the rectangle inside a 25-by-25 box enclosing the portion of the first quadrant in which the rectangle resides. Use the methods of the `Graphics` class to help output the `Rectangle`. If you feel ambitious, you might include methods to scale the size of the rectangle, rotate it and move it around within the designated portion of the first quadrant.

26.7 Create a class `HugeInteger` which uses a 40-element array of digits to store integers as large as 40 digits each. Provide methods `inputHugeInteger`, `outputHugeInteger`, `addHugeIntegers` and `subtractHugeIntegers`. For comparing `HugeInteger` objects, provide methods `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `isGreaterThanOrEqualTo` and `isLessThanOrEqualTo`—each of these is a “predicate” method that simply returns `true` if the relationship holds between the two `HugeIntegers` and returns `false` if the relationship does not hold. Provide a predicate method `isZero`. If you feel ambitious, also provide the method `multiplyHugeIntegers`, the method `divideHugeIntegers` and the method `modulusHugeIntegers`.

26.8 Create class `SavingsAccount`. Use a `static` class variable to store the `annualInterestRate` for all account holders. Each object of the class contains a `private` instance variable `savingsBalance` indicating the amount the saver currently has on deposit. Provide method `calculateMonthlyInterest` to calculate the monthly interest by multiplying the `savingsBalance` by `annualInterestRate` divided by 12; this interest should be added to `savingsBalance`. Provide a `static` method `modifyInterestRate` that sets the `annualInterestRate` to a new value. Write a driver program to test class `SavingsAccount`. Instant-

tiate two `savingsAccount` objects, `saver1` and `saver2`, with balances of \$2000.00 and \$3000.00, respectively. Set `annualInterestRate` to 4%, then calculate the monthly interest and print the new balances for each of the savers. Then set the `annualInterestRate` to 5% and calculate the next month's interest and print the new balances for each of the savers.

26.9 Create class `IntegerSet`. Each object of the class can hold integers in the range 0 through 100. A set is represented internally as an array of `boolean`s. Array element `a[i]` is `true` if integer `i` is in the set. Array element `a[j]` is `false` if integer `j` is not in the set. The no-argument constructor initializes a set to the so-called “empty set” (i.e., a set whose array representation contains all `false` values).

Provide the following methods: Method `unionOfIntegerSets` creates a third set which is the set-theoretic union of two existing sets (i.e., an element of the third set's array is set to `true` if that element is `true` in either or both of the existing sets; otherwise, the element of the third set is set to `false`). Method `intersectionOfIntegerSets` creates a third set which is the set-theoretic intersection of two existing sets i.e., an element of the third set's array is set to `false` if that element is `false` in either or both of the existing sets; otherwise, the element of the third set is set to `true`). Method `insertElement` inserts a new integer `k` into a set (by setting `a[k]` to `true`). Method `deleteElement` deletes integer `m` (by setting `a[m]` to `false`). Method `setPrint` prints a set as a list of numbers separated by spaces. Print only those elements that are present in the set. Print `---` for an empty set. Method `isEqualTo` determines if two sets are equal. Write a program to test your `IntegerSet` class. Instantiate several `IntegerSet` objects. Test that all your methods work properly.

26.10 It would be perfectly reasonable for the `Time1` class of Fig. 26.1 to represent the time internally as the number of seconds since midnight rather than the three integer values `hour`, `minute` and `second`. Clients could use the same `public` methods and get the same results. Modify the `Time1` class of Fig. 26.1 to implement the `Time1` as the number of seconds since midnight and show that there is no visible change to the clients of the class.

ANS:

```

1 // Exercise 26.10 Solution
2 // Time1 class definition
3
4 public class Time1 {
5     private int totalSeconds;
6
7     public Time1() { setTime( 0, 0, 0 ); }
8
9     public void setTime( int h, int m, int s )
10    {
11        int hour, minute, second;
12
13        hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
14        minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
15        second = ( ( s >= 0 && s < 60 ) ? s : 0 );
16
17        totalSeconds = hour * 3600 + minute * 60 + second;
18    }
19
20    public String toMilitaryString()
21    {
22        int hour, minute, temp;
23
24        hour = totalSeconds / 3600;
25        temp = totalSeconds % 3600;
26        minute = temp / 60;
27
28        return ( hour < 10 ? "0" : "" ) + hour +
29                ( minute < 10 ? "0" : "" ) + minute;
30    }
31
32    public String toString()
33    {
34        int hour, minute, second, temp;
35
36        hour = totalSeconds / 3600;
37        temp = totalSeconds % 3600;
38        minute = temp / 60;
39        second = temp % 60;
40
41        return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
42                ":" + ( minute < 10 ? "0" : "" ) + minute +
43                ":" + ( second < 10 ? "0" : "" ) + second +
44                ( hour < 12 ? " AM" : " PM" );
45    }
46 }

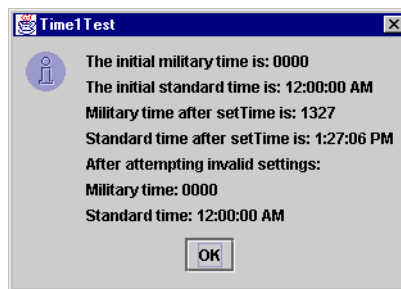
```

```

47 // Exercise 26.10 Solution
48 // TimeTest.java
49 // Class TimeTest to exercise class Time
50 import javax.swing.*;
51
52 public class TimeTest {
53     public static void main( String args[] )
54     {
55         Time1 t = new Time1();
56         String result = "";
57

```

```
58     result += "The initial military time is: " +
59             t.toMilitaryString();
60     result += "\nThe initial standard time is: " +
61             t.toString();
62
63     t.setTime( 13, 27, 6 );
64     result += "\nMilitary time after setTime is: " +
65             t.toMilitaryString();
66     result += "\nStandard time after setTime is: " +
67             t.toString();
68
69     t.setTime( 99, 99, 99 );
70     result += "\nAfter attempting invalid settings:";
71     result += "\nMilitary time: " + t.toMilitaryString();
72     result += "\nStandard time: " + t.toString();
73
74     JOptionPane.showMessageDialog(
75         null, result, "Time1Test",
76         JOptionPane.INFORMATION_MESSAGE );
77     System.exit( 0 );
78 }
79 }
```



26.11 (*Drawing Program*) Create a drawing applet that randomly draws lines, rectangles and ovals. For this purpose, create a set of “smart” shape classes where objects of these classes know how to draw themselves if provided with a `Graphics` object that tells them where to draw (i.e., the applet’s `Graphics` object allows a shape to draw on the applet’s background). The class names should be `MyLine`, `MyRect` and `MyOval`.

The data for class `MyLine` should include $x1$, $y1$, $x2$ and $y2$ coordinates. Method `drawLine` method of class `Graphics` will connect the two points supplied with a line. The data for classes `MyRect` and `MyOval` should include an upper-left x -coordinate value, an upper-left y -coordinate value, a *width* (must be nonnegative) and a *height* (must be nonnegative). All data in each class must be `private`.

In addition to the data, each class should define at least the following `public` methods:

- A constructor with no arguments that sets the coordinates to 0.
- A constructor with arguments that sets the coordinates to the supplied values.
- Set* methods for each individual piece of data that allow the programmer to independently set any piece of data in a shape (e.g., if you have an instance variable $x1$, you should have a method `setX1`).
- Get* methods for each individual piece of data that allow the programmer to independently retrieve any piece of data in a shape (e.g., if you have an instance variable $x1$, you should have a method `getX1`).
- A `draw` method with the first line

```
public void draw( Graphics g )
```

will be called from the applet’s `paint` method to draw a shape onto the screen.

The preceding methods are required. If you would like to provide more methods for flexibility, please do so.

Begin by defining class `MyLine` and an applet to test your classes. The applet should have a `MyLine` instance variable `line` that can refer to one `MyLine` object (created in the applet’s `init` method with random coordinates). The applet’s `paint` method should draw the shape with a statement like

```
line.draw( g );
```

where `line` is the `MyLine` reference and `g` is the `Graphics` object that the shape will use to draw itself on the applet.

Next, change the single `MyLine` reference into an array of `MyLine` references and hard code several `MyLine` objects into the program for drawing. The applet’s `paint` method should walk through the array of `MyLine` objects and draw every one.

After the preceding part is working, you should define the `MyOval` and `MyRect` classes and add objects of these classes into the `MyRect` and `MyOval` arrays. The applet’s `paint` method should walk through each array and draw every shape. Create five shapes of each type.

Once the applet is running, select `Reload` from the appletviewer’s `Applet` menu to reload the applet. This will cause the applet to choose new random numbers for the shapes and draw the shapes again.

In Chapter 27, we will modify this exercise to take advantage of the similarities between the classes and to avoid reinventing the wheel.

ANS:

```

1 // Exercise 26.11 Solution
2 // MyLine.java
3 // Definition of class MyLine
4 import java.awt.Graphics;
5
6 public class MyLine {
7     private int x1, x2;
8     private int y1, y2;
9
10    public MyLine()
11    {
12        x1 = 0;
13        y1 = 0;
14        x2 = 0;
15        y2 = 0;
16    }
17
18    public MyLine( int x1, int y1, int x2, int y2 )
19    {
20        setX1( x1 );
21        setX2( x2 );
22        setY1( y1 );

```



```

23     setY2( y2 );
24 }
25
26 public void setX1( int x1 )
27 { this.x1 = ( x1 >= 0 ? x1 : 0 ); }
28
29 public void setX2( int x2 )
30 { this.x2 = ( x2 >= 0 ? x2 : 0 ); }
31
32 public void setY1( int y1 )
33 { this.y1 = ( y1 >= 0 ? y1 : 0 ); }
34
35 public void setY2( int x2 )
36 { this.y2 = ( y2 >= 0 ? y2 : 0 ); }
37
38 public int getX1() { return x1; }
39 public int getX2() { return x2; }
40 public int getY1() { return y1; }
41 public int getY2() { return y2; }
42
43 public void draw( Graphics g )
44 {
45     g.drawLine( x1, y1, x2, y2 );
46 }
47 }

```

```

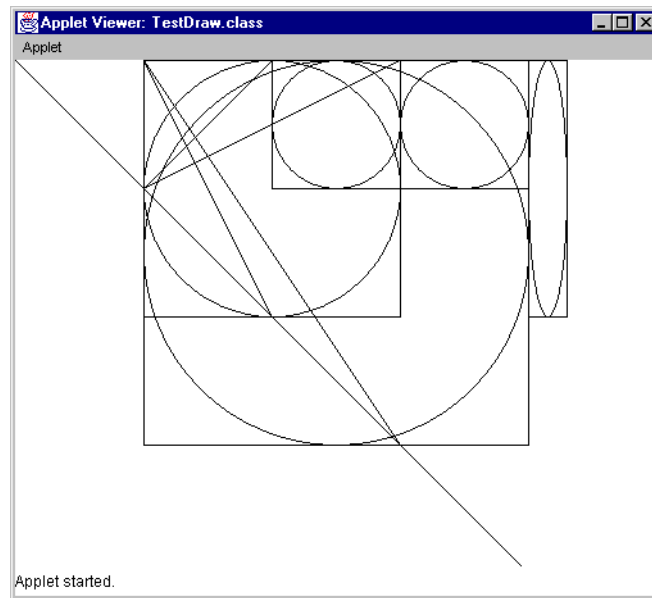
48 // Exercise 26.11 Solution
49 // MyOval.java
50 // Definition of class MyRect
51 import java.awt.Graphics;
52
53 public class MyOval {
54     private int length, width;
55     private int upperLeftX, upperLeftY;
56
57     public MyOval()
58     {
59         length = 0;
60         width = 0;
61         upperLeftX = 0;
62         upperLeftY = 0;
63     }
64
65     public MyOval( int x, int y, int l, int w )
66     {
67         setUpperLeftX( x );
68         setUpperLeftY( y );
69         setLength( l );
70         setWidth( w );
71     }
72
73     public void setLength( int len )
74     { length = ( len >= 0 ? len : 0 ); }
75
76     public void setUpperLeftX( int x )
77     { upperLeftX = ( x >= 0 ? x : 0 ); }
78
79     public void setUpperLeftY( int y )
80     { upperLeftY = ( y >= 0 ? y : 0 ); }
81
82     public void setWidth( int w )
83     { width = ( w >= 0 ? w : 0 ); }
84
85     public int getLength() { return length; }
86
87     public int getWidth() { return width; }

```

```
88
89     public int getUpperLeftX() { return upperLeftX; }
90
91     public int getUpperLeftY() { return upperLeftY; }
92
93     public void draw( Graphics g )
94     {
95         g.drawOval( upperLeftX, upperLeftY, length, width );
96     }
97 }
```

```
98 // Exercise 26.11 Solution
99 // MyRect.java
100 // Definition of class MyRect
101 import java.awt.Graphics;
102
103 public class MyRect {
104     private int length, width;
105     private int upperLeftX, upperLeftY;
106
107     public MyRect()
108     {
109         length = 0;
110         width = 0;
111         upperLeftX = 0;
112         upperLeftY = 0;
113     }
114
115     public MyRect( int x, int y, int l, int w )
116     {
117         setUpperLeftX( x );
118         setUpperLeftY( y );
119         setLength( l );
120         setWidth( w );
121     }
122
123     public void setLength( int len )
124     { length = ( len >= 0.0 ? len : 1 ); }
125
126     public void setUpperLeftX( int x )
127     { upperLeftX = ( x >= 0 ? x : 0 ); }
128
129     public void setUpperLeftY( int y )
130     { upperLeftY = ( y >= 0 ? y : 0 ); }
131
132     public void setWidth( int w )
133     { width = ( w >= 0 ? w : 1 ); }
134
135     public int getLength() { return length; }
136
137     public int getWidth() { return width; }
138
139     public int getUpperLeftX() { return upperLeftX; }
140
141     public int getUpperLeftY() { return upperLeftY; }
142
143     public void draw( Graphics g )
144     {
145         g.drawRect( upperLeftX, upperLeftY, length, width );
146     }
147 }
```

```
148 // Exercise 26.11 Solution
149 // Definition of class RectangleTest
150 import java.awt.*;
151 import javax.swing.*;
152
153 public class TestDraw extends JApplet {
154     private MyLine line[];
155     private MyOval oval[];
156     private MyRect rect[];
157
158     public void initDraw()
159     {
160         line = new MyLine[ 5 ];
161         line[ 0 ] = new MyLine( 100, 100, 200, 200 );
162         line[ 1 ] = new MyLine( 200, 200, 100, 100 );
163         line[ 2 ] = new MyLine( 300, 300, 100, 100 );
164         line[ 3 ] = new MyLine( 400, 400, 0, 0 );
165         line[ 4 ] = new MyLine( 100, 100, 300, 300 );
166
167         oval = new MyOval[ 5 ];
168         oval[ 0 ] = new MyOval( 100, 100, 200, 200 );
169         oval[ 1 ] = new MyOval( 200, 200, 100, 100 );
170         oval[ 2 ] = new MyOval( 300, 300, 100, 100 );
171         oval[ 3 ] = new MyOval( 400, 400, 30, 200 );
172         oval[ 4 ] = new MyOval( 100, 100, 300, 300 );
173
174         rect = new MyRect[ 5 ];
175         rect[ 0 ] = new MyRect( 100, 100, 200, 200 );
176         rect[ 1 ] = new MyRect( 200, 200, 100, 100 );
177         rect[ 2 ] = new MyRect( 300, 300, 100, 100 );
178         rect[ 3 ] = new MyRect( 400, 400, 30, 200 );
179         rect[ 4 ] = new MyRect( 100, 100, 300, 300 );
180     }
181
182     public void paint( Graphics g )
183     {
184         initDraw();
185
186         for ( int i = 0; i < line.length; i++ )
187             line[ i ].draw( g );
188
189         for ( int i = 0; i < oval.length; i++ )
190             oval[ i ].draw( g );
191
192         for ( int i = 0; i < rect.length; i++ )
193             rect[ i ].draw( g );
194     }
195 }
```



27

Java Object-Oriented Programming: Solutions

SOLUTIONS

27.3 Consider the class `Bicycle`. Given your knowledge of some common components of bicycles, show a class hierarchy in which the class `Bicycle` inherits from other classes, which, in turn, inherit from yet other classes. Discuss the instantiation of various objects of class `Bicycle`. Discuss inheritance from class `Bicycle` for other closely related subclasses.

ANS: Possible classes are displayed in bold.

`Bicycle` composed of:

`Handle bars`

`Seat`

`Frame`

`Wheels` composed of:

`Tires`

`Rims`

`Spokes`

`Pedals`

`Chain` composed of:

`Links`

`Brakes` composed of:

`Wires`

`Brake Pads`

`Brake Handles`

27.4 Define each of the following terms: single inheritance, multiple inheritance, interface, superclass and subclass.

ANS:

- a) Single inheritance is the process by which a class incorporates the attributes and behaviors of a previously defined class.
- b) Multiple inheritance is the process by which a class incorporates the attributes and behaviors of two or more previously defined classes.
- c) An interface is a collection of abstract methods that can be implemented to simulate multiple inheritance.
- d) A superclass is a class from which other classes inherit attributes and behaviors.
- e) A subclass is a class that has inherited attributes and behaviors from a superclass.

27.5 Discuss why casting a superclass reference to a subclass reference is potentially dangerous.

ANS: The reference must refer to an object of the subclass, before being used. When the compiler looks at an object through a subclass reference, it expects to see all the pieces of the subclass. However, if the superclass reference originally referred to a superclass object, the additional pieces added by the subclass do not exist. For this reason, an attempt to cast

a subclass reference, that refers to a subclass object, into a superclass reference results in a `ClassCastException` at execution time.

27.6 Distinguish between single inheritance and multiple inheritance. Why does Java not support multiple inheritance? What feature of Java helps realize the benefits of multiple inheritance?

ANS: Single inheritance inherits from one class only. Multiple inheritance inherits from two or more classes. Java does not support multiple inheritance because of the problems that can be encountered with multiple inheritance. However, Java does support interfaces which provide the benefits of multiple inheritance without the potential problems.

27.7 (*True/False*) A subclass is generally smaller than its superclass.

ANS: False. A subclass is usually larger because it normally adds more data and more functionality.

27.8 (*True/False*) A subclass object is also an object of that subclass's superclass.

ANS: True.

27.9 Rewrite the Point, Circle, Cylinder program of Fig. 27.4 as a Point, Square, Cube program. Do this two ways—once with inheritance and once with composition.

ANS:

```

1 // Exercies 27.9 -- Composition
2 // Point.java
3 // Definition of class Point
4
5 public class Point {
6     private double x, y; // coordinates of the Point
7
8     public Point( double a, double b ) { setPoint( a, b ); }
9
10    public void setPoint( double a, double b )
11    {
12        x = a;
13        y = b;
14    }
15
16    public double getX() { return x; }
17
18    public double getY() { return y; }
19
20    public String toString()
21    { return "[" + x + ", " + y + ""]; }
22
23    public String getName() { return "Point"; }
24 }

```

```

25 // Exercies 27.9 -- Composition
26 // Square.java
27 // Definition of class Square
28
29 public class Square {
30     private double side;
31     private Point p; // composition
32
33     public Square() { this( 0.0, 0.0, 0.0 ); }
34
35     public Square( double s, double a, double b )
36     {
37         p = new Point( a, b ); // instantiate point object
38         setSide( s );
39     }
40
41     public void setSide( double s )
42     { side = ( s >= 0 ? s : 0 ); }
43
44     public double getSide() { return side; }
45
46     public double area() { return Math.pow( side, 2 ); }
47
48     public String toString()
49     { return "Corner = " + p.toString() + "; Side = " + side; }
50
51     public String getName() { return "Square"; }
52
53     public String getPointName() { return p.getName(); }
54
55     public String getPointString() { return p.toString(); }
56 }

```

```

57 // Exercies 27.9 -- Composition
58 // Cube.java
59 // Definition of class Cube
60
61 public class Cube {
62     private double depth;
63     private Square s;        // composition
64
65     public Cube( double m, double a, double b )
66     {
67         s = new Square( m, a, b );
68         depth = m;
69     }
70
71     public double getDepth() { return depth; }
72
73     public double area() { return s.area() * 6; }
74
75     public double volume() { return s.area() * depth; }
76
77     public String toString()
78     { return s.toString() + "; Depth = " + depth; }
79
80     public String getName() { return "Cube"; }
81
82     public double getSquareArea() { return s.area(); }
83
84     public String getSquareName() { return s.getName(); }
85
86     public String getSquareString() { return s.toString(); }
87
88     public String getSPointString() { return s.getPointString(); }
89
90     public String getSPointName() { return s.getPointName(); }
91 }

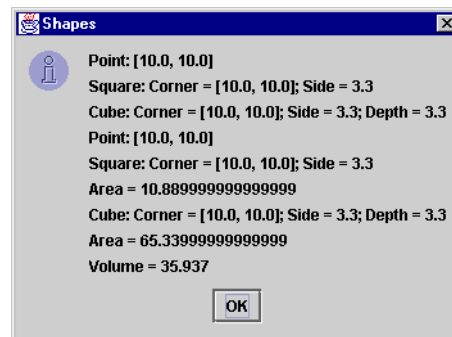
```

```

92 // Exercise 27.9 -- Composition
93 // Test.java
94 // Driver for point, square, cube composition program
95 import javax.swing.*;
96
97 public class Test {
98     public static void main( String args[] )
99     {
100         Cube cube = new Cube( 3.3, 10, 10 );
101         String result = "";
102
103         result += cube.getSPointName() + ": " +
104                 cube.getSPointString();
105
106         result += "\n" + cube.getSquareName() + ": " +
107                 cube.getSquareString();
108
109         result += "\n" + cube.getName() + ": " +
110                 cube.toString();
111
112         result += "\n" + cube.getSPointName() +
113                 ": " + cube.getSPointString();
114
115         result += "\n" + cube.getSquareName() +
116                 ": " + cube.getSquareString();
117         result += "\n" + "Area = " + cube.getSquareArea();
118
119         result += "\n" + cube.getName() +
120                 ": " + cube.toString();
121         result += "\n" + "Area = " + cube.area();
122         result += "\n" + "Volume = " + cube.volume();

```

```
123
124     JOptionPane.showMessageDialog(
125         null, result, "Shapes",
126         JOptionPane.INFORMATION_MESSAGE );
127     System.exit( 0 );
128 }
129 }
```



ANS:

```

1 // Exercies 27.9 -- Inheritance
2 // Point.java
3 // Definition of class Point
4
5 public class Point extends Shape {
6     protected double x, y;
7
8     public Point( double a, double b ) { setPoint( a, b ); }
9
10    public void setPoint( double a, double b )
11    {
12        x = a;
13        y = b;
14    }
15
16    public double getX() { return x; }
17
18    public double getY() { return y; }
19
20    public String toString()
21    { return "[" + x + ", " + y + "]; }
22
23    public String getName() { return "Point"; }
24 }

```

```

25 // Exercies 27.9 -- Inheritance
26 // Shape.java
27 // Definition of abstract base class Shape
28
29 public abstract class Shape {
30     public double area() { return 0.0; }
31     public double volume() { return 0.0; }
32     public abstract String getName();
33 }

```

```

34 // Exercies 27.9 -- Inheritance
35 // Square.java
36 // Definition of class Square
37
38 public class Square extends Point {
39     protected double side;
40
41     public Square()
42     { this( 0.0, 0.0, 0.0 ); }
43
44     public Square( double s, double a, double b )
45     {
46         super( a, b );
47         setSide( s );
48     }
49
50     public void setSide( double s )
51     { side = ( s >= 0 ? s : 0 ); }
52
53     public double getSide() { return side; }
54
55     public double area() { return Math.pow( side, 2 ); }
56
57     public String toString()
58     { return "Corner = " + super.toString() +
59       "; side = " + side; }
60 }

```

```

61 public String getName() { return "Square"; }
62 }

```

```

63 // Exercies 27.9 -- Inheritance
64 // Cube.java
65 // Definition of class Cylinder
66
67 public class Cube extends Square {
68     private double depth;
69
70     public Cube( double s, double a, double b )
71     {
72         super( s, a, b );
73         depth = s;
74     }
75
76     public double area() { return super.area() * 6; }
77
78     public double volume() { return super.area() * depth; }
79
80     public String toString()
81     { return super.toString() + "; depth = " + depth; }
82
83     public String getName() { return "Cube"; }
84 }

```

```

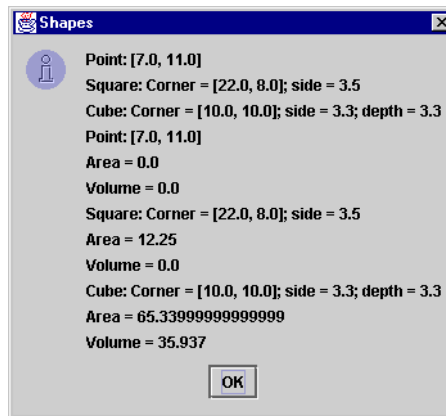
85 // Exercies 27.9 -- Inheritance
86 // Test.java
87 // Driver for point, square, cube hierarchy
88 import javax.swing.*;
89
90 public class Test {
91     public static void main( String args[] )
92     {
93         Point point = new Point( 7, 11 );
94         Square square = new Square( 3.5, 22, 8 );
95         Cube cube = new Cube( 3.3, 10, 10 );
96
97         Shape[] arrayOfShapes = new Shape[ 3 ];
98         String result = "";
99
100        arrayOfShapes[ 0 ] = point;
101        arrayOfShapes[ 1 ] = square;
102        arrayOfShapes[ 2 ] = cube;
103
104        result += point.getName() + ": " +
105                point.toString();
106
107        result += "\n" + square.getName() + ": " +
108                square.toString();
109
110        result += "\n" + cube.getName() + ": " +
111                cube.toString();
112
113        for ( int i = 0; i < 3; i++ ) {
114            result += "\n" + arrayOfShapes[ i ].getName() +
115                    ": " + arrayOfShapes[ i ].toString();
116            result += "\n" + "Area = " +
117                    arrayOfShapes[ i ].area();
118            result += "\n" + "Volume = " +
119                    arrayOfShapes[ i ].volume();
120        }
121

```

```

122     JOptionPane.showMessageDialog(
123         null, result, "Shapes",
124         JOptionPane.INFORMATION_MESSAGE );
125     System.exit( 0 );
126 }
127 }

```



27.10 In the chapter, we stated, “When a superclass method is inappropriate for a subclass, that method can be overridden in the subclass with an appropriate implementation.” If this is done, does the subclass-is-a-superclass-object relationship still hold? Explain your answer.

ANS: Yes, the subclass-is-a-superclass-object relationship still holds. In Java, it is not possible to break this relationship.

27.11 How is it that polymorphism enables you to program “in the general” rather than “in the specific”? Discuss the key advantages of programming “in the general.”

ANS: Polymorphism enables the programmer to concentrate on the processing of common operations that are applied to all data types in a class hierarchy without the knowledge of individual details of each data type. The general processing capabilities are separated from the internal details of each type. Programming in the general enables you to write more maintainable and modifyable systems. New data types can be added into the system as long as they belong to the portion of the class hierarchy being polymorphically processed.

27.12 Discuss the problems of programming with `switch` logic. Explain why polymorphism is an effective alternative to using `switch` logic.

ANS: The main problem with programming using the `switch` structure is the extensibility and maintainability of the program. A program containing many `switch` structures is difficult to modify. All the structures must be modified to handle the processing of an additional type or of one less type. Polymorphism determines the type of an object automatically, so it is not necessary to determine the type of an object to process the object in a generic manner.

27.13 Distinguish between inheriting interface and inheriting implementation. How do inheritance hierarchies designed for inheriting interface differ from those designed for inheriting implementation?

ANS: When a class inherits implementation, it inherits previously defined functionality from another class. When a class inherits interface, it inherits the definition of what the interface to the new class type should be. The implementation is then provided by the programmer defining the new class type. Inheritance hierarchies designed for inheriting implementation are used to reduce the amount of new code that is being written. Such hierarchies are commonly used to facilitate software reusability. Inheritance hierarchies designed for inheriting interface are used to write programs that perform generic processing of many class types. Such hierarchies are commonly used to facilitate software extensibility (i.e., new types can be added to the hierarchy without changing the generic processing capabilities of the program).

27.14 Distinguish between non-abstract methods and abstract methods.

ANS: A non-abstract method provides implementation. An abstract method does not provide any implementation.

27.15 (*True/False*) All methods in an abstract superclass must be declared `abstract`.

ANS: False. An abstract class must have at least one abstract method. Any number of methods in the class can be non-abstract.

27.16 Suggest one or more levels of **abstract** superclasses for the Shape hierarchy discussed in the beginning of this chapter (the first level is Shape and the second level consists of the classes TwoDimensionalShape and ThreeDimensionalShape).

27.17 How does polymorphism promote extensibility?

ANS: Polymorphism makes programs more extensible by making all method calls generic. When a new class type with the appropriate methods is added to the hierarchy, no changes need to be made to the generic method calls to enable processing of the new data type.

27.18 You have been asked to develop a flight simulator that will have elaborate graphical outputs. Explain why polymorphic programming would be especially effective for a problem of this nature.

27.19 (*Drawing Application*) Modify the drawing program of Exercise 26.11 to create a drawing application that draws random lines, rectangles and ovals. [Note: Like an applet, a JFrame has a `paint` method that you can override to draw on the background of the JFrame.]

For this exercise, modify the `MyLine`, `MyOval` and `MyRect` classes of Exercise 26.11 to create the class hierarchy in Fig. 27.8. The classes of the `MyShape` hierarchy should be “smart” shape classes where objects of these classes know how to draw themselves (if provided with a `Graphics` object that tells them where to draw). The only `switch` or `if/else` logic in this program should be to determine the type of shape object to create (use random numbers to pick the shape type and the coordinates of each shape). Once an object from this hierarchy is created, it will be manipulated for the rest of its lifetime as a superclass `MyShape` reference.

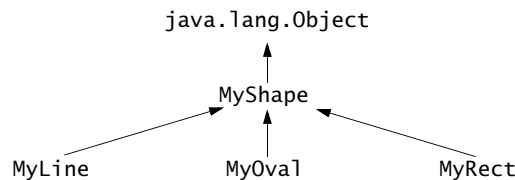


Fig. 27.8 The `MyShape` hierarchy.

Class `MyShape` in Fig. 27.8 *must* be **abstract**. The only data representing the coordinates of the shapes in the hierarchy should be defined in class `MyShape`. Lines, rectangles and ovals can all be drawn if you know two points in space. Lines require $x1$, $y1$, $x2$ and $y2$ coordinates. The `drawLine` method of the `Graphics` class will connect the two points supplied with a line. If you have the same four coordinate values ($x1$, $y1$, $x2$ and $y2$) for ovals and rectangles, you can calculate the four arguments needed to draw them. Each requires an upper-left x -coordinate value (minimum of the two x -coordinate values), an upper-left y -coordinate value (minimum of the two y -coordinate values), a *width* (difference between the two x -coordinate values; must be nonnegative) and a *height* (difference between the two y -coordinate values; must be nonnegative). [Note: In Chapter 29, each x,y pair will be captured using mouse events from mouse interactions between the user and the program’s background. These coordinates will be stored in an appropriate shape object as selected by the user. As you begin the exercise, you will use random coordinate values as arguments to the constructor.]

In addition to the data for the hierarchy, class `MyShape` should define at least the following methods:

- A constructor with no arguments that sets the coordinates to 0.
- A constructor with arguments that sets the coordinates to the supplied values.
- Set* methods for each individual piece of data that allow the programmer to independently set any piece of data for a shape in the hierarchy (e.g., if you have an instance variable `x1`, you should have a method `setX1`).
- Get* methods for each individual piece of data that allow the programmer to independently retrieve any piece of data for a shape in the hierarchy (e.g., if you have an instance variable `x1`, you should have a method `getX1`).
- The abstract method

```
public abstract void draw( Graphics g );
```

This method will be called from the program’s `paint` method to draw a shape onto the screen.

The preceding methods are required. If you would like to provide more methods for flexibility, please do so. However, be sure that any method you define in this class is a method that would be used by *all* shapes in the hierarchy.

All data *must* be **private** to class `MyShape` in this exercise (this forces you to use proper encapsulation of the data and provide proper *set/get* methods to manipulate the data). You are not allowed to define new data that can be derived from existing information. As explained previously, the upper-left x , upper-left y , *width* and *height* needed to draw an oval or rectangle can be calculated if you already know two points in space. All subclasses of `MyShape` should provide two constructors that mimic those provided by class `MyShape`.

Objects of the `MyOval` and `MyRect` classes should not calculate their upper-left *x*-coordinate, upper-left *y*-coordinate, *width* and *height* until they are about to draw. Never modify the *x1*, *y1*, *x2* and *y2* coordinates of a `MyOval` or `MyRect` object to prepare to draw them. Instead, use the temporary results of the calculations described above. This will help us enhance the program in Chapter 29 by allowing the user to select each shape's coordinates with the mouse.

There should be no `MyLine`, `MyOval` or `MyRect` references in the program—only `MyShape` references that refer to `MyLine`, `MyOval` and `MyRect` objects are allowed. The program should keep an array of `MyShape` references containing all shapes. The program's `paint` method should walk through the array of `MyShape` references and draw every shape (i.e., call every shape's `draw` method).

Begin by defining class `MyShape`, class `MyLine` and an application to test your classes. The application should have a `MyShape` instance variable that can refer to one `MyLine` object (created in the application's constructor). The `paint` method (for your subclass of `JFrame`) should draw the shape with a statement like

```
currentShape.draw( g );
```

where `currentShape` is the `MyShape` reference and `g` is the `Graphics` object that the shape will use to draw itself on the background of the window.

Next, change the single `MyShape` reference into an array of `MyShape` references and hard code several `MyLine` objects into the program for drawing. The application's `paint` method should walk through the array of shapes and draw every shape.

After the preceding part is working, you should define the `MyOval` and `MyRect` classes and add objects of these classes into the existing array. For now, all the shape objects should be created in the constructor for your subclass of `JFrame`. In Chapter 29, we will create the objects when the user chooses a shape and begins drawing it with the mouse.

ANS:

```

1 // Exercise 27.19 Solution
2 // MyShape.java
3
4 import java.awt.Graphics;
5
6 public abstract class MyShape extends Object {
7     private int x1, x2, y1, y2;
8
9     public MyShape()
10    {
11        setX1( 0 );
12        setX2( 0 );
13        setY1( 0 );
14        setY2( 0 );
15    }
16
17    public MyShape( int x1, int y1, int x2, int y2 )
18    {
19        setX1( x1 );
20        setX2( x2 );
21        setY1( y1 );
22        setY2( y2 );
23    }
24
25    public void setX1( int x1 ) { this.x1 = ( x1 >= 0 ? x1 : 0 ); }
26    public void setX2( int x2 ) { this.x2 = ( x2 >= 0 ? x2 : 0 ); }
27    public void setY1( int y1 ) { this.y1 = ( y1 >= 0 ? y1 : 0 ); }
28    public void setY2( int y2 ) { this.y2 = ( y2 >= 0 ? y2 : 0 ); }
29
30    public int getX1() { return x1; }
31    public int getX2() { return x2; }
32    public int getY1() { return y1; }
33    public int getY2() { return y2; }
34
35    public abstract void draw( Graphics g );
36 }

```

```
37 // Exercise 27.19 Solution
38 // MyLine.java
39 // Definition of class MyLine
40 import java.awt.Graphics;
41
42 public class MyLine extends MyShape {
43
44     public MyLine()
45     {
46         super ();
47     }
48
49     public MyLine( int x1, int y1, int x2, int y2 )
50     {
51         super (x1, y1, x2, y2);
52     }
53
54     public void draw( Graphics g )
55     {
56         g.drawLine( getX1(), getY1(), getX2(), getY2() );
57     }
58 }
```

```
59 // Exercise 27.19 Solution
60 // MyOval.java
61 import java.awt.Graphics;
62
63 public class MyOval extends MyShape {
64
65     public MyOval()
66     {
67         super();
68     }
69
70     public MyOval( int x1, int y1, int x2, int y2 )
71     {
72         super( x1, y1, x2, y2 );
73     }
74
75     public void draw( Graphics g )
76     {
77         g.drawOval( Math.min( getX1(), getX2() ),
78                   Math.min( getY1(), getY2() ),
79                   Math.abs( getY2() - getY1() ),
80                   Math.abs( getX2() - getX1() ) );
81     }
82 }
```

```
83 // Exercise 27.19 Solution
84 // MyRect.java
85 import java.awt.Graphics;
86
87 public class MyRect extends MyShape {
88
89     public MyRect()
90     {
91         super ();
92     }
93
94     public MyRect( int x1, int y1, int x2, int y2 )
95     {
96         super( x1, y1, x2, y2 );
97     }
98 }
```



```

99     public void draw( Graphics g )
100     {
101         g.drawRect( Math.min( getX1(),getX2() ),
102                   Math.min( getY1(), getY2() ),
103                   Math.abs( getY2() - getY1() ),
104                   Math.abs( getX2() - getX1() ) );
105     }
106 }

```

```

107 // Exercise 27.19 Solution
108 // MyShape.java
109
110 import java.awt.Graphics;
111
112 public abstract class MyShape extends Object {
113     private int x1, x2, y1, y2;
114
115     public MyShape()
116     {
117         setX1( 0 );
118         setX2( 0 );
119         setY1( 0 );
120         setY2( 0 );
121     }
122
123     public MyShape( int x1, int y1, int x2, int y2 )
124     {
125         setX1( x1 );
126         setX2( x2 );
127         setY1( y1 );
128         setY2( y2 );
129     }
130
131     public void setX1( int x1 ) { this.x1 = ( x1 >= 0 ? x1 : 0 ); }
132     public void setX2( int x2 ) { this.x2 = ( x2 >= 0 ? x2 : 0 ); }
133     public void setY1( int y1 ) { this.y1 = ( y1 >= 0 ? y1 : 0 ); }
134     public void setY2( int x2 ) { this.y2 = ( y2 >= 0 ? y2 : 0 ); }
135
136     public int getX1() { return x1; }
137     public int getX2() { return x2; }
138     public int getY1() { return y1; }
139     public int getY2() { return y2; }
140
141     public abstract void draw( Graphics g );
142 }

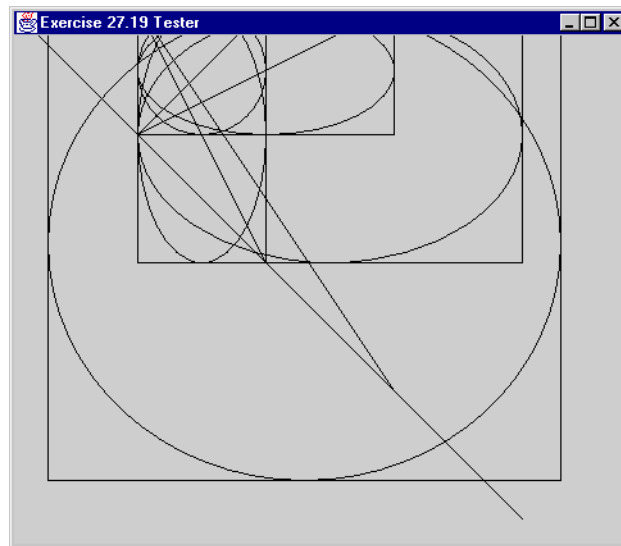
```

```

143 // Exercise 27.19 Solution
144 // TestDrawWindow.java
145
146 import java.awt.*;
147 import javax.swing.*;
148
149 public class TestDrawWindow extends JFrame {
150     private MyShape shape[];
151
152     public TestDrawWindow()
153     {
154         super( "Exercise 9.28 Tester" );
155         shape = new MyShape[ 15 ];
156
157         shape[ 0 ] = new MyLine( 100, 100, 200, 200 );
158         shape[ 1 ] = new MyLine( 200, 200, 100, 100 );
159         shape[ 2 ] = new MyLine( 300, 300, 100, 100 );
160         shape[ 3 ] = new MyLine( 400, 400, 0, 0 );

```

```
161     shape[ 4 ] = new MyLine( 100, 100, 300, 300 );
162
163     shape[ 5 ] = new MyOval( 100, 100, 200, 200 );
164     shape[ 6 ] = new MyOval( 200, 200, 100, 100 );
165     shape[ 7 ] = new MyOval( 300, 300, 100, 100 );
166     shape[ 8 ] = new MyOval( 400, 400, 30, 200 );
167     shape[ 9 ] = new MyOval( 100, 100, 300, 300 );
168
169     shape[ 10 ] = new MyRect( 100, 100, 200, 200 );
170     shape[ 11 ] = new MyRect( 200, 200, 100, 100 );
171     shape[ 12 ] = new MyRect( 300, 300, 100, 100 );
172     shape[ 13 ] = new MyRect( 400, 400, 30, 200 );
173     shape[ 14 ] = new MyRect( 100, 100, 300, 300 );
174
175     setDefaultCloseOperation( WindowConstants.DISPOSE_ON_CLOSE );
176 }
177
178 public static void main( String args[] )
179 {
180     TestDrawWindow window = new TestDrawWindow();
181     window.setSize( 500, 500 );
182     window.show();
183 }
184
185 public void paint( Graphics g )
186 {
187     for ( int i = 0; i < shape.length; i++ )
188         shape[ i ].draw( g );
189 }
190 }
```



28

Java Graphics and Java2D: Solutions

SOLUTIONS

28.4 Fill in the blanks in each of the following:

a) Class _____ of the Java2D API is used to define ovals.

ANS: `Ellipse2D`

b) Methods `draw` and `fill` of class `Graphics2D` require an object of type _____ as their argument.

ANS: `Shape`

c) The three constants that specify font style are _____, _____ and _____.

ANS: `Font.PLAIN`, `Font.BOLD` and `Font.ITALIC`

d) `Graphics2D` method _____ sets the painting color for Java2D shapes.

ANS: `setColor`

28.5 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The `drawPolygon` method automatically connects the endpoints of the polygon.

ANS: True.

b) The `drawLine` method draws a line between two points.

ANS: True.

c) The `fillArc` method uses degrees to specify the angle.

ANS: True.

d) In the Java coordinate system, y values increase from top to bottom.

ANS: True.

e) The `Graphics` class inherits directly from class `Object`.

ANS: True.

f) The `Graphics` class is an abstract class.

ANS: True.

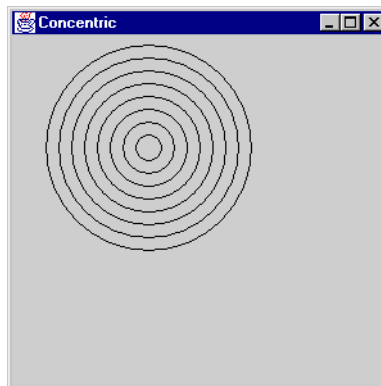
g) The `Font` class inherits directly from class `Graphics`.

ANS: False. Class `Font` inherits directly from class `Object`.

28.6 Write a program that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use the `drawOval` method of class `Graphics`.

ANS:

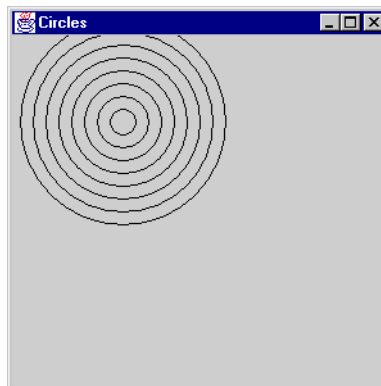
```
1 // Exercise 28.6 Solution
2 // Concentric.java
3 // This program draws concentric circles
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Concentric extends JFrame {
9
10     public Concentric()
11     {
12         super( "Concentric" );
13         setSize( 300, 300 );
14         show();
15     }
16
17     public void paint( Graphics g )
18     {
19         for ( int x = 0; x <= 160; x += 10 ) {
20             int y = 160 - ( x * 2 );
21             g.drawOval( x + 30, x + 30, y, y );
22         }
23     }
24
25     public static void main( String args[] )
26     {
27         Concentric app = new Concentric();
28
29         app.addWindowListener(
30             new WindowAdapter() {
31                 public void windowClosing( WindowEvent e )
32                 {
33                     System.exit( 0 );
34                 }
35             }
36         );
37     }
38 }
```



28.7 Write a program that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use the `drawArc` method.

ANS:

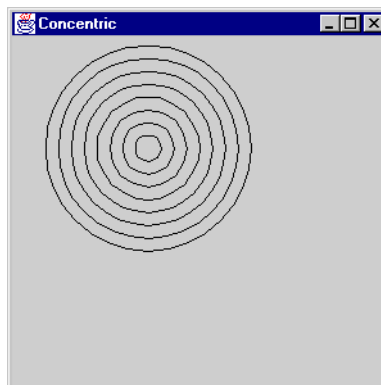
```
1 // Exercise 28.7 Solution
2 // Circles.java
3 // This program draws concentric circles
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Circles extends JFrame {
9
10     public Circles()
11     {
12         super( "Circles" );
13         setSize( 300, 300 );
14         show();
15     }
16
17     public void paint( Graphics g )
18     {
19         for ( int x = 0; x <= 160; x += 10 ) {
20             int y = 160 - ( x * 2 );
21
22             g.drawArc( x + 10, x + 10, y, y, 0, 360 );
23         }
24     }
25     public static void main( String args[] )
26     {
27         Circles app = new Circles();
28
29         app.addWindowListener(
30             new WindowAdapter() {
31                 public void windowClosing( WindowEvent e )
32                 {
33                     System.exit( 0 );
34                 }
35             }
36         );
37     }
38 }
```



28.8 Modify your solution to Exercise 28.6 to draw the ovals using instances of class `Ellipse2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
1 // Exercise 28.8 Solution
2 // Concentric.java
3 // This program draws concentric circles
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.awt.geom.*;
8 import java.awt.image.*;
9
10 public class Concentric extends JFrame {
11
12     public Concentric()
13     {
14         super( "Concentric" );
15         setSize( 300, 300 );
16         show();
17     }
18
19     public void paint( Graphics g )
20     {
21         // Create 2D by casting g to Graphics 2D
22         Graphics2D g2d = ( Graphics2D ) g;
23
24         for ( int x = 0; x <= 160; x += 10 ) {
25             int y = 160 - ( x * 2 );
26             g2d.draw( new Ellipse2D.Double( x + 30, x + 30, y, y ) );
27         }
28     }
29
30     public static void main( String args[] )
31     {
32         Concentric app = new Concentric();
33
34         app.addWindowListener(
35             new WindowAdapter() {
36                 public void windowClosing( WindowEvent e )
37                 {
38                     System.exit( 0 );
39                 }
40             }
41         );
42     }
43 }
```

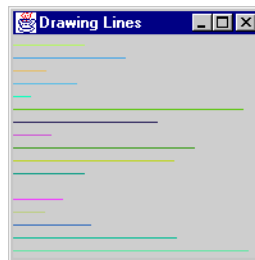


28.9 Write a program that draws lines of random lengths in random colors.

ANS:

```
1 // Exercise 28.9 Solution
2 // Lines1.java
3 // This program draws lines of random sizes and colors
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Lines1 extends JFrame {
9
10     public Lines1()
11     {
12         super( "Drawing Lines" );
13         setSize( 200, 200 );
14         show();
15     }
16
17     public void paint( Graphics g )
18     {
19         for ( int y = 10; y < 200; y += 10 ) {
20             int x1 = ( int ) ( 1 + Math.random() * 199 );
21
22             g.setColor( new Color( ( float ) Math.random(),
23                                   ( float ) Math.random(), ( float ) Math.random() ) );
24             g.drawLine( 1, y, x1, y );
25         }
26     }
27
28     public static void main( String args[] )
29     {
30         Lines1 app = new Lines1();
31
32         app.addWindowListener(
33             new WindowAdapter() {
34                 public void windowClosing( WindowEvent e )
35                 {
36                     System.exit( 0 );
37                 }
38             }
39         );
40     }
41 }
```

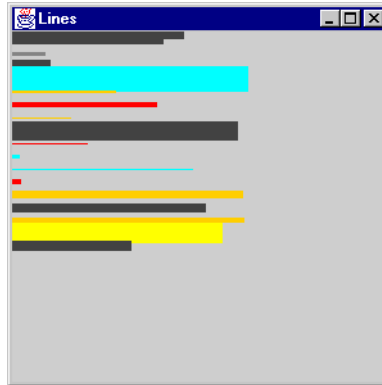
t



28.10 Modify your solution to Exercise 28.9 to draw random lines, in random colors and random line thicknesses. Use class `Line2D.Double` and method `draw` of class `Graphics2D` to draw the lines.

ANS:

```
1 // Exercise 28.10 Solution
2 // Lines.java
3 // This program draws lines of different colors
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.awt.geom.*;
8
9 public class Lines extends JFrame {
10     private Color colors[] = { Color.green, Color.cyan,
11                               Color.black, Color.yellow,
12                               Color.darkGray, Color.red,
13                               Color.orange, Color.gray,
14                               Color.pink, Color.magenta };
15
16     public Lines()
17     {
18         super( "Lines" );
19         setSize( 300, 300 );
20         show();
21     }
22
23     public void paint( Graphics g )
24     {
25         // Create 2D by casting g to Graphics 2D
26         Graphics2D g2d = ( Graphics2D ) g;
27
28         for ( int y = 10; y < 200; y += 10 ) {
29             int color = ( int ) ( Math.random() * 9 );
30
31             g2d.setColor( colors[ color ] );
32             int thickness = ( int ) ( Math.random() * 20 + 1 );
33             g2d.setStroke( new BasicStroke( thickness ) );
34             int x1 = ( int ) ( 1 + Math.random() * 199 );
35             g2d.draw( new Line2D.Double( 1, y, x1, y ) );
36         }
37     }
38
39     public static void main( String args[] )
40     {
41         Lines app = new Lines();
42
43         app.addWindowListener(
44             new WindowAdapter() {
45                 public void windowClosing( WindowEvent e )
46                 {
47                     System.exit( 0 );
48                 }
49             }
50         );
51     }
52 }
```



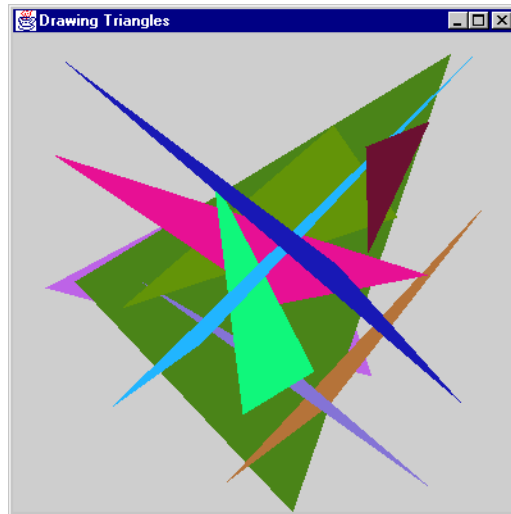
28.11 Write a program that displays randomly generated triangles in different colors. Each triangle should be filled with a different color. Use class `GeneralPath` and method `fill` of class `Graphics2D` to draw the triangles.

ANS:

```

1 // Exercise 28.11 Solution
2 // Triangles.java
3 import javax.swing.*;
4 import java.awt.event.*;
5 import java.awt.*;
6 import java.awt.geom.*;
7
8 public class Triangles extends JFrame {
9
10     public Triangles()
11     {
12         super( "Drawing Triangles" );
13         setSize( 400, 400 );
14         show();
15     }
16
17     public void paint( Graphics g )
18     {
19         GeneralPath triangle = new GeneralPath();
20
21         for ( int i = 0; i < 10; i++ ) {
22             // create a triangle from three random points
23             int x = ( int ) ( Math.random() * 375 + 25 );
24             int y = ( int ) ( Math.random() * 375 + 25 );
25             triangle.moveTo( x, y );
26
27             // second point
28             x = ( int ) ( Math.random() * 375 + 25 );
29             y = ( int ) ( Math.random() * 375 + 25 );
30             triangle.lineTo( x, y );
31
32             // third point
33             x = ( int ) ( Math.random() * 375 + 25 );
34             y = ( int ) ( Math.random() * 375 + 25 );
35             triangle.lineTo( x, y );
36
37             Graphics2D g2d = ( Graphics2D ) g;
38
39             // close the shape
40             triangle.closePath();
41
42             // choose a random color
43             g2d.setColor( new Color( ( int ) ( Math.random() * 256 ),
44                                     ( int ) ( Math.random() * 256 ),
45                                     ( int ) ( Math.random() * 256 ) ) );
46             g2d.fill( triangle );
47             triangle.reset();
48         }
49     }
50
51     public static void main( String args[] )
52     {
53         Triangles app = new Triangles();
54
55         app.addWindowListener(
56             new WindowAdapter() {
57                 public void windowClosing( WindowEvent e )
58                 {
59                     System.exit( 0 );
60                 }
61             }
62         );
63     }
64 }

```



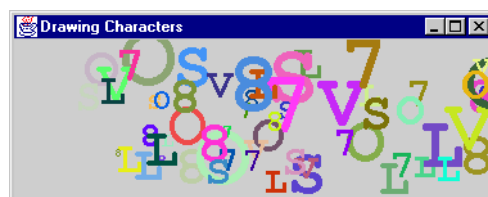
28.12 Write a program that randomly draws characters in different font sizes and colors.

ANS:

```

1 // Exercise 28.12 Solution
2 // Draw.java
3 // This program randomly draws characters
4 // Note: cover, resize, or restart the program
5 // repeatedly to see multiple characters drawn
6 import javax.swing.*;
7 import java.awt.*;
8 import java.awt.event.*;
9
10 public class Draw extends JFrame {
11     private final int DELAY = 4000000;
12
13     public Draw()
14     {
15         super( "Drawing Characters" );
16         setSize( 380, 150 );
17         show();
18     }
19
20     public void paint( Graphics g )
21     {
22         int fontSize = ( int ) ( 10 + Math.random() * 63 );
23         int x = ( int ) ( 30 + Math.random() * 341 );
24         int y = ( int ) ( 50 + Math.random() * 95 );
25         char letters[] = { 'V', 'O', 'L', 'S', '8', '7' };
26         Font f = new Font( "Monospaced", Font.BOLD, fontSize );
27
28         g.setColor( new Color( ( float ) Math.random(),
29                               ( float ) Math.random(),
30                               ( float ) Math.random() ) );
31         g.setFont( f );
32         g.drawChars( letters, ( int ) ( Math.random() * 6 ), 1, x, y );
33
34         for ( int h = 1; h < DELAY; h++ ) ; // slow things down
35         repaint();
36     }
37
38     public static void main( String args[] )
39     {
40         Draw app = new Draw();
41
42         app.addWindowListener(
43             new WindowAdapter() {
44                 public void windowClosing( WindowEvent e )
45                 {
46                     System.exit( 0 );
47                 }
48             }
49         );
50     }
51 }

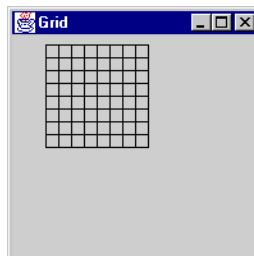
```



28.13 Write a program that draws an 8-by-8 grid. Use the drawLine method.

ANS:

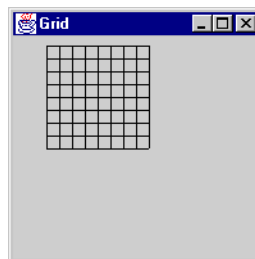
```
1 // Exercise 28.13 Solution
2 // Grid.java
3 // This program draws an 8 x 8 grid
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Grid extends JFrame {
9
10     public Grid()
11     {
12         super( "Grid" );
13         setSize( 200, 200 );
14         show();
15     }
16
17     public void paint( Graphics g )
18     {
19         int y = 30, x1 = 30;
20
21         // 9 lines are required for an 8 x 8 grid
22         for ( int r = 1; r <= 9; r++, y += 10 )
23             g.drawLine( 30, y, 110, y );
24
25         for ( int c = 1; c <= 9; c++, x1 += 10 )
26             g.drawLine( x1, 30, x1, 110 );
27     }
28
29     public static void main( String args[] )
30     {
31         Grid app = new Grid();
32
33         app.addWindowListener(
34             new WindowAdapter() {
35                 public void windowClosing( WindowEvent e )
36                 {
37                     System.exit( 0 );
38                 }
39             }
40         );
41     }
42 }
```



28.14 Modify your solution to Exercise 28.13 to draw the grid using instances of class `Line2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
1 // Exercise 28.14 Solution
2 // Grid.java
3 // This program draws an 8 x 8 grid
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.awt.geom.*;
8
9 public class Grid extends JFrame {
10
11     public Grid()
12     {
13         super( "Grid" );
14         setSize( 200, 200 );
15         show();
16     }
17
18     public void paint( Graphics g )
19     {
20         int y = 30, x1 = 30;
21         Graphics2D g2d = ( Graphics2D ) g;
22
23         // 9 lines are required for an 8 x 8 grid
24         for ( int r = 1; r <= 9; r++, y += 10 )
25             g2d.draw( new Line2D.Double( 30, y, 110, y ) );
26
27         for ( int c = 1; c <= 9; c++, x1 += 10 )
28             g2d.draw( new Line2D.Double( x1, 30, x1, 110 ) );
29     }
30
31     public static void main( String args[] )
32     {
33         Grid app = new Grid();
34
35         app.addWindowListener(
36             new WindowAdapter() {
37                 public void windowClosing( WindowEvent e )
38                 {
39                     System.exit( 0 );
40                 }
41             }
42         );
43     }
44 }
```

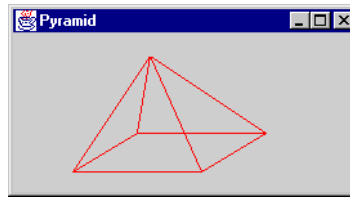


- 28.15** Write a program that draws a 10-by-10 grid. Use the `drawRect` method.
- 28.16** Modify your solution to Exercise 28.15 to draw the grid using instances of class `Rectangle2D.Double` and method `draw` of class `Graphics2D`.
- 28.17** Write a program that draws a tetrahedron (a pyramid). Use class `GeneralPath` and method `draw` of class `Graphics2D`.
ANS:

```

1 // Exercise 28.17 Solution
2 // Pyramid.java
3 // This program draws a tetrahedron
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.geom.*;
7 import java.awt.event.*;
8
9 public class Pyramid extends JFrame {
10
11     public Pyramid()
12     {
13         super( "Pyramid" );
14         setSize( 275, 150 );
15         show();
16     }
17
18     public void paint( Graphics g )
19     {
20         int basex[] = { 100, 200, 150, 50, 100 };
21         int basey[] = { 100, 100, 130, 130, 100 };
22         int x = 110, y = 40;
23
24         Graphics2D g2d = ( Graphics2D ) g;
25
26         GeneralPath tetra = new GeneralPath();
27
28         g2d.setColor( Color.red );
29
30         tetra.moveTo( basex[ 0 ], basey[ 0 ] );
31
32         for ( int i = 1; i < 5; i++ ) {
33             tetra.lineTo( x, y );
34             tetra.moveTo( basex[ i - 1 ], basey[ i - 1 ] );
35             tetra.lineTo( basex[ i ], basey[ i ] );
36         }
37
38         tetra.closePath();
39         g2d.draw( tetra );
40     }
41
42     public static void main( String args[] )
43     {
44         Pyramid app = new Pyramid();
45
46         app.addWindowListener(
47             new WindowAdapter() {
48                 public void windowClosing( WindowEvent e )
49                 {
50                     System.exit( 0 );
51                 }
52             }
53         );
54     }
55 }

```

28.18 Write a program that draws a cube. Use class `GeneralPath` and method `draw` of class `Graphics2D`.

28.19 Write an application that simulates a screen saver. The application should randomly draw lines using method `drawLine` of class `Graphics`. After drawing 100 lines, the application should clear itself and start drawing lines again. To allow the program to draw continuously, place a call to `repaint` as the last line in method `paint`. Do you notice any problems with this on your system?

ANS:

```

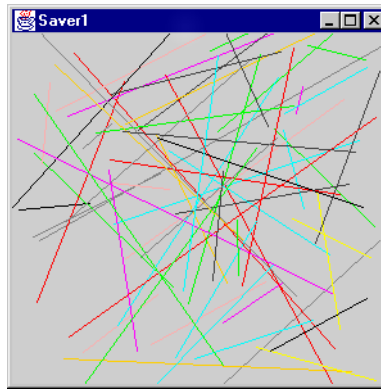
1 // Exercise 28.19 Solution
2 // Saver1.java
3 // Program simulates a simple screen saver
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.awt.geom.*;
8
9 public class Saver1 extends JFrame {
10     private final int DELAY = 4000000;
11     private final int XDIM = 300;
12     private final int YDIM = 300;
13     private int count;
14
15     public Saver1()
16     {
17         super( "Saver1" );
18         setSize( 300, 300 );
19         count = 0;
20         show();
21     }
22
23     public void paint( Graphics g )
24     {
25         int x, y, x1, y1;
26         Color colors[] = { Color.green, Color.cyan,
27                           Color.black, Color.yellow,
28                           Color.darkGray, Color.red,
29                           Color.orange, Color.gray,
30                           Color.pink, Color.magenta };
31
32         // assume html size is 200 x 200
33         x = ( int ) ( Math.random() * XDIM );
34         y = ( int ) ( Math.random() * YDIM );
35         x1 = ( int ) ( Math.random() * XDIM );
36         y1 = ( int ) ( Math.random() * YDIM );
37
38         g.setColor( colors[( int ) ( Math.random() * colors.length )] );
39         g.drawLine( x, y, x1, y1 );
40         ++count;
41
42         // slow the drawing down
43         for ( int q = 1; q < DELAY; q++ )
44             ; // do nothing
45
46         if ( count == 100 ) {
47             g.setColor( Color.white );
48             g.fillRect( 0, 0, XDIM, YDIM );
49             count = 0;
50         }

```

```

51
52     repaint();
53 }
54
55 public static void main( String args[] )
56 {
57     Saver1 app = new Saver1();
58
59     app.addWindowListener(
60         new WindowAdapter() {
61             public void windowClosing( WindowEvent e )
62             {
63                 System.exit( 0 );
64             }
65         }
66     );
67 }
68 }

```



28.20 Here is a peek ahead. Package `javax.swing` contains a class called `Timer` that is capable of calling method `actionPerformed` of interface `ActionListener` at a fixed time interval (specified in milliseconds). Modify your solution to Exercise 28.19 to remove the call to `repaint` from method `paint`. Define your class so it implements `ActionListener` (the `actionPerformed` method should simply call `repaint`). Define an instance variable of type `Timer` called `timer` in your class. In the constructor for your class, write the following statements:

```

timer = new Timer( 1000, this );
timer.start();

```

This creates an instance of class `Timer` that will call `this` object's `actionPerformed` method every 1000 milliseconds (i.e., every second).

28.21 Modify your solution to Exercise 28.20 to enable the user to enter the number of random lines that should be drawn before the application clears itself and starts drawing lines again. Use a `JTextField` to obtain the value. The user should be able to type a new number into the `JTextField` at any time during the program's execution. [Note: Combining Swing GUI components and drawing leads to interesting problems for which we present solutions in Chapter 29]. For now, the first line of your `paint` method should be

```

super.paint( g );

```

to ensure that the GUI components are displayed properly. You will notice that some of the randomly drawn lines will obscure the `JTextField`. Use an inner class definition to perform event handling for the `JTextField`.

28.22 Modify your solution to Exercise 28.20 to randomly choose different shapes to display (use methods of class `Graphics`).]

28.23 Modify your solution to Exercise 28.22 to use classes and drawing capabilities of the Java2D API. For shapes such as rectangles and ellipses, draw them with randomly generated gradients (use class `GradientPaint` to generate the gradient).

28.24 Write a program that uses method `drawPolyline` to draw a spiral.

28.25 Write a program that inputs four numbers and graphs the numbers as a pie chart. Use class `Arc2D.Double` and method `fill` of class `Graphics2D` to perform the drawing. Draw each piece of the pie in a separate color.

28.26 Write an applet that inputs four numbers and graphs the numbers as a bar graph. Use class `Rectangle2D.Double` and method `fill` of class `Graphics2D` to perform the drawing. Draw each bar in a different color.

29

Java Graphical User Interface Components: Solutions

SOLUTIONS

- 29.4** Fill in the blanks in each of the following:
- a) The `JTextField` class inherits directly from _____.
ANS: `JTextComponent`.
 - b) The layout managers discussed in this chapter are _____, _____ and _____.
ANS: `FlowLayout`, `BorderLayout` and `GridLayout`.
 - c) Container method _____ attaches a GUI component to a container.
ANS: `add`.
 - d) Method _____ is called when a mouse button is released (without moving the mouse).
ANS: `mouseClicked`.
- 29.5** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) Only one layout manager can be used per Container.
ANS: True.
 - b) GUI components can be added to a Container in any order in a `BorderLayout`.
ANS: True.
 - c) Graphics method `setFont` is used to set the font for text fields.
ANS: False. Component method `setFont` is used.
 - d) A `Mouse` object contains a method called `mouseDragged`.
ANS: False. A `Mouse` object is not provided by Java.
- 29.6** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) A `JApplet` does not have a content pane.
ANS: False. A `JApplet` does have a content pane.
 - b) A `JPanel` is a `JComponent`.
ANS: True.
 - c) A `JPanel` is a `Component`.
ANS: True.
 - d) A `JLabel` is a `Container`.
ANS: True.
 - e) An `AbstractButton` is a `JButton`.
ANS: False. A `JButton` is an `AbstractButton`.
 - f) A `JTextField` is an `Object`.
ANS: True.

29.7 Find any error(s) in each of the following and explain how to correct it (them).

a) `import javax.swing.* // include swing package`

ANS: Semicolon is missing after the asterick.

b) `panelObject.GridLayout(8, 8); // set GridLayout`

ANS: The GridLayout constructor cannot be used in this manner. The correct statement should be:

`panelObject.getContentPane().setLayout(new GridLayout(8, 8));`

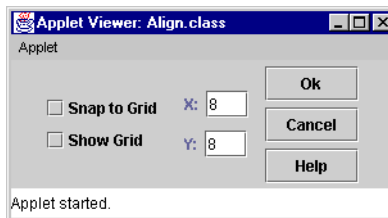
c) `c.setLayout(new FlowLayout(FlowLayout.DEFAULT));`

ANS: Class FlowLayout does not contain static constant DEFAULT.

d) `c.add(eastButton, EAST); // BorderLayout`

ANS: EAST should be BorderLayout.EAST.

29.8 Create the following GUI. You do not have to provide any functionality.



ANS:

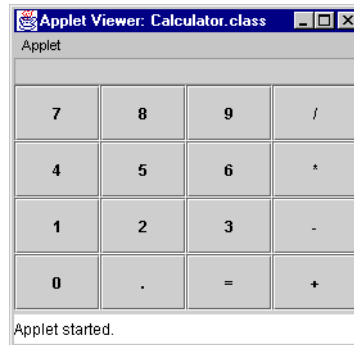
```

1 // Exercise 29.8 Solution
2 // Align.java
3 // This program creates a simple GUI
4 import javax.swing.*;
5 import java.awt.*;
6
7 public class Align extends JApplet {
8     private JButton ok, cancel, help;
9     private JTextField xValue, yValue;
10    private JCheckBox snap, show;
11    private JLabel xLabel, yLabel;
12    private JPanel checkPanel, buttonPanel,
13                fieldPanel1, fieldPanel2,
14                fieldPanel;
15
16    public void init()
17    {
18        // build checkPanel
19        snap = new JCheckBox( "Snap to Grid" );
20        show = new JCheckBox( "Show Grid" );
21        checkPanel = new JPanel();
22        checkPanel.setLayout( new GridLayout( 2, 1 ) );
23        checkPanel.add( snap );
24        checkPanel.add( show );
25
26        // build field panel1
27        xLabel = new JLabel( "X: " );
28        xValue = new JTextField( "8", 3 );
29        fieldPanel1 = new JPanel();
30        fieldPanel1.setLayout( new FlowLayout( FlowLayout.CENTER, 3, 5 ) );
31        fieldPanel1.add( xLabel );
32        fieldPanel1.add( xValue );
33
34        yLabel = new JLabel( "Y: " );
35        yValue = new JTextField( "8", 3 );
36        fieldPanel2 = new JPanel();
37        fieldPanel2.setLayout( new FlowLayout( FlowLayout.CENTER, 3, 5 ) );
38        fieldPanel2.add( yLabel );
39        fieldPanel2.add( yValue );
40
41        fieldPanel = new JPanel();
42        fieldPanel.setLayout( new BorderLayout() );

```

```
43     fieldPanel.add( fieldPanel1, BorderLayout.NORTH );
44     fieldPanel.add( fieldPanel2, BorderLayout.SOUTH );
45
46     // build button panel
47     ok = new JButton( "Ok" );
48     cancel = new JButton( "Cancel" );
49     help = new JButton( "Help" );
50     buttonPanel = new JPanel();
51     buttonPanel.setLayout( new GridLayout( 3, 1, 10, 5 ) );
52     buttonPanel.add( ok );
53     buttonPanel.add( cancel );
54     buttonPanel.add( help );
55
56     // set layout for applet
57     getContentPane().setLayout(
58         new FlowLayout( FlowLayout.CENTER, 10, 5 ) );
59     getContentPane().add( checkPanel );
60     getContentPane().add( fieldPanel );
61     getContentPane().add( buttonPanel );
62 }
63 }
```

29.9 Create the following GUI. You do not have to provide any functionality.



ANS:

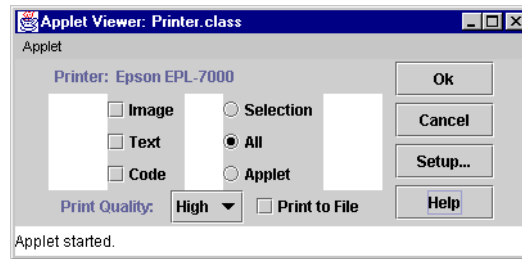
```

1 // Solution exercise 29.9
2 // Calculator.java
3 // This program creates a simple GUI
4 // html: width = 270 height = 200
5 import javax.swing.*;
6 import java.awt.*;
7
8 public class Calculator extends JApplet {
9     private JButton keys[];
10    private JPanel keyPad;
11    private JTextField lcd;
12
13    public void init()
14    {
15        lcd = new JTextField( 20 );
16        keyPad = new JPanel();
17        keys = new JButton[ 16 ];
18
19        lcd.setEditable( false );
20
21        for ( int i = 0; i <= 9; i++ )
22            keys[ i ] = new JButton( String.valueOf( i ) );
23
24        keys[ 10 ] = new JButton( "/" );
25        keys[ 11 ] = new JButton( "*" );
26        keys[ 12 ] = new JButton( "-" );
27        keys[ 13 ] = new JButton( "+" );
28        keys[ 14 ] = new JButton( "=" );
29        keys[ 15 ] = new JButton( "." );
30
31        // set keyPad layout to grid layout
32        keyPad.setLayout( new GridLayout( 4, 4 ) );
33
34        for ( int i = 7; i <= 10; i++ ) // 7, 8, 9, 10
35            keyPad.add( keys[ i ] ); // divide
36
37        for ( int i = 4; i <= 6; i++ ) // 4, 5, 6
38            keyPad.add( keys[ i ] );
39
40        keyPad.add( keys[ 11 ] ); // multiply
41
42        for ( int i = 1; i <= 3; i++ ) // 1, 2, 3
43            keyPad.add( keys[ i ] );
44
45        keyPad.add( keys[ 12 ] ); // subtract
46
47        keyPad.add( keys[ 0 ] ); // 0
48
49        for ( int i = 15; i >= 13; i-- )
50            keyPad.add( keys[ i ] ); // ., =, add

```

```
51
52 // set applet layout to border layout
53 getContentPane().setLayout( new BorderLayout() );
54 getContentPane().add( lcd, BorderLayout.NORTH );
55 getContentPane().add( keypad, BorderLayout.CENTER );
56     }
57 }
```

29.10 Create the following GUI. You do not have to provide any functionality.



ANS:

```

1 // Exercise 29.10 Solution
2 // Printer.java
3 // This program creates a simple GUI
4 // html: width = 400 height = 130
5 import javax.swing.*;
6 import java.awt.*;
7
8 public class Printer extends JApplet {
9     private JButton b1, b2, b3, b4;
10    private JCheckBox c1, c2, c3, c4;
11    private JRadioButton rb1, rb2, rb3;
12    private ButtonGroup radioGroup;
13    private JComboBox q;
14    private JLabel label1, label2;
15    private JPanel p1, p2, p3, p4, p5, p6, p7, p8;
16
17    public void init()
18    {
19        // build left north panel
20        label1 = new JLabel( "Printer: Epson EPL-7000" );
21        p1 = new JPanel();
22        p1.setLayout( new FlowLayout( FlowLayout.LEFT ) );
23        p1.add( label1 );
24
25        // build right east panel
26        b1 = new JButton( "Ok" );
27        b2 = new JButton( "Cancel" );
28        b3 = new JButton( "Setup..." );
29        b4 = new JButton( "Help" );
30        p2 = new JPanel();
31        p2.setLayout( new GridLayout( 4, 1, 5, 5 ) );
32        p2.add( b1 );
33        p2.add( b2 );
34        p2.add( b3 );
35        p2.add( b4 );
36
37        // build left south panel
38        label2 = new JLabel( "Print Quality: " );
39        q = new JComboBox();
40        q.addItem( "High" );
41        c1 = new JCheckBox( "Print to File" );
42        p3 = new JPanel();
43        p3.setLayout( new FlowLayout( FlowLayout.CENTER, 10, 0 ) );
44        p3.add( label2 );
45        p3.add( q );
46        p3.add( c1 );
47
48        // build left east panel
49        c2 = new JCheckBox( "Image" );
50        c3 = new JCheckBox( "Text" );
51        c4 = new JCheckBox( "Code" );
52        p4 = new JPanel();

```

```
53     p4.setLayout( new BorderLayout( ) );
54     p4.add( c2, BorderLayout.NORTH );
55     p4.add( c3, BorderLayout.CENTER );
56     p4.add( c4, BorderLayout.SOUTH );
57
58     // build left west panel
59     p5 = new JPanel();
60     p5.setLayout( new BorderLayout() );
61     p5.add( rb1 = new JRadioButton( "Selection", false ),
62           BorderLayout.NORTH );
63     p5.add( rb2 = new JRadioButton( "All", true ),
64           BorderLayout.CENTER );
65     p5.add( rb3 = new JRadioButton( "Applet", false ),
66           BorderLayout.SOUTH );
67     // Group the radio buttons
68     radioGroup=new ButtonGroup();
69     radioGroup.add( rb1 );
70     radioGroup.add( rb2 );
71     radioGroup.add( rb3 );
72
73     // build left center
74     p8 = new JPanel();
75     p8.setLayout( new FlowLayout( FlowLayout.CENTER, 30, 0 ) );
76     p8.setBackground( Color.white );
77     p8.add( p4 );
78     p8.add( p5 );
79
80     // setup left panel
81     p6 = new JPanel();
82     p6.setLayout( new BorderLayout() );
83     p6.add( p1, BorderLayout.NORTH );
84     p6.add( p8, BorderLayout.CENTER );
85     p6.add( p3, BorderLayout.SOUTH );
86
87     // setup applet layout
88     p7 = new JPanel();
89     p7.setLayout( new FlowLayout( FlowLayout.CENTER, 10, 0 ) );
90     p7.add( p6 );
91     p7.add( p2 );
92
93     getContentPane().add( p7 );
94 }
95 }
```

29.11 Write a temperature conversion program that converts from Fahrenheit to Celsius. The Fahrenheit temperature should be entered from the keyboard (via a `JTextField`). A `JLabel` should be used to display the converted temperature. Use the following formula for the conversion:

$$\text{Celsius} = 5 / 9 \times (\text{Fahrenheit} - 32)$$

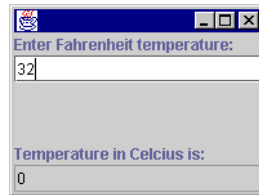
ANS:

```

1 // Exercise 29.11 Solution
2 // Convert.java
3 // Temperature conversion program
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Convert extends JFrame {
9     private JPanel p;
10    private JLabel label1, label2;
11    private JTextField temperatureF;
12    private JTextField temperatureC;
13
14    public Convert()
15    {
16        label1 = new JLabel( "Enter Fahrenheit temperature:" );
17        label2 = new JLabel( "Temperature in Celsius is:" );
18        temperatureF = new JTextField( 10 );
19        temperatureF.addActionListener(
20            new ActionListener() {
21                public void actionPerformed( ActionEvent e )
22                {
23                    int celcius, temp;
24
25                    temp = Integer.parseInt( temperatureF.getText() );
26                    celcius = ( int ) ( 5.0f / 9.0f * ( temp - 32 ) );
27                    temperatureC.setText( String.valueOf( celcius ) );
28                }
29            }
30        );
31
32        temperatureC = new JTextField( 10 );
33        temperatureC.setEditable( false );
34
35        p = new JPanel();
36        p.setLayout( new BorderLayout() );
37        p.add( temperatureF, BorderLayout.NORTH );
38        p.add( label2, BorderLayout.SOUTH );
39
40        Container c = getContentPane();
41        c.setLayout( new BorderLayout() );
42        c.add( label1, BorderLayout.NORTH );
43        c.add( p, BorderLayout.CENTER );
44        c.add( temperatureC, BorderLayout.SOUTH );
45        setSize( 200, 150 );
46        show();
47    }
48
49    public static void main ( String args[] )
50    {
51        Convert app = new Convert();
52
53        app.addWindowListener(
54            new WindowAdapter() {
55                public void windowClosing( WindowEvent e )
56                {
57                    System.exit( 0 );
58                }
59            }
60        );

```

```
61     }  
62 }
```



29.12 Write an application that allows the user to draw a rectangle by dragging the mouse on the application window. The upper-left coordinate should be the location where the user presses the mouse button, and the lower-right coordinate should be the location where the user releases the mouse button. Also display the area of the rectangle in a `JLabel` in the `SOUTH` region of a `BorderLayout` out. All drawing should be done on a subclass of `JPanel`. Use the following formula for the area:

$$\text{area} = \text{width} \times \text{height}$$

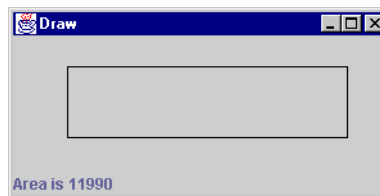
ANS:

```

1 // Exercise 29.12 Solution
2 // Draw.java
3 // Program draws a rectangle with the mouse
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Draw extends JFrame {
9     private int topX, topY;
10    private int width, height;
11    private int bottomX, bottomY;
12    protected JLabel status;
13
14    public Draw()
15    {
16        super( "Draw" );
17        topX = topY = 0;
18        addMouseListener( new MouseHandler( this ) );
19
20        status = new JLabel();
21        getContentPane().add( status, BorderLayout.SOUTH );
22        setSize( 300, 150 );
23        show();
24    }
25
26    public int getTopX() { return topX; }
27    public int getTopY() { return topY; }
28    public int getWidth() { return width; }
29    public int getHeight() { return height; }
30    public int getBottomX() { return bottomX; }
31    public int getBottomY() { return bottomY; }
32    public void setTopX( int x ) { topX = x; }
33    public void setTopY( int y ) { topY = y; }
34    public void setBottomX( int x ) { bottomX = x; }
35    public void setBottomY( int y ) { bottomY = y; }
36    public void setWidth( int w ) { width = w; }
37    public void setHeight( int h ) { height = h; }
38
39    public void paint( Graphics g )
40    {
41        super.paint( g );
42
43        g.drawRect( topX, topY, width, height );
44    }
45
46    public static void main( String args[] )
47    {
48        Draw app = new Draw();
49
50        app.addWindowListener(
51            new WindowAdapter() {
52                public void windowClosing( WindowEvent e )
53                {
54                    System.exit( 0 );
55                }
56            }
57        );
58    }
59 }

```

```
60
61 class MouseHandler extends MouseAdapter {
62     private Draw draw;
63
64     public MouseHandler( Draw d ) { draw = d; }
65
66     public void mouseReleased( MouseEvent e )
67     {
68         draw.setBottomX( e.getX() );
69         draw.setBottomY( e.getY() );
70         draw.setWidth( Math.abs( draw.getTopX() - draw.getBottomX() ) );
71         draw.setHeight( Math.abs( draw.getTopY() - draw.getBottomY() ) );
72         draw.setTopX( Math.min( draw.getTopX(), draw.getBottomX() ) );
73         draw.setTopY( Math.min( draw.getTopY(), draw.getBottomY() ) );
74         draw.status.setText( "Area is " + ( draw.getWidth() * draw.getHeight() ) );
75         draw.repaint();
76     }
77
78     public void mousePressed( MouseEvent e )
79     {
80         draw.setTopX( e.getX() );
81         draw.setTopY( e.getY() );
82     }
83 }
```



29.13 Write a program that displays a circle of random size and calculates and displays the area, radius, diameter and circumference. Use the following equations: $diameter = 2 \times radius$, $area = \pi \times radius^2$, $circumference = 2 \times \pi \times radius$. Use the constant `Math.PI` for pi (π). All drawing should be done on a subclass of `JPanel` and the results of the calculations should be displayed in a read-only `JTextArea`.

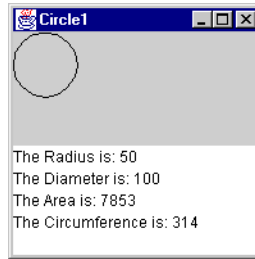
ANS:

```

1 // Exercise 29.13 Solution
2 // Circle1.java
3 // Program draws a circle of a random
4 // diameter and displays the area, diameter,
5 // and circumference.
6 import javax.swing.*;
7 import java.awt.*;
8 import java.awt.event.*;
9
10 public class Circle1 extends JFrame {
11     private CircleCanvas theCanvas;
12     private JTextArea display;
13
14     public Circle1()
15     {
16         super( "Circle1" );
17         theCanvas = new CircleCanvas();
18         display = new JTextArea( 5, 30 );
19
20         display.setText( "The Radius is: " + theCanvas.getRadius() +
21             "\nThe Diameter is: " + theCanvas.getDiameter()
22             + "\nThe Area is: " + theCanvas.getArea() +
23             "\nThe Circumference is: " +
24             theCanvas.getCircumference() );
25
26         getContentPane().add( theCanvas, BorderLayout.CENTER );
27         getContentPane().add( display, BorderLayout.SOUTH );
28         setSize( 200, 200 );
29         show();
30     }
31
32     public static void main( String args[] )
33     {
34         Circle1 app = new Circle1();
35
36         app.addWindowListener(
37             new WindowAdapter() {
38                 public void windowClosing( WindowEvent e )
39                 {
40                     System.exit( 0 );
41                 }
42             }
43         );
44     }
45 }
46
47 class CircleCanvas extends JPanel {
48     private int radius;
49
50     public CircleCanvas()
51     {
52         radius = ( int )( 1 + Math.random() * 100 );
53         setSize( 100, 100 );
54     }
55
56     public void paintComponent( Graphics g )
57     { g.drawOval( 0, 0, radius, radius ); }
58
59     public int getDiameter() { return ( 2 * radius ); }
60

```

```
61     public int getCircumference()
62     { return ( int )( 2 * Math.PI * radius ); }
63
64     public int getArea()
65     { return ( int )( radius * radius * Math.PI ); }
66
67     public int getRadius() { return radius; }
68 }
```



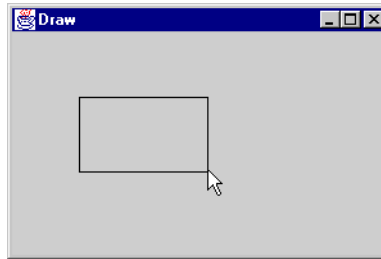
29.14 Write a program that uses `System.out.println` statements to print out events as they occur. Provide a `JComboBox` with a minimum of four items. The user should be able to choose an event to “monitor” from the `JComboBox`. When that particular event occurs, display information about the event in a message dialog box. Use method `toString` on the event object to convert it to a string representation.

29.15 Write a program using methods from interface `MouseListener` that allows the user to press the mouse button, drag the mouse and release the mouse button. When the mouse is released, draw a rectangle with the appropriate upper-left corner, width and height. (*Hint:* The `mousePressed` method should capture the set of coordinates at which the user presses and holds the mouse button initially, and the `mouseReleased` method should capture the set of coordinates at which the user releases the mouse button. Both methods should store the appropriate coordinate values. All drawing should be done on a subclass of `JPanel` and all calculations of the width, height and upper-left corner should be performed by the `paintComponent` method before the shape is drawn).

ANS:

```
1 // Exercise 29.15 Solution
2 // Draw.java
3 // Program draws a rectangle with the mouse
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Draw extends JFrame {
9     private int topX, topY;
10    private int width, height, upperX, upperY;
11    private int bottomX, bottomY;
12
13    public Draw()
14    {
15        super( "Draw" );
16        addMouseListener( new MouseHandler() );
17        setSize( 300, 200 );
18        show();
19    }
20
21    public void setTopX( int x ) { topX = x; }
22    public void setTopY( int y ) { topY = y; }
23    public void setBottomX( int x ) { bottomX = x; }
24    public void setBottomY( int y ) { bottomY = y; }
25
26    public void paint( Graphics g )
27    {
28        super.paint( g );
29
30        width = Math.abs( topX - bottomX );
31        height = Math.abs( topY - bottomY );
32        upperX = Math.min( topX, bottomX );
33        upperY = Math.min( topY, bottomY );
34
35        g.drawRect( upperX, upperY, width, height );
36    }
37
38    public static void main( String args[] )
39    {
40        Draw app = new Draw();
41
42        app.addWindowListener(
43            new WindowAdapter() {
44                public void windowClosing( WindowEvent e )
45                {
46                    System.exit( 0 );
47                }
48            }
49        );
50    }
51
52    private class MouseHandler extends MouseAdapter {
53        public void mouseReleased( MouseEvent e )
54        {
55            setBottomX( e.getX() );
56            setBottomY( e.getY() );
57            repaint();
58        }
59    }
```

```
59  
60     public void mousePressed( MouseEvent e )  
61     {  
62         setTopX( e.getX() );  
63         setTopY( e.getY() );  
64     }  
65 }  
66 }
```



29.16 Modify Exercise 29.15 to provide a “rubber-banding” effect. As the user drags the mouse, the user should be able to see the current size of the rectangle to know exactly what the rectangle will look like when the mouse button is released. (*Hint*: Method `mouseDragged` should perform the same tasks as `mouseReleased`).

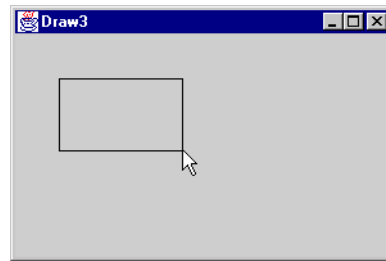
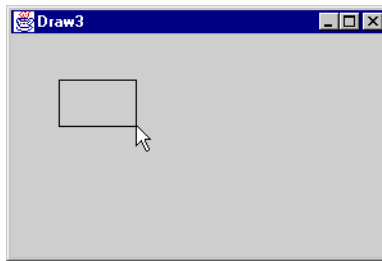
ANS:

```

1 // Exercise 29.16 Solution
2 // Draw3.java
3 // Program draws a rectangle with the mouse
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Draw3 extends JFrame {
9     private int topX, topY;
10    private int width, height, upperX, upperY;
11    private int bottomX, bottomY;
12
13    public Draw3()
14    {
15        super( "Draw3" );
16        addMouseListener( new MouseHandler() );
17        addMouseMotionListener( new MouseMotionHandler() );
18        setSize( 300, 200 );
19        show();
20    }
21
22    public void setTopX( int x ) { topX = x; }
23    public void setTopY( int y ) { topY = y; }
24    public void setBottomX( int x ) { bottomX = x; }
25    public void setBottomY( int y ) { bottomY = y; }
26
27    public void paint( Graphics g )
28    {
29        super.paint( g );
30
31        width = Math.abs( topX - bottomX );
32        height = Math.abs( topY - bottomY );
33        upperX = Math.min( topX, bottomX );
34        upperY = Math.min( topY, bottomY );
35
36        g.drawRect( upperX, upperY, width, height );
37    }
38
39    public static void main( String args[] )
40    {
41        Draw3 app = new Draw3();
42
43        app.addWindowListener(
44            new WindowAdapter() {
45                public void windowClosing( WindowEvent e )
46                {
47                    System.exit( 0 );
48                }
49            }
50        );
51    }
52
53    private class MouseHandler extends MouseAdapter {
54        public void mouseReleased( MouseEvent e )
55        {
56            setBottomX( e.getX() );
57            setBottomY( e.getY() );
58            repaint();
59        }
60
61        public void mousePressed( MouseEvent e )
62        {

```

```
63         setTopX( e.getX() );
64         setTopY( e.getY() );
65     }
66 }
67
68 private class MouseMotionHandler extends MouseMotionAdapter {
69     public void mouseDragged( MouseEvent e )
70     {
71         setBottomX( e.getX() );
72         setBottomY( e.getY() );
73         repaint();
74     }
75 }
76 }
```



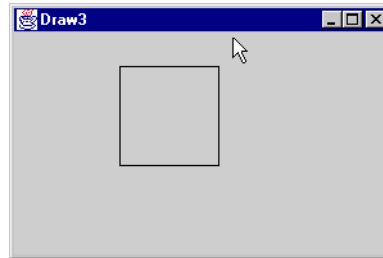
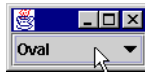
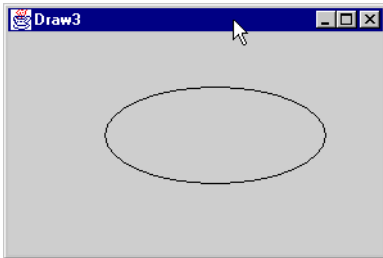
29.17 Modify Exercise 29.16 to allow the user to select which shape to draw. A JComboBox should provide options including at least rectangle, oval, line and rounded rectangle.

ANS:

```
1 // Exercise 29.17 Solution
2 // Draw3.java
3 // Program draws a rectangle with the mouse
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Draw3 extends JFrame {
9     private int topX, topY;
10    private int width, height, upperX, upperY;
11    private int bottomX, bottomY;
12    private final int CIRCLE = 0;
13    private final int SQUARE = 1;
14    private final int OVAL = 2;
15    private final int RECTANGLE = 3;
16    private JComboBox choice;
17    private int shape;
18    private ToolWindow tools;
19
20    public Draw3()
21    {
22        super( "Draw3" );
23        addMouseListener( new MouseHandler() );
24        addMouseMotionListener( new MouseMotionHandler() );
25
26        // set default shape to Circle
27        shape = CIRCLE;
28
29        setSize( 300, 200 );
30        show();
31        tools = new ToolWindow();
32    }
33
34    public void setTopX( int x ) { topX = x; }
35    public void setTopY( int y ) { topY = y; }
36    public void setBottomX( int x ) { bottomX = x; }
37    public void setBottomY( int y ) { bottomY = y; }
38    public void setShape( int s ) { shape = s; }
39
40    public void paint( Graphics g )
41    {
42        super.paint( g );
43
44        width = Math.abs( topX - bottomX );
45        height = Math.abs( topY - bottomY );
46        upperX = Math.min( topX, bottomX );
47        upperY = Math.min( topY, bottomY );
48
49        switch ( shape ) {
50            case CIRCLE:
51                g.drawOval( upperX, upperY, width, width );
52                break;
53            case SQUARE:
54                g.drawRect( upperX, upperY, width, width );
55                break;
56            case OVAL:
57                g.drawOval( upperX, upperY, width, height );
58                break;
59            case RECTANGLE:
60                g.drawRect( upperX, upperY, width, height );
61                break;
62        }
63    }
}
```

```
64
65 public static void main( String args[] )
66 {
67     Draw3 app = new Draw3();
68
69     app.addWindowListener(
70         new WindowAdapter() {
71             public void windowClosing( WindowEvent e )
72             {
73                 System.exit( 0 );
74             }
75         }
76     );
77 }
78
79 private class ToolWindow extends JFrame {
80     public ToolWindow()
81     {
82         choice = new JComboBox();
83         choice.addItem( "Circle" );
84         choice.addItem( "Square" );
85         choice.addItem( "Oval" );
86         choice.addItem( "Rectangle" );
87
88         choice.addItemListener(
89             new ItemListener() {
90                 public void itemStateChanged( ItemEvent e )
91                 {
92                     setShape( choice.getSelectedIndex() );
93                     repaint();
94                 }
95             }
96         );
97
98         // set default shape to Circle
99         shape = CIRCLE;
100
101         Container c = getContentPane();
102         c.setLayout( new BorderLayout() );
103         c.add( choice, BorderLayout.SOUTH );
104
105         pack();
106         setLocation( 300, 0 );
107         show();
108     }
109 }
110
111 private class MouseHandler extends MouseAdapter {
112     public void mouseReleased( MouseEvent e )
113     {
114         setBottomX( e.getX() );
115         setBottomY( e.getY() );
116         repaint();
117     }
118
119     public void mousePressed( MouseEvent e )
120     {
121         setTopX( e.getX() );
122         setTopY( e.getY() );
123     }
124 }
125
126 private class MouseMotionHandler extends MouseMotionAdapter {
127     public void mouseDragged( MouseEvent e )
128     {
129         setBottomX( e.getX() );
130         setBottomY( e.getY() );
131         repaint();
132     }
133 }
```

```
133     }  
134 }
```



29.18 Modify Exercise 29.17 to allow the user to select the drawing color from a JColorChooser dialog box.

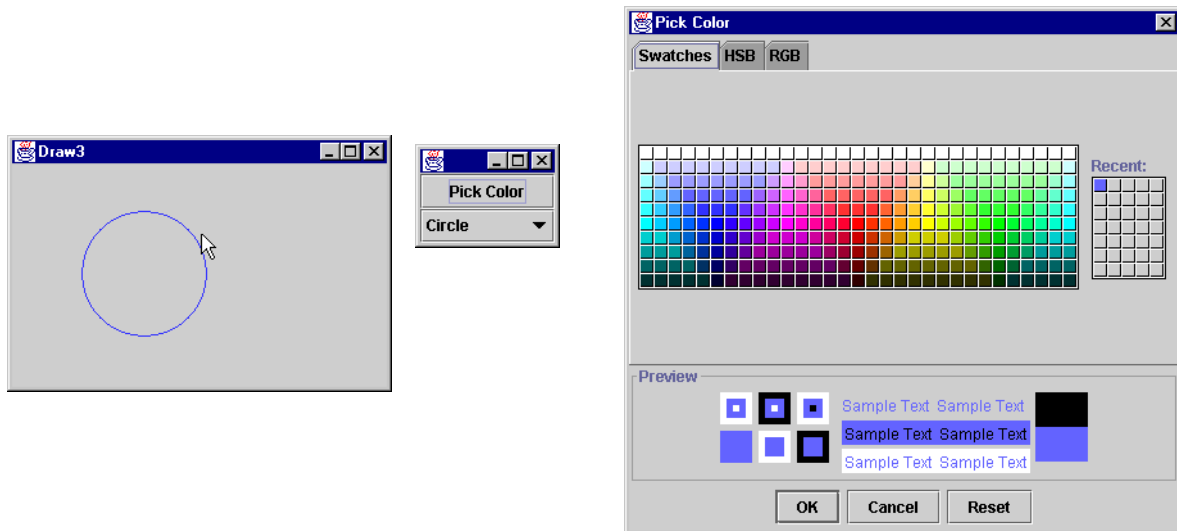
ANS:

```
1 // Exercise 29.18 Solution
2 // Draw3.java
3 // Program draws a rectangle with the mouse
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Draw3 extends JFrame {
9     private int topX, topY;
10    private int width, height, upperX, upperY;
11    private int bottomX, bottomY;
12    private final int CIRCLE = 0;
13    private final int SQUARE = 1;
14    private final int OVAL = 2;
15    private final int RECTANGLE = 3;
16    private JComboBox choice;
17    private int shape;
18    private ToolWindow tools;
19    private JButton chooseColor;
20    private Color color;
21
22    public Draw3()
23    {
24        super( "Draw3" );
25        addMouseListener( new MouseHandler() );
26        addMouseMotionListener( new MouseMotionHandler() );
27
28        // set default shape to Circle
29        shape = CIRCLE;
30
31        setSize( 300, 200 );
32        show();
33        tools = new ToolWindow();
34    }
35
36    public void setTopX( int x ) { topX = x; }
37    public void setTopY( int y ) { topY = y; }
38    public void setBottomX( int x ) { bottomX = x; }
39    public void setBottomY( int y ) { bottomY = y; }
40    public void setShape( int s ) { shape = s; }
41
42    public void paint( Graphics g )
43    {
44        super.paint( g );
45
46        g.setColor( color );
47
48        width = Math.abs( topX - bottomX );
49        height = Math.abs( topY - bottomY );
50        upperX = Math.min( topX, bottomX );
51        upperY = Math.min( topY, bottomY );
52
53        switch ( shape ) {
54            case CIRCLE:
55                g.drawOval( upperX, upperY, width, width );
56                break;
57            case SQUARE:
58                g.drawRect( upperX, upperY, width, width );
59                break;
60            case OVAL:
61                g.drawOval( upperX, upperY, width, height );
62                break;
63            case RECTANGLE:
64                g.drawRect( upperX, upperY, width, height );
65                break;
```



```
66     }
67 }
68
69 public static void main( String args[] )
70 {
71     Draw3 app = new Draw3();
72
73     app.addWindowListener(
74         new WindowAdapter() {
75             public void windowClosing( WindowEvent e )
76             {
77                 System.exit( 0 );
78             }
79         }
80     );
81 }
82
83 private class ToolWindow extends JFrame {
84     public ToolWindow()
85     {
86         choice = new JComboBox();
87         choice.addItem( "Circle" );
88         choice.addItem( "Square" );
89         choice.addItem( "Oval" );
90         choice.addItem( "Rectangle" );
91
92         choice.addItemListener(
93             new ItemListener() {
94                 public void itemStateChanged( ItemEvent e )
95                 {
96                     setShape( choice.getSelectedIndex() );
97                     repaint();
98                 }
99             }
100        );
101
102        Container c = getContentPane();
103        c.add( choice, BorderLayout.SOUTH );
104
105        chooseColor = new JButton( "Pick Color" );
106        chooseColor.addActionListener(
107            new ActionListener() {
108                public void actionPerformed( ActionEvent e )
109                {
110                    color = JColorChooser.showDialog( null, "Pick Color", Color.black );
111                }
112            }
113        );
114        c.add( chooseColor, BorderLayout.NORTH );
115
116        pack();
117        setLocation( 300, 0 );
118        show();
119    }
120 }
121
122 private class MouseHandler extends MouseAdapter {
123     public void mouseReleased( MouseEvent e )
124     {
125         setBottomX( e.getX() );
126         setBottomY( e.getY() );
127         repaint();
128     }
129
130     public void mousePressed( MouseEvent e )
131     {
132         setTopX( e.getX() );
133         setTopY( e.getY() );
134     }
135 }
```

```
135     }  
136  
137     private class MouseMotionHandler extends MouseMotionAdapter {  
138         public void mouseDragged( MouseEvent e )  
139         {  
140             setBottomX( e.getX() );  
141             setBottomY( e.getY() );  
142             repaint();  
143         }  
144     }  
145 }
```



29.19 Modify Exercise 29.18 to allow the user to specify if a shape should be filled or empty when it is drawn. The user should click a JCheckBox to indicate filled or empty.

ANS:

```

1 // Exercise 29.19 Solution
2 // Draw3.java
3 // Program draws a rectangle with the mouse
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Draw3 extends JFrame {
9     private int topX, topY;
10    private int width, height, upperX, upperY;
11    private int bottomX, bottomY;
12    private final int CIRCLE = 0;
13    private final int SQUARE = 1;
14    private final int OVAL = 2;
15    private final int RECTANGLE = 3;
16    private JComboBox choice;
17    private int shape;
18    private ToolWindow tools;
19    private JButton chooseColor;
20    private Color color;
21    private JCheckBox filled;
22
23    public Draw3()
24    {
25        super( "Draw3" );
26        addMouseListener( new MouseHandler() );
27        addMouseMotionListener( new MouseMotionHandler() );
28
29        // set default shape to Circle
30        shape = CIRCLE;
31
32        tools = new ToolWindow();
33        setSize( 300, 200 );
34        show();
35    }
36
37    public void setTopX( int x ) { topX = x; }
38    public void setTopY( int y ) { topY = y; }
39    public void setBottomX( int x ) { bottomX = x; }
40    public void setBottomY( int y ) { bottomY = y; }
41    public void setShape( int s ) { shape = s; }
42
43    public void paint( Graphics g )
44    {
45        super.paint( g );
46
47        g.setColor( color );
48
49        width = Math.abs( topX - bottomX );
50        height = Math.abs( topY - bottomY );
51        upperX = Math.min( topX, bottomX );
52        upperY = Math.min( topY, bottomY );
53
54        if ( filled.isSelected() )
55            switch ( shape ) {
56                case CIRCLE:
57                    g.fillOval( upperX, upperY, width, width );
58                    break;
59                case SQUARE:
60                    g.fillRect( upperX, upperY, width, width );
61                    break;
62                case OVAL:
63                    g.fillOval( upperX, upperY, width, height );
64                    break;

```

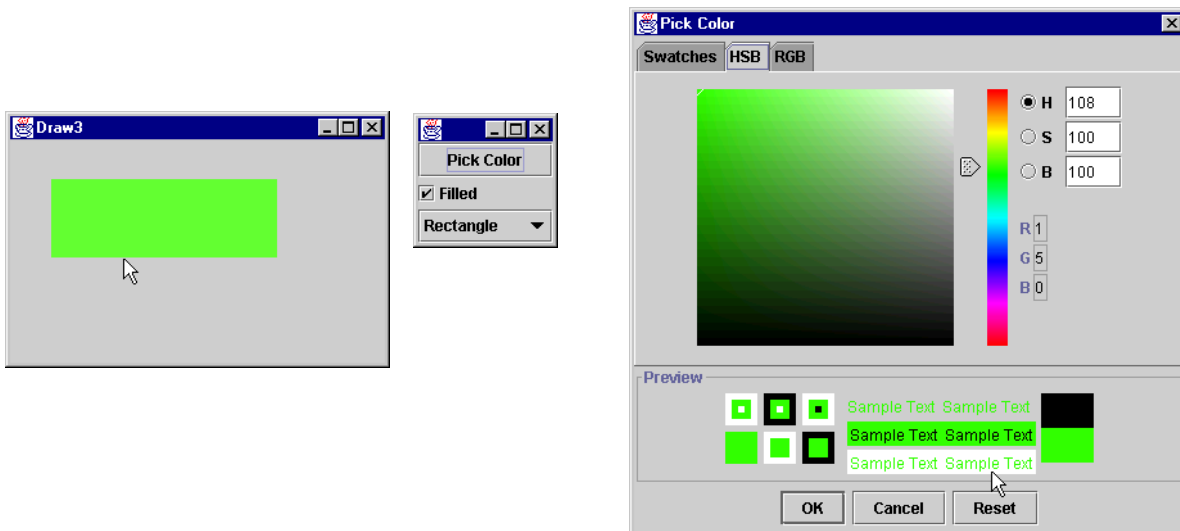
```
65         case RECTANGLE:
66             g.fillRect( upperX, upperY, width, height );
67             break;
68     }
69     else
70     switch ( shape ) {
71         case CIRCLE:
72             g.drawOval( upperX, upperY, width, width );
73             break;
74         case SQUARE:
75             g.drawRect( upperX, upperY, width, width );
76             break;
77         case OVAL:
78             g.drawOval( upperX, upperY, width, height );
79             break;
80         case RECTANGLE:
81             g.drawRect( upperX, upperY, width, height );
82             break;
83     }
84 }
85
86 public static void main( String args[] )
87 {
88     Draw3 app = new Draw3();
89
90     app.addWindowListener(
91         new WindowAdapter() {
92             public void windowClosing( WindowEvent e )
93             {
94                 System.exit( 0 );
95             }
96         }
97     );
98 }
99
100 private class ToolWindow extends JFrame {
101     public ToolWindow()
102     {
103         choice = new JComboBox();
104         choice.addItem( "Circle" );
105         choice.addItem( "Square" );
106         choice.addItem( "Oval" );
107         choice.addItem( "Rectangle" );
108
109         choice.addItemListener(
110             new ItemListener() {
111                 public void itemStateChanged( ItemEvent e )
112                 {
113                     setShape( choice.getSelectedIndex() );
114                     repaint();
115                 }
116             }
117         );
118
119         Container c = getContentPane();
120         c.add( choice, BorderLayout.SOUTH );
121
122         chooseColor = new JButton( "Pick Color" );
123         chooseColor.addActionListener(
124             new ActionListener() {
125                 public void actionPerformed( ActionEvent e )
126                 {
127                     color = JColorChooser.showDialog( null, "Pick Color", Color.black );
128                 }
129             }
130         );
131         c.add( chooseColor, BorderLayout.NORTH );
132     }

```

```

133         filled = new JCheckBox( "Filled" );
134         c.add( filled, BorderLayout.CENTER );
135
136         pack();
137         setLocation( 300, 0 );
138         show();
139     }
140 }
141
142 private class MouseHandler extends MouseAdapter {
143     public void mouseReleased( MouseEvent e )
144     {
145         setBottomX( e.getX() );
146         setBottomY( e.getY() );
147         repaint();
148     }
149
150     public void mousePressed( MouseEvent e )
151     {
152         setTopX( e.getX() );
153         setTopY( e.getY() );
154     }
155 }
156
157 private class MouseMotionHandler extends MouseMotionAdapter {
158     public void mouseDragged( MouseEvent e )
159     {
160         setBottomX( e.getX() );
161         setBottomY( e.getY() );
162         repaint();
163     }
164 }
165 }

```



29.20 (*Complete Drawing Application*) Using the techniques developed in Exercises 29.12 through 29.19, create a complete drawing program. The program should use the GUI components of this chapter to enable the user to select the shape, color and fill characteristics. For this program, create your own classes (like those in the class hierarchy described in Exercise 27.19) from which objects will be created to store each shape the user draws. The classes should store the location, dimensions and color of each shape and should indicate if the shape is filled or unfilled. Your classes should all derive from a class called `MyShape` that has all the common features of every shape type. Every subclass of `MyShape` should have its own method `draw`, which returns `void` and receives a `Graphics` object as its argument. Create a subclass of `JPanel` called `DrawPanel` for drawing the shapes. When the `DrawPanel`'s `paintComponent` method is called, it should walk through the array of shapes and display each shape by polymorphically calling the shape's `draw` method (passing the `Graphics` object as an argument). Each shape's `draw` method should know how to draw the shape. As a minimum, your program should provide the following classes: `MyLine`, `MyOval`, `MyRect`, `MyRoundRect`. Design the class hierarchy for maximum software reuse and place all your classes in the package `shapes`. Import this package into your program. Each shape should be stored in an array of `MyShape` objects, where `MyShape` is the superclass in your hierarchy of shape classes (see Exercise 27.19).

29.21 Modify Exercise 29.20 to provide an `Undo` button that can be used repeatedly to undo the last painting operation. If there are no shapes in the array of shapes, the `Undo` button should be disabled.

30

Java Multimedia: Images, Animation, and Audio: Solutions

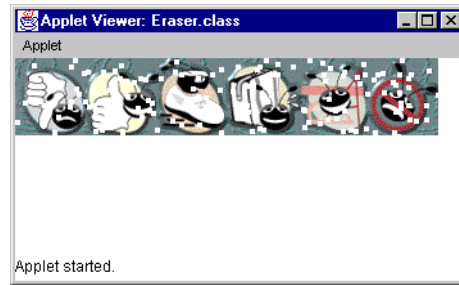
EXERCISES

- 30.3** Describe how to make an animation “browser friendly.”
ANS: Begin the animation in the `start` method and suspend/terminate the animation in the `stop` method.
- 30.4** Discuss the various aspects of flicker elimination in Java.
ANS: Flickering can be reduced or eliminated by overriding the `paint` method and using graphics double buffering techniques.
- 30.5** Explain the technique of graphics double buffering.
ANS: Method `createImage` is used to create an empty image. The graphics context of the empty image is retrieved with a call to `getGraphics`. The empty image can then be used to store pixels drawn on the image with the `Graphics` object that was obtained via `getGraphics`. When the image is complete, it can be displayed using method `drawImage`. Swing components such as `JPanel` have built-in double buffering.
- 30.6** Describe the Java methods for playing and manipulating audio clips.
ANS: The applet `play` method. The `AudioClip` interface methods: `play`, `loop` and `stop`.
- 30.7** (*Animation*) Create a general-purpose Java animation program. Your program should allow the user to specify the sequence of frames to be displayed, the speed at which the images are displayed, audios that should be played while the animation is running and so on.
- 30.8** (*Screensaver*) Use animation of a series of your favorite images to create a screensaver program. Create various special effects that explode the images, spin the images, fade them in and out, move them off the edge of the screen, and the like.

30.9 (*Randomly Erasing an Image*) Suppose an image is displayed in a rectangular screen area. One way to erase the image is simply to set every pixel to the same color immediately, but this is a dull visual effect. Write a Java program that displays an image then erases it by using random-number generation to select individual pixels to erase. After most of the image is erased, erase all of the remaining pixels at once. You can refer to individual pixels by having a line that starts and ends at the same point. You might try several variants of this problem. For example, you might display lines randomly or you might display shapes randomly to erase regions of the screen.

ANS:

```
1 // Exercise 30.9 Solution
2 // Eraser.java
3 // Program randomly covers up an image.
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Eraser extends JApplet implements ActionListener {
9     private ImageIcon image;
10    private int imageWidth, imageHeight, count;
11    private int numberOfTimes;
12    private boolean showImage = true;
13    private Timer t;
14
15    public void init()
16    {
17        image = new ImageIcon( "icons2.gif" );
18        t = new Timer( 10, this );
19        t.start();
20
21        imageWidth = image.getIconWidth();
22        imageHeight = image.getIconHeight();
23        numberOfTimes = imageWidth * imageHeight / 8;
24    }
25
26    public void paint( Graphics g )
27    {
28        if ( showImage == true ) {
29            image.paintIcon( this, getGraphics(), 0, 0 );
30            showImage = false;
31        }
32
33        g.setColor( getBackground() );
34        g.fillRect( ( int ) ( Math.random() * imageWidth ),
35                  ( int ) ( Math.random() * imageHeight ), 4, 4 );
36    }
37
38    public void actionPerformed( ActionEvent e )
39    {
40        repaint();
41
42        if ( count == numberOfTimes ) {
43            showImage = true;
44            count = 0;
45        }
46
47        ++count;
48    }
49 }
```



30.10 (*Text Flasher*) Create a Java program that repeatedly flashes text on the screen. Do this by interspersing the text with a plain background color image. Allow the user to control the “blink speed” and the background color or pattern.

ANS:

```

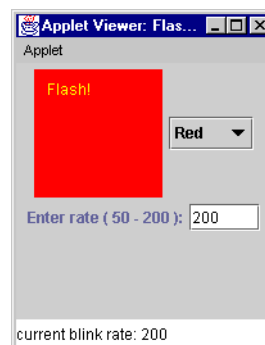
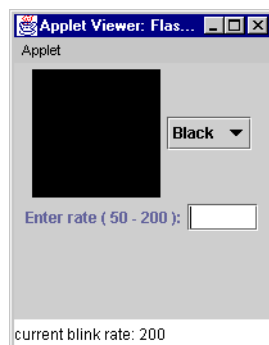
1 // Exercise 30.10 Solution
2 // Flash.java
3 // Program flashes text.
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Flash extends JApplet
9     implements ActionListener, ItemListener {
10     private MyCanvas theCanvas;
11     private JComboBox colorSelect;
12     private JLabel prompt;
13     private JTextField input;
14
15     public void init()
16     {
17         prompt = new JLabel( "Enter rate ( 50 - 200 ):" );
18         input = new JTextField( 5 );
19         input.addActionListener( this );
20         theCanvas = new MyCanvas();
21         String items[] = { "Black", "Red", "Blue", "Green" };
22         colorSelect = new JComboBox( items );
23         colorSelect.addItemListener( this );
24         Container c = getContentPane();
25         c.setLayout( new FlowLayout() );
26         c.add( theCanvas );
27         c.add( colorSelect );
28         c.add( prompt );
29         c.add( input );
30     }
31
32     public void itemStateChanged( ItemEvent e )
33     {
34         Color c;
35
36         if ( e.getItem().equals( "Black" ) )
37             c = Color.black;
38         else if ( e.getItem().equals( "Red" ) )
39             c = Color.red;
40         else if ( e.getItem().equals( "Blue" ) )
41             c = Color.blue;
42         else
43             c = Color.green;
44
45         theCanvas.setBackground( c );
46     }
47
48     public void actionPerformed( ActionEvent e )
49     {
50         theCanvas.setSleepTime( Integer.parseInt( input.getText() ) );
51         showStatus( "current blink rate: " + theCanvas.getSleepTime() );
52     }
53 }
54
55 class MyCanvas extends JPanel implements ActionListener {
56     private String text;
57     private Timer t;

```

```

58     private Color c = Color.black;
59     boolean flash = true;
60
61     public MyCanvas()
62     {
63         setBackground( Color.black );
64         t = new Timer( 150, this );
65         t.start();
66         text = "Flash!";
67         setSize( 100, 100 );
68         setOpaque( true );
69     }
70
71     public synchronized void paintComponent( Graphics g )
72     {
73         super.paintComponent( g );
74
75         if ( flash ) {
76             g.setColor( Color.yellow );
77             g.drawString( text, 10, 20 );
78         }
79     }
80
81     public synchronized void actionPerformed((ActionEvent e )
82     {
83         flash = !flash;
84         repaint();
85     }
86
87     public void setSleepTime( int time )
88         { t.setDelay( time >= 50 && time <= 200 ? time : 150 ); }
89
90     public int getSleepTime() { return t.getDelay(); }
91
92     public Dimension getPreferredSize()
93     {
94         return new Dimension( 100, 100 );
95     }
96 }

```



30.11 (*Image Flasher*) Create a Java program that repeatedly flashes an image on the screen. Do this by interspersing the image with a plain background color image.

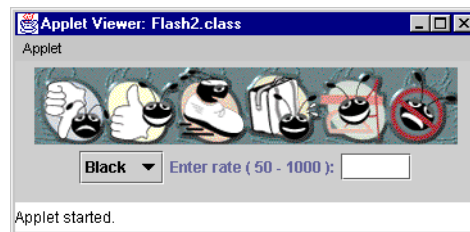
ANS:

```

1 // Exercise 30.11 Solution
2 // Flash2.java
3 // Program flashes text.
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Flash2 extends JApplet
9     implements ActionListener, ItemListener {
10     private MyCanvas theCanvas;
11     private JComboBox colorSelect;
12     private JLabel prompt;
13     private JTextField input;
14
15     public void init()
16     {
17         prompt = new JLabel( "Enter rate ( 50 - 1000 ):" );
18         input = new JTextField( 5 );
19         input.addActionListener( this );
20         theCanvas = new MyCanvas();
21         String items[] = { "Black", "Red", "Blue", "Green" };
22         colorSelect = new JComboBox( items );
23         colorSelect.addItemListener( this );
24         Container c = getContentPane();
25         c.setLayout( new FlowLayout() );
26         c.add( theCanvas );
27         c.add( colorSelect );
28         c.add( prompt );
29         c.add( input );
30     }
31
32     public void itemStateChanged( ItemEvent e )
33     {
34         Color c;
35
36         if ( e.getItem().equals( "Black" ) )
37             c = Color.black;
38         else if ( e.getItem().equals( "Red" ) )
39             c = Color.red;
40         else if ( e.getItem().equals( "Blue" ) )
41             c = Color.blue;
42         else
43             c = Color.green;
44
45         theCanvas.setBackground( c );
46     }
47
48     public void actionPerformed( ActionEvent e )
49     {
50         theCanvas.setSleepTime( Integer.parseInt( input.getText() ) );
51         showStatus( "current blink rate: " + theCanvas.getSleepTime() );
52     }
53 }
54
55 class MyCanvas extends JPanel implements ActionListener {
56     private ImageIcon image;
57     private Timer t;
58     boolean flash = true;
59
60     public MyCanvas()
61     {
62         setBackground( Color.black );
63         image = new ImageIcon( "icons2.gif" );

```

```
64     t = new Timer( 500, this );
65     t.start();
66 }
67
68 public synchronized void paintComponent( Graphics g )
69 {
70     super.paintComponent( g );
71
72     if ( flash )
73         g.drawImage( image.getImage(), 0, 0, this );
74 }
75
76 public synchronized void actionPerformed((ActionEvent e )
77 {
78     flash = !flash;
79     repaint();
80 }
81
82 public void setSleepTime( int time )
83     { t.setDelay( time >= 50 && time <= 1000 ? time : 500 ); }
84
85 public int getSleepTime() { return t.getDelay(); }
86
87 public Dimension getPreferredSize()
88 {
89     return new Dimension( image.getIconWidth(),
90                           image.getIconHeight() );
91 }
92 }
```



30.12 (*Digital Clock*) Implement a program that displays a digital clock on the screen. You might add options to scale the clock; display day, month and year; issue an alarm; play certain audios at designated times and the like.

ANS:

```
1 // Exercise 30.12 Solution
2 // DigitalClock.java
3 // Program creates a digital clock.
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.util.*;
8
9 public class DigitalClock extends JApplet
10     implements ActionListener {
11     private String theTime;
12     private Timer t;
13
14     public void init()
15     {
16         theTime = "";
17         t = new Timer( 1000, this );
18     }
19
20     public void paint( Graphics g )
21     {
22         super.paint( g ); // clears the background
23
24         g.drawString( theTime, 20, 50 );
25     }
26
27     public void start()
28     {
29         t.start();
30     }
31
32     public void stop()
33     {
34         t.stop();
35     }
36
37     public void actionPerformed((ActionEvent e )
38     {
39         theTime = new Date().toString();
40         repaint();
41     }
42 }
```



30.13 (*Calling attention to an image*) If you want to emphasize an image, you might place a row of simulated light bulbs around your image. You can let the light bulbs flash in unison or you can let them fire on and off in sequence one after the other.

ANS:

```

1 // Exercise 30.13 Solution
2 // Flash3.java
3 // Program highlights an image.
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Flash3 extends JApplet {
9     private MyCanvas theCanvas;
10
11     public void init()
12     {
13         ImageIcon image1 = new ImageIcon( "icons2.gif" );
14
15         int width = image1.getIconWidth() + 20;
16         int height = image1.getIconHeight() + 20;
17         Image image2 = createImage( width, height );
18         Image image3 = createImage( width, height );
19
20         theCanvas = new MyCanvas( image1.getImage(), image2, image3,
21                                 width, height );
22         getContentPane().add( theCanvas, BorderLayout.CENTER );
23     }
24 }
25
26 class MyCanvas extends JPanel implements ActionListener {
27     private Image img, img2, img3;
28     private Graphics graph2, graph3;
29     private boolean flashSwitch;
30     private Timer t;
31
32     public MyCanvas( Image i, Image i2, Image i3, int w, int h )
33     {
34         t = new Timer( 300, this );
35         t.start();
36         flashSwitch = true;
37         setSize( w, h );
38         img = i;
39         img2 = i2;
40         img3 = i3;
41         createBuffers( w, h );
42     }
43
44     public void createBuffers( int w, int h )
45     {
46         graph2 = img2.getGraphics();
47         graph3 = img3.getGraphics();
48
49         graph2.setColor( Color.black );
50         graph2.fillRect( 0, 0, w, h );
51         graph3.setColor( Color.black );
52         graph3.fillRect( 0, 0, w, h );
53
54         int count = 0;
55
56         for ( int x = 0; x < w; x += 10 ) {
57             for ( int y = 0; y < h; y += 10 ) {
58
59                 // Change ++count to y to get the effect of
60                 // all the lights "turning off" then "turning on"
61                 // Also the line below that alternates the lights
62                 // should be commented out or removed.

```



```
63         if ( ++count % 2 == 0 ) {
64             graph2.setColor( Color.yellow );
65             graph3.setColor( Color.white );
66         }
67         else {
68             graph2.setColor( Color.white );
69             graph3.setColor( Color.yellow );
70         }
71
72         graph2.fillOval( x, y, 10, 10 );
73         graph3.fillOval( x, y, 10, 10 );
74     }
75
76     // Allow the lights to alternate
77     count = ( count % 2 == 0 ? 1 : 0 );
78 }
79
80 graph2.drawImage( img, 10, 10, this );
81 graph3.drawImage( img, 10, 10, this );
82 }
83
84 public void paintComponent( Graphics g )
85 {
86     super.paintComponent( g );
87
88     if ( flashSwitch )
89         g.drawImage( img2, 0, 0, this );
90     else
91         g.drawImage( img3, 0, 0, this );
92 }
93
94 public void actionPerformed((ActionEvent e )
95 {
96     flashSwitch = !flashSwitch;
97     repaint();
98 }
99 }
```



30.14 *Image Zooming*) Create a program that enables you to zoom in on, or away from, an image.

ANS:

```

1 // Exercise 30.14 Solution
2 // Zoom.java
3 // Program zooms an image.
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class Zoom extends JApplet
9     implements ItemListener {
10     private MyCanvas theCanvas;
11     private JPanel p;
12     private JComboBox select;
13     private int width, height;
14
15     public void init()
16     {
17         ImageIcon image1 = new ImageIcon( "icons2.gif" );
18
19         p = new JPanel();
20         String items[] = { "50%", "100%", "200%", "300%" };
21         select = new JComboBox( items );
22         select.addItemListener( this );
23         p.add( select );
24
25         width = image1.getIconWidth() / 2;
26         height = image1.getIconHeight() / 2;
27
28         theCanvas = new MyCanvas( image1.getImage(), width, height );
29         Container c = getContentPane();
30         c.add( theCanvas, BorderLayout.CENTER );
31         c.add( p, BorderLayout.SOUTH );
32     }
33
34     public void itemStateChanged( ItemEvent e )
35     {
36         if ( e.getItem().equals( "50%" ) )
37             theCanvas.setWidthHeight( ( int ) ( width * .5 ), ( int ) ( height * .5 ) );
38         else if ( e.getItem().equals( "100%" ) )
39             theCanvas.setWidthHeight( width, height );
40         else if ( e.getItem().equals( "200%" ) )
41             theCanvas.setWidthHeight( width * 2, height * 2 );
42         else // 300%
43             theCanvas.setWidthHeight( width * 3, height * 3 );
44     }
45 }
46
47 class MyCanvas extends JPanel {
48     private Image img;
49     private int imgWidth, imgHeight;
50
51     public MyCanvas( Image i, int w, int h )
52     {
53         setBackground( Color.green );
54         setSize( w, h );
55         img = i;
56         setWidthHeight( w, h );
57     }
58
59     public void setWidthHeight( int w, int h )
60     {
61         imgWidth = w;
62         imgHeight = h;
63         repaint();
64     }

```

```
65  
66 public void paintComponent( Graphics g )  
67 {  
68     super.paintComponent( g );  
69     g.drawImage( img, 0, 0, imgWidth, imgHeight, this );  
70 }  
71 }
```

