APPENDICES

INTRODUCTION

Now that you've become more intimately involved with your Commodore 64, we want you to know that our customer support does not stop here. You may not know it, but Commodore has been in business for over 23 years. In the 1970's we introduced the first self-contained personal computer (the PET). We have since become the leading computer company in many countries of the world. Our ability to design and manufacture our own computer chips allows us to bring you new and better personal computers at prices way below what you'd expect for this level of technical excellence.

Commodore is committed to supporting not only you, the end user, but also the dealer you bought your computer from, magazines which publish how-to articles showing you new applications or techniques, and . . . importantly . . . software developers who produce programs on cartridge, disk and tape for use with your computer. We encourage you to establish or join a Commodore "user club" where you can learn new techniques, exchange ideas and share discoveries. We publish two separate magazines which contain programming tips, information on new products and ideas for computer applications. (See Appendix N).

In North America, Commodore provides a "Commodore Information Network" on the CompuServe Information Service . . . to access this network, all you need is your Commodore 64 computer and our low cost VICMODEM telephone interface cartridge (or other compatible modem).

The following APPENDICES contain charts, tables, and other information which help you program your Commodore 64 faster and more efficiently. They also include important information on the wide variety of Commodore products you may be interested in, and a bibliography listing of over 20 books and magazines which can help you develop your programming skills and keep you current on the latest information concerning your computer and peripherals.

APPENDIX A

COMMODORE 64 ACCESSORIES AND SOFTWARE

ACCESSORIES

The Commodore 64 will support Commodore VIC 20 storage devices and accessories—DATASSETTE recorder, disk drive, modem, printer — so your system can expand to keep pace with changing needs.

- Datasette Recorder—This low cost tape unit enables programs and data to be stored on cassette tape, and played back at a later time. The datasette can also be used to play pre-written programs.
- Disk—The single disk unit uses standard 5¼-inch floppy diskettes, about the size of a 45 RPM record, to store programs and data. Disks allow faster access to data and hold up to 170,000 characters of information each. Disk units are "intelligent," meaning they have their own microprocessor and memory. Disks require no resources from the Commodore 64, such as using part of main memory.
- Modem—A low-cost communication device,the VICMODEM allows access to other computers over ordinary telephone lines. Users will have access to the full resources of large data bases such as The Source, CompuServe, and Dow Jones News Retrieval Service (North America only).
- Printer—The VIC printer produces printed copies of programs, data, or graphics. This 30 character per second dot-matrix printer uses plain tractor feed paper and other inexpensive supplies. The printer attaches directly to the Commodore 64 without any additional interfaces.
- Interface Cartridges—A number of specialized cartridges will be available for the Commodore 64 to allow various standard devices such as modems, printers, controllers, and instruments to be attached to the system.

With a special IEEE-488 Cartridge, the Commodore 64 will support the full range of CBM peripherals including disk units and printers.

Additionally, a Z80 cartridge will allow you to run CP/M* on the Commodore 64, giving you access to the largest base of microcomputer applications available.

SOFTWARE

Several categories of software will be offered for the Commodore 64, providing you with a wide variety of personal, entertainment, and educational applications to choose from.

BUSINESS AIDS

- An Electronic Spreadsheet package will allow you to plan budgets, and perform "what if?" analysis. And with the optional graphic program, meaningful graphs may be created from the spreadsheet data.
- Financial planning, such as loan amortization, will be easily handled with the Financial Planning Package.
- A number of Professional Time Management programs will help manage appointments and work load.
- Easy-to-use Data Base programs will allow you to keep track of information . . . mailing lists . . . phone lists . . . inventories . . . and organize information in a useful form.
- Professional Word Processing programs will turn the Commodore 64 into a full-featured word processor. Typing and revising memos, letters, and other text material become a breeze.

ENTERTAINMENT

- The highest quality games will be available on plug-in cartridges for the Commodore 64, providing hours of enjoyment. These programs make use of the high resolution graphics and full sound range possible with the Commodore 64.
- Your Commodore 64 allows you all the fun and excitement available on MAX games because these two machines have completely compatible cartridges.

---

*CP/M is a registered trademark of Digital Research Inc.

EDUCATION

- The Commodore 64 is a tutor that never tires and always gives personal attention. Besides access to much of the vast PET educational programs, additional educational languages that will be available for the Commodore 64 include PILOT, LOGO and other key advanced packages.

APPENDIX B

ADVANCED CASSETTE OPERATION

Besides saving copies of your programs on tape, the Commodore 64 can also store the values of variables and other items of data, in a group called a FILE. This allows you to store even more information than could be held in the computer's main memory at one time.

Statements used with data files are OPEN, CLOSE, PRINT#, INPUT#, and GET#. The system variable ST (status) is used to check for tape markers.

In writing data to tape, the same concepts are used as when displaying information on the computer's screen. But instead of PRINTing information on the screen, the information is PRINTed on tape using a variation of the PRINT command—PRINT#.

The following program illustrates how this works:

```
10 PRINT "WRITE-TO-TAPE-PROGRAM"
20 OPEN 1,1,1,"DATA FILE"
30 PRINT "TYPE DATA TO BE STORED OR TYPE STOP"
50 PRINT
60 INPUT "DATA";A$
70 PRINT #1, A$
80 IF A$ <>"STOP" THEN 50
90 PRINT
100 PRINT "CLOSING FILE"
110 CLOSE 1
```

The first thing that you must do is OPEN a file (in this case DATA FILE). Line 10 handles that.

The program prompts for the data you want to save on tape in line 60. Line 70 writes what you typed—held in A$—onto the tape. And the process continues.

If you type STOP, line 110 CLOSES the file.

To retrieve the information, rewind the tape, and try this:

```
10 PRINT "READ-TAPE-PROGRAM"
20 OPEN 1,1,0,"DATA FILE"
30 PRINT "FILE OPEN"
40 PRINT
50 INPUT#1, A$
60 PRINT A$
70 IF A$ = "STOP" THEN END
80 GOTO 40
```

Again, the file "DATA FILE" first must be OPENed. In line 50 the program INPUTs A$ from tape and also PRINTs A$ on the screen. Then the whole process is repeated until "STOP" is found, which ENDs the program.

A variation of GET—GET#—can also be used to read the data back from tape. Replace lines 50-80 in the program above with:

```
50 GET#1, A$
60 IF A$ = "" THEN END
70 PRINT A$, ASC(A$)
80 GOTO 50
```

APPENDIX C

COMMODORE 64 BASIC

This manual has given you an introduction to the BASIC language—enough for you to get a feel for computer programming and some of the vocabulary involved. This appendix gives a complete list of the rules (SYNTAX) of Commodore 64 BASIC, along with concise descriptions. Please experiment with these commands. Remember, you can't do any permanent damage to the computer by just typing in programs, and the best way to learn computing is by doing.

This appendix is divided into sections according to the different types of operations in BASIC. These include:

1. **Variables and Operators:** describes the different type of variables, legal variable names, and arithmetic and logical operators.
2. **Commands:** describes the commands used to work with programs, edit, store, and erase them.
3. **Statements:** describes the BASIC program statements used in numbered lines of programs.
4. **Functions:** describes the string, numeric, and print functions.

VARIABLES

The Commodore 64 uses three types of variables in BASIC. These are real numeric, integer numeric, and string (alphanumeric) variables.

Variable names may consist of a single letter, a letter followed by a number, or two letters.

An integer variable is specified by using the percent (%) sign after the variable name. String variables have the dollar sign ($) after their name.

EXAMPLES

Real Variable Names: A, A5, BZ
Integer Variable Names: A%, A5%, BZ%

String Variable Names: A$, A5$, BZ$

Arrays are lists of variables with the same name, using extra numbers to specify the element of the array. Arrays are defined using the DIM statement, and may contain floating point, integer, or string variables. The array variable name is followed by a set of parentheses ( ) enclosing the number of variables in the list.

A(7), BZ%(11), A$(50), PT(20,20)

NOTE: There are three variable names which are reserved for use by the Commodore 64, and may not be defined by you. These variables are: ST, TI, and TI$. ST is a status variable which relates to input/output operations. The value of ST will change if there is a problem loading a program from disk or tape.

TI and TI$ are variables which relate to the real-time clock built into the Commodore 64. The variable TI is updated every 1/60th of a second. It starts at 0 when the computer is turned on, and is reset only by changing the value of TI$.

TI$ is a string which is constantly updated by the system. The first two characters contain the number of hours, the 3rd and 4th characters the number of minutes, and the 5th and 6th characters are the number of seconds. This variable can be given any numeric value, and will be updated from that point.

TI$ = "101530" sets the clock to 10:15 and 30 seconds AM.

This clock is erased when the computer is turned off, and starts at zero when the system is turned back on.

OPERATORS

The arithmetic operators include the following signs:

+ Addition
− Subtraction
* Multiplication
/ Division
↑ Raising to a power (exponentiation)

On a line containing more than one operator, there is a set order in which operations always occur. If several operations are used together on the same line, the computer assigns priorities as follows: First, exponentiation. Next, multiplication and division, and last, addition and subtraction.

You can change the order of operations by enclosing within parentheses the calculation to be performed first. Operations enclosed in parentheses will take place before other operations.

There are also operations for equalities and inequalities:

= Equal To
< Less Than
> Greater Than
<= Less Than or Equal To
>= Greater Than or Equal To
<> Not Equal To

Finally, there are three logical operators:

AND
OR
NOT

These are used most often to join multiple formulas in IF . . . THEN statements. For example:

IF A = B AND C = D THEN 100 (Requires both parts to be true)

IF A = B OR C = D THEN 100 (Allows either part to be true)

COMMANDS

CONT (Continue)

This command is used to restart the execution of a program which has been stopped by either using the STOP key, a STOP statement, or an END statement within the program. The program will restart at the exact place from where it left off.

CONT will not work if you have changed or added lines to the program (or even just moved the cursor), or if the program halted due to an error, or if you caused an error before trying to restart the program. In these cases you will get a CAN'T CONTINUE ERROR.

LIST

The LIST command allows you to look at lines of a BASIC program in memory. You can ask for the entire program to be displayed, or only certain line numbers.

LIST	Shows entire program
LIST 10–	Shows only from line 10 until end
LIST 10	Shows only line 10
LIST –10	Shows lines from beginning until 10
LIST 10-20	Shows line from 10 to 20, inclusive

LOAD

This command is used to transfer a program from tape or disk into memory so the program can be used. If you just type LOAD and hit RETURN, the first program found on the cassette unit will be placed in memory. The command may be followed by a program name enclosed within quotes. The name may then be followed by a comma and a number or numeric variable, which acts as a device number to indicate where the program is coming from.

If no device number is given, the Commodore 64 assumes device #1, which is the cassette unit. The other device commonly used with the LOAD command is the disk drive, which is device #8.

LOAD	Reads in the next program on tape
LOAD "HELLO"	Searches tape for program called HELLO, and loads program, if found
LOAD A$	Looks for program whose name is in the variable A$
LOAD "HELLO",8	Looks for program called HELLO on the disk drive
LOAD "*",8	Looks for first program on disk

NEW

This command erases the entire program in memory, and also clears out any variables that may have been used. Unless the program was SAVEd, it is lost. **BE CAREFUL WHEN YOU USE THIS COMMAND.**

The NEW command can also be used as a BASIC program statement. When the program reaches this line, the program is erased. This is useful if you want to leave everything neat when the program is done.

RUN

This command causes execution of a program, once the program is loaded into memory. If there is no line number following RUN, the computer will start with the lowest line number. If a line number is designated, the program will start executing from the specified line.

RUN	Starts program at lowest line number
RUN 100	Starts execution at line 100
RUN X	UNDEFINED STATEMENT ERROR. You must always specify an actual line number, not a variable representation

SAVE

This command will store the program currently in memory on cassette or disk. If you just type SAVE and RETURN, the program will be SAVEd on cassette. The computer has no way of knowing if there is a program already on that tape, so be careful with your tapes or you may erase a valuable program.

If you type SAVE followed by a name in quotes or a string variable, the computer will give the program that name, so it can be more easily located and retrieved in the future. The name may also be followed by a device number.

After the device number, there can be a comma and a second number, either 0 or 1. If the second number is 1, the Commodore 64 will put an END-OF-TAPE marker after your program. This signals the computer not to look any further on the tape if you were to give an additional LOAD command. If you try to LOAD a program and the computer finds one of these markers, you will get a FILE NOT FOUND ERROR.

SAVE	Stores program to tape without name
SAVE "HELLO"	Stores on tape with name HELLO
SAVE AS	Stores on tape with name in A$
SAVE "HELLO",8	Stores on disk with name HELLO
SAVE "HELLO",1,1	Stores on tape with name HELLO and follows program with END-OF-TAPE marker

VERIFY

This command causes the computer to check the program on disk or tape against the one in memory. This is proof that the program is actually SAVEd, in case the tape or disk is bad, or something went wrong during the SAVE. VERIFY without anything after the command causes the Commodore 64 to check the next program on tape, regardless of name, against the program in memory.

VERIFY followed by a program name, or a string variable, will search for that program and then check. Device numbers can also be included with the verify command.

VERIFY	Checks the next program on tape
VERIFY "HELLO"	Searches for HELLO, checks against memory
VERIFY "HELLO",8	Searches for HELLO on disk, then checks

STATEMENTS

CLOSE

This command completes and closes any files used by OPEN statements. The number following CLOSE is the file number to be closed.

CLOSE 2	Only file #2 is closed

CLR

This command will erase any variables in memory, but leaves the program itself intact. This command is automatically executed when a RUN command is given.

CMD

CMD sends the output which normally would go to the screen (i.e., PRINT statements, LISTs, but not POKEs onto the screen) to another device instead. This could be a printer, or a data file on tape or disk. This device or file must be OPENed first. The CMD command must be followed by a number or numeric variable referring to the file.

OPEN 1,4	OPENs device #4, which is the printer
CMD 1	All normal output now goes to printer
LIST	The program listing now goes to the printer, not the screen

To send output back to the screen, CLOSE the file with CLOSE 1.

DATA

This statement is followed by a list of items to be used by READ statements. Items may be numeric values or text strings, and items are separated by commas. String items need not be inside quote marks unless they contain space, colon, or comma. If two commas have nothing between them, the value will be READ as a zero for a number, or an empty string.

```
DATA 12, 14.5, "HELLO, MOM", 3.14, PART1
```

DEF FN

This command allows you to define a complex calculation as a function with a short name. In the case of a long formula that is used many times within the program, this can save time and space.

The function name will be FN and any legal variable name (1 or 2 characters long). First you must define the function using the statement DEF followed by the function name. Following the name is a set of parentheses enclosing a numeric variable. Then follows the actual formula that you want to define, with the variable in the proper spot. You can then "call" the formula, substituting any number for the variable.

```
1Ø DEF FNA(X) = 12*(34.75 - X/.3)
2Ø PRINT FNA(7)
```

7 is inserted where
X is in the formula

For this example, the result would be 137.

DIM

When you use more than 11 elements of an array, you must execute a DIM statement for the array. Keep in mind that the whole array takes up room in memory, so don't create an array much larger than you'll need. To figure the number of variables created with DIM, multiply the total number of elements in each dimension of the array.

```
10 DIM A$(40), B7(15), CC%(4,4,4)
          ↑         ↑          ↑
     41 ELEMENTS  16 ELEMENTS  125 ELEMENTS
```

You can dimension more than one array in a DIM statement. However, be careful not to dimension an array more than once.

END

When a program encounters an END statement, the program halts, as if it ran out of lines. You may use CONT to restart the program.

FOR. . .TO. . .STEP

This statement works with the NEXT statement to repeat a section of the program a set number of times. The format is:

FOR (Var. Name)=(Start of Count) TO (End of Count) STEP(Count By)

The loop variable will be added to or subtracted from during the program. Without any STEP specified, STEP is assumed to be 1. The start count and end count are the limits to the value of the loop variable.

```
1Ø FOR L = 1 TO 1Ø STEP .1
2Ø PRINT L
3Ø NEXT L
```

The end of the loop value may be followed by the word STEP and another number or variable. In this case, the value following STEP is added each time instead of 1. This allows you to count backwards, or by fractions.

GET

The GET statement allows you to get data from the keyboard, one character at a time. When GET is executed, the character that is typed is assigned to the variable. If no character is typed, then a null (empty) character is assigned.

GET is followed by a variable name, usually a string variable. If a numeric variable was used and a nonnumeric key depressed, the program would halt with an error message. The GET statement may be placed into a loop, checking for any empty result. This loop will continue until a key is hit.

10 GET A$: IF A$ ="" THEN 10

GET#

The GET# statement is used with a previously OPENed device or file, to input one character at a time from that device or file.

GET #1,A$

This would input one character from a data file.

GOSUB

This statement is similar to GOTO, except the computer remembers which program line it last executed before the GOSUB. When a line with a RETURN statement is encountered, the program jumps back to the statement immediately following the GOSUB. This is useful if there is a routine in your program that occurs in several parts of the program. Instead of typing the routine over and over, execute GOSUBs each time the routine is needed.

20 GOSUB 800

GOTO OR GO TO

When a statement with the GOTO command is reached, the next line to be executed will be the one with the line number following the word GOTO.

IF. . .THEN

IF. . .THEN lets the computer analyze a situation and take two possible courses of action, depending on the outcome. If the expression is true, the statement following THEN is executed. This may be any BASIC statement.

If the expression is false, the program goes directly to the next line.

The expression being evaluated may be a variable or formula, in which case it is considered true if nonzero, and false if zero. In most cases, there is an expression involving relational operators (=, <, >, <=, >=, <>, AND, OR, NOT).

```
10 IF X > 10 THEN END
```

INPUT

The INPUT statement allows the program to get data from the user, assigning that data to a variable. The program will stop, print a question mark (?) on the screen, and wait for the user to type in the answer and hit RETURN.

INPUT is followed by a variable name, or a list of variable names, separated by commas. A message may be placed within quote marks, before the list of variable names to be INPUT. If more than one variable is to be INPUT, they must be separated by commas when typed.

```
10 INPUT "PLEASE ENTER YOUR FIRST NAME ";A$
20 PRINT "ENTER YOUR CODE NUMBER"; : INPUT B
```

INPUT#

INPUT# is similar to INPUT, but takes data from a previously OPENed file or device.

```
10 INPUT#1, A
```

LET

LET is hardly ever used in programs, since it is optional, but the statement is the heart of all BASIC programs. The variable name which is to be assigned the result of a calculation is on the left side of the equal sign, and the formula on the right.

```
10 LET A = 5
20 LET D$ = "HELLO"
```

NEXT

NEXT is always used in conjunction with the FOR statement. When the program reaches a NEXT statement, it checks the FOR statement to see if the limit of the loop has been reached. If the loop is not finished, the loop variable is increased by the specified STEP value. If the loop is finished, execution proceeds with the statement following NEXT.

NEXT may be followed by a variable name, or list of variable names, separated by commas. If there are no names listed, the last loop started is the one being completed. If variables are given, they are completed in order from left to right.

```
10 FOR X = 1 TO 100: NEXT
```

ON

This command turns the GOTO and GOSUB commands into special versions of the IF statement. ON is followed by a formula, which is evaluated. If the result of the calculation is one, the first line on the list is executed; if the result is 2, the second line is executed, and so on. If the result is 0, negative, or larger than the list of numbers, the next line executed will be the statement following the ON statement.

```
10 INPUT X
20 ON X GOTO 10,20,30,40,50
```

OPEN

The OPEN statement allows the Commodore 64 to access devices such as the cassette recorder and disk for data, a printer, or even the screen. OPEN is followed by a number (0-255), to which all following statements will refer. There is usually a second number after the first, which is the device number.

The device numbers are:

Ø	Screen
1	Cassette
4	Printer
8	Disk

Following the device number may be a third number, separated again by a comma, which is the secondary address. In the case of the cassette, this is 0 for read, 1 for write, and 2 for write with end-of-tape marker.

In the case of the disk, the number refers to the buffer, or channel, number. In the printer, the secondary address controls features like expanded printing. See the Commodore 64 Programmer's Reference Manual for more details.

1Ø OPEN 1,Ø	OPENs the SCREEN as a device
2Ø OPEN 2,1,Ø,"D"	OPENs the cassette for reading, file to be searched for is D
3Ø OPEN 3,4	OPENs the printer
4Ø OPEN 4,8,15	OPENs the data channel on the disk

Also see: CLOSE, CMD, GET#, INPUT#, and PRINT#, system variable ST, and Appendix B.

POKE

POKE is always followed by two numbers, or formulas. The first location is a memory location; the second number is a decimal value from 0 to 255, which will be placed in the memory location, replacing any previously stored value.

```
1Ø POKE 53281,Ø

2Ø S=4Ø96*13
3Ø POKE S+29,8
```

PRINT

The PRINT statement is the first one most people learn to use, but there are a number of variations to be aware of. PRINT can be followed by:

Text String with quotes
Variable names
Functions
Punctuation marks

Punctuation marks are used to help format the data on the screen. The comma divides the screen into four columns, while the semicolon suppresses all spacing. Either mark can be the last symbol on a line. This results in the next thing PRINTed acting as if it were a continuation of the same PRINT statement.

```
1Ø PRINT "HELLO"
2Ø PRINT "HELLO",A$
3Ø PRINT A+B
```

```
4Ø PRINT J;
6Ø PRINT A,B,C,D
```

Also see: POS, SPC and TAB functions

PRINT#

There are a few differences between this statement and PRINT. PRINT# is followed by a number, which refers to the device or data file previously OPENed. This number is followed by a comma and a list to be printed. The comma and semicolon have the same effect as they do in PRINT. Please note that some devices may not work with TAB and SPC.

```
1ØØ PRINT#1,"DATA VALUES"; A%, B1, C$
```

READ

READ is used to assign information from DATA statements to variables, so the information may be put to use. Care must be taken to avoid READing strings where READ is expecting a number, which will give a TYPE MISMATCH ERROR.

REM (Remark)

REMark is a note to whomever is reading a LIST of the program. It may explain a section of the program, or give additional instructions. REM statements in no way affect the operation of the program, except to add to its length. REM may be followed by any text.

RESTORE

When executed in a program, the pointer to which an item in a DATA statement will be READ next is reset to the first item in the list. This gives you the ability to re-READ the information. RESTORE stands by itself on a line.

RETURN

This statement is always used in conjunction with GOSUB. When the program encounters a RETURN, it will go to the statement immediately following the GOSUB command. If no GOSUB was previously issued, a RETURN WITHOUT GOSUB ERROR will occur.

STOP

This statement will halt program execution. The message, BREAK IN xxx will be displayed, where xxx is the line number containing STOP. The program may be restarted by using the CONT command. STOP is normally used in debugging a program.

SYS

SYS is followed by a decimal number or numeric value in the range 0-65535. The program will then begin executing the machine language program starting at that memory location. This is similar to the USR function, but does not allow parameter passing.

WAIT

WAIT is used to halt the program until the contents of a memory location changes in a specific way. WAIT is followed by a memory location (X) and up to two variables. The format is:

```
WAIT X,Y,Z
```

The contents of the memory location are first exclusive-ORed with the third number, if present, and then logically ANDed with the second number. If the result is zero, the program goes back to that memory location and checks again. When the result is nonzero, the program continues with the next statement.

NUMERIC FUNCTIONS

ABS(X) (absolute value)

ABS returns the absolute value of the number, without its sign (+ or −). The answer is always positive.

ATN(X) (arctangent)

Returns the angle, measured in radians, whose tangent is X.

COS(X) (cosine)

Returns the value of the cosine of X, where X is an angle measured in radians.

EXP(X)

Returns the value of the mathematical constant e(2.71827183) raised to the power of X.

FNxx(X)

Returns the value of the user-defined function xx created in a DEF FNxx(X) statement.

INT(X)

Returns the truncated value of X, that is, with all the decimal places to the right of the decimal point removed. The result will always be less than, or equal to, X. Thus, any negative numbers with decimal places will become the integer less than their current value.

LOG(X) (logarithm)

Will return the natural log of X. The natural log to the base e (see EXP(X)). To convert to log base 10, simply divide by LOG(10).

PEEK(X)

Used to find out contents of memory location X, in the range 0-65535, giving a result from 0-255. PEEK is often used in conjunction with the POKE statement.

RND(X) (random number)

RND(X) returns a random number in the range 0-1. The first random number should be generated by the formula RND(−TI) to start things off differently every time. After this, X should be a 1 or any positive number. If X is zero, the result will be the same random number as the last one.

A negative value for X will reseed the generator. The use of the same negative number for X will result in the same sequence of "random" numbers.

The formula for generating a number between X and Y is:

N = RND(1)*(Y−X)+X

where,

Y is the upper limit

X is the lower range of numbers desired.

SGN(X) (sign)

This function returns the sign (positive, negative, or zero) of X. The result will be +1 if positive, 0 if zero, and −1 if negative.

SIN(X) (sine)

SIN(X) is the trigonometric sine function. The result will be the sine of X, where X is an angle in radians.

SQR(X) (square root)

This function will return the square root of X, where X is a positive number or 0. If X is negative, an ILLEGAL QUANTITY ERROR results.

TAN(X) (tangent)

The result will be the tangent of X, where X is an angle in radians.

USR(X)

When this function is used, the program jumps to a machine language program whose starting point is contained in memory locations. The parameter X is passed to the machine language program, which will return another value back to the BASIC program. Refer to the Commodore 64 Programmer's Reference Manual for more details on this function and machine language programming.

STRING FUNCTIONS

ASC(X$)

This function will return the ASCII code of the first character of X$.

CHR$(X)

This is the opposite of ASC, and returns a string character whose ASCII code is X.

LEFT$(X$,X)

Returns a string containing the leftmost X characters of $X.

LEN(X$)

Returned will be the number of characters (including spaces and other symbols) in the string X$.

MID$(X$,S,X)

This will return a string containing X characters starting from the Sth character in X$.

RIGHT$(X$,X)

Returns the rightmost X characters in X$.

STR$(X)

This will return a string which is identical to the PRINTed version of X.

VAL(X$)

This function converts X$ into a number, and is essentially the inverse operation from STR$. The string is examined from the leftmost character to the right, for as many characters as are in recognizable number format.

```
1Ø X = VAL("123.456")           X = 123.456
1Ø X = VAL("12A13B")            X = 12
1Ø X = VAL("RIUØ17")            X = Ø
1Ø X = VAL ("-1.23.45.67")      X = -1.23
```

OTHER FUNCTIONS

FRE(X)

This function returns the number of unused bytes available in memory, regardless of the value of X. Note that FRE(X) will read out n negative numbers if the number of unused bytes is over 32K.

POS(X)

This function returns the number of the column (0-39) at which the next PRINT statement will begin on the screen. X may have any value and is not used.

SPC(X)

This is used in a PRINT statement to skip X spaces forward.

TAB(X)

TAB is also used in a PRINT statement; the next item to be PRINTed will be in column X.

APPENDIX D

ABBREVIATIONS FOR BASIC KEYWORDS

As a time-saver when typing in programs and commands, Commodore 64 BASIC allows the user to abbreviate most keywords. The abbreviation for PRINT is a question mark. The abbreviations for other words are made by typing the first one or two letters of the word, followed by the SHIFTed next letter of the word. If the abbreviations are used in a program line, the keyword will LIST in the full form.

Command	Abbreviation	Looks like this on screen	Command	Abbreviation	Looks like this on screen
ABS	A SHIFT B	A	END	E SHIFT N	E
AND	A SHIFT N	A	EXP	E SHIFT X	E
ASC	A SHIFT S	A	FN	NONE	FN
ATN	A SHIFT T	A	FOR	F SHIFT O	F
CHR$	C SHIFT H	C	FRE	F SHIFT R	F
CLOSE	CL SHIFT O	CL	GET	G SHIFT E	G
CLR	C SHIFT L	C	GET#	NONE	GET#
CMD	C SHIFT M	C	GOSUB	GO SHIFT S	GO
CONT	C SHIFT O	C	GOTO	G SHIFT O	G
COS	NONE	COS	IF	NONE	IF
DATA	D SHIFT A	D	INPUT	NONE	INPUT
DEF	D SHIFT E	D	INPUT#	I SHIFT N	I
DIM	D SHIFT I	D	INT	NONE	INT

Command	Abbreviation	Looks like this on screen	Command	Abbreviation	Looks like this on screen
LEFT$	LE SHIFT F	LE	RIGHT$	R SHIFT I	R
LEN	NONE	LEN	RND	R SHIFT N	R
LET	L SHIFT E	L	RUN	R SHIFT U	R
LIST	L SHIFT I	L	SAVE	S SHIFT A	S ♠
LOAD	L SHIFT O	L	SGN	S SHIFT G	S
LOG	NONE	LOG	SIN	S SHIFT I	S
MID$	M SHIFT I	M	SPC(	S SHIFT P	S
NEW	NONE	NEW	SQR	S SHIFT Q	S ●
NEXT	N SHIFT E	N	STATUS	ST	ST
NOT	N SHIFT O	N	STEP	ST SHIFT E	ST
ON	NONE	ON	STOP	S SHIFT T	S
OPEN	O SHIFT P	O	STR$	ST SHIFT R	ST
OR	NONE	OR	SYS	S SHIFT Y	S
PEEK	P SHIFT E	P	TAB(	T SHIFT A	T ♠
POKE	P SHIFT O	P	TAN	NONE	TAN
POS	NONE	POS	THEN	T SHIFT H	T
PRINT	?	?	TIME	TI	TI
PRINT#	P SHIFT R	P	TIME$	TI$	TI$
READ	R SHIFT E	R	USR	U SHIFT S	U ♥
REM	NONE	REM	VAL	V SHIFT A	V ♠
RESTORE	RE SHIFT S	RE ♥	VERIFY	V SHIFT E	V
RETURN	RE SHIFT T	RE	WAIT	W SHIFT A	W ♠

APPENDIX E

SCREEN DISPLAY CODES

The following chart lists all of the characters built into the Commodore 64 character sets. It shows which numbers should be POKEd into screen memory (locations 1024-2023) to get a desired character. Also shown is which character corresponds to a number PEEKed from the screen.

Two character sets are available, but only one set at a time. This means that you cannot have characters from one set on the screen at the same time you have characters from the other set displayed. The sets are switched by holding down the SHIFT and C= keys simultaneously.

From BASIC, POKE 53272,21 will switch to upper case mode and POKE 53272,23 switches to lower case.

Any number on the chart may also be displayed in REVERSE. The reverse character code may be obtained by adding 128 to the values shown.

If you want to display a solid circle at location 1504, POKE the code for the circle (81) into location 1504: POKE 1504,81.

There is a corresponding memory location to control the color of each character displayed on the screen (locations 55296-56295). To change the color of the circle to yellow (color code 7) you would POKE the corresponding memory location (55776) with the character color: POKE 55776,7.

Refer to Appendix G for the complete screen and color memory maps, along with color codes.

SCREEN CODES

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	C	c	3	F	f	6
A	a	1	D	d	4	G	g	7
B	b	2	E	e	5	H	h	8

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
I	i	9	%		37		A	65
J	j	10	&		38		B	66
K	k	11	'		39		C	67
L	l	12	(		40		D	68
M	m	13	)		41		E	69
N	n	14	*		42		F	70
O	o	15	+		43		G	71
P	p	16	,		44		H	72
Q	q	17	–		45		I	73
R	r	18	.		46		J	74
S	s	19	/		47		K	75
T	t	20	0		48		L	76
U	u	21	1		49		M	77
V	v	22	2		50		N	78
W	w	23	3		51		O	79
X	x	24	4		52		P	80
Y	y	25	5		53		Q	81
Z	z	26	6		54		R	82
[		27	7		55		S	83
£		28	8		56		T	84
]		29	9		57		U	85
↑		30	:		58		V	86
←		31	;		59		W	87
SPACE		32	<		60		X	88
!		33	=		61		Y	89
"		34	>		62		Z	90
#		35	?		63			91
$		36			64			92

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
		93			105			117
		94			106			118
		95			107			119
SPACE		96			108			120
		97			109			121
		98			110		✓	122
		99			111			123
		100			112			124
		101			113			125
		102			114			126
		103			115			127
		104			116			

Codes from 128-255 are reversed images of codes 0-127.

APPENDIX F

ASCII AND CHR$ CODES

This appendix shows you what characters will appear if you PRINT CHR$(X), for all possible values of X. It will also show the values obtained by typing PRINT ASC("x"), where x is any character you can type. This is useful in evaluating the character received in a GET statement, converting upper/lower case, and printing character based commands (like switch to upper/lower case) that could not be enclosed in quotes.

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0	CRSR ↓	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
DISABLES SHIFT C=	8		25	*	42	;	59
ENABLES SHIFT C=	9		26	+	43	<	60
	10		27	,	44	=	61
	11	RED	28	−	45	>	62
	12	CRSR →	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
SWITCH TO LOWER CASE	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
D	68	♠	97		126		155
E	69		98		127	PUR	156
F	70		99		128	CRSR	157
G	71		100	♠	129	YEL	158
H	72		101		130	CYN	159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84	●	113	SWITCH TO UPPER CASE	142		171
U	85		114		143		172
V	86	♥	115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91	♣	120		149		178
£	92		121		150		179
]	93	♦	122		151		180
↑	94		123	♣	152		181
←	95		124		153		182
	96		125	♦	154		183

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	184		186		188		190
	185		187		189		191

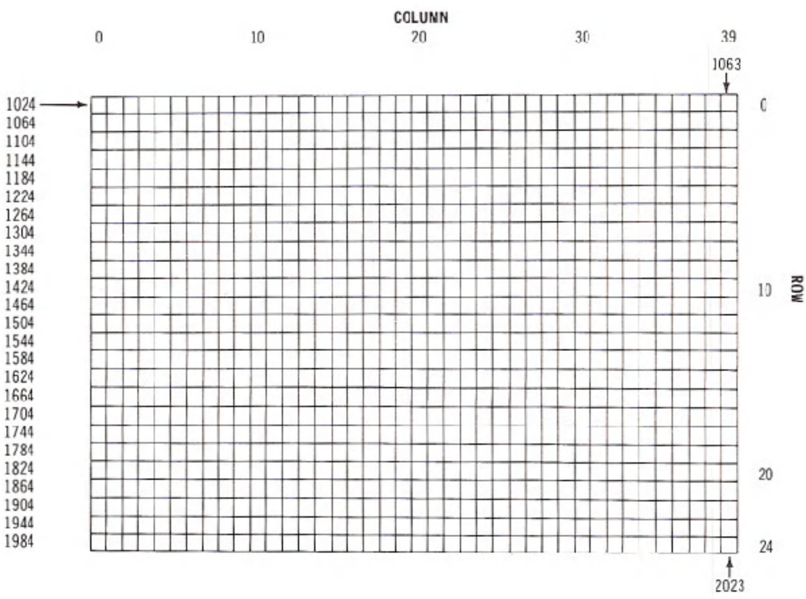
CODES	**192-223**	**SAME AS**	**96-127**
CODES	**224-254**	**SAME AS**	**160-190**
CODE	**255**	**SAME AS**	**126**

APPENDIX G

SCREEN AND COLOR MEMORY MAPS

The following charts list which memory locations control placing characters on the screen, and the locations used to change individual character colors, as well as showing character color codes.

SCREEN MEMORY MAP

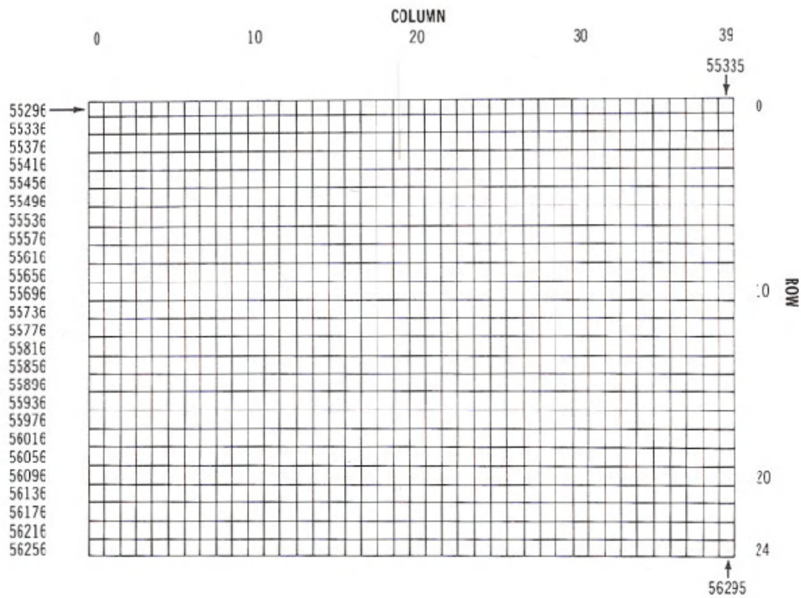


The actual values to POKE into a color memory location to change a character's color are:

0	BLACK
1	WHITE
2	RED
3	CYAN
4	PURPLE
5	GREEN
6	BLUE
7	YELLOW
8	ORANGE
9	BROWN
10	Light RED
11	GRAY 1
12	GRAY 2
13	Light GREEN
14	Light BLUE
15	GRAY 3

For example, to change the color of a character located at the upper left-hand corner of the screen to red, type: POKE 55296,2.

COLOR MEMORY MAP



APPENDIX H

DERIVING MATHEMATICAL FUNCTIONS

Functions that are not intrinsic to Commodore 64 BASIC may be calculated as follows:

FUNCTION	BASIC EQUIVALENT
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(−X*X+1))
INVERSE COSINE	ARCCOS(X)=−ATN(X/SQR (−X*X +1)) +π/2
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X−1))
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X−1)) +(SGN(X)−1*π/2
INVERSE COTANGENT	ARCOT(X)=ATN(X)+π/2
HYPERBOLIC SINE	SINH(X)=(EXP(X)−EXP(−X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(−X))/2
HYPERBOLIC TANGENT	TANH(X)=EXP(−X)/(EXP(x)+EXP (−X))*2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(−X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)−EXP(−X))
HYPERBOLIC COTANGENT	COTH(X)=EXP(−X)/(EXP(X) −EXP(−X))*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X−1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1−X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR (−X*X+1)+1/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR (X*X+1/x)
INVERSE HYPERBOLIC COTAN-GENT	ARCCOTH(X)=LOG((X+1)/(x−1))/2

APPENDIX I

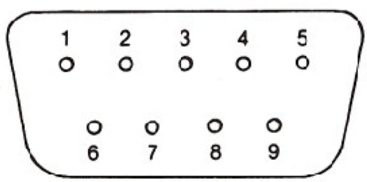
PINOUTS FOR INPUT/OUTPUT DEVICES

This appendix is designed to show you what connections may be made to the Commodore 64.

1) Game I/O
2) Cartridge Slot
3) Audio/Video
4) Serial I/O (Disk/Printer)
5) Modulator Output
6) Cassette
7) User Port

Control Port 1

Pin	Type	Note
1	JOYA0	
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY	
6	BUTTON A/LP	
7	+5V	MAX. 50mA
8	GND	
9	POT AX	



Control Port 2

Pin	Type	Note
1	JOYB0	
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY	
6	BUTTON B	
7	+5V	MAX. 50mA
8	GND	
9	POT BX	

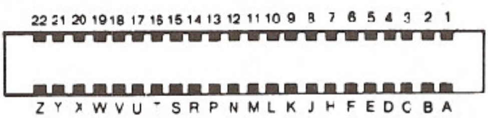
Cartridge Expansion Slot

Pin	Type
12	BA
13	$\overline{\text{DMA}}$
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

Pin	Type
1	GND
2	+5V
3	+5V
4	$\overline{\text{IRQ}}$
5	R/$\overline{\text{W}}$
6	Dot Clock
7	I/O 1
8	$\overline{\text{GAME}}$
9	$\overline{\text{EXROM}}$
10	I/O 2
11	$\overline{\text{ROML}}$

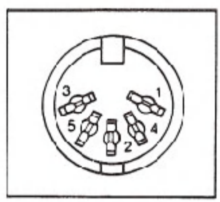
Pin	Type
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

Pin	Type
A	GND
B	$\overline{\text{ROMH}}$
C	$\overline{\text{RESET}}$
D	$\overline{\text{NMI}}$
E	S 02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10



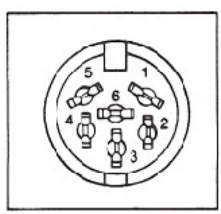
Audio/Video

Pin	Type	Note
1	LUMINANCE	
2	GND	
3	AUDIO OUT	
4	VIDEO OUT	
5	AUDIO IN	



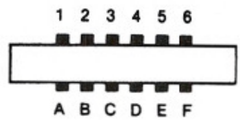
Serial I/O

Pin	Type
1	SERIAL $\overline{\text{SRQIN}}$
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	$\overline{\text{RESET}}$



Cassette

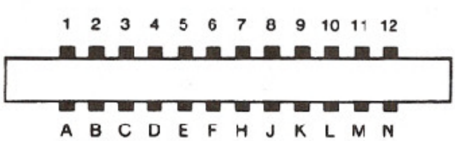
Pin	Type
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE



User I/O

Pin	Type	Note
1	GND	
2	+5V	MAX. 100 mA
3	$\overline{\text{RESET}}$	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	$\overline{\text{PC2}}$	
9	SER. ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	

Pin	Type	Note
A	GND	
B	$\overline{\text{FLAG2}}$	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	



APPENDIX J

PROGRAMS TO TRY

We've included a number of useful programs for you to try with your Commodore 64. These programs will prove both entertaining and useful.

```
100 print"Øjotto        jim butterfield"
120 input"Øwant instructions";z$:ifasc(z$)=78goto250
130 print"Øtry to guess the mystery 5-letter word"
140 print"Øyou must guess only legal 5-letter"
150 print"words, too..."
160 print"you will be told the number of matches"
170 print"(or 'jots') of your guess."
180 print"Øhint: the trick is to vary slightly"
190 print"  from one guess to the next; so that"
200 print"  if you guess 'batch' and get 2 jots"
210 print"  you might try 'botch' or 'chart'"
220 print"  for the next guess..."
250 data bxbsf,ipccz,dbdif,esfbe,pggbm
260 data hpshf,ibudi,djwjm,kpmmz,lbzbl
270 data sbkbi,mfwfm,njnjd,boofy,qjcfs
280 data rvftu,sjwfs,qsftt,puufs,fwfou
290 data xfbwf,fyupm,nvtiz,afcsb,gjaaz
300 data uijdl,esvol,gmppe,ujhfs,gblfs
310 data cppui,mzjoh,trvbu,hbvaf,pxjoh
320 data uisff,tjhiu,bymft,hsvnq,bsfob
330 data rvbsu,dsffq,cfmdi,qsftt,tqbsl
340 data sbebs,svsbm,tnfmm,gspxo,esjgu
400 n=50
410 dim n$(n),z(5),y(5)
420 for j=1ton:readn$(j):nextj
430 t=ti
440 t=t/1000:ift>=1thengoto440
450 z=rnd(-t)
500 g=0:n$=n$(rnd(1)*n+1)
510 print "Øi have a five letter word:":ifr>0goto560
520 print "guess (with legal words)"
530 print "and i'll tell you how many"
540 print "'jots', or matching letters,"
550 print "you have...."
560 g=g+1:input "your word";z$
570 if len(z$)<>5thenprint"you must guess a
    5-letter word!":goto560
580 v=0:h=0:m=0
590 forj=1to5
600 z=asc(mid$(z$,j,1)):y=asc(mid$(n$,j,1))-1:ify=64theny=90
610 ifz<65orz>90thenprint"that's not a word!":goto560
620 ifz=65orz=69orz=73orz=79orz=85orz=89thenv=v+1
630 ifz=ythenm=m+1
640 z(j)=z:y(j)=y:nextj
650 ifm=5goto800
660 ifv=0orv=5thenprint"come on..what kind of
    a word is that?":goto560
670 for j=1to5:y=y(j)
680 for k=1to5:ify=z(k)thenh=h+1:z(k)=0:goto700
690 next k
700 next j
710 print"■▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯";H;"JOTS"
720 ifg<30goto560
730 print"i'd.better tell you.. word was '";
740 forj=1to5:printchr$(y(j));:nextj
750 print"'":goto810
800 print"you got it in only";g;"guesses."
810 input"Øanother word";z$
820 r=1:ifasc(z$)<>78goto500
```

```
1 rem *** sequence
2 rem
3 rem *** from pet user group
4 rem *** software exchange
5 rem *** po box 371
6 rem *** montgomeryville, pa 18936
7 rem
50 dim a$(26)
100 z$="abcdefghijklmnopqrstuvwxyz"
110 z1$="12345678901234567890123456"
200 print"█▮▮enter length of string to be sequenced▮"
220 input "maximum length is 26 ";s%
230 if s%<1 or s%>26 then 200
240 s=s%
300 for i=1 to s
310 a$(i)=mid$(z$,i,1)
320 next i
400 rem randomize string
420 for i=1 to s
430 k=int(rnd(1)*s+1)
440 t$=a$(i)
450 a$(i)=a$(k)
460 a$(k)=t$
470 next i
480 gosub 950
595 t=0
600 rem reverse substring
605 t=t+1
610 input "how many to reverse ";r%
620 if r%=0 goto 900
630 if r%>0 and r%<=s goto 650
640 print "must be between 1 and ";s: goto 610
650 r=int(r%/2)
660 for i=1 to r
670 t$=a$(i)
680 a$(i)=a$(r%-i+1)
690 a$(r%-i+1)=t$
700 next i
750 gosub 950
800 c=1: for i=2 to s
810 if a$(i)>a$(i-1) goto 830
820 c=0
830 next i
840 if c=0 goto 600
850 print "▮you did it in ";t;" tries"
900 rem check for another game
910 input "▮want to play again ";y$
920 if left$(y$,1)="y" or y$="ok" or y$="1" goto 200
930 end
950 print
960 print left$(z1$,s)
970 for i=1 to s: print a$(i);:next i
980 print "▮"
990 return
```

This program courtesy of Gene Deals

```
 90 REM PIANO KEYBOARD
100 PRINT"■ ■ ■ ■ | ■ ■ ■ | ■ ■ | ■ ■ "
110 PRINT" ■ ■ ■ | ■ ■ ■ | ■ ■ | ■ ■ "
120 PRINT" ■ ■ ■ | ■ ■ ■ | ■ ■ | ■ ■ "
130 PRINT" ■ | | | | | | | | | | | | "
140 PRINT" ■Q |W |E |R |T |Y |U |I |O |P |@ |* |↑"
150 PRINT"■'SPACE' FOR SOLO OR POLYPHONIC'
160 PRINT"■'F1,F3,F5,F7' OCTAVE SELECTION"
170 PRINT"■'F2,F4.F6,F8' WAVEFORM■"
180 PRINT"HANG ON, SETTING UP FREQUENCY TABLE..."
190 S=13*4096+1024:DIMF(26):DIMK(255)
200 FORI=0TO28:POKES+I,0:NEXT
210 F1=7040:FORI=1TO26:F(27-I)=F1*5.8+30:F1=F1/2↑(1/12):NEXT
220 K$="Q2W3ER5T6Y7UI9O0P@-*£↑"
230 FORI=1TOLEN(K$):K(ASC(MID$(K$,I)))=I:NEXT
240 PRINT"■                                         "
250 AT=0:DE=0:SU=15:RE=9:SV=SU*16+RE:AV=AT*16+DE:
    WV=16:W=0:M=1:OC=4:HB=256:Z=0
260 FORI=0TO2:POKES+5+I*7,AT*16+DE:POKES+6+I*7,SU*16+RE
270 POKES+2+I*7,4000AND255:POKES+3+I*7,4000/256:NEXT
280 POKES+24,15:REM+16+64:POKES+23,7
300 GETA$:IFA$=""THEN300
310 FR=F(K(ASC(A$)))/M:T=V*7:CR=S+T+4:IFFR=ZTHEN500
320 POKES+6+T,Z:REM FINISH DEC/SUS
325 POKES+5+T,Z:REM FINISH ATT/REL
330 POKECR,8:POKECR,0:REM FIX OFF
340 POKES+T,FR-HB*INT(FR/HB):REM SET LO
350 POKES+1+T,FR/HB:REM SET HI
360 POKES+6+T,SV:REM SET DEC/SUS
365 POKES+5+T,AV:REM SET ATT/REL
370 POKECR,WV+1:FORI=1TO50*AT:NEXT
375 POKECR,WV:REM PULSE
380 IFP=1THENV=V+1:IFV=3THENV=0
400 GOTO300
500 IFA$="■"THENM=1:OC=4:GOTO300
510 IFA$="■"THENM=2:OC=3:GOTO300
520 IFA$="■"THENM=4:OC=2:GOTO300
530 IFA$="■"THENM=8:OC=1:GOTO300
540 IFA$="■"THENW=0:WV=16:GOTO300
550 IFA$="■"THENW=1:WV=32:GOTO300
560 IFA$="■"THENW=2:WV=64:GOTO300
570 IFA$="■"THENW=3:WV=128:GOTO300
580 IFA$=" "THENP=1-P:GOTO300
590 IFA$="■"THEN200
600 GOTO300
800 PRINT"HIT A KEY"
810 GETA$:IFA$=""THEN810:WAIT FOR A KEY
820 PRINTA$:RETURN
```

NOTES:

Line 100 uses (SHIFT CLR/HOME), (CTRL 9),(CTRL]),(SHIFT B).
Line 150 uses (CRSR DOWN)
Line 240 uses (CRSR UP)
Line 500 uses (f1)
Line 510 uses (f3)
Line 520 uses (f5)
Line 530 uses (f7)
Line 540 uses (f2)
Line 550 uses (f4)
Line 560 uses (f6)
Line 570 uses (f8)
Line 590 uses (SHIFT CLR/HOME)

APPENDIX K

CONVERTING STANDARD BASIC PROGRAMS TO COMMODORE 64 BASIC

If you have programs written in a BASIC other than Commodore BASIC, some minor adjustments may be necessary before running them on the Commodore-64. We've included some hints to make the conversion easier.

String Dimensions

Delete all statements that are used to declare the length of strings. A statement such as DIM A$(I,J), which dimensions a string array for J elements of length I, should be converted to the Commodore BASIC statement DIM A$(J).

Some BASICs use a comma or ampersand for string concatenation. Each of these must be changed to a plus sign, which is the Commodore BASIC operator for string concatenation.

In Commodore-64 BASIC, the MID$, RIGHT$, and LEFT$ functions are used to take substrings of strings. Forms such as A$(I) to access the Ith character in A$, or A$(I,J) to take a substring of A$ from position I to J, must be changed as follows:

Other BASIC	Commodore 64 BASIC
A$(I) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,I+1)
A$(I,J) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,J+1)

Multiple Assignments

To set B and C equal to zero, some BASICs allow statements of the form:

```
1Ø LET B=C=Ø
```

Commodore 64 BASIC would interpret the second equal sign as a logical operator and set B = −1 if C = 0. Instead, convert this statement to:

```
1Ø C=Ø : B=Ø
```

Multiple Statements

Some BASICs use a backslash (\ ) to separate multiple statements on a line. With Commodore 64 BASIC, separate all statements by a colon (:).

MAT Functions

Programs using the MAT functions available on some BASICs must be rewritten using FOR. . .NEXT loops to execute properly.

APPENDIX L

ERROR MESSAGES

This appendix contains a complete list of the error messages generated by the Commodore-64, with a description of causes.

BAD DATA String data was received from an open file, but the program was expecting numeric data.

BAD SUBSCRIPT The program was trying to reference an element of an array whose number is outside of the range specified in the DIM statement.

CAN'T CONTINUE The CONT command will not work, either because the program was never RUN, there has been an error, or a line has been edited.

DEVICE NOT PRESENT The required I/O device was not available for an OPEN, CLOSE, CMD, PRINT#, INPUT#, or GET#.

DIVISION BY ZERO Division by zero is a mathematical oddity and not allowed.

EXTRA IGNORED Too many items of data were typed in response to an INPUT statement. Only the first few items were accepted.

FILE NOT FOUND If you were looking for a file on tape, and END-OF-TAPE marker was found. If you were looking on disk, no file with that name exists.

FILE NOT OPEN The file specified in a CLOSE, CMD, PRINT#, INPUT#, or GET#, must first be OPENed.

FILE OPEN An attempt was made to open a file using the number of an already open file.

FORMULA TOO COMPLEX The string expression being evaluated should be split into at least two parts for the system to work with, or a formula has too many parentheses.

ILLEGAL DIRECT The INPUT statement can only be used within a program, and not in direct mode.

ILLEGAL QUANTITY A number used as the argument of a function or statement is out of the allowable range.

LOAD There is a problem with the program on tape.

NEXT WITHOUT FOR This is caused by either incorrectly nesting loops or having a variable name in a NEXT statement that doesn't correspond with one in a FOR statement.

NOT INPUT FILE An attempt was made to INPUT or GET data from a file which was specified to be for output only.

NOT OUTPUT FILE An attempt was made to PRINT data to a file which was specified as input only.

OUT OF DATA A READ statement was executed but there is no data left unREAD in a DATA statement.

OUT OF MEMORY There is no more RAM available for program or variables. This may also occur when too many FOR loops have been nested, or when there are too many GOSUBs in effect.

OVERFLOW The result of a computation is larger than the largest number allowed, which is 1.70141884E+38.

REDIM'D ARRAY An array may only be DIMensioned once. If an array variable is used before that array is DIM'd, an automatic DIM operation is performed on that array setting the number of elements to ten, and any subsequent DIMs will cause this error.

REDO FROM START Character data was typed in during an INPUT statement when numeric data was expected. Just re-type the entry so that it is correct, and the program will continue by itself.

RETURN WITHOUT GOSUB A RETURN statement was encountered, and no GOSUB command has been issued.

STRING TOO LONG A string can contain up to 255 characters.

?SYNTAX ERROR A statement is unrecognizable by the Commodore 64. A missing or extra parenthesis, misspelled keywords, etc.

TYPE MISMATCH This error occurs when a number is used in place of a string, or vice-versa.

UNDEF'D FUNCTION A user defined function was referenced, but it has never been defined using the DEF FN statement.

UNDEF'D STATEMENT An attempt was made to GOTO or GOSUB or RUN a line number that doesn't exist.

VERIFY The program on tape or disk does not match the program currently in memory.

APPENDIX M

MUSIC NOTE VALUES

This appendix contains a complete list of Note#, actual note, and the values to be POKEd into the HI FREQ and LOW FREQ registers of the sound chip to produce the indicated note.

MUSICAL NOTE		OSCILLATOR FREQ		
NOTE	OCTAVE	DECIMAL	HI	LOW
0	C–0	268	1	12
1	C#–0	284	1	28
2	D–0	301	1	45
3	D#–0	318	1	62
4	E–0	337	1	81
5	F–0	358	1	102
6	F#–0	379	1	123
7	G–0	401	1	145
8	G#–0	425	1	169
9	A–0	451	1	195
10	A#–0	477	1	221
11	B–0	506	1	250
16	C–1	536	2	24
17	C#–1	568	2	56
18	D–1	602	2	90
19	D#–1	637	2	125
20	E–1	675	2	163
21	F–1	716	2	204
22	F#–1	758	2	246
23	G–1	803	3	35
24	G#–1	851	3	83
25	A–1	902	3	134
26	A#–1	955	3	187
27	B–1	1012	3	244
32	C–2	1072	4	48

MUSICAL NOTE		OSCILLATOR FREQ		
NOTE	OCTAVE	DECIMAL	HI	LOW
33	C#–2	1136	4	112
34	D–2	1204	4	180
35	D#–2	1275	4	251
36	E–2	1351	5	71
37	F–2	1432	5	152
38	F#–2	1517	5	237
39	G–2	1607	6	71
40	G#–2	1703	6	167
41	A–2	1804	7	12
42	A#–2	1911	7	119
43	B–2	2025	7	233
48	C–3	2145	8	97
49	C#–3	2273	8	225
50	D–3	2408	9	104
51	D#–3	2551	9	247
52	E 3	2703	10	143
53	F–3	2864	11	48
54	F#–3	3034	11	218
55	G–3	3215	12	143
56	G#–3	3406	13	78
57	A–3	3608	14	24
58	A#–3	3823	14	239
59	B–3	4050	15	210
64	C–4	4291	16	195
65	C#–4	4547	17	195
66	D–4	4817	18	209
67	D#–4	5103	19	239
68	E–4	5407	21	31
69	F–4	5728	22	96
70	F#–4	6069	23	181
71	G–4	6430	25	30
72	G#–4	6812	26	156
73	A–4	7217	28	49
74	A#–4	7647	29	223
75	B–4	8101	31	165
80	C–5	8583	33	135
81	C#–5	9094	35	134

MUSICAL NOTE		OSCILLATOR FREQ		
NOTE	OCTAVE	DECIMAL	HI	LOW
82	C–0	9634	37	162
83	C#–0	10207	39	223
84	D–0	10814	42	62
85	F–5	11457	44	193
86	F#–5	12139	47	107
87	G–5	12860	50	60
88	G#–5	13625	53	57
89	A–5	14435	56	99
90	A#–5	15294	59	190
91	B–5	16203	63	75
96	C–6	17167	67	15
97	C#–6	18188	71	12
98	D–6	19269	75	69
99	D#–6	20415	79	191
100	E–6	21629	84	125
101	F–6	22915	89	131
102	F#–6	24278	94	214
103	G–6	25721	100	121
104	G#–6	27251	106	115
105	A–6	28871	112	199
106	A#–6	30588	119	124
107	B–6	32407	126	151
112	C–7	34334	134	30
113	C#–7	36376	142	24
114	D–7	38539	150	139
115	D#–7	40830	159	126
116	E–7	43258	168	250
117	F–7	45830	179	6
118	F#–7	48556	189	172
119	G–7	51443	200	243
120	G#–7	54502	212	230
121	A–7	57743	225	143
122	A#–7	61176	238	248
123	B–7	64814	253	46

FILTER SETTINGS

Location	Contents
54293	Low cutoff frequency (0–7)
54294	High cutoff frequency (0–255)
54295	Resonance (bits 4–7) Filter voice 3 (bit 2) Filter voice 2 (bit 1) Filter voice 1 (bit 0)
54296	High pass (bit 6) Bandpass (bit 5) Low pass (bit 4) Volume (bits 0–3)

APPENDIX N

BIBLIOGRAPHY

Addison-Wesley	"BASIC and the Personal Computer", Dwyer and Critchfield
Compute	"Compute's First Book of PET/CBM"
Cowbay Computing	"Feed Me, I'm Your PET Computer", Carol Alexander
	"Looking Good with Your PET", Carol Alexander
	"Teacher's PET—Plans, Quizzes, and Answers"
Creative Computing	"Getting Acquainted With Your VIC 20", T. Hartnell
Dilithium Press	"BASIC Basic-English Dictionary for the PET", Larry Noonan
	"PET BASIC", Tom Rugg and Phil Feldman
Faulk Baker Associates	"MOS Programming Manual", MOS Technology
Hayden Book Co.	"BASIC From the Ground Up", David E. Simon
	"I Speak BASIC to My PET", Aubrey Jones, Jr.
	"Library of PET Subroutines", Nick Hampshire
	"PET Graphics", Nick Hampshire
	"BASIC Conversions Handbook, Apple, TRS-80, and PET", David A. Brain, Phillip R. Oviatt, Paul J. Paquin, and Chandler P. Stone

Howard W. Sams	"The Howard W. Sams Crash Course in Microcomputers", Louis E. Frenzel, Jr.
	"Mostly BASIC: Applications for Your PET", Howard Berenbon
	"PET Interfacing", James M. Downey and Steven M. Rogers
	"VIC 20 Programmer's Reference Guide", A. Finkel, P. Higginbottom, N. Harris, and M. Tomczyk
Little, Brown & Co.	"Computer Games for Businesses, Schools, and Homes", J. Victor Nagigian, and William S. Hodges
	"The Computer Tutor: Learning Activities for Homes and Schools", Gary W. Orwig, University of Central Florida, and William S. Hodges
McGraw-Hill	"Hands-On BASIC With a PET", Herbert D. Peckman
	"Home and Office Use of VisiCalc", D. Castlewitz, and L. Chisauki
Osborne/McGraw-Hill	"PET/CBM Personal Computer Guide", Carroll S. Donahue
	"PET Fun and Games", R. Jeffries and G. Fisher
	"PET and the IEEE", A. Osborne and C. Donahue
	"Some Common BASIC Programs for the PET", L. Poole, M. Borchers, and C. Donahue
	"Osborne CP/M User Guide", Thom Hogan
	"CBM Professional Computer Guide"
	"The PET Personal Guide"
	"The 8086 Book", Russell Rector and George Alexy
P. C. Publications	"Beginning Self-Teaching Computer Lessons"

Prentice-Hall	"The PET Personal Computer for Beginners", S. Dunn and V. Morgan
Reston Publishing Co.	"PET and the IEEE 488 Bus (GPIB)", Eugene Fisher and C. W. Jensen
	"PET BASIC—Training Your PET Computer", Ramon Zamora, Wm. F. Carrie, and B. Allbrecht
	"PET Games and Recreation", M. Ogelsby, L. Lindsey, and D. Kunkin
	"PET BASIC", Richard Huskell
	"VIC Games and Recreation"
Telmas Courseware Ratings	"BASIC and the Personal Computer", T. A. Dwyer, and M. Critchfield
Total Information Services	"Understanding Your PET/CBM, Vol. 1, BASIC Programming"
	"Understanding Your VIC", David Schultz

Commodore Magazines provide you with the most up-to-date information for your Commodore 64. Two of the most popular publications that you should seriously consider subscribing to are:

COMMODORE—*The Microcomputer Magazine* is published bi-monthly and is available by subscription ($15.00 per year, U.S., and $25.00 per year, worldwide).

POWER/PLAY—*The Home Computer Magazine* is published quarterly and is available by subscription ($10.00 per year, U.S., and $15.00 per year worldwide).

APPENDIX O

SPRITE REGISTER MAP

Register # Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7							S0X0	SPRITE 0 X Component
1	1	S0Y7							S0Y0	SPRITE 0 Y Component
2	2	S1X7							S1X0	SPRITE 1 X
3	3	S1Y7							S1Y0	SPRITE 1 Y
4	4	S2X7							S2X0	SPRITE 2 X
5	5	S2Y7							S2Y0	SPRITE 2 Y
6	6	S3X7							S3X0	SPRITE 3 X
7	7	S3Y7							S3Y0	SPRITE 3 Y
8	8	S4X7							S4X0	SPRITE 4 X
9	9	S4Y7							S4Y0	SPRITE 4 Y
10	A	S5X7							S5X0	SPRITE 5 X
11	B	S5Y7							S5Y0	SPRITE 5 Y
12	C	S6X7							S6X0	SPRITE 6 X
13	D	S6Y7							S6Y0	SPRITE 6 Y
14	E	S7X7							S7X0	SPRITE 7 X Component
15	F	S7Y7							S7Y0	SPRITE 7 Y Component
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	MSB of X COORD.
17	11	RC8	ECM	BMM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	Y SCROLL MODE
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	RASTER
19	13	LPX7							LPX0	LIGHT PEN X
20	14	LPY7							LPY0	LIGHT PEN Y

Register # Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
21	15	SE7							SE0	SPRITE ENABLE (ON/OFF)
22	16	N.C.	N.C.	RST	MCM	CSEL	XSCL2	XSCL1	XSCL0	X SCROLL MODE
23	17	SEXY7							SEXY0	SPRITE EXPAND Y
24	18	VS13	VS12	VS11	VS10	CB13	CB12	CB11	N.C.	SCREEN Character Memory
25	19	IRQ	N.C.	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Interupt Request's
26	1A	N.C.	N.C.	N.C.	N.C.	MLPI	MISSC	MISBC	MRIRQ	Interupt Request MASKS
27	1B	BSP7							BSP0	Background-Sprite PRIORITY
28	1C	SCM7							SCM0	MULTICOLOR SPRITE SELECT
29	1D	SEXX7							SEXX0	SPRITE EXPAND X
30	1E	SSC7							SSC0	Sprite-Sprite COLLISION
31	1F	SBC7							SBC0	Sprite-Background COLLISION

				COLOR CODES	DEC	HEX			COLOR
32	20	0	0	BLACK	EXT 1				EXTERIOR COL
33	21	1	1	WHITE	BKGD0				
34	22	2	2	RED	BKGD1				
35	23	3	3	CYAN	BKGD2				
36	24	4	4	PURPLE	BKGD3				
37	25	5	5	GREEN	SMC 0				SPRITE MULTICOLOR 0
38	26	6	6	BLUE	SMC 1				1
39	27	7	7	YELLOW	S0COL				SPRITE 0 COLOR
40	28	8	8	ORANGE	S1COL				1
41	29	9	9	BROWN	S2COL				2
42	2A	10	A	LT RED	S3COL				3
43	2B	11	B	GRAY 1	S4COL				4
44	2C	12	C	GRAY 2	S5COL				5
45	2D	13	D	LT GREEN	S6COL				6
46	2E	14	E	LT BLUE	S7COL				7
		15	F	GRAY 3					

LEGEND:
ONLY COLORS 0-7 MAY BE USED IN MULTICOLOR CHARACTER MODE

APPENDIX P

COMMODORE 64 SOUND CONTROL SETTINGS

This handy table gives you the key numbers you need to use in your sound programs, according to which of the Commodore 64's 3 voices you want to use. To set or adjust a sound control in your BASIC program, just POKE the number from the second column, followed by a comma (,) and a number from the chart . . . like this: POKE 54276,17 (Selects a Triangle Waveform for VOICE 1).

Remember that you must set the VOLUME before you can generate sound. POKE54296 followed by a number from 0 to 15 sets the volume for all 3 voices.

It takes 2 separate POKEs to generate each musical note . . . for example POKE54273,34:POKE54272,75 designates low C in the sample scale below.

Also . . . you aren't limited to the numbers shown in the tables. If 34 doesn't sound "right" for a low C, try 35. To provide a higher SUSTAIN or ATTACK rate than those shown, add two or more SUSTAIN numbers together. (Examples: POKE54277,96 combines two attack rates (32 and 64) for a combined higher attack rate . . . but . . . POKE54277,20 provides a low attack rate (16) and a medium decay rate (4).

SETTING VOLUME—SAME FOR ALL 3 VOICES

VOLUME CONTROL	POKE54296	Settings range from 0 (off) to 15 (loudest)

VOICE NUMBER 1

TO CONTROL THIS SETTING:	POKE THIS NUMBER:	FOLLOWED BY ONE OF THESE NUMBERS (0 to 15 . . . or . . . 0 to 255 depending on range)

TO PLAY A NOTE		C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#
HIGH FREQUENCY	54273	34	36	38	40	43	45	48	51	54	57	61	64	68	72
LOW FREQUENCY	54272	75	85	126	200	52	198	127	97	111	172	126	188	149	169

WAVEFORM	POKE	TRIANGLE	SAWTOOTH	PULSE	NOISE
	54276	17	33	65	129

PULSE RATE (Pulse Waveform)		
HI PULSE	54275	A value of 0 to 15 (for Pulse waveform only)
LO PULSE	54274	A value of 0 to 255 (for Pulse waveform only)

ATTACK/DECAY	POKE	ATK4	ATK3	ATK2	ATK1	DEC4	DEC3	DEC2	DEC1
	54277	128	64	32	16	8	4	2	1

SUSTAIN/RELEASE	POKE	SUS4	SUS3	SUS2	SUS1	REL4	REL3	REL2	REL1
	54278	128	64	32	16	8	4	2	1

VOICE NUMBER 2

TO PLAY A NOTE		C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#
HIGH FREQUENCY	54280	34	36	38	40	43	45	48	51	54	57	61	64	68	72
LOW FREQUENCY	54279	75	85	126	200	52	198	127	97	111	172	126	188	149	169

WAVEFORM	POKE	TRIANGLE	SAWTOOTH	PULSE	NOISE
	54283	17	33	65	129

PULSE RATE		
HI PULSE	54282	A value of 0 to 15 (for Pulse waveform only)
LO PULSE	54281	A value of 0 to 255 (for Pulse waveform only)

ATTACK/DECAY	POKE	ATK4	ATK3	ATK2	ATK1	DEC4	DEC3	DEC2	DEC1
	54284	128	64	32	16	8	4	2	1

SUSTAIN/RELEASE	POKE	SUS4	SUS3	SUS2	SUS1	REL4	REL3	REL2	REL1
	54285	128	64	32	16	8	4	2	1

VOICE NUMBER 3

TO PLAY A NOTE	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#
HIGH FREQUENCY	54287 34	36	38	40	43	45	48	51	54	57	61	64	68	72
LOW FREQUENCY	54286 75	85	126	200	52	198	127	97	111	172	126	188	149	169

WAVEFORM	POKE	TRIANGLE	SAWTOOTH	PULSE	NOISE
	54290	17	33	65	129

PULSE RATE		
HI PULSE	54289	A value of 0 to 15 (for Pulse waveform only)
LO PULSE	54288	A value of 0 to 255 (for Pulse waveform only)

ATTACK/DECAY	POKE	ATK4	ATK3	ATK2	ATK1	DEC4	DEC3	DEC2	DEC1
	54291	128	64	32	16	8	4	2	1

SUSTAIN/RELEASE	POKE	SUS4	SUS3	SUS2	SUS1	REL4	REL3	REL2	REL1
	54292	128	64	32	16	8	4	2	1

TRY THESE SETTINGS TO SIMULATE DIFFERENT INSTRUMENTS

Instrument	Waveform	Attack/Decay	Sustain/Release	Pulse Rate
Piano	Pulse	9	0	Hi-0, Lo-255
Flute	Triange	96	0	Not applicable
Harpsichord	Sawtooth	9	0	Not applicable
Xylophone	Triangle	9	0	Not applicable
Organ	Triangle	0	240	Not applicable
Colliape	Triangle	0	240	Not applicable
Accordian	Triangle	102	0	Not applicable
Trumpet	Sawtooth	96	0	Not applicable

MEANINGS OF SOUND TERMS

ADSR—Attack/Decay/Sustain/Release

Attack—rate sound rises to peak volume

Decay—rate sound falls from peak volume to Sustain level

Sustain—prolong note at certain volume

Release—rate at which volume falls from Sustain level

Waveform—"shape" of sound wave

Pulse—tone quality of Pulse Waveform

NOTE: Attack/Decay and Sustain/Release settings should always be POKEd in your program **BEFORE** the Waveform is POKEd.

INDEX

A

Abbreviations, BASIC commands, 130, 131
Accessories, viii, 106-108
Addition, 23, 26-27, 113
AND operator, 114
Animation, 43-44, 65-66, 69-75, 132, 138-139
Arithmetic, Operators, 23, 26-27, 113-114
Arithmetic, Formulas, 23, 26-27, 113, 120, 140
Arrays, 95-103
ASC function, 128, 135-137
ASCII character codes, 135-137

B

BASIC
  abbreviations, 130-131
  commands, 114-117
  numeric functions, 125-127
  operators, 113-114
  other functions, 129
  statements, 117-125
  string functions, 128
  variables, 112-113
Bibliography, 156-158
Binary arithmetic, 75-77
Bit, 75-76
Business aids, 108
Byte, 76

C

Calculations, 22-29
Cassette tape recorder (audio), viii, 3, 18-20, 21
Cassette tape recorder (video), 7
Cassette, port 3
CHR$ function, 36-37, 46-47, 53, 58-60, 113, 128, 135-137, 148
CLR statement, 117
CLR/HOME key, 15
Clock, 113
CLOSE statement, 117
Color
  adjustment, 11-12
  CHR$ codes, 58
  keys, 56-57
  memory map, 64, 139
  PEEKS and POKES, 60-61
  screen and border, 60-63, 138
Commands, BASIC, 114-117
Commodore key, (see graphics keys)
Connections
  optional, 6-7
  rear, 2-3
  side panel, 2
  TV/Monitor, 3-5
CONT command, 114
ConTRL key, 11, 16
COSine function, 126
CuRSoR keys, 10, 15
Correcting errors, 34
Cursor, 10

D

DATASSETTE recorder, (see cassette tape recorder)
Data, loading and saving (disk), 18-21
Data, loading and saving (tape), 18-21
DATA statement, 92-94, 118
DEFine statement, 118
Delay loop, 61, 65
DELete key, 15
DIMension statement, 118-119
Division, 23, 26, 27, 113
Duration, (see For . . . Next)

E

Editing programs, 15, 34
END statement, 119
Equal, not-equal-to, signs, 23, 26-27, 114
Equations, 114
Error messages, 22-23, 150-151
Expansion port, 141-142
EXPonent function, 126
Exponentiation, 25-27, 113

F

Files, (DATASSETTE), 21, 110-111
Files, (disk), 21, 110-111
FOR statement, 119
FRE function, 129
Functions, 125-129

G

Game controls and ports, 2-3, 141
GET statement, 47-48, 119-120
GET# statement, 120
Getting started, 13-29
GOSUB statement, 120
GOTO (GO TO) statement, 32-34, 120

Graphic keys, 17, 56-57, 61, 132-137
Graphic symbols, (see graphic keys)
Greater than, 114

H

Hyperbolic functions, 140

I

IEEE-488 Interface, 2-3, 141
IF . . . THEN statement, 37-39, 120-121
INPUT statement, 45-47, 121
INPUT#, 121
INSert key, 15
INTeger function, 126
Integer variable, 112
I/O pinouts, 141-143
I/O ports, 2-7, 141-143

J

Joysticks, 2-3, 141

K

Keyboard, 14-17

L

LEFT$ function, 128
LENgth function, 128
Less than, 114
LET statement, 121
LIST command, 33-34, 115
LOAD command, 115
LOADing programs on tape, 18-20
LOGarithm function, 126
Loops, 39-40, 43-45
Lower case characters, 14-17

M

Mathematics
  formulas, 23-27
  function table, 140
  symbols, 24-27, 38, 114
Memory expansion, 2-4, 142
Memory maps, 62-65
MID$ function, 128
Modulator, RF, 4-7
Multiplication, 24, 113
Music, 79-90

N

Names
  program, 18-21
  variable, 34-37
NEW command, 115
NEXT statement, 121-122
NOT operator, 114
Numeric variables, 36-37

O

ON statement, 122
OPEN statement, 122
Operators
  arithmetic, 113
  logical, 114
  relational, 114

P

Parentheses, 28
PEEK function, 60-62
Peripherals, viii, 2-8, 107-109
POKE statement, 60-61
Ports, I/O, 2-3, 141-143
POS function, 129
PRINT statement, 23-29, 123-124
PRINT#, 124
Programs
  editing, 15, 34
  line numbering, 32-33
  loading/saving (DATASSETTE), 18-21
  loading/saving (disk), 18-21
Prompt, 45

Q

Quotation marks, 22

R

RaNDom function, 48-53, 126
Random numbers, 48-53
READ statement, 124
REMark statement, 124
Reserved words, (see Command statements)
Restore key, 15, 18
RESTORE statement, 124
Return key, 15, 18
RETURN statement, 124
RIGHT$ function, 128
RUN command, 116
RUN/STOP key, 16-17

S

SAVE command, 21, 116
Saving programs (DATASSETTE), 21
Saving programs (disk), 21
Screen memory maps, 62-63, 138
SGN, function, 127
Shift key, 14-15, 17
SINe function, 127
Sound effects, 89-90
SPC function, 129

SPRITE EDITOR, vii, 69-76
SPRITE graphics, vii, 69-76
SQuaRe function, 127
STOP command, 125
STOP key, 16-17
String variables, 36-37, 112-113
STR$ function, 128
Subscripted variables, 95-98, 112-113
Subtraction, 24, 113
Syntax error, 22
SYS statement, 125

T

TAB function, 129
TAN function, 127
TI variable, 113
TI$ variable, 113
Time clock, 113
TV connections, 3-7

U

Upper/Lower Case mode, 14
USR function, 127
User defined function, (see DEF)

V

VALue function, 128
Variables
  array, 95-103, 113
  dimensions, 98-103, 113
  floating point, 95-103, 113
  integer, 95-103, 112
  numeric, 95-103, 112
  string ($), 95-103, 112
VERIFY command, 117
Voice, 80-90, 162-164

W

WAIT command, 125
Writing to tape, 110

Z

Z-80, vii, 108

Commodore hopes you've enjoyed the COMMODORE 64 USER'S GUIDE. Although this manual contains some programming information and tips, it is NOT intended to be a Programmer's Reference Manual. For those of you who are advanced programmers and computer hobbyists Commodore suggests that you consider purchasing the COMMODORE 64 PROGRAMMER'S REFERENCE GUIDE available through your local Commodore dealer.

In addition updates and corrections as well as programming hints and tips are available in the COMMODORE and POWER PLAY magazines, on the COMMODORE database of the COMPUSERVE INFORMATION NETWORK, accessed through a VICMODEM.

COMMODORE 64 QUICK REFERENCE CARD

SIMPLE VARIABLES

Type	Name	Range
Real	XY	±1.70141183E+38
		±2.93873588E−39
Integer	XY%	±32767
String	XY$	0 to 255 characters

X is a letter (A-Z), Y is a letter or number (0-9). Variable names can be more than 2 characters, but only the first two are recognized.

ARRAY VARIABLES

Type	Name
Single Dimension	XY(5)
Two-Dimension	XY(5,5)
Three-Dimension	XY(5,5,5)

Arrays of up to eleven elements (subscripts 0-10) can be used where needed. Arrays with more than eleven elements need to be DIMensioned.

ALGEBRAIC OPERATORS

= Assigns value to variable
− Negation
↑ Exponentiation
* Multiplication
/ Division
+ Addition
− Subtraction

RELATIONAL AND LOGICAL OPERATORS

=	Equal
<>	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
NOT	Logical "Not"
AND	Logical "And"
OR	Logical "Or"

Expression equals 1 if true, 0 if false.

SYSTEM COMMANDS

LOAD "NAME"	Loads a program from tape
SAVE "NAME"	Saves a program on tape
LOAD "NAME",8	Loads a program from disk
SAVE "NAME",8	Saves a program to disk
VERIFY "NAME"	Verifies that program was SAVEd without errors
RUN	Executes a program
RUN xxx	Executes program starting at line xxx
STOP	Halts execution
END	Ends execution
CONT	Continues program execution from line where program was halted
PEEK(X)	Returns contents of memory location X
POKE X,Y	Changes contents of location X to value Y
SYS xxxxx	Jumps to execute a machine language program, starting at xxxxx
WAIT X,Y,Z	Program waits until contents of location X, when EORed with Z and ANDed with Y, is nonzero.
USR(X)	Passes value of X to a machine language subroutine

EDITING AND FORMATTING COMMANDS

LIST	Lists entire program
LIST A-B	Lists from line A to line B
REM Message	Comment message can be listed but is ignored during program execution
TAB(X)	Used in PRINT statements. Spaces X positions on screen
SPC(X)	PRINTs X blanks on line
POS(X)	Returns current cursor position
CLR/HOME	Positions cursor to left corner of screen
SHIFT CLR/HOME	Clears screen and places cursor in "Home" position
SHIFT INST/DEL	Inserts space at current cursor position
INST/DEL	Deletes character at current cursor position
CTRL	When used with numeric color key, selects text color. May be used in PRINT statement.
CRSR Keys	Moves cursor up, down, left, right on screen
Commodore Key	When used with SHIFT selects between upper/lower case and graphic display mode. When used with numeric color key, selects optional text color

ARRAYS AND STRINGS

DIM A(X,Y,Z)	Sets maximum subscripts for A; reserves space for (X+1)*(Y+1)*(Z+1) elements starting at A(0,0,0)
LEN (X$)	Returns number of characters in X$
STR$(X)	Returns numeric value of X, converted to a string
VAL(X$)	Returns numeric value of A$, up to first nonnumeric character
CHR$(X)	Returns ASCII character whose code is X
ASC(X$)	Returns ASCII code for first character of X$
LEFT$(A$,X)	Returns leftmost X characters of A$
RIGHT$(A$,X)	Returns rightmost X characters of A$
MID$(A$,X,Y)	Returns Y characters of A$ starting at character X

INPUT/OUTPUT COMMANDS

INPUT A$ OR A	PRINTs '?' on screen and waits for user to enter a string or value
INPUT "ABC";A	PRINTs message and waits for user to enter value. Can also INPUT A$
GET A$ or A	Waits for user to type one-character value; no RETURN needed
DATA A,"B",C	Initializes a set of values that can be used by READ statement
READ A$ or A	Assigns next DATA value to A$ or A
RESTORE	Resets data pointer to start READing the DATA list again
PRINT "A= ";A	PRINTs string 'A= ' and value of A ';' suppresses spaces - ',' tabs data to next field.

PROGRAM FLOW

GOTO X	Branches to line X
IF A=3 THEN 10	IF assertion is true THEN execute following part of statement. IF false, execute next line number
FOR A=1 TO 10 STEP 2 : NEXT	Executes all statements between FOR and corresponding NEXT, with A going from 1 to 10 by 2. Step size is 1 unless specified
NEXT A	Defines end of loop. A is optional
GOSUB 2000	Branches to subroutine starting at line 2000
RETURN	Marks end of subroutine. Returns to statement following most recent GOSUB
ON X GOTO A,B	Branches to Xth line number on list. If X = 1 branches to A, etc.
ON X GOSUB A,B	Branches to subroutine at Xth line number in list

ABOUT THE COMMODORE 64 USER'S GUIDE . . .

Outstanding color . . . sound synthesis . . . graphics . . . computing capabilities . . . the synergistic marriage of state-of-the-art technologies. These features make the Commodore 64 the most advanced personal computer in its class.

The **Commodore 64 User's Guide** helps you get started in computing, even if you've never used a computer before. Through clear, step-by-step instructions, you are given an insight into the BASIC language and how the Commodore 64 can be put to a myriad of uses.

For those already familiar with microcomputers, the advanced programming sections and appendices explain the enhanced features of the Commodore 64 and how to get the most of these expanded capabilities.

commodore COMPUTER

Commodore Business Machines, Inc.—Computer Systems Division, 487 Devon Park Drive, Wayne, PA 19087.

DISTRIBUTED BY

Howard W. Sams & Co., Inc.

4300 W. 62nd Street, Indianapolis, Indiana 46268 USA

$12.95/22010

ISBN: 0-672-22010-5