



**MICROCHIP**

---

**16-Bit Language Tools Libraries  
Reference Manual**

---

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

**Trademarks**

The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, flexPWR, JukeBlox, KEELOQ, KEELOQ logo, Kleer, LANCheck, MediaLB, MOST, MOST logo, MPLAB, OptoLyzer, PIC, PICSTART, PIC<sup>32</sup> logo, RightTouch, SpyNIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

The Embedded Control Solutions Company and mTouch are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, ECAN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, KleerNet, KleerNet logo, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, RightTouch logo, REAL ICE, SQI, Serial Quad I/O, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademarks of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2004-2014, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-63276-906-0

**QUALITY MANAGEMENT SYSTEM  
CERTIFIED BY DNV  
= ISO/TS 16949 =**

*Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC<sup>®</sup> MCUs and dsPIC<sup>®</sup> DSCs, KEELOQ<sup>®</sup> code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

**Table of Contents**

<b>Preface</b> .....	<b>5</b>
<b>Chapter 1. Library Overview</b>	
1.1 Introduction .....	11
1.2 OMF-Specific Libraries/Start-up Modules .....	12
1.3 Start-up Code .....	12
1.4 DSP Library .....	12
1.5 16-Bit Peripheral Libraries .....	12
1.6 Standard C Libraries with Math and Support Functions .....	13
1.7 Fixed-Point Math Functions .....	13
1.8 Compiler Built-in Functions .....	13
<b>Chapter 2. Standard C Libraries</b>	
2.1 Introduction .....	15
2.2 Using the Standard C Libraries .....	16
2.3 <assert.h> diagnostics .....	17
2.4 <ctype.h> character handling .....	18
2.5 <errno.h> errors .....	27
2.6 <float.h> floating-point characteristics .....	28
2.7 <limits.h> implementation-defined limits .....	33
2.8 <locale.h> localization .....	35
2.9 <setjmp.h> non-local jumps .....	36
2.10 <signal.h> signal handling .....	37
2.11 <stdarg.h> variable argument lists .....	43
2.12 <stddef.h> common definitions .....	45
2.13 <stdio.h> input and output .....	47
2.14 <stdlib.h> utility functions .....	97
2.15 <string.h> string functions .....	122
2.16 <time.h> date and time functions .....	145
<b>Chapter 3. Standard C Libraries - Math Functions</b>	
3.1 Introduction .....	153
3.2 Using the Standard C Libraries .....	153
3.3 <math.h> mathematical functions .....	155
<b>Chapter 4. Standard C Libraries - Support Functions</b>	
4.1 Introduction .....	197
4.2 Using the Support Functions .....	198
4.3 Standard C Library Helper Functions .....	199
4.4 Standard C Library Functions That Require Modification .....	204
4.5 Functions/Constants to Support A Simulated UART .....	205

# 16-Bit Language Tools Libraries

---

4.6 Functions for Erasing and Writing EEDATA Memory .....	208
4.7 Functions for Erasing and Writing Flash Memory .....	211
4.8 Functions for Specialized Copying and Initialization .....	216
<b>Chapter 5. Fixed-Point Math Functions</b>	
5.1 Introduction .....	219
5.2 Overview of Fixed-Point Data Formats .....	220
5.3 Using the Fixed-Point Libraries .....	223
5.4 <libq.h> mathematical functions .....	225
<b>Appendix A. ASCII Character Set .....</b>	<b>245</b>
<b>Index .....</b>	<b>252</b>
<b>Worldwide Sales and Service .....</b>	<b>266</b>

---

---

## Preface

---

---

### NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site ([www.microchip.com](http://www.microchip.com)) to obtain the latest documentation available.

Documents are identified with a “DS” number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is “DSXXXXXXXXA”, where “XXXXXXXX” is the document number and “A” is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

## INTRODUCTION

This chapter contains general information that will be useful to know before using 16-bit libraries. Items discussed include:

- Document Layout
- Conventions Used in this Guide
- Recommended Reading
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support

## DOCUMENT LAYOUT

This document describes how to use GNU language tools to write code for 16-bit applications. The document is organized in the following order:

- **Chapter 1: Library Overview** – gives an overview of libraries. Some are described further in this document, while others are described in other documents or online Help files.
- **Chapter 2: Standard C Libraries** – lists the library functions and macros for standard C operation.
- **Chapter 3: Standard C Libraries – Math Functions** – lists the math functions for standard C operation.
- **Chapter 4: Standard C Libraries – Support Functions** – lists standard C library helper functions.
- **Chapter 5: Fixed-Point Math Functions** – lists the fixed-point library math functions.
- **Appendix A: ASCII Character Set**

# 16-Bit Language Tools Libraries

---

## CONVENTIONS USED IN THIS GUIDE

The following conventions may appear in this documentation:

### DOCUMENTATION CONVENTIONS

Description	Represents	Examples
<b>Arial font:</b>		
Italic	Referenced books	<i>MPLAB<sup>®</sup> IDE User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic with right angle bracket	A menu path	<u><i>File&gt;Save</i></u>
Bold	A dialog button	Click <b>OK</b>
	A tab	Click the <b>Power</b> tab
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
<b>Courier New font:</b>		
Plain	Sample source code	#define START
	Filenames	autoexec.bat
	File paths	c:\mcc18\h
	Keywords	_asm, _endasm, static
	Command-line options	-Opa+, -Opa-
	Bit values	0, 1
	Constants	0xFF, 'A'
Italic	A variable argument	<i>file.o</i> , where <i>file</i> can be any valid filename
Square brackets [ ]	Optional arguments	mpasmwin [options] <i>file</i> [options]
Curly brackets and pipe character: {   }	Choice of mutually exclusive arguments; an OR selection	errorlevel {0 1}
Ellipses...	Replaces repeated text	var_name [, var_name...]
	Represents code supplied by user	void main (void) { ... }

## RECOMMENDED READING

This documentation describes how to use 16-bit libraries. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

### Readme Files

For the latest information on Microchip tools, read the associated Readme files (HTML files) included with the software.

### 16-Bit Language Tools Getting Started (DS70094)

A guide to installing and working with the Microchip language tools for 16-bit devices. Examples using the 16-bit simulator SIM30 (a component of MPLAB SIM) are provided.

### MPLAB<sup>®</sup> Assembler, Linker and Utilities for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide (DS51317)

A guide to using the 16-bit assembler, object linker and various utilities, including the 16-bit archiver/librarian.

### MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide (DS51284)

A guide to using the 16-bit C compiler. The 16-bit linker is used with this tool.

### Device-Specific Documentation

The Microchip website contains many documents that describe 16-bit device functions and features. Among these are:

- Individual and family data sheets
- Family reference manuals
- Programmer's reference manuals

### C Standards Information

American National Standard for Information Systems – *Programming Language – C*. American National Standards Institute (ANSI), 11 West 42nd. Street, New York, New York, 10036.

This standard specifies the form and establishes the interpretation of programs expressed in the programming language C. Its purpose is to promote portability, reliability, maintainability and efficient execution of C language programs on a variety of computing systems.

### C Reference Manuals

Harbison, Samuel P. and Steele, Guy L., *C A Reference Manual*, Fourth Edition, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Second Edition. Prentice Hall, Englewood Cliffs, N.J. 07632.

Kochan, Steven G., *Programming In ANSI C*, Revised Edition. Hayden Books, Indianapolis, Indiana 46268.

Plauger, P.J., *The Standard C Library*, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Van Sickle, Ted., *Programming Microcontrollers in C*, First Edition. LLH Technology Publishing, Eagle Rock, Virginia 24085.

# 16-Bit Language Tools Libraries

---

## THE MICROCHIP WEB SITE

Microchip provides online support via our web site at [www.microchip.com](http://www.microchip.com). This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

## DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at [www.microchip.com](http://www.microchip.com), click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debuggers. This includes MPLAB ICD 3 in-circuit debuggers and PICKit™ 3 debug express.
- **MPLAB® IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include production programmers such as MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger and MPLAB PM3 device programmers. Also included are nonproduction development programmers such as PICSTART® Plus and PICKit 2 and 3.



## CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at:

<http://www.microchip.com/support>.

# 16-Bit Language Tools Libraries

---

NOTES:

---

---

## Chapter 1. Library Overview

---

---

### 1.1 INTRODUCTION

A library is a collection of functions grouped for reference and ease of linking. See the “MPLAB® XC16 Assembler, Linker and Utilities User’s Guide” (DS50002106) for more information about making and using libraries.

#### 1.1.1 Compiler Installation Locations

The majority of the libraries discussed in this manual come with the MPLAB® XC16 C Compiler, which is installed by default in the following locations:

- Windows OS 32-bit - C:\Program Files\Microchip\xc16\x.xx
- Windows OS 64-bit - C:\Program Files (x86)\Microchip\xc16\x.xx
- Mac OS - Applications/microchip/xc16/x.xx
- Linux OS - /opt/microchip/xc16/x.xx

(where *x.xx* is the version number).

#### 1.1.2 Assembly Code Applications

Free versions of the 16-bit language tool libraries are available from the Microchip web site. DSP and 16-bit peripheral libraries are provided with object files and source code. A math library (containing functions from the standard C header file <math.h>) is provided as an object file only. The complete standard C library is provided with the MPLAB XC16 C Compiler.

#### 1.1.3 C Code Applications

The 16-bit language tool libraries are included in the `lib` subdirectory of the MPLAB XC16 C Compiler install directory (**Section 1.1.1 “Compiler Installation Locations”**).

These libraries can be linked directly into an application with a 16-bit linker.

#### 1.1.4 Chapter Organization

This chapter is organized in the following order:

- OMF-Specific Libraries/Start-up Modules
- Start-up Code
- DSP Library
- 16-Bit Peripheral Libraries
- Standard C Libraries with Math and Support Functions
- Fixed-Point Math Functions
- Compiler Built-in Functions

# 16-Bit Language Tools Libraries

---

## 1.2 OMF-SPECIFIC LIBRARIES/START-UP MODULES

Library files and start-up modules are specific to OMF (Object Module Format). An OMF can be one of the following:

- COFF – This is the default.
- ELF – The debugging format used for ELF object files is DWARF 2.0.

There are two ways to select the OMF:

1. Set an environment variable called `XC16_OMF` for all tools.
2. Select the OMF on the command line when invoking the tool, i.e., `-omf=omf` or `-momf=omf`.

16-bit tools will first look for generic library files when building your application (no OMF specification). If these cannot be found, the tools will look at your OMF specifications and determine which library file to use.

As an example, if `libdsp.a` is not found and no environment variable or command-line option is set, the file, `libdsp-coff.a`, will be used by default.

## 1.3 START-UP CODE

In order to initialize variables in data memory, the linker creates a data initialization template. This template must be processed at start-up, before the application proper takes control. For C programs, this function is performed by the start-up modules in `libpic30-coff.a` (either `crt0.o` or `crt1.o`) or `libpic30-elf.a` (either `crt0.eo` or `crt1.eo`). Assembly language programs can utilize these modules directly by linking with the desired start-up module file. The source code for the start-up modules is provided in corresponding `.s` files.

The primary start-up module (`crt0`) initializes all variables (variables without initializers are set to zero as required by the ANSI standard) except for variables in the persistent data section. The alternate start-up module (`crt1`) performs no data initialization.

For more on start-up code, see the “MPLAB<sup>®</sup> XC16 Assembler, Linker and Utilities User’s Guide” (DS52106) and, for C applications, the “MPLAB<sup>®</sup> XC16 C Compiler User’s Guide” (DS00052071).

## 1.4 DSP LIBRARY

The DSP library (`libdsp-omf.a`) provides a set of digital signal processing operations to a program targeted for execution on a dsPIC30F digital signal controller (DSC). In total, 49 functions are supported by the DSP Library.

Documentation for these libraries is provided in HTML Help files. Examples of use may also be provided. By default, the documentation is found in the `docs\dsp_lib` subdirectory of the MPLAB XC16 C Compiler install directory (see **Section 1.1.1 “Compiler Installation Locations”**).

## 1.5 16-BIT PERIPHERAL LIBRARIES

The 16-bit software and hardware peripheral libraries provide functions and macros for setting up and controlling 16-bit peripherals. These libraries are processor-specific and of the form `libpDevice-omf.a`, where *Device* is the 16-bit device number (e.g., `libp30F6014-coff.a` for the dsPIC30F6014 device) and *omf* is either `coff` or `elf`.

Documentation for these libraries is provided in HTML Help files. Examples of use are also provided in each file. By default, the documentation is found in the `docs\periph_lib` subdirectory of the MPLAB XC16 C Compiler install directory (see **Section 1.1.1 “Compiler Installation Locations”**).

## 1.6 STANDARD C LIBRARIES WITH MATH AND SUPPORT FUNCTIONS

A complete set of ANSI-89 conforming libraries are provided. The standard C library files are `libc-omf.a` (written by Dinkumware, an industry leader) and `libm-omf.a` (math functions, written by Microchip).

Additionally, some 16-bit standard C library helper functions, and standard functions that must be modified for use with 16-bit devices, are in `libpic30-omf.a`.

A typical C application will require these libraries. Documentation for these library functions is contained in this manual.

## 1.7 FIXED-POINT MATH FUNCTIONS

Fixed-point math functions may be found in the library file `libq-omf.a`. Documentation for these library functions is contained in this manual.

## 1.8 COMPILER BUILT-IN FUNCTIONS

The MPLAB XC16 C Compiler contains built-in functions that, to the developer, work like library functions. These functions are listed in the *"MPLAB<sup>®</sup> XC16 C Compiler Users' Guide"* (DS50002071).

# 16-Bit Language Tools Libraries

---

NOTES:

---

---

## Chapter 2. Standard C Libraries

---

---

### 2.1 INTRODUCTION

Standard ANSI C library functions are contained in the file `libc-omf.a`, where *omf* will be `coff` or `elf` depending upon the selected object module format.

#### 2.1.1 Assembly Code Applications

A free version of the math functions library and header file is available from the Microchip web site. No source code is available with this free version.

#### 2.1.2 C Code Applications

The MPLAB XC16 C Compiler install directory (**Section 1.1.1 “Compiler Installation Locations”**) contains the following subdirectories with library-related files:

- `lib` – standard C library files
- `src\libm` – source code for math library functions, batch file to rebuild the library
- `support\h` – header files for libraries

In addition, there is a file, `ResourceGraphs.pdf`, which contains diagrams of resources used by each function, located in `lib`.

#### 2.1.3 Chapter Organization

This chapter is organized as follows:

- Using the Standard C Libraries
- `<assert.h>` diagnostics
- `<ctype.h>` character handling
- `<errno.h>` errors
- `<float.h>` floating-point characteristics
- `<limits.h>` implementation-defined limits
- `<locale.h>` localization
- `<setjmp.h>` non-local jumps
- `<signal.h>` signal handling
- `<stdarg.h>` variable argument lists
- `<stddef.h>` common definitions
- `<stdio.h>` input and output
- `<stdlib.h>` utility functions
- `<string.h>` string functions
- `<time.h>` date and time functions

# 16-Bit Language Tools Libraries

---

## 2.2 USING THE STANDARD C LIBRARIES

Building an application which utilizes the standard C libraries requires two types of files: header files and library files.

### 2.2.1 Header Files

All standard C library entities are declared or defined in one or more standard headers (See list in **Section 2.1.3 “Chapter Organization”**.) To make use of a library entity in a program, write an include directive that names the relevant standard header.

The contents of a standard header is included by naming it in an include directive, as in:

```
#include <stdio.h> /* include I/O facilities */
```

The standard headers can be included in any order. Do not include a standard header within a declaration. Do not define macros that have the same names as keywords before including a standard header.

A standard header never includes another standard header.

### 2.2.2 Library Files

The archived library files contain all the individual object files for each library function.

When linking an application, the library file must be provided as an input to the linker (using the `--library` or `-l` linker option) such that the functions used by the application may be linked into the application.

A typical C application will require three library files: `libc-omf.a`, `libm-omf.a`, and `libpic30-omf.a`. (See **Section 1.2 “OMF-Specific Libraries/Start-up Modules”** for more on OMF-specific libraries.) These libraries will be included automatically if linking is performed using the compiler.

**Note:** Some standard library functions require a heap. These include the standard I/O functions that open files and the memory allocation functions. See the “*MPLAB® XC16 Assembler, Linker and Utilities User’s Guide*” (DS52106) and “*MPLAB® XC16 C Compiler User’s Guide*” (DS00052071) for more information on the heap.



## 2.3 <ASSERT.H> DIAGNOSTICS

The header file `assert.h` consists of a single macro that is useful for debugging logic errors in programs. By using the `assert` statement in critical locations where certain conditions should be true, the logic of the program may be tested.

Assertion testing may be turned off without removing the code by defining `NDEBUG` before including `<assert.h>`. If the macro `NDEBUG` is defined, `assert()` is ignored and no code is generated.

---

### **assert**

---

**Description:** If the expression is false, an assertion message is printed to `stderr` and the program is aborted.

**Include:** `<assert.h>`

**Prototype:** `void assert(int expression);`

**Argument:** `expression` The expression to test.

**Remarks:** The expression evaluates to zero or non-zero. If zero, the assertion fails and a message is printed to `stderr`. The message includes the source file name (`__FILE__`), the source line number (`__LINE__`), the expression being evaluated and the message. The macro then calls the function `abort()`. If the macro `_VERBOSE_DEBUGGING` is defined, a message will be printed to `stderr` each time `assert()` is called.

**Example:** `#include <assert.h> /* for assert */`

```
int main(void)
{
    int a;

    a = 2 * 2;
    assert(a == 4); /* if true-nothing prints */
    assert(a == 6); /* if false-print message */
                    /* and abort */
}
```

**Output:**

```
sampassert.c:9 a == 6 -- assertion failed
ABRT
```

with `_VERBOSE_DEBUGGING` defined:

```
sampassert.c:8 a == 4 -- OK
sampassert.c:9 a == 6 -- assertion failed
ABRT
```

# 16-Bit Language Tools Libraries

---

## 2.4 <CTYPE.H> CHARACTER HANDLING

The header file `ctype.h` consists of functions that are useful for classifying and mapping characters. Characters are interpreted according to the Standard C locale.

---

### isalnum

---

**Description:** Test for an alphanumeric character.  
**Include:** `<ctype.h>`  
**Prototype:** `int isalnum(int c);`  
**Argument:** `c` The character to test.  
**Return Value:** Returns a non-zero integer value if the character is alphanumeric; otherwise, returns a zero.  
**Remarks:** Alphanumeric characters are included within the ranges A-Z, a-z or 0-9.  
**Example:**

```
#include <ctype.h> /* for isalnum */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = '3';
    if (isalnum(ch))
        printf("3 is an alphanumeric\n");
    else
        printf("3 is NOT an alphanumeric\n");

    ch = '#';
    if (isalnum(ch))
        printf("# is an alphanumeric\n");
    else
        printf("# is NOT an alphanumeric\n");
}
```

**Output:**

```
3 is an alphanumeric
# is NOT an alphanumeric
```

---

### isalpha

---

**Description:** Test for an alphabetic character.  
**Include:** `<ctype.h>`  
**Prototype:** `int isalpha(int c);`  
**Argument:** `c` The character to test.  
**Return Value:** Returns a non-zero integer value if the character is alphabetic; otherwise, returns zero.  
**Remarks:** Alphabetic characters are included within the ranges A-Z or a-z.

---

## isalpha (Continued)

---

**Example:**

```
#include <ctype.h> /* for isalpha */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'B';
    if (isalpha(ch))
        printf("B is alphabetic\n");
    else
        printf("B is NOT alphabetic\n");

    ch = '#';
    if (isalpha(ch))
        printf("# is alphabetic\n");
    else
        printf("# is NOT alphabetic\n");
}
```

**Output:**

```
B is alphabetic
# is NOT alphabetic
```

---

## isctrl

---

**Description:** Test for a control character.

**Include:** <ctype.h>

**Prototype:** int isctrl(int c);

**Argument:** c character to test.

**Return Value:** Returns a non-zero integer value if the character is a control character; otherwise, returns zero.

**Remarks:** A character is considered to be a control character if its ASCII value is in the range 0x00 to 0x1F inclusive, or 0x7F.

**Example:**

```
#include <ctype.h> /* for isctrl */
#include <stdio.h> /* for printf */

int main(void)
{
    char ch;

    ch = 'B';
    if (isctrl(ch))
        printf("B is a control character\n");
    else
        printf("B is NOT a control character\n");

    ch = '\t';
    if (isctrl(ch))
        printf("A tab is a control character\n");
    else
        printf("A tab is NOT a control character\n");
}
```

**Output:**

```
B is NOT a control character
a tab is a control character
```

# 16-Bit Language Tools Libraries

---

---

---

## isdigit

---

**Description:** Test for a decimal digit.

**Include:** <ctype.h>

**Prototype:** int isdigit(int c);

**Argument:** c character to test.

**Return Value:** Returns a non-zero integer value if the character is a digit; otherwise, returns zero.

**Remarks:** A character is considered to be a digit character if it is in the range of '0'-'9'.

**Example:**

```
#include <ctype.h> /* for isdigit */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = '3';
    if (isdigit(ch))
        printf("3 is a digit\n");
    else
        printf("3 is NOT a digit\n");

    ch = '#';
    if (isdigit(ch))
        printf("# is a digit\n");
    else
        printf("# is NOT a digit\n");
}
```

**Output:**

```
3 is a digit
# is NOT a digit
```

---

## isgraph

---

**Description:** Test for a graphical character.

**Include:** <ctype.h>

**Prototype:** int isgraph (int c);

**Argument:** c character to test

**Return Value:** Returns a non-zero integer value if the character is a graphical character; otherwise, returns zero.

**Remarks:** A character is considered to be a graphical character if it is any printable character except a space.

**Example:**

```
#include <ctype.h> /* for isgraph */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;
```

## isgraph (Continued)

---

```
ch = '3';
if (isgraph(ch))
    printf("3 is a graphical character\n");
else
    printf("3 is NOT a graphical character\n");

ch = '#';
if (isgraph(ch))
    printf("# is a graphical character\n");
else
    printf("# is NOT a graphical character\n");

ch = ' ';
if (isgraph(ch))
    printf("a space is a graphical character\n");
else
    printf("a space is NOT a graphical character\n");
}
```

**Output:**

```
3 is a graphical character
# is a graphical character
a space is NOT a graphical character
```

---

## islower

---

**Description:** Test for a lowercase alphabetic character.

**Include:** <ctype.h>

**Prototype:** int islower (int c);

**Argument:** c character to test

**Return Value:** Returns a non-zero integer value if the character is a lowercase alphabetic character; otherwise, returns zero.

**Remarks:** A character is considered to be a lowercase alphabetic character if it is in the range of 'a'-'z'.

**Example:**

```
#include <ctype.h> /* for islower */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'B';
    if (islower(ch))
        printf("B is lowercase\n");
    else
        printf("B is NOT lowercase\n");

    ch = 'b';
    if (islower(ch))
        printf("b is lowercase\n");
    else
        printf("b is NOT lowercase\n");
}
```

# 16-Bit Language Tools Libraries

---

---

---

## islower (Continued)

---

**Output:**

B is NOT lowercase  
b is lowercase

---

## isprint

---

**Description:** Test for a printable character (includes a space).  
**Include:** <ctype.h>  
**Prototype:** int isprint (int c);  
**Argument:** c character to test  
**Return Value:** Returns a non-zero integer value if the character is printable; otherwise, returns zero.  
**Remarks:** A character is considered to be a printable character if it is in the range 0x20 to 0x7e inclusive.  
**Example:**

```
#include <ctype.h> /* for isprint */
#include <stdio.h> /* for printf */

int main(void)
{

    int ch;

    ch = '&';
    if (isprint(ch))
        printf("& is a printable character\n");
    else
        printf("& is NOT a printable character\n");

    ch = '\t';
    if (isprint(ch))
        printf("a tab is a printable character\n");
    else
        printf("a tab is NOT a printable character\n");
}
```

**Output:**

& is a printable character  
a tab is NOT a printable character

---

## ispunct

---

**Description:** Test for a punctuation character.  
**Include:** <ctype.h>  
**Prototype:** int ispunct (int c);  
**Argument:** c character to test  
**Return Value:** Returns a non-zero integer value if the character is a punctuation character; otherwise, returns zero.  
**Remarks:** A character is considered to be a punctuation character if it is a printable character which is neither a space nor an alphanumeric character. Punctuation characters consist of the following:  
!"#\$%&'();<=>?@[\\]\*+,-./:^\_{}~

---

## ispunct (Continued)

---

**Example:**

```
#include <ctype.h> /* for ispunct */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = '&';
    if (ispunct(ch))
        printf("& is a punctuation character\n");
    else
        printf("& is NOT a punctuation character\n");

    ch = '\t';
    if (ispunct(ch))
        printf("a tab is a punctuation character\n");
    else
        printf("a tab is NOT a punctuation character\n");
}

Output:
& is a punctuation character
a tab is NOT a punctuation character
```

---

## isspace

---

**Description:** Test for a white-space character.

**Include:** <ctype.h>

**Prototype:** int isspace (int c);

**Argument:** c character to test

**Return Value:** Returns a non-zero integer value if the character is a white-space character; otherwise, returns zero.

**Remarks:** A character is considered to be a white-space character if it is one of the following: space (' '), form feed ('\f'), newline ('\n'), carriage return ('\r'), horizontal tab ('\t'), or vertical tab ('\v').

**Example:**

```
#include <ctype.h> /* for isspace */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = '&';
    if (isspace(ch))
        printf("& is a white-space character\n");
    else
        printf("& is NOT a white-space character\n");

    ch = '\t';
    if (isspace(ch))
        printf("a tab is a white-space character\n");
    else
        printf("a tab is NOT a white-space character\n");
}
```

# 16-Bit Language Tools Libraries

---

---

---

## isspace (Continued)

---

**Output:**

& is NOT a white-space character  
a tab is a white-space character

---

## isupper

---

**Description:** Test for an uppercase letter.

**Include:** <ctype.h>

**Prototype:** int isupper (int c);

**Argument:** c character to test

**Return Value:** Returns a non-zero integer value if the character is an uppercase alphabetic character; otherwise, returns zero.

**Remarks:** A character is considered to be an uppercase alphabetic character if it is in the range of 'A'-'Z'.

**Example:** #include <ctype.h> /\* for isupper \*/  
#include <stdio.h> /\* for printf \*/

```
int main(void)
{
    int ch;

    ch = 'B';
    if (isupper(ch))
        printf("B is uppercase\n");
    else
        printf("B is NOT uppercase\n");

    ch = 'b';
    if (isupper(ch))
        printf("b is uppercase\n");
    else
        printf("b is NOT uppercase\n");
}
```

**Output:**

B is uppercase  
b is NOT uppercase

---

## isxdigit

---

**Description:** Test for a hexadecimal digit.

**Include:** <ctype.h>

**Prototype:** int isxdigit (int c);

**Argument:** c character to test

**Return Value:** Returns a non-zero integer value if the character is a hexadecimal digit; otherwise, returns zero.

**Remarks:** A character is considered to be a hexadecimal digit character if it is in the range of '0'-'9', 'A'-'F', or 'a'-'f'. Note: The list does not include the leading 0x because 0x is the prefix for a hexadecimal number but is not an actual hexadecimal digit.

---



## isxdigit (Continued)

---

**Example:**

```
#include <ctype.h> /* for isxdigit */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'B';
    if (isxdigit(ch))
        printf("B is a hexadecimal digit\n");
    else
        printf("B is NOT a hexadecimal digit\n");

    ch = 't';
    if (isxdigit(ch))
        printf("t is a hexadecimal digit\n");
    else
        printf("t is NOT a hexadecimal digit\n");
}
```

**Output:**

```
B is a hexadecimal digit
t is NOT a hexadecimal digit
```

---

## tolower

---

**Description:** Convert a character to a lowercase alphabetical character.

**Include:** <ctype.h>

**Prototype:** int tolower (int *c*);

**Argument:** *c* The character to convert to lowercase.

**Return Value:** Returns the corresponding lowercase alphabetical character if the argument was originally uppercase; otherwise, returns the original character.

**Remarks:** Only uppercase alphabetical characters may be converted to lowercase.

**Example:**

```
#include <ctype.h> /* for tolower */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'B';
    printf("B changes to lowercase %c\n",
        tolower(ch));

    ch = 'b';
    printf("b remains lowercase %c\n",
        tolower(ch));

    ch = '@';
    printf("@ has no lowercase, ");
    printf("so %c is returned\n", tolower(ch));
}
```

# 16-Bit Language Tools Libraries

---

---

---

## tolower (Continued)

---

**Output:**

B changes to lowercase b  
b remains lowercase b  
@ has no lowercase, so @ is returned

---

## toupper

---

**Description:** Convert a character to an uppercase alphabetical character.  
**Include:** <ctype.h>  
**Prototype:** int toupper (int c);  
**Argument:** c The character to convert to uppercase.  
**Return Value:** Returns the corresponding uppercase alphabetical character if the argument was originally lowercase; otherwise, returns the original character.  
**Remarks:** Only lowercase alphabetical characters may be converted to uppercase.  
**Example:**

```
#include <ctype.h> /* for toupper */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'b';
    printf("b changes to uppercase %c\n",
          toupper(ch));

    ch = 'B';
    printf("B remains uppercase %c\n",
          toupper(ch));

    ch = '@';
    printf("@ has no uppercase, ");
    printf("so %c is returned\n", toupper(ch));
}
```

**Output:**

b changes to uppercase B  
B remains uppercase B  
@ has no uppercase, so @ is returned

## 2.5 <ERRNO.H> ERRORS

The header file `errno.h` consists of macros that provide error codes that are reported by certain library functions (see individual functions). The variable `errno` may return any value greater than zero. To test if a library function encounters an error, the program should store the zero value in `errno` immediately before calling the library function. The value should be checked before another function call could change the value. At program start-up, `errno` is zero. Library functions will never set `errno` to zero.

---

### EDOM

---

<b>Description:</b>	Represents a domain error.
<b>Include:</b>	<code>&lt;errno.h&gt;</code>
<b>Remarks:</b>	EDOM represents a domain error, which occurs when an input argument is outside the domain in which the function is defined.

---

### ERANGE

---

<b>Description:</b>	Represents an overflow or underflow error.
<b>Include:</b>	<code>&lt;errno.h&gt;</code>
<b>Remarks:</b>	ERANGE represents an overflow or underflow error, which occurs when a result is too large or too small to be stored.

---

### errno

---

<b>Description:</b>	Contains the value of an error when an error occurs in a function.
<b>Include:</b>	<code>&lt;errno.h&gt;</code>
<b>Remarks:</b>	The variable <code>errno</code> is set to a non-zero integer value by a library function when an error occurs. At program start-up, <code>errno</code> is set to zero. <code>Errno</code> should be reset to zero prior to calling a function that sets it.

# 16-Bit Language Tools Libraries

---

## 2.6 <FLOAT.H> FLOATING-POINT CHARACTERISTICS

The header file `float.h` consists of macros that specify various properties of floating-point types. These properties include number of significant figures, size limits and what rounding mode is used.

---

### DBL\_DIG

---

**Description:** Number of decimal digits of precision in a double precision floating-point value.

**Include:** `<float.h>`

**Value:** 6 by default, 15 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

### DBL\_EPSILON

---

**Description:** The difference between 1.0 and the next larger representable double precision floating-point value

**Include:** `<float.h>`

**Value:** 1.192093e-07 by default, 2.220446e-16 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

### DBL\_MANT\_DIG

---

**Description:** Number of base-`FLT_RADIX` digits in a double precision floating-point significand.

**Include:** `<float.h>`

**Value:** 24 by default, 53 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

### DBL\_MAX

---

**Description:** Maximum finite double precision floating-point value.

**Include:** `<float.h>`

**Value:** 3.402823e+38 by default, 1.797693e+308 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

## DBL\_MAX\_10\_EXP

---

**Description:** Maximum integer value for a double precision floating-point exponent in base 10.

**Include:** `<float.h>`

**Value:** 38 by default, 308 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

## DBL\_MAX\_EXP

---

**Description:** Maximum integer value for a double precision floating-point exponent in base `FLT_RADIX`.

**Include:** `<float.h>`

**Value:** 128 by default, 1024 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

## DBL\_MIN

---

**Description:** Minimum double precision floating-point value.

**Include:** `<float.h>`

**Value:** 1.175494e-38 by default, 2.225074e-308 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

## DBL\_MIN\_10\_EXP

---

**Description:** Minimum negative integer value for a double precision floating-point exponent in base 10.

**Include:** `<float.h>`

**Value:** -37 by default, -307 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

# 16-Bit Language Tools Libraries

---

---

---

## DBL\_MIN\_EXP

---

**Description:** Minimum negative integer value for a double precision floating-point exponent in base `FLT_RADIX`.

**Include:** `<float.h>`

**Value:** -125 by default, -1021 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

---

---

## FLT\_DIG

---

**Description:** Number of decimal digits of precision in a single precision floating-point value.

**Include:** `<float.h>`

**Value:** 6

---

---

## FLT\_EPSILON

---

**Description:** The difference between 1.0 and the next larger representable single precision floating-point value.

**Include:** `<float.h>`

**Value:** 1.192093e-07

---

---

## FLT\_MANT\_DIG

---

**Description:** Number of base-`FLT_RADIX` digits in a single precision floating-point significand.

**Include:** `<float.h>`

**Value:** 24

---

---

## FLT\_MAX

---

**Description:** Maximum finite single precision floating-point value.

**Include:** `<float.h>`

**Value:** 3.402823e+38

---

---

## FLT\_MAX\_10\_EXP

---

**Description:** Maximum integer value for a single precision floating-point exponent in base 10.

**Include:** `<float.h>`

**Value:** 38

---

---

## FLT\_MAX\_EXP

---

**Description:** Maximum integer value for a single precision floating-point exponent in base FLT\_RADIX.

**Include:** <float.h>

**Value:** 128

---

---

## FLT\_MIN

---

**Description:** Minimum single precision floating-point value.

**Include:** <float.h>

**Value:** 1.175494e-38

---

---

## FLT\_MIN\_10\_EXP

---

**Description:** Minimum negative integer value for a single precision floating-point exponent in base 10.

**Include:** <float.h>

**Value:** -37

---

---

## FLT\_MIN\_EXP

---

**Description:** Minimum negative integer value for a single precision floating-point exponent in base FLT\_RADIX.

**Include:** <float.h>

**Value:** -125

---

---

## FLT\_RADIX

---

**Description:** Radix of exponent representation.

**Include:** <float.h>

**Value:** 2

**Remarks:** The base representation of the exponent is base-2 or binary.

---

---

## FLT\_ROUNDS

---

**Description:** Represents the rounding mode for floating-point operations.

**Include:** <float.h>

**Value:** 1

**Remarks:** Rounds to the nearest representable value.

---

---

## LDBL\_DIG

---

**Description:** Number of decimal digits of precision in a long double precision floating-point value.

**Include:** <float.h>

**Value:** 15

---

# 16-Bit Language Tools Libraries

---

---

---

## LDBL\_EPSILON

---

**Description:** The difference between 1.0 and the next larger representable long double precision floating-point value.

**Include:** <float.h>

**Value:** 2.220446e-16

---

## LDBL\_MANT\_DIG

---

**Description:** Number of base-FLT\_RADIX digits in a long double precision floating-point significand.

**Include:** <float.h>

**Value:** 53

---

## LDBL\_MAX

---

**Description:** Maximum finite long double precision floating-point value.

**Include:** <float.h>

**Value:** 1.797693e+308

---

## LDBL\_MAX\_10\_EXP

---

**Description:** Maximum integer value for a long double precision floating-point exponent in base 10.

**Include:** <float.h>

**Value:** 308

---

## LDBL\_MAX\_EXP

---

**Description:** Maximum integer value for a long double precision floating-point exponent in base FLT\_RADIX.

**Include:** <float.h>

**Value:** 1024

---

## LDBL\_MIN

---

**Description:** Minimum long double precision floating-point value.

**Include:** <float.h>

**Value:** 2.225074e-308

---

## LDBL\_MIN\_10\_EXP

---

**Description:** Minimum negative integer value for a long double precision floating-point exponent in base 10.

**Include:** <float.h>

**Value:** -307

---



---

## LDBL\_MIN\_EXP

---

**Description:** Minimum negative integer value for a long double precision floating-point exponent in base `FLT_RADIX`.

**Include:** `<float.h>`

**Value:** -1021

## 2.7 <LIMITS.H> IMPLEMENTATION-DEFINED LIMITS

The header file `limits.h` consists of macros that define the minimum and maximum values of integer types. Each of these macros can be used in `#if` preprocessing directives.

---

## CHAR\_BIT

---

**Description:** Number of bits to represent type `char`.

**Include:** `<limits.h>`

**Value:** 8

---

## CHAR\_MAX

---

**Description:** Maximum value of a `char`.

**Include:** `<limits.h>`

**Value:** 127

---

## CHAR\_MIN

---

**Description:** Minimum value of a `char`.

**Include:** `<limits.h>`

**Value:** -128

---

## INT\_MAX

---

**Description:** Maximum value of an `int`.

**Include:** `<limits.h>`

**Value:** 32767

---

## INT\_MIN

---

**Description:** Minimum value of an `int`.

**Include:** `<limits.h>`

**Value:** -32768

---

## LLONG\_MAX

---

**Description:** Maximum value of a long long `int`.

**Include:** `<limits.h>`

**Value:** 9223372036854775807

# 16-Bit Language Tools Libraries

---

---

---

## LLONG\_MIN

---

**Description:** Minimum value of a long long int.  
**Include:** <limits.h>  
**Value:** -9223372036854775808

---

---

## LONG\_MAX

---

**Description:** Maximum value of a long int.  
**Include:** <limits.h>  
**Value:** 2147483647

---

---

## LONG\_MIN

---

**Description:** Minimum value of a long int.  
**Include:** <limits.h>  
**Value:** -2147483648

---

---

## MB\_LEN\_MAX

---

**Description:** Maximum number of bytes in a multibyte character.  
**Include:** <limits.h>  
**Value:** 1

---

---

## SCHAR\_MAX

---

**Description:** Maximum value of a signed char.  
**Include:** <limits.h>  
**Value:** 127

---

---

## SCHAR\_MIN

---

**Description:** Minimum value of a signed char.  
**Include:** <limits.h>  
**Value:** -128

---

---

## SHRT\_MAX

---

**Description:** Maximum value of a short int.  
**Include:** <limits.h>  
**Value:** 32767

---

---

## SHRT\_MIN

---

**Description:** Minimum value of a short int.  
**Include:** <limits.h>  
**Value:** -32768

---

---

## UCHAR\_MAX

---

**Description:** Maximum value of an unsigned char.  
**Include:** <limits.h>  
**Value:** 255

---

---

## UINT\_MAX

---

**Description:** Maximum value of an unsigned int.  
**Include:** <limits.h>  
**Value:** 65535

---

---

## ULLONG\_MAX

---

**Description:** Maximum value of a long long unsigned int.  
**Include:** <limits.h>  
**Value:** 18446744073709551615

---

---

## ULONG\_MAX

---

**Description:** Maximum value of a long unsigned int.  
**Include:** <limits.h>  
**Value:** 4294967295

---

---

## USHRT\_MAX

---

**Description:** Maximum value of an unsigned short int.  
**Include:** <limits.h>  
**Value:** 65535

---

## 2.8 <LOCALE.H> LOCALIZATION

This compiler defaults to the C locale and does not support any other locales; therefore, it does not support the header file `locale.h`. The following would normally be found in this file:

- struct lconv
- NULL
- LC\_ALL
- LC\_COLLATE
- LC\_CTYPE
- LC\_MONETARY
- LC\_NUMERIC
- LC\_TIME
- localeconv
- setlocale

# 16-Bit Language Tools Libraries

---

## 2.9 <SETJMP.H> NON-LOCAL JUMPS

The header file `setjmp.h` consists of a type, a macro and a function that allow control transfers to occur that bypass the normal function call and return process.

---

### jmp\_buf

---

**Description:** A type that is an array used by `setjmp` and `longjmp` to save and restore the program environment.

**Include:** `<setjmp.h>`

**Prototype:** `typedef int jmp_buf[_NSETJMP];`

**Remarks:** `_NSETJMP` is defined as `16 + 2` that represents 16 registers and a 32-bit return address.

---

### setjmp

---

**Description:** A macro that saves the current state of the program for later use by `longjmp`.

**Include:** `<setjmp.h>`

**Prototype:** `#define setjmp(jmp_buf env)`

**Argument:** `env` variable where environment is stored

**Return Value:** If the return is from a direct call, `setjmp` returns zero. If the return is from a call to `longjmp`, `setjmp` returns a non-zero value.  
**Note:** If the argument `val` from `longjmp` is 0, `setjmp` returns 1.

**Example:** See `longjmp`.

---

### longjmp

---

**Description:** A function that restores the environment saved by `setjmp`.

**Include:** `<setjmp.h>`

**Prototype:** `void longjmp(jmp_buf env, int val);`

**Arguments:** `env` variable where environment is stored  
`val` value to be returned to `setjmp` call.

**Remarks:** The value parameter, `val`, should be non-zero. If `longjmp` is invoked from a nested signal handler (that is, invoked as a result of a signal raised during the handling of another signal), the behavior is undefined.

## 2.10 <SIGNAL.H> SIGNAL HANDLING

The header file `signal.h` consists of a type, several macros and two functions that specify how the program handles signals while it is executing. A signal is a condition that may be reported during the program execution. Signals are synchronous, occurring under software control via the `raise` function.

A signal may be handled by:

- Default handling (`SIG_DFL`); the signal is treated as a fatal error and execution stops
- Ignoring the signal (`SIG_IGN`); the signal is ignored and control is returned to the user application
- Handling the signal with a function designated via `signal`.

By default, all signals are handled by the default handler, which is identified by `SIG_DFL`.

The type `sig_atomic_t` is an integer type that the program accesses atomically. When this type is used with the keyword `volatile`, the signal handler can share the data objects with the rest of the program.

---

### **sig\_atomic\_t**

---

**Description:** A type used by a signal handler.  
**Include:** `<signal.h>`  
**Prototype:** `typedef int sig_atomic_t;`

---

---

### **SIG\_DFL**

---

**Description:** Used as the second argument and/or the return value for `signal` to specify that the default handler should be used for a specific signal.  
**Include:** `<signal.h>`

---

---

### **SIG\_ERR**

---

**Description:** Used as the return value for `signal` when it cannot complete a request due to an error.  
**Include:** `<signal.h>`

---

---

### **SIG\_IGN**

---

**Description:** Used as the second argument and/or the return value for `signal` to specify that the signal should be ignored.  
**Include:** `<signal.h>`

---

# 16-Bit Language Tools Libraries

---

---

---

## SIGABRT

---

<b>Description:</b>	Name for the abnormal termination signal.
<b>Include:</b>	<signal.h>
<b>Prototype:</b>	#define SIGABRT
<b>Remarks:</b>	<p>SIGABRT represents an abnormal termination signal and is used in conjunction with <code>raise</code> or <code>signal</code>. The default <code>raise</code> behavior (action identified by <code>SIG_DFL</code>) is to output to the standard error stream:</p> <pre>abort - terminating</pre> <p>See the example accompanying <code>signal</code> to see general usage of signal names and signal handling.</p>
<b>Example:</b>	<pre>#include &lt;signal.h&gt; /* for raise, SIGABRT */ #include &lt;stdio.h&gt; /* for printf */  int main(void) {     raise(SIGABRT);     printf("Program never reaches here."); }</pre> <p><b>Output:</b> ABRT</p> <p><b>Explanation:</b> ABRT stands for “abort”.</p>

---

## SIGFPE

---

<b>Description:</b>	Signals floating-point errors such as for division by zero or result out of range.
<b>Include:</b>	<signal.h>
<b>Prototype:</b>	#define SIGFPE
<b>Remarks:</b>	<p>SIGFPE is used as an argument for <code>raise</code> and/or <code>signal</code>. When used, the default behavior is to print an arithmetic error message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See <code>signal</code> for an example of a user-defined function.</p>
<b>Example:</b>	<pre>#include &lt;signal.h&gt; /* for raise, SIGFPE */ #include &lt;stdio.h&gt; /* for printf */  int main(void) {     raise(SIGFPE);     printf("Program never reaches here"); }</pre> <p><b>Output:</b> FPE</p> <p><b>Explanation:</b> FPE stands for “floating-point error”.</p>

---

## SIGILL

---

**Description:** Signals illegal instruction.

**Include:** <signal.h>

**Prototype:** #define SIGILL

**Remarks:** SIGILL is used as an argument for `raise` and/or `signal`. When used, the default behavior is to print an invalid executable code message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user-defined function.

**Example:**

```
#include <signal.h> /* for raise, SIGILL */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGILL);
    printf("Program never reaches here");
}
```

**Output:**  
ILL

**Explanation:**  
ILL stands for “illegal instruction”.

---

## SIGINT

---

**Description:** Interrupt signal.

**Include:** <signal.h>

**Prototype:** #define SIGINT

**Remarks:** SIGINT is used as an argument for `raise` and/or `signal`. When used, the default behavior is to print an interruption message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user-defined function.

**Example:**

```
#include <signal.h> /* for raise, SIGINT */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGINT);
    printf("Program never reaches here.");
}
```

**Output:**  
INT

**Explanation:**  
INT stands for “interruption”.

# 16-Bit Language Tools Libraries

---

---

---

## SIGSEGV

---

**Description:** Signals invalid access to storage.

**Include:** `<signal.h>`

**Prototype:** `#define SIGSEGV`

**Remarks:** SIGSEGV is used as an argument for `raise` and/or `signal`. When used, the default behavior is to print an invalid storage request message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user-defined function.

**Example:**

```
#include <signal.h> /* for raise, SIGSEGV */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGSEGV);
    printf("Program never reaches here.");
}

Output:
SEGV

Explanation:
SEGV stands for "invalid storage access".
```

---

## SIGTERM

---

**Description:** Signals a termination request.

**Include:** `<signal.h>`

**Prototype:** `#define SIGTERM`

**Remarks:** SIGTERM is used as an argument for `raise` and/or `signal`. When used, the default behavior is to print a termination request message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user-defined function.

**Example:**

```
#include <signal.h> /* for raise, SIGTERM */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGTERM);
    printf("Program never reaches here.");
}

Output:
TERM

Explanation:
TERM stands for "termination request".
```



## raise

---

**Description:** Reports a synchronous signal.  
**Include:** <signal.h>  
**Prototype:** int raise(int sig);  
**Argument:** sig signal name  
**Return Value:** Returns a 0 if successful; otherwise, returns a non-zero value.  
**Remarks:** raise sends the signal identified by sig to the executing program.

**Example:**

```
#include <signal.h> /* for raise, signal, */
                    /* SIGILL, SIG_DFL */
#include <stdlib.h> /* for div, div_t */
#include <stdio.h> /* for printf */
#include <p30f6014.h> /* for INTCON1bits */
```

```
void __attribute__((__interrupt__))
_MathError(void)
{
    raise(SIGILL);
    INTCON1bits.MATHERR = 0;
}

void illegalinsn(int idsig)
{
    printf("Illegal instruction executed\n");
    exit(1);
}

int main(void)
{
    int x, y;
    div_t z;

    signal(SIGILL, illegalinsn);
    x = 7;
    y = 0;
    z = div(x, y);
    printf("Program never reaches here");
}
```

**Output:**

Illegal instruction executed

**Explanation:**

This example requires the linker script, p30f6014.gld. There are three parts to this example.

First, an interrupt handler is written for the interrupt vector, `_MathError`, to handle a math error by sending an illegal instruction, `signal (SIGILL)`, to the executing program. The last statement in the interrupt handler clears the exception flag.

Second, the function `illegalinsn` will print an error message and call `exit`.

Third, in `main`, `signal (SIGILL, illegalinsn)` sets the handler for `SIGILL` to the function `illegalinsn`.

When a math error occurs due to a divide by zero, the `_MathError` interrupt vector is called, which in turn will raise a signal that will call the handler function for `SIGILL`: the function `illegalinsn`. Thus, error messages are printed and the program is terminated.

# 16-Bit Language Tools Libraries

---

---

## signal

---

**Description:** Controls interrupt signal handling.

**Include:** <signal.h>

**Prototype:** void (\*signal(int sig, void(\*func)(int)))(int);

**Arguments:** sig signal name  
func function to be executed

**Return Value:** Returns the previous value of *func*.

**Example:**

```
#include <signal.h> /* for signal, raise, */
                    /* SIGINT, SIGILL, */
                    /* SIG_IGN, and SIGFPE */
#include <stdio.h> /* for printf */

/* Signal handler function */
void mysigint(int id)
{
    printf("SIGINT received\n");
}

int main(void)
{
    /* Override default with user defined function */
    signal(SIGINT, mysigint);
    raise(SIGINT);

    /* Ignore signal handler */
    signal(SIGILL, SIG_IGN);
    raise(SIGILL);
    printf("SIGILL was ignored\n");

    /* Use default signal handler */
    raise(SIGFPE);
    printf("Program never reaches here.");
}
```

**Output:**

```
SIGINT received
SIGILL was ignored
FPE
```

**Explanation:**

The function `mysigint` is the user-defined signal handler for `SIGINT`. Inside the main program, the function `signal` is called to set up the signal handler (`mysigint`) for the signal `SIGINT` that will override the default actions. The function `raise` is called to report the signal `SIGINT`. This causes the signal handler for `SIGINT` to use the user-defined function (`mysigint`) as the signal handler so it prints the "SIGINT received" message.

Next, the function `signal` is called to set up the signal handler `SIG_IGN` for the signal `SIGILL`. The constant `SIG_IGN` is used to indicate the signal should be ignored. The function `raise` is called to report the signal `SIGILL` that is ignored.

The function `raise` is called again to report the signal `SIGFPE`. Since there is no user-defined function for `SIGFPE`, the default signal handler is used so the message "FPE" is printed (which stands for "arithmetic error - terminating"). Then, the calling program is terminated. The `printf` statement is never reached.

## 2.11 <STDARG.H> VARIABLE ARGUMENT LISTS

The header file `stdarg.h` supports functions with variable argument lists. This allows functions to have arguments without corresponding parameter declarations. There must be at least one named argument. The variable arguments are represented by ellipses (...). An object of type `va_list` must be declared inside the function to hold the arguments. `va_start` will initialize the variable to an argument list, `va_arg` will access the argument list and `va_end` will end the use of the argument.

---

### `va_list`

---

**Description:** The type `va_list` declares a variable that will refer to each argument in a variable-length argument list.

**Include:** `<stdarg.h>`

**Example:** See `va_arg`.

---

### `va_arg`

---

**Description:** Gets the current argument.

**Include:** `<stdarg.h>`

**Prototype:** `#define va_arg(va_list ap, Ty)`

**Argument:** `ap` pointer to list of arguments

`Ty` type of argument to be retrieved

**Return Value:** Returns the current argument

**Remarks:** `va_start` must be called before `va_arg`.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdarg.h> /* for va_arg, va_start,
                    va_list, va_end */
```

```
void tprint(const char *fmt, ...)
{
    va_list ap;

    va_start(ap, fmt);
    while (*fmt)
    {
        switch (*fmt)
        {
```

# 16-Bit Language Tools Libraries

---

---

## va\_arg (Continued)

---

```
        case '%':
            fmt++;
            if (*fmt == 'd')
            {
                int d = va_arg(ap, int);
                printf("<%d> is an integer\n",d);
            }
            else if (*fmt == 's')
            {
                char *s = va_arg(ap, char*);
                printf("<%s> is a string\n", s);
            }
            else
            {
                printf("%%c is an unknown format\n",
                    *fmt);
            }
            fmt++;
            break;
        default:
            printf("%c is unknown\n", *fmt);
            fmt++;
            break;
    }
}
va_end(ap);
}
```

```
int main(void)
{
    tprint("%d%s.%c", 83, "This is text.", 'a');
}
```

### Output:

```
<83> is an integer
<This is text.> is a string
. is unknown
%c is an unknown format
```

---

## va\_end

---

**Description:** Ends the use of *ap*.  
**Include:** <stdarg.h>  
**Prototype:** #define va\_end(va\_list ap)  
**Argument:** *ap* pointer to list of arguments  
**Remarks:** After a call to *va\_end*, the argument list pointer, *ap*, is considered to be invalid. Further calls to *va\_arg* should not be made until the next *va\_start*. In the 16-bit compiler, *va\_end* does nothing, so this call is not necessary but should be used for readability and portability.  
**Example:** See *va\_arg*.

---

## va\_start

---

**Description:** Sets the argument pointer *ap* to first optional argument in the variable-length argument list.  
**Include:** <stdarg.h>  
**Prototype:** #define va\_start(va\_list ap, last\_arg)  
**Argument:** *ap* pointer to list of arguments  
*last\_arg* last named argument before the optional arguments  
**Example:** See *va\_arg*.

---

## 2.12 <STDDEF.H> COMMON DEFINITIONS

The header file *stddef.h* consists of several types and macros that are of general use in programs.

---

### ptrdiff\_t

---

**Description:** The type of the result of subtracting two pointers.  
**Include:** <stddef.h>

---

### size\_t

---

**Description:** The type of the result of the *sizeof* operator.  
**Include:** <stddef.h>

---

### wchar\_t

---

**Description:** A type that holds a wide character value.  
**Include:** <stddef.h>

---

### NULL

---

**Description:** The value of a null pointer constant.  
**Include:** <stddef.h>

---

# 16-Bit Language Tools Libraries

---

---

---

## offsetof

---

**Description:** Gives the offset of a structure member from the beginning of the structure.

**Include:** `<stddef.h>`

**Prototype:** `#define offsetof(T, mbr)`

**Arguments:** *T* name of structure  
*mbr* name of member in structure T

**Return Value:** Returns the offset in bytes of the specified member (*mbr*) from the beginning of the structure.

**Remarks:** The macro `offsetof` is undefined for bitfields. An error message will occur if bitfields are used.

**Example:**

```
#include <stddef.h> /* for offsetof */
#include <stdio.h> /* for printf */
```

```
struct info {
    char item1[5];
    int item2;
    char item3;
    float item4;
};

int main(void)
{
    printf("Offset of item1 = %d\n",
        offsetof(struct info,item1));
    printf("Offset of item2 = %d\n",
        offsetof(struct info,item2));
    printf("Offset of item3 = %d\n",
        offsetof(struct info,item3));
    printf("Offset of item4 = %d\n",
        offsetof(struct info,item4));
}
```

**Output:**

```
Offset of item1 = 0
Offset of item2 = 6
Offset of item3 = 8
Offset of item4 = 10
```

**Explanation:**

This program shows the offset in bytes of each structure member from the start of the structure. Although `item1` is only 5 bytes (`char item1[5]`), padding is added so the address of `item2` falls on an even boundary. The same occurs with `item3`; it is 1 byte (`char item3`) with 1 byte of padding.

## 2.13 <STDIO.H> INPUT AND OUTPUT

The header file `stdio.h` consists of types, macros and functions that provide support to perform input and output operations on files and streams. When a file is opened, it is associated with a stream. A stream is a pipeline for the flow of data into and out of files. Because different systems use different properties, the stream provides more uniform properties to allow reading and writing of the files.

Streams can be text streams or binary streams. Text streams consist of a sequence of characters divided into lines. Each line is terminated with a newline (`'\n'`) character. The characters may be altered in their internal representation, particularly in regards to line endings. Binary streams consist of sequences of bytes of information. The bytes transmitted to the binary stream are not altered. There is no concept of lines - the file is just a series of bytes.

At start-up, three streams are automatically opened: `stdin`, `stdout` and `stderr`. `stdin` provides a stream for standard input, `stdout` is standard output and `stderr` is the standard error. Additional streams may be created with the `fopen` function. See `fopen` for the different types of file access that are permitted. These access types are used by `fopen` and `freopen`.

The type `FILE` is used to store information about each opened file stream. It includes such things as error indicators, end-of-file indicators, file position indicators and other internal status information needed to control a stream. Many functions in the `stdio` use `FILE` as an argument.

There are three types of buffering: unbuffered, line buffered and fully buffered. Unbuffered means a character or byte is transferred one at a time. Line buffered collects and transfers an entire line at a time (i.e., the newline character indicates the end of a line). Fully buffered allows blocks of an arbitrary size to be transmitted. The functions `setbuf` and `setvbuf` control file buffering.

The `stdio.h` file also contains functions that use input and output formats. The input formats, or scan formats, are used for reading data. Their descriptions can be found under `scanf`, but they are also used by `fscanf` and `sscanf`. The output formats, or print formats, are used for writing data. Their descriptions can be found under `printf`. These print formats are also used by `fprintf`, `sprintf`, `vfprintf`, `vprintf` and `vsprintf`.

### 2.13.1 Compiler Options

Certain compiler options may affect how standard I/O performs. In an effort to provide a more tailored version of the formatted I/O routines, the tool chain may convert a call to a `printf` or `scanf` style function to a different call. The options are summarized below:

- The `-msmart-io` option, when enabled, will attempt to convert `printf`, `scanf` and other functions that use the input output formats to an integer-only variant. The functionality is the same as that of the C standard forms, minus the support for floating-point output. `-msmart-io=0` disables this feature and no conversion will take place. `-msmart-io=1` or `-msmart-io` (the default) will convert a function call if it can be proven that an I/O function will never be presented with a floating-point conversion. `-msmart-io=2` is more optimistic than the default, and will assume that non-constant format strings or otherwise unknown format strings will not contain a floating-point format. In the event that `-msmart-io=2` is used with a floating-point format, the format letter will appear as literal text and its corresponding argument will not be consumed.

# 16-Bit Language Tools Libraries

---

- `-fno-short-double` will cause the compiler to generate calls to formatted I/O routines that support `double` as if it were a `long double` type.

Mixing modules compiled with these options may result in a larger executable size or incorrect execution if large and small double-sized data is shared across modules.

## 2.13.2 Customizing STDIO

The standard I/O relies on helper functions described in **Chapter 4. “Standard C Libraries - Support Functions”**. These functions include `read()`, `write()`, `open()`, and `close()` which are called to read, write, open or close handles that are associated with standard I/O `FILE` pointers. The sources for these libraries are provided for you to customize as you wish.

It is recommended that these customized functions be allocated in a named section that begins with `.lib`. This will cause the linker to allocate them near to the rest of the library functions, which is where they ought to be.

The simplest way to redirect standard I/O to the peripheral of your choice is to select one of the default handles already in use. Also, you could open files with a specific name via `fopen()` by rewriting `open()` to return a new handle to be recognized by `read()` or `write()` as appropriate.

If only a specific peripheral is required, then you could associate handle `1 == stdout` or `2 == stderr` to another peripheral by writing the correct code to talk to the interested peripheral.

A complete generic solution might be:

```
/* should be in a header file */
enum my_handles {
    handle_stdin,
    handle_stdout,
    handle_stderr,
    handle_can1,
    handle_can2,
    handle_spi1,
    handle_spi2,
};

int __attribute__((__weak__, __section__(".libc")))
open(const char *name, int access, int mode) {
    switch (name[0]) {
        case 'i' : return handle_stdin;
        case 'o' : return handle_stdout;
        case 'e' : return handle_stderr;
        case 'c' : return handle_can1;
        case 'C' : return handle_can2;
        case 's' : return handle_spi1;
        case 'S' : return handle_spi2;
        default: return handle_stderr;
    }
}
```

Single letters were used in this example because they are faster to check and use less memory. However, if memory is not an issue, you could use `strcmp` to compare full names.



In `write()`, you would write:

```
int __attribute__((__section__(".libc.write")))
write(int handle, void *buffer, unsigned int len) {
    int i;
    volatile UxMODEBITS *umode = &U1MODEbits;
    volatile UxSTABITS *ustatus = &U1STAbits;
    volatile unsigned int *txreg = &U1TXREG;
    volatile unsigned int *brg = &U1BRG;

    switch (handle)
    {
    default:
    case 0:
    case 1:
    case 2:
        if ((__C30_UART != 1) && (&U2BRG)) {
            umode = &U2MODEbits;
            ustatus = &U2STAbits;
            txreg = &U2TXREG;
            brg = &U2BRG;
        }
        if ((umode->UARTEN) == 0)
        {
            *brg = 0;
            umode->UARTEN = 1;
        }
        if ((ustatus->UTXEN) == 0)
        {
            ustatus->UTXEN = 1;
        }
        for (i = len; i; --i)
        {
            while ((ustatus->TRMT) ==0);
            *txreg = *(char*)buffer++;
        }
        break;
    case handle_can1: /* code to support can1 */
        break;
    case handle_can2: /* code to support can2 */
        break;
    case handle_spi1: /* code to support spi1 */
        break;
    case handle_spi2: /* code to support spi2 */
        break;
    }
    return(len);
}
```

where you would fill in the appropriate code as specified in the comments.

Now you can use the generic C STDIO features to write to another port:

```
FILE *can1 = fopen("c","w");
fprintf(can1,"This will be output through the can\n");
```

# 16-Bit Language Tools Libraries

---

## 2.13.3 STDIO Functions

---

### FILE

**Description:** Stores information for a file stream.  
**Include:** <stdio.h>

---

### fpos\_t

**Description:** Type of a variable used to store a file position.  
**Include:** <stdio.h>

---

### size\_t

**Description:** The result type of the `sizeof` operator.  
**Include:** <stdio.h>

---

### \_IOFBF

**Description:** Indicates full buffering.  
**Include:** <stdio.h>  
**Remarks:** Used by the function, `setvbuf`.

---

### \_IOLBF

**Description:** Indicates line buffering.  
**Include:** <stdio.h>  
**Remarks:** Used by the function, `setvbuf`.

---

### \_IONBF

**Description:** Indicates no buffering.  
**Include:** <stdio.h>  
**Remarks:** Used by the function, `setvbuf`.

---

### BUFSIZ

**Description:** Defines the size of the buffer used by the function `setbuf`.  
**Include:** <stdio.h>  
**Value:** 512

---

### EOF

**Description:** A negative number indicating the end-of-file has been reached or to report an error condition.  
**Include:** <stdio.h>  
**Remarks:** If an end-of-file is encountered, the end-of-file indicator is set. If an error condition is encountered, the error indicator is set. Error conditions include write errors and input or read errors.

---

---

## FILENAME\_MAX

---

**Description:** Maximum number of characters in a filename including the null terminator.

**Include:** <stdio.h>

**Value:** 260

---

---

## FOPEN\_MAX

---

**Description:** Defines the maximum number of files that can be simultaneously open.

**Include:** <stdio.h>

**Value:** 8

**Remarks:** stderr, stdin and stdout are included in the FOPEN\_MAX count.

---

---

## L\_tmpnam

---

**Description:** Defines the number of characters for the longest temporary filename created by the function tmpnam.

**Include:** <stdio.h>

**Value:** 16

**Remarks:** L\_tmpnam is used to define the size of the array used by tmpnam.

---

---

## NULL

---

**Description:** The value of a null pointer constant.

**Include:** <stdio.h>

---

---

## SEEK\_CUR

---

**Description:** Indicates that fseek should seek from the current position of the file pointer.

**Include:** <stdio.h>

**Example:** See example for fseek.

---

---

## SEEK\_END

---

**Description:** Indicates that fseek should seek from the end of the file.

**Include:** <stdio.h>

**Example:** See example for fseek.

---

---

## SEEK\_SET

---

**Description:** Indicates that fseek should seek from the beginning of the file.

**Include:** <stdio.h>

**Example:** See example for fseek.

---

---

## stderr

---

**Description:** File pointer to the standard error stream.

**Include:** <stdio.h>

---

# 16-Bit Language Tools Libraries

---

---

---

## stdin

---

**Description:** File pointer to the standard input stream.

**Include:** <stdio.h>

---

## stdout

---

**Description:** File pointer to the standard output stream.

**Include:** <stdio.h>

---

## TMP\_MAX

---

**Description:** The maximum number of unique filenames the function `tmpnam` can generate.

**Include:** <stdio.h>

**Value:** 32

---

---

## clearerr

---

**Description:** Resets the error indicator for the stream.

**Include:** <stdio.h>

**Prototype:** void clearerr(FILE \*stream);

**Argument:** stream stream to reset error indicators

**Remarks:** The function clears the end-of-file and error indicators for the given stream (i.e., feof and ferror will return false after the function clearerr is called).

**Example:**

```
/* This program tries to write to a file that is */
/* readonly. This causes the error indicator to */
/* be set. The function ferror is used to check */
/* the error indicator. The function clearerr is */
/* used to reset the error indicator so the next */
/* time ferror is called it will not report an */
/* error. */
#include <stdio.h> /* for ferror, clearerr, */
                  /* printf, fprintf, fopen, */
                  /* fclose, FILE, NULL */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("sampclearerr.c", "r")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fprintf(myfile, "Write this line to the "
                "file.\n");
        if (ferror(myfile))
            printf("Error\n");
        else
            printf("No error\n");
        clearerr(myfile);
        if (ferror(myfile))
            printf("Still has Error\n");
        else
            printf("Error indicator reset\n");

        fclose(myfile);
    }
}
```

**Output:**

```
Error
Error indicator reset
```

---

## fclose

---

**Description:** Close a stream.

**Include:** <stdio.h>

**Prototype:** int fclose(FILE \*stream);

**Argument:** stream pointer to the stream to close

**Return Value:** Returns 0 if successful; otherwise, returns EOF if any errors were detected.

**Remarks:** fclose writes any buffered output to the file.  
This function requires a heap.

**Example:**

```
#include <stdio.h> /* for fopen, fclose, */
                  /* printf, FILE, NULL, EOF */

int main(void)
{
    FILE *myfile1, *myfile2;
    int y;

    if ((myfile1 = fopen("afile1", "w+")) == NULL)
        printf("Cannot open afile1\n");
    else
    {
        printf("afile1 was opened\n");

        y = fclose(myfile1);
        if (y == EOF)
            printf("afile1 was not closed\n");
        else
            printf("afile1 was closed\n");
    }
}
```

**Output:**

```
afile1 was opened
afile1 was closed
```

## fdopen

---

**Description:** Associate a stream with a file descriptor.

**Include:** `<stdio.h>`

**Prototype:** `FILE *fdopen(int fil-des, const char *mode);`

**Arguments:** *fil-des* file descriptor  
*mode* type of access permitted

**Return Value:** Returns a pointer to the open stream. If the function fails, a null pointer is returned.

**Remarks:** The following are types of file access:

- `r -` opens an existing text file for reading
- `w -` opens an empty text file for writing.
- `a -` opens a text file for appending.
- `rb -` opens an existing binary file for reading.
- `wb -` opens an empty binary file for writing.
- `ab -` opens a binary file for appending.
- `r+ -` opens an existing text file for reading and writing.
- `w+ -` opens an empty text file for reading and writing.
- `a+ -` opens a text file for reading and appending.
- `r+b or rb+ -` opens an existing binary file for reading and writing.
- `w+b or wb+ -` opens an empty binary file for reading and writing.
- `a+b or ab+ -` opens a binary file for reading and appending.

This function requires a heap.

**Example:**

```
#include <stdio.h> /* for fdopen, fclose, */
                /* printf, FILE,      */
                /* NULL, EOF          */

int main(void)
{
    FILE *myfile1, *myfile2;
    int y;
```

## fdopen (Continued)

---

```
if ((myfile1 = fdopen("afile1", "r")) == NULL)
    printf("Cannot open afile1\n");
else
{
    printf("afile1 was opened\n");
    y = fclose(myfile1);
    if (y == EOF)
        printf("afile1 was not closed\n");
    else
        printf("afile1 was closed\n");
}

if ((myfile1 = fdopen("afile1", "w+")) == NULL)
    printf("Second try, cannot open afile1\n");
else
{
    printf("Second try, afile1 was opened\n");
    y = fclose(myfile1);
    if (y == EOF)
        printf("afile1 was not closed\n");
    else
        printf("afile1 was closed\n");
}

if ((myfile2 = fdopen("afile2", "a+")) == NULL)
    printf("Cannot open afile2\n");
else
{
    printf("afile2 was opened\n");
    y = fclose(myfile2);
    if (y == EOF)
        printf("afile2 was not closed\n");
    else
        printf("afile2 was closed\n");
}
}
```

### Output:

```
Cannot open afile1
Second try, afile1 was opened
afile1 was closed
Cannot open afile2
```

### Explanation:

afile1 must exist before it can be opened for reading (r) or the fdopen function will fail.

If the fdopen function opens a file for writing (w+) that doesn't exist, it will be created and then opened. If the file does exist, it cannot be over-written and will be appended.

If the fdopen function attempts to append a file (a+) that doesn't exist, it will NOT be created and fdopen will fail. If the file does exist, it will be opened for appending.



## feof

---

**Description:** Tests for end-of-file.  
**Include:** <stdio.h>  
**Prototype:** int feof(FILE \*stream);  
**Argument:** stream stream to check for end-of-file  
**Return Value:** Returns non-zero if stream is at the end-of-file; otherwise, returns zero.  
**Example:**

```
#include <stdio.h> /* for feof, fgetc, fputc, */
                  /* fopen, fclose, FILE, */
                  /* NULL */

int main(void)
{
    FILE *myfile;
    int y = 0;

    if ( (myfile = fopen( "afile.txt", "rb" )) == NULL )
        printf( "Cannot open file\n" );
    else
    {
        for (;;)
        {
            y = fgetc(myfile);
            if (feof(myfile))
                break;
            fputc(y, stdout);
        }
        fclose( myfile );
    }
}
```

**Input:**

Contents of afile.txt (used as input):  
This is a sentence.

**Output:**

This is a sentence.

# 16-Bit Language Tools Libraries

---

---

## **fclose**

---

**Description:** Tests if error indicator is set.

**Include:** <stdio.h>

**Prototype:** int fclose(FILE \*stream);

**Argument:** stream pointer to FILE structure

**Return Value:** Returns a non-zero value if error indicator is set; otherwise, returns a zero.

**Example:**

```
/* This program tries to write to a file that is */
/* readonly. This causes the error indicator to */
/* be set. The function fclose is used to check */
/* the error indicator and find the error. The */
/* function clearerr is used to reset the error */
/* indicator so the next time fclose is called */
/* it will not report an error. */
```

```
#include <stdio.h> /* for fclose, clearerr, */
                  /* printf, fprintf, */
                  /* fopen, fclose, */
                  /* FILE, NULL */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samclearerr.c", "r")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fprintf(myfile, "Write this line to the "
                "file.\n");
        if (fclose(myfile))
            printf("Error\n");
        else
            printf("No error\n");
        clearerr(myfile);
        if (fclose(myfile))
            printf("Still has Error\n");
        else
            printf("Error indicator reset\n");

        fclose(myfile);
    }
}
```

**Output:**

```
Error
Error indicator reset
```

---

## fflush

---

**Description:** Flushes the buffer in the specified stream.

**Include:** <stdio.h>

**Prototype:** `int fflush(FILE *stream);`

**Argument:** *stream* pointer to the stream to flush.

**Return Value:** Returns EOF if a write error occurs; otherwise, returns zero for success.

**Remarks:** If *stream* is a null pointer, all output buffers are written to files. `fflush` has no effect on an unbuffered stream.

---

---

## fgetc

---

**Description:** Get a character from a stream

**Include:** <stdio.h>

**Prototype:** `int fgetc(FILE *stream);`

**Argument:** *stream* pointer to the open stream

**Return Value:** Returns the character read or EOF if a read error occurs or end-of-file is reached.

**Remarks:** The function reads the next character from the input stream, advances the file-position indicator and returns the character as an unsigned char converted to an int.

**Example:**

```
#include <stdio.h> /* for fgetc, printf, */
                  /* fclose, FILE, */
                  /* NULL, EOF */

int main(void)
{
    FILE *buf;
    char y;

    if ((buf = fopen("afile.txt", "r")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        y = fgetc(buf);
        while (y != EOF)
        {
            printf("%c|", y);
            y = fgetc(buf);
        }
        fclose(buf);
    }
}
```

**Input:**

Contents of afile.txt (used as input):

Short

Longer string

**Output:**

```
S|h|o|r|t|
|L|o|n|g|e|r| |s|t|r|i|n|g|
|
```

# 16-Bit Language Tools Libraries

---

---

## fgetpos

---

**Description:** Gets the stream's file position.

**Include:** <stdio.h>

**Prototype:** int fgetpos(FILE \*stream, fpos\_t \*pos);

**Arguments:** stream target stream  
pos position-indicator storage

**Return Value:** Returns 0 if successful; otherwise, returns a non-zero value.

**Remarks:** The function stores the file-position indicator for the given stream in \*pos if successful, otherwise, fgetpos sets errno.

**Example:**

```
/* This program opens a file and reads bytes at */
/* several different locations. The fgetpos */
/* function notes the 8th byte. 21 bytes are */
/* read then 18 bytes are read. Next the */
/* fsetpos function is set based on the */
/* fgetpos position and the previous 21 bytes */
/* are reread. */
```

```
#include <stdio.h> /* for fgetpos, fread, */
                  /* printf, fopen, fclose, */
                  /* FILE, NULL, perror, */
                  /* fpos_t, sizeof */
```

```
int main(void)
{
    FILE *myfile;
    fpos_t pos;
    char buf[25];

    if ((myfile = fopen("sampfgetpos.c", "rb")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fread(buf, sizeof(char), 8, myfile);
        if (fgetpos(myfile, &pos) != 0)
            perror("fgetpos error");
        else
        {
            fread(buf, sizeof(char), 21, myfile);
            printf("Bytes read: %.21s\n", buf);
            fread(buf, sizeof(char), 18, myfile);
            printf("Bytes read: %.18s\n", buf);
        }

        if (fsetpos(myfile, &pos) != 0)
            perror("fsetpos error");

        fread(buf, sizeof(char), 21, myfile);
        printf("Bytes read: %.21s\n", buf);
        fclose(myfile);
    }
}
```

**Output:**

```
Bytes read: program opens a file
Bytes read: and reads bytes at
Bytes read: program opens a file
```

## fgets

---

**Description:** Get a string from a stream.

**Include:** <stdio.h>

**Prototype:** `char *fgets(char *s, int n, FILE *stream);`

**Arguments:**

- `s` pointer to the storage string
- `n` maximum number of characters to read
- `stream` pointer to the open stream.

**Return Value:** Returns a pointer to the string `s` if successful; otherwise, returns a null pointer.

**Remarks:** The function reads characters from the input stream and stores them into the string pointed to by `s` until it has read `n-1` characters, stores a newline character or sets the end-of-file or error indicators. If any characters were stored, a null character is stored immediately after the last read character in the next element of the array. If `fgets` sets the error indicator, the array contents are indeterminate.

**Example:**

```
#include <stdio.h> /* for fgets, printf, */
                  /* fopen, fclose, */
                  /* FILE, NULL */

#define MAX 50

int main(void)
{
    FILE *buf;
    char s[MAX];

    if ((buf = fopen("afile.txt", "r")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        while (fgets(s, MAX, buf) != NULL)
        {
            printf("%s|", s);
        }
        fclose(buf);
    }
}
```

**Input:**

Contents of `afile.txt` (used as input):

Short

Longer string

**Output:**

Short

|Longer string

|

# 16-Bit Language Tools Libraries

---

---

## fopen

---

**Description:** Opens a file.

**Include:** <stdio.h>

**Prototype:** FILE \*fopen(const char \*filename, const char \*mode);

**Arguments:** filename name of the file  
mode type of access permitted

**Return Value:** Returns a pointer to the open stream. If the function fails a null pointer is returned.

**Remarks:** The following are types of file access:

- r - opens an existing text file for reading
- w - opens an empty text file for writing. (An existing file will be overwritten).
- a - opens a text file for appending. (A file is created if it doesn't exist).
- rb - opens an existing binary file for reading.
- wb - opens an empty binary file for writing. (An existing file will be overwritten).
- ab - opens a binary file for appending. (A file is created if it doesn't exist).
- r+ - opens an existing text file for reading and writing.
- w+ - opens an empty text file for reading and writing. (An existing file will be overwritten).
- a+ - opens a text file for reading and appending. (A file is created if it doesn't exist).
- r+b or rb+ - opens an existing binary file for reading and writing.
- w+b or wb+ - opens an empty binary file for reading and writing. (An existing file will be overwritten).
- a+b or ab+ - opens a binary file for reading and appending. (A file is created if it doesn't exist).

This function requires a heap.

**Example:**

```
#include <stdio.h> /* for fopen, fclose, */
                    /* printf, FILE, */
                    /* NULL, EOF */

int main(void)
{
    FILE *myfile1, *myfile2;
    int y;
```

## fopen (Continued)

---

```
if ((myfile1 = fopen("afile1", "r")) == NULL)
    printf("Cannot open afile1\n");
else
{
    printf("afile1 was opened\n");
    y = fclose(myfile1);
    if (y == EOF)
        printf("afile1 was not closed\n");
    else
        printf("afile1 was closed\n");
}

if ((myfile1 = fopen("afile1", "w+")) == NULL)
    printf("Second try, cannot open afile1\n");
else
{
    printf("Second try, afile1 was opened\n");
    y = fclose(myfile1);
    if (y == EOF)
        printf("afile1 was not closed\n");
    else
        printf("afile1 was closed\n");
}

if ((myfile2 = fopen("afile2", "w+")) == NULL)
    printf("Cannot open afile2\n");
else
{
    printf("afile2 was opened\n");
    y = fclose(myfile2);
    if (y == EOF)
        printf("afile2 was not closed\n");
    else
        printf("afile2 was closed\n");
}
}
```

### Output:

```
Cannot open afile1
Second try, afile1 was opened
afile1 was closed
afile2 was opened
afile2 was closed
```

### Explanation:

afile1 must exist before it can be opened for reading (r) or the fopen function will fail. If the fopen function opens a file for writing (w+) it does not have to already exist. If it doesn't exist, it will be created and then opened.

# 16-Bit Language Tools Libraries

---

---

## **fprintf**

---

**Description:** Prints formatted data to a stream.

**Include:** <stdio.h>

**Prototype:** `int fprintf(FILE *stream, const char *format, ...);`

**Arguments:**

<i>stream</i>	pointer to the stream in which to output data
<i>format</i>	format control string
...	optional arguments

**Return Value:** Returns number of characters generated or a negative number if an error occurs.

**Remarks:** The format argument has the same syntax and use that it has in `print`.

**Example:**

```
#include <stdio.h> /* for fopen, fclose, */
                    /* fprintf, printf, */
                    /* FILE, NULL */

int main(void)
{
    FILE *myfile;
    int y;
    char s[]="Print this string";
    int x = 1;
    char a = '\n';

    if ((myfile = fopen("afile", "w")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        y = fprintf(myfile, "%s %d time%c", s, x, a);

        printf("Number of characters printed "
               "to file = %d",y);

        fclose(myfile);
    }
}
```

**Output:**

Number of characters printed to file = 25  
Contents of afile:  
Print this string 1 time



---

## fputc

---

**Description:** Puts a character to the stream.

**Include:** <stdio.h>

**Prototype:** `int fputc(int c, FILE *stream);`

**Arguments:** *c* character to be written  
*stream* pointer to the open stream

**Return Value:** Returns the character written or EOF if a write error occurs.

**Remarks:** The function writes the character to the output stream, advances the file-position indicator and returns the character as an unsigned char converted to an int.

**Example:** `#include <stdio.h> /* for fputc, EOF, stdout */`

```
int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;

    x = 0;

    for (y = buf; (x != EOF) && (*y != '\\0'); y++)
    {
        x = fputc(*y, stdout);
        fputc('|', stdout);
    }
}
```

**Output:**

```
T|h|i|s| |i|s| |t|e|x|t|
|
```

---

## fputs

---

**Description:** Puts a string to the stream.

**Include:** <stdio.h>

**Prototype:** `int fputs(const char *s, FILE *stream);`

**Arguments:** *s* string to be written  
*stream* pointer to the open stream

**Return Value:** Returns a non-negative value if successful; otherwise, returns EOF.

**Remarks:** The function writes characters to the output stream up to but not including the null character.

**Example:** `#include <stdio.h> /* for fputs, stdout */`

```
int main(void)
{
    char buf[] = "This is text\n";

    fputs(buf, stdout);
    fputs("|", stdout);
}
```

**Output:**

```
This is text
|
```

# 16-Bit Language Tools Libraries

---

---

## fread

---

**Description:** Reads data from the stream.

**Include:** <stdio.h>

**Prototype:** `size_t fread(void *ptr, size_t size, size_t nelem, FILE *stream);`

**Arguments:**

- `ptr` pointer to the storage buffer
- `size` size of item
- `nelem` maximum number of items to be read
- `stream` pointer to the stream

**Return Value:** Returns the number of complete elements read up to `nelem` whose size is specified by `size`.

**Remarks:** The function reads characters from a given stream into the buffer pointed to by `ptr` until the function stores `size * nelem` characters or sets the end-of-file or error indicator. `fread` returns `n/size` where `n` is the number of characters it read. If `n` is not a multiple of `size`, the value of the last element is indeterminate. If the function sets the error indicator, the file-position indicator is indeterminate.

**Example:**

```
#include <stdio.h> /* for fread, fwrite, */
                    /* printf, fopen, fclose, */
                    /* sizeof, FILE, NULL */

int main(void)
{
    FILE *buf;
    int x, numwrote, numread;
    double nums[10], readnums[10];

    if ((buf = fopen("afile.out", "w+")) != NULL)
    {
        for (x = 0; x < 10; x++)
        {
            nums[x] = 10.0/(x + 1);
            printf("10.0/%d = %f\n", x+1, nums[x]);
        }

        numwrote = fwrite(nums, sizeof(double),
                          10, buf);
        printf("Wrote %d numbers\n\n", numwrote);
        fclose(buf);
    }
    else
        printf("Cannot open afile.out\n");
}
```

## fread (Continued)

---

```
if ((buf = fopen("afile.out", "r+")) != NULL)
{
    numread = fread(readnums, sizeof(double),
                    10, buf);
    printf("Read %d numbers\n", numread);
    for (x = 0; x < 10; x++)
    {
        printf("%d * %f = %f\n", x+1, readnums[x],
              (x + 1) * readnums[x]);
    }
    fclose(buf);
}
else
    printf("Cannot open afile.out\n");
}
```

### Output:

```
10.0/1 = 10.000000
10.0/2 = 5.000000
10.0/3 = 3.333333
10.0/4 = 2.500000
10.0/5 = 2.000000
10.0/6 = 1.666667
10.0/7 = 1.428571
10.0/8 = 1.250000
10.0/9 = 1.111111
10.0/10 = 1.000000
Wrote 10 numbers
```

```
Read 10 numbers
1 * 10.000000 = 10.000000
2 * 5.000000 = 10.000000
3 * 3.333333 = 10.000000
4 * 2.500000 = 10.000000
5 * 2.000000 = 10.000000
6 * 1.666667 = 10.000000
7 * 1.428571 = 10.000000
8 * 1.250000 = 10.000000
9 * 1.111111 = 10.000000
10 * 1.000000 = 10.000000
```

### Explanation:

This program uses `fwrite` to save 10 numbers to a file in binary form. This allows the numbers to be saved in the same pattern of bits as the program is using, which provides more accuracy and consistency. Using `fprintf` would save the numbers as text strings, which could cause the numbers to be truncated. Each number is divided into 10 to produce a variety of numbers. Retrieving the numbers with `fread` to a new array and multiplying them by the original number shows the numbers were not truncated in the save process.

# 16-Bit Language Tools Libraries

---

---

---

## freopen

---

**Description:** Reassigns an existing stream to a new file.

**Include:** <stdio.h>

**Prototype:** FILE \*freopen(const char \*filename, const char \*mode, FILE \*stream);

**Arguments:** *filename* name of the new file  
*mode* type of access permitted  
*stream* pointer to the currently open stream

**Return Value:** Returns a pointer to the new open file. If the function fails a null pointer is returned.

**Remarks:** The function closes the file associated with the stream as though `fclose` was called. Then it opens the new file as though `fopen` was called. `freopen` will fail if the specified stream is not open. See `fopen` for the possible types of file access. This function requires a heap.

**Example:**

```
#include <stdio.h> /* for fopen, freopen, */
                    /* printf, fclose, */
                    /* FILE, NULL */

int main(void)
{
    FILE *myfile1, *myfile2;
    int y;

    if ((myfile1 = fopen("afile1", "w+")) == NULL)
        printf("Cannot open afile1\n");
    else
    {
        printf("afile1 was opened\n");

        if ((myfile2 = freopen("afile2", "w+",
            myfile1)) == NULL)
        {
            printf("Cannot open afile2\n");
            fclose(myfile1);
        }
        else
        {
            printf("afile2 was opened\n");
            fclose(myfile2);
        }
    }
}
```

**Output:**

```
afile1 was opened
afile2 was opened
```

**Explanation:**

This program uses `myfile2` to point to the stream when `freopen` is called, so if an error occurs, `myfile1` will still point to the stream and can be closed properly. If the `freopen` call is successful, `myfile2` can be used to close the stream properly.

---

## fscanf

---

**Description:** Scans formatted text from a stream.

---

## fscanf (Continued)

---

**Include:** `<stdio.h>`

**Prototype:** `int fscanf(FILE *stream, const char *format, ...);`

**Arguments:** *stream* pointer to the open stream from which to read data  
*format* format control string  
... optional arguments

**Return Value:** Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if end-of-file is encountered before the first conversion or if an error occurs.

**Remarks:** The format argument has the same syntax and use that it has in `scanf`.

**Example:**

```
#include <stdio.h> /* for fopen, fscanf, */
                  /* fclose, fprintf, */
                  /* fseek, printf, FILE, */
                  /* NULL, SEEK_SET */
```

```
int main(void)
{
    FILE *myfile;
    char s[30];
    int x;
    char a;

    if ((myfile = fopen("afile", "w+")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        fprintf(myfile, "%s %d times%c",
                "Print this string", 100, '\n');

        fseek(myfile, 0L, SEEK_SET);

        fscanf(myfile, "%s", s);
        printf("%s\n", s);
        fscanf(myfile, "%s", s);
        printf("%s\n", s);
        fscanf(myfile, "%s", s);
        printf("%s\n", s);
        fscanf(myfile, "%d", &x);
        printf("%d\n", x);
        fscanf(myfile, "%s", s);
        printf("%s\n", s);
        fscanf(myfile, "%c", a);
        printf("%c\n", a);

        fclose(myfile);
    }
}
```

**Input:**  
Contents of afile:  
Print this string 100 times

---

## fscanf (Continued)

---

**Output:**

Print  
this  
string  
100  
times

---

---

## fseek

---

**Description:** Moves file pointer to a specific location.

**Include:** <stdio.h>

**Prototype:** `int fseek(FILE *stream, long offset, int mode);`

**Arguments:**

<i>stream</i>	stream in which to move the file pointer.
<i>offset</i>	value to add to the current position
<i>mode</i>	type of seek to perform

**Return Value:** Returns 0 if successful; otherwise, returns a non-zero value and set `errno`.

**Remarks:** `mode` can be one of the following:  
`SEEK_SET` – seeks from the beginning of the file  
`SEEK_CUR` – seeks from the current position of the file pointer  
`SEEK_END` – seeks from the end of the file

**Example:**

```
#include <stdio.h> /* for fseek, fgets,      */
                    /* printf, fopen, fclose, */
                    /* FILE, NULL, perror,   */
                    /* SEEK_SET, SEEK_CUR,   */
                    /* SEEK_END              */

int main(void)
{
    FILE *myfile;
    char s[70];
    int y;

    myfile = fopen("afile.out", "w+");
    if (myfile == NULL)
        printf("Cannot open afile.out\n");
    else
    {
        fprintf(myfile, "This is the beginning, "
                    "this is the middle and "
                    "this is the end.");

        y = fseek(myfile, 0L, SEEK_SET);
        if (y)
            perror("Fseek failed");
        else
        {
            fgets(s, 22, myfile);
            printf("\n%s\n\n", s);
        }
    }
}
```

---

## fseek (Continued)

---

```
y = fseek(myfile, 2L, SEEK_CUR);
if (y)
    perror("Fseek failed");
else
{
    fgets(s, 70, myfile);
    printf("\n%s\n\n", s);
}

y = fseek(myfile, -16L, SEEK_END);
if (y)
    perror("Fseek failed");
else
{
    fgets(s, 70, myfile);
    printf("\n%s\n\n", s);
}
fclose(myfile);
}
```

### Output:

"This is the beginning"

"this is the middle and this is the end."

"this is the end."

### Explanation:

The file `afile.out` is created with the text, "This is the beginning, this is the middle and this is the end".

The function `fseek` uses an offset of zero and `SEEK_SET` to set the file pointer to the beginning of the file. `fgets` then reads 22 characters which are "This is the beginning", and adds a null character to the string.

Next, `fseek` uses an offset of two and `SEEK_CURRENT` to set the file pointer to the current position plus two (skipping the comma and space). `fgets` then reads up to the next 70 characters. The first 39 characters are "this is the middle and this is the end". It stops when it reads EOF and adds a null character to the string.

Finally, `fseek` uses an offset of negative 16 characters and `SEEK_END` to set the file pointer to 16 characters from the end of the file. `fgets` then reads up to 70 characters. It stops at the EOF after reading 16 characters "this is the end", and adds a null character to the string.

---

## fsetpos

---

<b>Description:</b>	Sets the stream's file position.
<b>Include:</b>	<code>&lt;stdio.h&gt;</code>
<b>Prototype:</b>	<code>int fsetpos(FILE *stream, const fpos_t *pos);</code>
<b>Arguments:</b>	<i>stream</i> target stream <i>pos</i> position-indicator storage as returned by an earlier call to <code>fgetpos</code>
<b>Return Value:</b>	Returns 0 if successful; otherwise, returns a non-zero value.
<b>Remarks:</b>	The function sets the file-position indicator for the given stream in <i>pos</i> if successful; otherwise, <code>fsetpos</code> sets <code>errno</code> .

## fsetpos (Continued)

---

```
Example:      /* This program opens a file and reads bytes at */
                /* several different locations. The fgetpos      */
                /* function notes the 8th byte. 21 bytes are    */
                /* read then 18 bytes are read. Next, the      */
                /* fsetpos function is set based on the        */
                /* fgetpos position and the previous 21 bytes  */
                /* are reread.                                  */

#include <stdio.h> /* for fgetpos, fread,      */
                  /* printf, fopen, fclose, */
                  /* FILE, NULL, perror,    */
                  /* fpos_t, sizeof        */

int main(void)
{
    FILE    *myfile;
    fpos_t  pos;
    char    buf[25];

    if ((myfile = fopen("sampfgetpos.c", "rb")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fread(buf, sizeof(char), 8, myfile);
        if (fgetpos(myfile, &pos) != 0)
            perror("fgetpos error");
        else
        {
            fread(buf, sizeof(char), 21, myfile);
            printf("Bytes read: %.21s\n", buf);
            fread(buf, sizeof(char), 18, myfile);
            printf("Bytes read: %.18s\n", buf);
        }

        if (fsetpos(myfile, &pos) != 0)
            perror("fsetpos error");

        fread(buf, sizeof(char), 21, myfile);
        printf("Bytes read: %.21s\n", buf);
        fclose(myfile);
    }
}
```

### Output:

```
Bytes read: program opens a file
Bytes read: and reads bytes at
Bytes read: program opens a file
```



## ftell

---

**Description:** Gets the current position of a file pointer.

**Include:** <stdio.h>

**Prototype:** long ftell(FILE \*stream);

**Argument:** *stream* stream in which to get the current file position

**Return Value:** Returns the position of the file pointer if successful; otherwise, returns -1.

**Example:**

```
#include <stdio.h> /* for ftell, fread,      */
                  /* fprintf, printf,     */
                  /* fopen, fclose, sizeof, */
                  /* FILE, NULL */

int main(void)
{
    FILE *myfile;
    char s[75];
    long y;

    myfile = fopen("afile.out", "w+");
    if (myfile == NULL)
        printf("Cannot open afile.out\n");
    else
    {
        fprintf(myfile, "This is a very long sentence "
                    "for input into the file named "
                    "afile.out for testing.");

        fclose(myfile);

        if ((myfile = fopen("afile.out", "rb")) != NULL)
        {
            printf("Read some characters:\n");
            fread(s, sizeof(char), 29, myfile);
            printf("\t\"%s\"\n", s);

            y = ftell(myfile);
            printf("The current position of the "
                "file pointer is %ld\n", y);
            fclose(myfile);
        }
    }
}
```

**Output:**

```
Read some characters:
    "This is a very long sentence "
The current position of the file pointer is 29
```

# 16-Bit Language Tools Libraries

---

---

## **fwrite**

---

**Description:** Writes data to the stream.

**Include:** <stdio.h>

**Prototype:** `size_t fwrite(const void *ptr, size_t size, size_t nelem, FILE *stream);`

**Arguments:**

- `ptr` pointer to the storage buffer
- `size` size of item
- `nelem` maximum number of items to be read
- `stream` pointer to the open stream

**Return Value:** Returns the number of complete elements successfully written, which will be less than `nelem` only if a write error is encountered.

**Remarks:** The function writes characters to a given stream from a buffer pointed to by `ptr` up to `nelem` elements whose size is specified by `size`. The file position indicator is advanced by the number of characters successfully written. If the function sets the error indicator, the file-position indicator is indeterminate.

**Example:**

```
#include <stdio.h> /* for fread, fwrite,      */
                    /* printf, fopen, fclose, */
                    /* sizeof, FILE, NULL     */

int main(void)
{
    FILE *buf;
    int x, numwrote, numread;
    double nums[10], readnums[10];

    if ((buf = fopen("afile.out", "w+")) != NULL)
    {
        for (x = 0; x < 10; x++)
        {
            nums[x] = 10.0/(x + 1);
            printf("10.0/%d = %f\n", x+1, nums[x]);
        }

        numwrote = fwrite(nums, sizeof(double),
                          10, buf);
        printf("Wrote %d numbers\n\n", numwrote);
        fclose(buf);
    }
    else
        printf("Cannot open afile.out\n");
}
```

## **fwrite (Continued)**

---

```
if ((buf = fopen("afile.out", "r+")) != NULL)
{
    numread = fread(readnums, sizeof(double),
                    10, buf);
    printf("Read %d numbers\n", numread);
    for (x = 0; x < 10; x++)
    {
        printf("%d * %f = %f\n", x+1, readnums[x],
              (x + 1) * readnums[x]);
    }
    fclose(buf);
}
else
    printf("Cannot open afile.out\n");
}
```

### **Output:**

```
10.0/1 = 10.000000
10.0/2 = 5.000000
10.0/3 = 3.333333
10.0/4 = 2.500000
10.0/5 = 2.000000
10.0/6 = 1.666667
10.0/7 = 1.428571
10.0/8 = 1.250000
10.0/9 = 1.111111
10.0/10 = 1.000000
Wrote 10 numbers
```

```
Read 10 numbers
1 * 10.000000 = 10.000000
2 * 5.000000 = 10.000000
3 * 3.333333 = 10.000000
4 * 2.500000 = 10.000000
5 * 2.000000 = 10.000000
6 * 1.666667 = 10.000000
7 * 1.428571 = 10.000000
8 * 1.250000 = 10.000000
9 * 1.111111 = 10.000000
10 * 1.000000 = 10.000000
```

### **Explanation:**

This program uses `fwrite` to save 10 numbers to a file in binary form. This allows the numbers to be saved in the same pattern of bits as the program is using which provides more accuracy and consistency. Using `fprintf` would save the numbers as text strings, which could cause the numbers to be truncated. Each number is divided into 10 to produce a variety of numbers. Retrieving the numbers with `fread` to a new array and multiplying them by the original number shows the numbers were not truncated in the save process.

# 16-Bit Language Tools Libraries

---

---

## getc

---

**Description:** Get a character from the stream.

**Include:** <stdio.h>

**Prototype:** int getc(FILE \*stream);

**Argument:** stream pointer to the open stream

**Return Value:** Returns the character read or EOF if a read error occurs or end-of-file is reached.

**Remarks:** getc is the same as the function fgetc.

**Example:**

```
#include <stdio.h> /* for getc, printf, */
                    /* fopen, fclose, */
                    /* FILE, NULL, EOF */
```

```
int main(void)
{
    FILE *buf;
    char y;

    if ((buf = fopen("afile.txt", "r")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        y = getc(buf);
        while (y != EOF)
        {
            printf("%c|", y);
            y = getc(buf);
        }
        fclose(buf);
    }
}
```

**Input:**

Contents of afile.txt (used as input):

Short

Longer string

**Output:**

S|h|o|r|t|

|L|o|n|g|e|r| |s|t|r|i|n|g|

|

---

## getchar

---

**Description:** Get a character from `stdin`.

**Include:** `<stdio.h>`

**Prototype:** `int getchar(void);`

**Return Value:** Returns the character read or EOF if a read error occurs or end-of-file is reached.

**Remarks:** Same effect as `fgetc` with the argument `stdin`.

**Example:** `#include <stdio.h> /* for getchar, printf */`

```
int main(void)
{
    char y;

    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
}
```

**Input:**

Contents of `UartIn.txt` (used as `stdin` input for simulator):

Short

Longer string

**Output:**

S|h|o|r|t|

---

## gets

---

**Description:** Get a string from `stdin`.

**Include:** `<stdio.h>`

**Prototype:** `char *gets(char *s);`

**Argument:** `s` pointer to the storage string

**Return Value:** Returns a pointer to the string `s` if successful; otherwise, returns a null pointer.

**Remarks:** The function reads characters from the stream `stdin` and stores them into the string pointed to by `s` until it reads a newline character (which is not stored) or sets the end-of-file or error indicators. If any characters were read, a null character is stored immediately after the last read character in the next element of the array. If `gets` sets the error indicator, the array contents are indeterminate.

# 16-Bit Language Tools Libraries

---

---

---

## gets (Continued)

---

**Example:** `#include <stdio.h> /* for gets, printf */`

```
int main(void)
{
    char y[50];

    gets(y) ;
    printf("Text: %s\n", y);
}
```

**Input:**

Contents of UartIn.txt (used as stdin input for simulator):

Short

Longer string

**Output:**

Text: Short

---

## perror

---

**Description:** Prints an error message to stderr.

**Include:** `<stdio.h>`

**Prototype:** `void perror(const char *s);`

**Argument:** `s` string to print

**Return Value:** None.

**Remarks:** The string `s` is printed followed by a colon and a space. Then, an error message based on `errno` is printed followed by an newline.

**Example:** `#include <stdio.h> /* for perror, fopen, */  
/* fclose, printf, */  
/* FILE, NULL */`

```
int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samp.fil", "r+")) == NULL)
        perror("Cannot open samp.fil");
    else
        printf("Success opening samp.fil\n");

    fclose(myfile);
}
```

**Output:**

Cannot open samp.fil: file open error

## printf

---

<b>Description:</b>	Prints formatted text to <code>stdout</code> .
<b>Include:</b>	<code>&lt;stdio.h&gt;</code>
<b>Prototype:</b>	<code>int printf(const char *format, ...);</code>
<b>Arguments:</b>	<code>format</code> format control string ...            optional arguments
<b>Return Value:</b>	Returns number of characters generated or a negative number if an error occurs.
<b>Remarks:</b>	<p>There must be exactly the same number of arguments as there are format specifiers. If there are less arguments than match the format specifiers, the output is undefined. If there are more arguments than match the format specifiers, the remaining arguments are discarded. Each format specifier begins with a percent sign followed by optional fields and a required type as shown here:</p> <pre>    %[flags][width][.precision][size]type</pre> <p><code>flags</code></p> <ul style="list-style-type: none"><li>-      left justify the value within a given field width</li><li>0      Use 0 for the pad character instead of space (which is the default)</li><li>+      generate a plus sign for positive signed values</li><li>space generate a space or signed values that have neither a plus nor a minus sign</li><li>#      to prefix 0 on an octal conversion, to prefix 0x or 0X on a hexadecimal conversion, or to generate a decimal point and fraction digits that are otherwise suppressed on a floating-point conversion</li></ul> <p><code>width</code></p> <p>specify the number of characters to generate for the conversion. If the asterisk (*) is used instead of a decimal number, the next argument (which must be of type <code>int</code>) will be used for the field width. If the result is less than the field width, pad characters will be used on the left to fill the field. If the result is greater than the field width, the field is expanded to accommodate the value without padding.</p> <p><code>precision</code></p> <p>The field width can be followed with dot (.) and a decimal integer representing the precision that specifies one of the following:</p> <ul style="list-style-type: none"><li>- minimum number of digits to generate on an integer conversion</li><li>- number of fraction digits to generate on an e, E, or f conversion</li><li>- maximum number of significant digits to generate on a g or G conversion</li><li>- maximum number of characters to generate from a C string on an s conversion</li></ul> <p>If the period appears without the integer, the integer is assumed to be zero. If the asterisk (*) is used instead of a decimal number, the next argument (which must be of type <code>int</code>) will be used for the precision.</p>

# 16-Bit Language Tools Libraries

---

---

## printf (Continued)

---

size	
h modifier –	used with type d, i, o, u, x, X; converts the value to a short int or unsigned short int
h modifier –	used with n; specifies that the pointer points to a short int
l modifier –	used with type d, i, o, u, x, X; converts the value to a long int or unsigned long int
l modifier –	used with n; specifies that the pointer points to a long int
l modifier –	used with c; specifies a wide character
l modifier –	used with type e, E, f, F, g, G; converts the value to a double
ll modifier –	used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int
ll modifier –	used with n; specifies that the pointer points to a long long int
L modifier –	used with e, E, f, g, G; converts the value to a long double
type	
d, i	signed int
o	unsigned int in octal
u	unsigned int in decimal
x	unsigned int in lowercase hexadecimal
X	unsigned int in uppercase hexadecimal
e, E	double in scientific notation
f	double decimal notation
g, G	double (takes the form of e, E or f as appropriate)
c	char - a single character
s	string
p	value of a pointer
n	the associated argument shall be an integer pointer into which is placed the number of characters written so far. No characters are printed.
%	A % character is printed

This function requires a heap.

### Example:

```
#include <stdio.h> /* for printf */

int main(void)
{
    /* print a character right justified in a 3 */
    /* character space. */
    printf("%3c\n", 'a');

    /* print an integer, left justified (as */
    /* specified by the minus sign in the format */
    /* string) in a 4 character space. Print a */
    /* second integer that is right justified in */
    /* a 4 character space using the pipe (|) as */
    /* a separator between the integers. */
    printf("%-4d|%4d\n", -4, 4);

    /* print a number converted to octal in 4 */
    /* digits. */
    printf("%.4o\n", 10);
}
```



## printf (Continued)

```

/* print a number converted to hexadecimal */
/* format with a 0x prefix. */
printf("%#x\n", 28);

/* print a float in scientific notation */
printf("%E\n", 1.1e20);

/* print a float with 2 fraction digits */
printf("%.2f\n", -3.346);

/* print a long float with %E, %e, or %f */
/* whichever is the shortest version */
printf("%Lg\n", .02L);
}

```

### Output:

```

a
-4 | 4
0012
0x1c
1.100000E+20
-3.35
0.02

```

## putc

**Description:** Puts a character to the stream.

**Include:** <stdio.h>

**Prototype:** int putc(int *c*, FILE \**stream*);

**Arguments:** *c* character to be written  
*stream* pointer to FILE structure

**Return Value:** Returns the character or EOF if an error occurs or end-of-file is reached.

**Remarks:** putc is the same as the function fputc.

**Example:** #include <stdio.h> /\* for putc, EOF, stdout \*/

```

int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;

    x = 0;

    for (y = buf; (x != EOF) && (*y != '\0'); y++)
    {
        x = putc(*y, stdout);
        putc('|', stdout);
    }
}

```

### Output:

```

T|h|i|s| |i|s| |t|e|x|t|
|

```

# 16-Bit Language Tools Libraries

---

---

---

## putchar

---

**Description:** Put a character to `stdout`.

**Include:** `<stdio.h>`

**Prototype:** `int putchar(int c);`

**Argument:** `c` character to be written

**Return Value:** Returns the character or EOF if an error occurs or end-of-file is reached.

**Remarks:** Same effect as `fputc` with `stdout` as an argument.

**Example:**

```
#include <stdio.h> /* for putchar, printf, */
                    /* EOF, stdout          */
```

```
int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;

    x = 0;

    for (y = buf; (x != EOF) && (*y != '\0'); y++)
        x = putchar(*y);
}
```

**Output:**  
This is text

---

## puts

---

**Description:** Put a string to `stdout`.

**Include:** `<stdio.h>`

**Prototype:** `int puts(const char *s);`

**Argument:** `s` string to be written

**Return Value:** Returns a non-negative value if successful; otherwise, returns EOF.

**Remarks:** The function writes characters to the stream `stdout`. A newline character is appended. The terminating null character is not written to the stream.

**Example:**

```
#include <stdio.h> /* for puts */
```

```
int main(void)
{
    char buf[] = "This is text\n";

    puts(buf);
    puts("|");
}
```

**Output:**  
This is text

|

---

## remove

---

**Description:** Deletes the specified file.  
**Include:** <stdio.h>  
**Prototype:** int remove(const char \*filename);  
**Argument:** filename name of file to be deleted  
**Return Value:** Returns 0 if successful; otherwise, returns -1.  
**Remarks:** If filename does not exist or is open, remove will fail.  
**Example:** #include <stdio.h> /\* for remove, printf \*/

```
int main(void)
{
    if (remove("myfile.txt") != 0)
        printf("Cannot remove file");
    else
        printf("File removed");
}
```

**Output:**  
File removed

---

## rename

---

**Description:** Renames the specified file.  
**Include:** <stdio.h>  
**Prototype:** int rename(const char \*old, const char \*new);  
**Arguments:** old pointer to the old name  
new pointer to the new name  
**Return Value:** Return 0 if successful; otherwise, returns a non-zero value.  
**Remarks:** The new name must not already exist in the current working directory, the old name must exist in the current working directory.  
**Example:** #include <stdio.h> /\* for rename, printf \*/

```
int main(void)
{
    if (rename("myfile.txt", "newfile.txt") != 0)
        printf("Cannot rename file");
    else
        printf("File renamed");
}
```

**Output:**  
File renamed

# 16-Bit Language Tools Libraries

---

---

---

## rewind

---

**Description:** Resets the file pointer to the beginning of the file.

**Include:** <stdio.h>

**Prototype:** void rewind(FILE \*stream);

**Argument:** stream stream to reset the file pointer

**Remarks:** The function calls fseek(stream, 0L, SEEK\_SET) and then clears the error indicator for the given stream.

**Example:**

```
#include <stdio.h> /* for rewind, fopen, */
                  /* fscanff, fclose, */
                  /* fprintf, printf, */
                  /* FILE, NULL */

int main(void)
{
    FILE *myfile;
    char s[] = "cookies";
    int x = 10;

    if ((myfile = fopen("afile", "w+")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        fprintf(myfile, "%d %s", x, s);
        printf("I have %d %s.\n", x, s);

        /* set pointer to beginning of file */
        rewind(myfile);
        fscanff(myfile, "%d %s", &x, &s);
        printf("I ate %d %s.\n", x, s);

        fclose(myfile);
    }
}
```

**Output:**  
I have 10 cookies.  
I ate 10 cookies.

## scanf

---

<b>Description:</b>	Scans formatted text from <code>stdin</code> .
<b>Include:</b>	<code>&lt;stdio.h&gt;</code>
<b>Prototype:</b>	<code>int scanf(const char *format, ...);</code>
<b>Argument:</b>	<code>format</code> format control string ...            optional arguments
<b>Return Value:</b>	Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if an input failure is encountered before the first.
<b>Remarks:</b>	Each format specifier begins with a percent sign followed by optional fields and a required type as shown here: <pre>  %[ * ][width][modifier]type</pre> <p>*</p> <p>indicates assignment suppression. This will cause the input field to be skipped and no assignment made.</p> <p>width</p> <p>specify the maximum number of input characters to match for the conversion, not including white space that can be skipped.</p> <p>modifier</p> <p>h modifier – used with type d, i, o, u, x, X; converts the value to a short int or unsigned short int.</p> <p>h modifier – used with n; specifies that the pointer points to a short int</p> <p>l modifier – used with type d, i, o, u, x, X; converts the value to a long int or unsigned long int</p> <p>l modifier – used with n; specifies that the pointer points to a long int</p> <p>l modifier – used with c; specifies a wide character</p> <p>l modifier – used with type e, E, f, F, g, G; converts the value to a double</p> <p>ll modifier – used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int</p> <p>ll modifier – used with n; specifies that the pointer points to a long long int</p> <p>L modifier – used with e, E, f, g, G; converts the value to a long double</p>

# 16-Bit Language Tools Libraries

---

---

## scanf (Continued)

---

type	
d,i	signed int
o	unsigned int in octal
u	unsigned int in decimal
x	unsigned int in lowercase hexadecimal
X	unsigned int in uppercase hexadecimal
e,E	double in scientific notation
f	double decimal notation
g,G	double (takes the form of e, E or f as appropriate)
c	char - a single character
s	string
p	value of a pointer
n	the associated argument shall be an integer pointer into, which is placed the number of characters read so far. No characters are scanned.
[...]	character array. Allows a search of a set of characters. A caret (^) immediately after the left bracket ( [ ) inverts the scanset and allows any ASCII character except those specified between the brackets. A dash character (-) may be used to specify a range beginning with the character before the dash and ending the character after the dash. A null character can not be part of the scanset.
%	A % character is scanned

### Example:

```
For MPLAB SIM simulator.
#include <stdio.h> /* for scanf, printf */
#include <libpic30.h>

int main(void)
{
    int number, items;
    char letter;
    char color[30], string[30];
    float salary;

    __attach_input_file("UartIn.txt");
    printf("Enter your favorite number, "
           "favorite letter, ");
    printf("favorite color desired salary "
           "and SSN:\n");
    items = scanf("%d %c %[A-Za-z] %f %s", &number,
                 &letter, &color, &salary, &string);

    printf("Number of items scanned = %d\n", items);
    printf("Favorite number = %d, ", number);
    printf("Favorite letter = %c\n", letter);
    printf("Favorite color = %s, ", color);
    printf("Desired salary = $%.2f\n", salary);
    printf("Social Security Number = %s, ", string);
}

If not using the simulator, remove these lines:
#include <libpic30.h>
__attach_input_file("uart_in.txt");
```

**Input:**  
Contents of UartIn.txt (used as stdin input for simulator):  
5 T Green 300000 123-45-6789

---

## scanf (Continued)

---

**Output:**

```
Enter your favorite number, favorite letter,  
favorite color, desired salary and SSN:  
Number of items scanned = 5  
Favorite number = 5, Favorite letter = T  
Favorite color = Green, Desired salary = $300000.00  
Social Security Number = 123-45-6789
```

---

## setbuf

---

**Description:** Defines how a stream is buffered.

**Include:** <stdio.h>

**Prototype:** void setbuf(FILE \*stream, char \*buf);

**Arguments:** *stream* pointer to the open stream  
*buf* user allocated buffer

**Remarks:** setbuf must be called after fopen but before any other function calls that operate on the stream. If *buf* is a null pointer, setbuf calls the function setvbuf(stream, 0, \_IONBF, BUFSIZ) for no buffering; otherwise setbuf calls setvbuf(stream, buf, \_IOFBF, BUFSIZ) for full buffering with a buffer of size BUFSIZ. See setvbuf. This function requires a heap.

**Example:**

```
#include <stdio.h> /* for setbuf, printf, */  
                  /* fopen, fclose, */  
                  /* FILE, NULL, BUFSIZ */  
  
int main(void)  
{  
    FILE *myfile1, *myfile2;  
    char buf[BUFSIZ];  
  
    if ((myfile1 = fopen("afile1", "w+")) != NULL)  
    {  
        setbuf(myfile1, NULL);  
        printf("myfile1 has no buffering\n");  
        fclose(myfile1);  
    }  
  
    if ((myfile2 = fopen("afile2", "w+")) != NULL)  
    {  
        setbuf(myfile2, buf);  
        printf("myfile2 has full buffering");  
        fclose(myfile2);  
    }  
}
```

**Output:**

```
myfile1 has no buffering  
myfile2 has full buffering
```

# 16-Bit Language Tools Libraries

---

---

## setvbuf

---

**Description:** Defines the stream to be buffered and the buffer size.

**Include:** <stdio.h>

**Prototype:** `int setvbuf(FILE *stream, char *buf, int mode, size_t size);`

**Arguments:**

<i>stream</i>	pointer to the open stream
<i>buf</i>	user allocated buffer
<i>mode</i>	type of buffering
<i>size</i>	size of buffer

**Return Value:** Returns 0 if successful

**Remarks:** setvbuf must be called after fopen but before any other function calls that operate on the stream. For mode, use one of the following:  
\_IOFBF – for full buffering  
\_IOLBF – for line buffering  
\_IONBF – for no buffering  
This function requires a heap.

**Example:**

```
#include <stdio.h> /* for setvbuf, fopen, */
                    /* printf, FILE, NULL, */
                    /* _IONBF, _IOFBF      */

int main(void)
{
    FILE *myfile1, *myfile2;
    char buf[256];

    if ((myfile1 = fopen("afile1", "w+")) != NULL)
    {
        if (setvbuf(myfile1, NULL, _IONBF, 0) == 0)
            printf("myfile1 has no buffering\n");
        else
            printf("Unable to define buffer stream "
                "and/or size\n");
    }
    fclose(myfile1);

    if ((myfile2 = fopen("afile2", "w+")) != NULL)
    {
        if (setvbuf(myfile2, buf, _IOFBF, sizeof(buf)) ==
            0)
            printf("myfile2 has a buffer of %d "
                "characters\n", sizeof(buf));
        else
            printf("Unable to define buffer stream "
                "and/or size\n");
    }
    fclose(myfile2);
}
```

**Output:**

```
myfile1 has no buffering
myfile2 has a buffer of 256 characters
```



---

## sprintf

---

**Description:** Prints formatted text to a string.

**Include:** <stdio.h>

**Prototype:** `int sprintf(char *s, const char *format, ...);`

**Arguments:**

<i>s</i>	storage string for output
<i>format</i>	format control string
...	optional arguments

**Return Value:** Returns the number of characters stored in *s* excluding the terminating null character.

**Remarks:** The format argument has the same syntax and use that it has in `printf`.

**Example:** `#include <stdio.h> /* for sprintf, printf */`

```
int main(void)
{
    char sbuf[100], s[]="Print this string";
    int x = 1, y;
    char a = '\n';

    y = sprintf(sbuf, "%s %d time%c", s, x, a);

    printf("Number of characters printed to "
          "string buffer = %d\n", y);
    printf("String = %s\n", sbuf);
}
```

**Output:**

Number of characters printed to string buffer = 25  
String = Print this string 1 time

---

## sscanf

---

**Description:** Scans formatted text from a string.

**Include:** <stdio.h>

**Prototype:** `int sscanf(const char *s, const char *format, ...);`

**Arguments:**

<i>s</i>	storage string for input
<i>format</i>	format control string
...	optional arguments

**Return Value:** Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if an input error is encountered before the first conversion.

**Remarks:** The format argument has the same syntax and use that it has in `scanf`.

# 16-Bit Language Tools Libraries

---

---

---

## sscanf (Continued)

---

**Example:**

```
#include <stdio.h> /* for sscanf, printf */

int main(void)
{
    char s[] = "5 T green 3000000.00";
    int number, items;
    char letter;
    char color[10];
    float salary;

    items = sscanf(s, "%d %c %s %f", &number, &letter,
                  &color, &salary);

    printf("Number of items scanned = %d\n", items);
    printf("Favorite number = %d\n", number);
    printf("Favorite letter = %c\n", letter);
    printf("Favorite color = %s\n", color);
    printf("Desired salary = $%.2f\n", salary);
}
```

**Output:**

```
Number of items scanned = 4
Favorite number = 5
Favorite letter = T
Favorite color = green
Desired salary = $3000000.00
```

---

## tmpfileprintf

---

**Description:** Creates a temporary file.

**Include:** <stdio.h>

**Prototype:** FILE \*tmpfile(void)

**Return Value:** Returns a stream pointer if successful; otherwise, returns a NULL pointer.

**Remarks:** tmpfile creates a file with a unique filename. The temporary file is opened in w+b (binary read/write) mode. It will automatically be removed when exit is called; otherwise the file will remain in the directory. This function requires a heap.

**Example:**

```
#include <stdio.h> /* for tmpfile, printf, */
                          /* FILE, NULL */

int main(void)
{
    FILE *mytmpfile;

    if ((mytmpfile = tmpfile()) == NULL)
        printf("Cannot create temporary file");
    else
        printf("Temporary file was created");
}
```

**Output:**

```
Temporary file was created
```

---

## tmpnam

---

**Description:** Creates a unique temporary filename.

**Include:** `<stdio.h>`

**Prototype:** `char *tmpnam(char *s);`

**Argument:** `s` pointer to the temporary name

**Return Value:** Returns a pointer to the filename generated and stores the filename in `s`. If it can not generate a filename, the NULL pointer is returned.

**Remarks:** The created filename will not conflict with an existing file name. Use `L_tmpnam` to define the size of array the argument of `tmpnam` points to.

**Example:**

```
#include <stdio.h> /* for tmpnam, L_tmpnam, */
                  /* printf, NULL */
```

```
int main(void)
{
    char *myfilename;
    char mybuf[L_tmpnam];
    char *myptr = (char *) &mybuf;

    if ((myfilename = tmpnam(myptr)) == NULL)
        printf("Cannot create temporary file name");
    else
        printf("Temporary file %s was created",
              myfilename);
}
```

**Output:**

Temporary file ctm00001.tmp was created

---

---

## ungetc

---

**Description:** Pushes character back onto stream.

**Include:** `<stdio.h>`

**Prototype:** `int ungetc(int c, FILE *stream);`

**Argument:** `c` character to be pushed back  
`stream` pointer to the open stream

**Return Value:** Returns the pushed character if successful; otherwise, returns EOF.

**Remarks:** The pushed back character will be returned by a subsequent read on the stream. If more than one character is pushed back, they will be returned in the reverse order of their pushing. A successful call to a file positioning function (`fseek`, `fsetpos` or `rewind`) cancels any pushed back characters. Only one character of push back is guaranteed. Multiple calls to `ungetc` without an intervening read or file positioning operation may cause a failure.

---

# 16-Bit Language Tools Libraries

---

---

## ungetc (Continued)

---

**Example:**

```
#include <stdio.h> /* for ungetc, fgetc, */
                  /* printf, fopen, fclose, */
                  /* FILE, NULL, EOF */

int main(void)
{
    FILE *buf;
    char y, c;

    if ((buf = fopen("afile.txt", "r")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        y = fgetc(buf);
        while (y != EOF)
        {
            if (y == 'r')
            {
                c = ungetc(y, buf);
                if (c != EOF)
                {
                    printf("2");
                    y = fgetc(buf);
                }
            }
            printf("%c", y);
            y = fgetc(buf);
        }
        fclose(buf);
    }
}
```

**Input:**

Contents of afile.txt (used as input):

Short

Longer string

**Output:**

Sho2rt

Longe2r st2ring

## vasprintf

---

<b>Description:</b>	Prints formatted text to a string using a variable length argument list.
<b>Include:</b>	<stdio.h> <stdarg.h>
<b>Prototype:</b>	<code>int vasprintf(char **s, const char *format, va_list ap);</code>
<b>Arguments:</b>	<i>s</i> storage string for output <i>format</i> format control string <i>ap</i> pointer to a list of arguments
<b>Return Value:</b>	Returns number of characters stored in <i>s</i> excluding the terminating null character.
<b>Remarks:</b>	The format argument has the same syntax and use that it has in <code>printf</code> . To access the variable length argument list, the <i>ap</i> variable must be initialized by the macro <code>va_start</code> and may be reinitialized by additional calls to <code>va_arg</code> . This must be done before the <code>vasprintf</code> function is called. Invoke <code>va_end</code> after the function returns. For more details, see <code>stdarg.h</code> . This function requires a heap.
<b>Example:</b>	<pre>#include &lt;stdio.h&gt; /* for vasprintf, printf */ #include &lt;stdarg.h&gt; /* for va_start, va_end */  void errmsg(const char *fmt, ...) {     va_list ap;     char buf[100];     char **bufptr;     bufptr = &amp;buf;      va_start(ap, fmt);     vasprintf(bufptr, fmt, ap);     va_end(ap);     printf("Error: %s", buf); }  int main(void) {     int num = 3;      errmsg("The letter '%c' is not %s\n", 'a',           "an integer value.");     errmsg("Requires %d%s\n", num,           " or more characters.\n"); }</pre>

### Output:

```
Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.
```

# 16-Bit Language Tools Libraries

---

---

---

## fprintf

---

**Description:** Prints formatted data to a stream using a variable length argument list.

**Include:** <stdio.h>  
<stdarg.h>

**Prototype:** int fprintf(FILE \*stream, const char \*format, va\_list ap);

**Arguments:** *stream* pointer to the open stream  
*format* format control string  
*ap* pointer to a list of arguments

**Return Value:** Returns number of characters generated or a negative number if an error occurs.

**Remarks:** The format argument has the same syntax and use that it has in printf.  
To access the variable length argument list, the *ap* variable must be initialized by the macro *va\_start* and may be reinitialized by additional calls to *va\_arg*. This must be done before the *fprintf* function is called. Invoke *va\_end* after the function returns. For more details, see *stdarg.h*.  
This function requires a heap.

**Example:**

```
#include <stdio.h> /* for fprintf, fopen, */
                  /* fclose, printf, */
                  /* FILE, NULL */

#include <stdarg.h> /* for va_start, */
                  /* va_list, va_end */

FILE *myfile;

void errmsg(const char *fmt, ...)
{
    va_list ap;

    va_start(ap, fmt);
    fprintf(myfile, fmt, ap);
    va_end(ap);
}

int main(void)
{
    int num = 3;

    if ((myfile = fopen("afile.txt", "w")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        errmsg("Error: The letter '%c' is not %s\n", 'a',
              "an integer value.");
        errmsg("Error: Requires %d%s%c", num,
              " or more characters.", '\n');
    }
    fclose(myfile);
}
```

**Output:**

Contents of afile.txt

Error: The letter 'a' is not an integer value.  
Error: Requires 3 or more characters.

## **vprintf**

---

**Description:** Prints formatted text to `stdout` using a variable length argument list.

**Include:** `<stdio.h>`  
`<stdarg.h>`

**Prototype:** `int vprintf(const char *format, va_list ap);`

**Arguments:** `format` format control string  
`ap` pointer to a list of arguments

**Return Value:** Returns number of characters generated or a negative number if an error occurs.

**Remarks:** The format argument has the same syntax and use that it has in `printf`.  
To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vprintf` function is called. Invoke `va_end` after the function returns. For more details, see `stdarg.h`

**Example:**

```
#include <stdio.h> /* for vprintf, printf */
#include <stdarg.h> /* for va_start,      */
                  /* va_list, va_end    */
```

```
void errmsg(const char *fmt, ...)
{
    va_list ap;

    va_start(ap, fmt);
    printf("Error: ");
    vprintf(fmt, ap);
    va_end(ap);
}

int main(void)
{
    int num = 3;

    errmsg("The letter '%c' is not %s\n", 'a',
           "an integer value.");
    errmsg("Requires %d%s\n", num,
           " or more characters.\n");
}
```

**Output:**

```
Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.
```

# 16-Bit Language Tools Libraries

---

---

---

## vsprintf

---

<b>Description:</b>	Prints formatted text to a string using a variable length argument list.
<b>Include:</b>	<stdio.h> <stdarg.h>
<b>Prototype:</b>	int vsprintf(char * <i>s</i> , const char * <i>format</i> , va_list <i>ap</i> );
<b>Arguments:</b>	<i>s</i> storage string for output <i>format</i> format control string <i>ap</i> pointer to a list of arguments
<b>Return Value:</b>	Returns number of characters stored in <i>s</i> excluding the terminating null character.
<b>Remarks:</b>	The format argument has the same syntax and use that it has in <code>printf</code> . To access the variable length argument list, the <i>ap</i> variable must be initialized by the macro <code>va_start</code> and may be reinitialized by additional calls to <code>va_arg</code> . This must be done before the <code>vsprintf</code> function is called. Invoke <code>va_end</code> after the function returns. For more details, see <code>stdarg.h</code> This function requires a heap.
<b>Example:</b>	<pre>#include &lt;stdio.h&gt;    /* for vsprintf, printf */ #include &lt;stdarg.h&gt;  /* for va_start,        */                      /* va_list, va_end        */</pre>

```
void errormsg(const char *fmt, ...)
{
    va_list ap;
    char buf[100];

    va_start(ap, fmt);
    vsprintf(buf, fmt, ap);
    va_end(ap);
    printf("Error: %s", buf);
}

int main(void)
{
    int num = 3;

    errormsg("The letter '%c' is not %s\n", 'a',
             "an integer value.");
    errormsg("Requires %d%s\n", num,
             " or more characters.\n");
}
```

### Output:

```
Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.
```



## 2.14 <STDLIB.H> UTILITY FUNCTIONS

The header file `stdlib.h` consists of types, macros and functions that provide text conversions, memory management, searching and sorting abilities and other general utilities.

---

### **div\_t**

---

**Description:** A type that holds a quotient and remainder of a signed integer division with operands of type `int`.

**Include:** `<stdlib.h>`

**Prototype:** `typedef struct { int quot, rem; } div_t;`

**Remarks:** This is the structure type returned by the function, `div`.

---

### **ldiv\_t**

---

**Description:** A type that holds a quotient and remainder of a signed integer division with operands of type `long`.

**Include:** `<stdlib.h>`

**Prototype:** `typedef struct { long quot, rem; } ldiv_t;`

**Remarks:** This is the structure type returned by the function, `ldiv`.

---

### **size\_t**

---

**Description:** The type of the result of the `sizeof` operator.

**Include:** `<stdlib.h>`

---

### **wchar\_t**

---

**Description:** A type that holds a wide character value.

**Include:** `<stdlib.h>`

---

### **EXIT\_FAILURE**

---

**Description:** Reports unsuccessful termination.

**Include:** `<stdlib.h>`

**Remarks:** `EXIT_FAILURE` is a value for the `exit` function to return an unsuccessful termination status.

**Example:** See `exit` for example of use.

---

### **EXIT\_SUCCESS**

---

**Description:** Reports successful termination.

**Include:** `<stdlib.h>`

**Remarks:** `EXIT_SUCCESS` is a value for the `exit` function to return a successful termination status.

**Example:** See `exit` for example of use.

---

# 16-Bit Language Tools Libraries

---

---

---

## MB\_CUR\_MAX

---

**Description:** Maximum number of characters in a multibyte character.  
**Include:** <stdlib.h>  
**Value:** 1

---

---

## NULL

---

**Description:** The value of a null pointer constant.  
**Include:** <stdlib.h>

---

---

## RAND\_MAX

---

**Description:** Maximum value capable of being returned by the `rand` function.  
**Include:** <stdlib.h>  
**Value:** 32767

---

---

## abort

---

**Description:** Aborts the current process.  
**Include:** <stdlib.h>  
**Prototype:** `void abort(void);`  
**Remarks:** `abort` will cause the processor to reset.  
**Example:**

```
#include <stdio.h> /* for fopen, fclose, */
                  /* printf, FILE, NULL */
#include <stdlib.h> /* for abort          */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samp.fil", "r")) == NULL)
    {
        printf("Cannot open samp.fil\n");
        abort();
    }
    else
        printf("Success opening samp.fil\n");

    fclose(myfile);
}
```

**Output:**  
Cannot open samp.fil  
ABRT

---

---

## abs

---

**Description:** Calculates the absolute value.  
**Include:** <stdlib.h>  
**Prototype:** int abs(int i);  
**Argument:** *i* integer value  
**Return Value:** Returns the absolute value of *i*.  
**Remarks:** A negative number is returned as positive; a positive number is unchanged.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for abs */

int main(void)
{
    int i;

    i = 12;
    printf("The absolute value of %d is %d\n",
        i, abs(i));

    i = -2;
    printf("The absolute value of %d is %d\n",
        i, abs(i));

    i = 0;
    printf("The absolute value of %d is %d\n",
        i, abs(i));
}
```

**Output:**

```
The absolute value of 12 is 12
The absolute value of -2 is 2
The absolute value of 0 is 0
```

---

## atexit

---

**Description:** Registers the specified function to be called when the program terminates normally.  
**Include:** <stdlib.h>  
**Prototype:** int atexit(void(\*func)(void));  
**Argument:** *func* function to be called  
**Return Value:** Returns a zero if successful; otherwise, returns a non-zero value.  
**Remarks:** For the registered functions to be called, the program must terminate with the `exit` function call.

**Example:**

```
#include <stdio.h> /* for scanf, printf */
#include <stdlib.h> /* for atexit, exit */

void good_msg(void);
void bad_msg(void);
void end_msg(void);
```

# 16-Bit Language Tools Libraries

---

---

## atexit (Continued)

---

```
int main(void)
{
    int number;

    atexit(end_msg);
    printf("Enter your favorite number:");
    scanf("%d", &number);
    printf(" %d\n", number);
    if (number == 5)
    {
        printf("Good Choice\n");
        atexit(good_msg);
        exit(0);
    }
    else
    {
        printf("%d!?\n", number);
        atexit(bad_msg);
        exit(0);
    }
}

void good_msg(void)
{
    printf("That's an excellent number\n");
}

void bad_msg(void)
{
    printf("That's an awful number\n");
}

void end_msg(void)
{
    printf("Now go count something\n");
}
```

### Input:

With contents of UartIn.txt (used as stdin input for simulator):

5

### Output:

Enter your favorite number: 5

Good Choice

That's an excellent number

Now go count something

### Input:

With contents of UartIn.txt (used as stdin input for simulator):

42

### Output:

Enter your favorite number: 42

42!?

That's an awful number

Now go count something

## atof

---

**Description:** Converts a string to a double precision floating-point value.

**Include:** <stdlib.h>

**Prototype:** double atof(const char \*s);

**Argument:** s pointer to the string to be converted

**Return Value:** Returns the converted value if successful; otherwise, returns 0.

**Remarks:** The number may consist of the following:

```
[whitespace] [sign] digits [.digits]
[ { e | E } [sign] digits]
```

optional `whitespace` followed by an optional `sign`, then a sequence of one or more `digits` with an optional decimal point, followed by one or more optional `digits` and an optional `e` or `E` followed by an optional signed exponent. The conversion stops when the first unrecognized character is reached. The conversion is the same as `strtod(s,0)` except it does no error checking so `errno` will not be set.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for atof */

int main(void)
{
    char a[] = " 1.28";
    char b[] = "27.835e2";
    char c[] = "Number1";
    double x;

    x = atof(a);
    printf("String = \"%s\" float = %f\n", a, x);

    x = atof(b);
    printf("String = \"%s\" float = %f\n", b, x);

    x = atof(c);
    printf("String = \"%s\" float = %f\n", c, x);
}
```

**Output:**

```
String = "1.28" float = 1.280000
String = "27.835:e2" float = 2783.500000
String = "Number1" float = 0.000000
```

# 16-Bit Language Tools Libraries

---

---

## atoi

---

**Description:** Converts a string to an integer.

**Include:** <stdlib.h>

**Prototype:** int atoi(const char \*s);

**Argument:** s string to be converted

**Return Value:** Returns the converted integer if successful; otherwise, returns 0.

**Remarks:** The number may consist of the following:  
[whitespace] [sign] digits  
optional whitespace followed by an optional sign, then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to (int) strtol(s,0,10), except it does no error checking so errno will not be set.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for atoi */

int main(void)
{
    char a[] = " -127";
    char b[] = "Number1";
    int x;

    x = atoi(a);
    printf("String = \"%s\" \tint = %d\n", a, x);

    x = atoi(b);
    printf("String = \"%s\" \tint = %d\n", b, x);
}
```

**Output:**

```
String = " -127"          int = -127
String = "Number1"       int = 0
```

---

## atol

---

**Description:** Converts a string to a long integer.

**Include:** <stdlib.h>

**Prototype:** long atol(const char \*s);

**Argument:** s string to be converted

**Return Value:** Returns the converted long integer if successful; otherwise, returns 0.

**Remarks:** The number may consist of the following:  
[whitespace] [sign] digits  
optional whitespace followed by an optional sign, then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to (int) strtol(s,0,10), except it does no error checking so errno will not be set.

---

## atol (Continued)

---

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for atol */

int main(void)
{
    char a[] = " -123456";
    char b[] = "2Number";
    long x;

    x = atol(a);
    printf("String = \"%s\"   int = %ld\n", a, x);

    x = atol(b);
    printf("String = \"%s\"   int = %ld\n", b, x);
}

Output:
String = " -123456"       int = -123456
String = "2Number"       int = 2
```

---

## bsearch

---

**Description:** Performs a binary search.

**Include:** <stdlib.h>

**Prototype:**

```
void *bsearch(const void *key, const void *base,
              size_t nelem, size_t size,
              int (*cmp)(const void *ck, const void *ce));
```

**Arguments:**

<i>key</i>	object to search for
<i>base</i>	pointer to the start of the search data
<i>nelem</i>	number of elements
<i>size</i>	size of elements
<i>cmp</i>	pointer to the comparison function
<i>ck</i>	pointer to the key for the search
<i>ce</i>	pointer to the element being compared with the key.

**Return Value:** Returns a pointer to the object being searched for if found; otherwise, returns NULL.

**Remarks:** The value returned by the compare function is <0 if *ck* is less than *ce*, 0 if *ck* is equal to *ce* or >0 if *ck* is greater than *ce*. In the following example, `qsort` is used to sort the list before `bsearch` is called. `bsearch` requires the list to be sorted according to the comparison function. This `comp` uses ascending order.

# 16-Bit Language Tools Libraries

---

---

## bsearch (Continued)

---

```
Example:      #include <stdlib.h> /* for bsearch, qsort */
                #include <stdio.h> /* for printf, sizeof */

                #define NUM 7

                int comp(const void *e1, const void *e2);

                int main(void)
                {
                    int list[NUM] = {35, 47, 63, 25, 93, 16, 52};
                    int x, y;
                    int *r;

                    qsort(list, NUM, sizeof(int), comp);

                    printf("Sorted List:  ");
                    for (x = 0; x < NUM; x++)
                        printf("%d ", list[x]);

                    y = 25;
                    r = bsearch(&y, list, NUM, sizeof(int), comp);
                    if (r)
                        printf("\nThe value %d was found\n", y);
                    else
                        printf("\nThe value %d was not found\n", y);

                    y = 75;
                    r = bsearch(&y, list, NUM, sizeof(int), comp);
                    if (r)
                        printf("\nThe value %d was found\n", y);
                    else
                        printf("\nThe value %d was not found\n", y);
                }

                int comp(const void *e1, const void *e2)
                {

                    const int * a1 = e1;
                    const int * a2 = e2;

                    if (*a1 < *a2)
                        return -1;
                    else if (*a1 == *a2)
                        return 0;
                    else
                        return 1;
                }
```

### Output:

```
Sorted List:  16 25 35 47 52 63 93
The value 25 was found

The value 75 was not found
```



## calloc

---

**Description:** Allocates an array in memory and initializes the elements to 0.

**Include:** <stdlib.h>

**Prototype:** void \*calloc(size\_t *nelem*, size\_t *size*);

**Arguments:** *nelem* number of elements  
*size* length of each element

**Return Value:** Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

**Remarks:** Memory returned by `calloc` is aligned correctly for any size data element and is initialized to zero. This function requires a heap.

**Example:**

```
/* This program allocates memory for the      */
/* array 'i' of long integers and initializes */
/* them to zero.                             */
```

```
#include <stdio.h> /* for printf, NULL */
#include <stdlib.h> /* for calloc, free */
```

```
int main(void)
{
    int x;
    long *i;

    i = (long *)calloc(5, sizeof(long));
    if (i != NULL)
    {
        for (x = 0; x < 5; x++)
            printf("i[%d] = %ld\n", x, i[x]);
        free(i);
    }
    else
        printf("Cannot allocate memory\n");
}
```

**Output:**

```
i[0] = 0
i[1] = 0
i[2] = 0
i[3] = 0
i[4] = 0
```

---

## div

---

**Description:** Calculates the quotient and remainder of two numbers.

**Include:** <stdlib.h>

**Prototype:** div\_t div(int *numer*, int *denom*);

**Arguments:** *numer* numerator  
*denom* denominator

**Return Value:** Returns the quotient and the remainder.

**Remarks:** The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator (quot \* denom + rem = numer). Division by zero will invoke the math exception error, which, by default, will cause a Reset. Write a math error handler to do something else.

# 16-Bit Language Tools Libraries

---

---

## div (Continued)

---

**Example:**

```
#include <stdlib.h> /* for div, div_t */
#include <stdio.h> /* for printf */

void __attribute__((__interrupt__))
_MathError(void)
{
    printf("Illegal instruction executed\n");
    abort();
}

int main(void)
{
    int x, y;
    div_t z;

    x = 7;
    y = 3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\n", z.quot, z.rem);

    x = 7;
    y = -3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\n", z.quot, z.rem);

    x = -5;
    y = 3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\n", z.quot, z.rem);

    x = 7;
    y = 7;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\n", z.quot, z.rem);

    x = 7;
    y = 0;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\n", z.quot, z.rem);
}
```

## div (Continued)

---

**Output:**

```
For div(7, 3)
The quotient is 2 and the remainder is 1

For div(7, -3)
The quotient is -2 and the remainder is 1

For div(-5, 3)
The quotient is -1 and the remainder is -2

For div(7, 7)
The quotient is 1 and the remainder is 0

For div(7, 0)
Illegal instruction executed
ABRT
```

---

## exit

---

**Description:** Terminates program after clean up.

**Include:** <stdlib.h>

**Prototype:** void exit(int *status*);

**Argument:** *status* exit status

**Remarks:** exit calls any functions registered by atexit in reverse order of registration, flushes buffers, closes stream, closes any temporary files created with tmpfile and resets the processor. This function is customizable. See pic30-libs.

**Example:**

```
#include <stdio.h> /* for fopen, printf, */
                    /* FILE, NULL */
#include <stdlib.h> /* for exit */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samp.fil", "r" )) == NULL)
    {
        printf("Cannot open samp.fil\n");
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("Success opening samp.fil\n");
        exit(EXIT_SUCCESS);
    }
    printf("This will not be printed");
}
```

**Output:**

```
Cannot open samp.fil
```

# 16-Bit Language Tools Libraries

---

---

## free

---

**Description:** Frees memory.

**Include:** <stdlib.h>

**Prototype:** void free(void \*ptr);

**Argument:** ptr points to memory to be freed

**Remarks:** Frees memory previously allocated with calloc, malloc or realloc. If free is used on space that has already been deallocated (by a previous call to free or by realloc) or on space not allocated with calloc, malloc or realloc, the behavior is undefined. This function requires a heap.

**Example:**

```
#include <stdio.h> /* for printf, sizeof, */
/* NULL */
#include <stdlib.h> /* for malloc, free */

int main(void)
{
    long *i;

    if ((i = (long *)malloc(50 * sizeof(long))) ==
        NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        free(i);
        printf("Memory freed\n");
    }
}
```

**Output:**

Memory allocated  
Memory freed

---

## getenv

---

**Description:** Get a value for an environment variable.

**Include:** <stdlib.h>

**Prototype:** char \*getenv(const char \*name);

**Argument:** name name of environment variable

**Return Value:** Returns a pointer to the value of the environment variable if successful; otherwise, returns a null pointer.

**Remarks:** This function must be customized to be used as described (see pic30-libs). By default, there are no entries in the environment list for getenv to find.

---

## getenv (Continued)

---

**Example:**

```
#include <stdio.h> /* for printf, NULL */
#include <stdlib.h> /* for getenv */

int main(void)
{
    char *incvar;

    incvar = getenv("INCLUDE");
    if (incvar != NULL)
        printf("INCLUDE environment variable = %s\n",
            incvar);
    else
        printf("Cannot find environment variable "
            "INCLUDE ");
}
```

**Output:**

Cannot find environment variable INCLUDE

---

## labs

---

**Description:** Calculates the absolute value of a long integer.  
**Include:** <stdlib.h>  
**Prototype:** long labs(long i);  
**Argument:** i long integer value  
**Return Value:** Returns the absolute value of i.  
**Remarks:** A negative number is returned as positive; a positive number is unchanged.  
**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for labs */

int main(void)
{
    long i;

    i = 123456;
    printf("The absolute value of %7ld is %6ld\n",
        i, labs(i));

    i = -246834;
    printf("The absolute value of %7ld is %6ld\n",
        i, labs(i));

    i = 0;
    printf("The absolute value of %7ld is %6ld\n",
        i, labs(i));
}
```

**Output:**

The absolute value of 123456 is 123456  
The absolute value of -246834 is 246834  
The absolute value of 0 is 0

---

# 16-Bit Language Tools Libraries

---

---

## ldiv

---

**Description:** Calculates the quotient and remainder of two long integers.

**Include:** <stdlib.h>

**Prototype:** `ldiv_t ldiv(long numer, long denom);`

**Arguments:** *numer* numerator  
*denom* denominator

**Return Value:** Returns the quotient and the remainder.

**Remarks:** The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator (quot \* denom + rem = numer). If the denominator is zero, the behavior is undefined.

**Example:**

```
#include <stdlib.h> /* for ldiv, ldiv_t */
#include <stdio.h> /* for printf */

int main(void)
{
    long x,y;
    ldiv_t z;

    x = 7;
    y = 3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = -3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = -5;
    y = 3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = 7;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = 0;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n",
           z.quot, z.rem);
}
```

---

## ldiv (Continued)

---

**Output:**

```
For ldiv(7, 3)
The quotient is 2 and the remainder is 1

For ldiv(7, -3)
The quotient is -2 and the remainder is 1

For ldiv(-5, 3)
The quotient is -1 and the remainder is -2

For ldiv(7, 7)
The quotient is 1 and the remainder is 0

For ldiv(7, 0)
The quotient is -1 and the remainder is 7
```

**Explanation:**

In the last example (`ldiv(7, 0)`) the denominator is zero, the behavior is undefined.

---

## malloc

---

**Description:** Allocates memory.  
The default implementation of `malloc` will require an additional 4 bytes of heap memory per allocation.  
The legacy library's `malloc` will use an additional 2 bytes of heap memory per allocation.

**Include:** `<stdlib.h>`

**Prototype:** `void *malloc(size_t size);`

**Argument:** `size` number of characters to allocate

**Return Value:** Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

**Remarks:** `malloc` does not initialize memory it returns.  
This function requires a heap.

**Example:**

```
#include <stdio.h> /* for printf, sizeof, */
                    /* NULL */
#include <stdlib.h> /* for malloc, free */

int main(void)
{
    long *i;

    if ((i = (long *)malloc(50 * sizeof(long))) ==
        NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        free(i);
        printf("Memory freed\n");
    }
}
```

**Output:**

```
Memory allocated
Memory freed
```

# 16-Bit Language Tools Libraries

---

---



---

## mblen

---

**Description:** Gets the length of a multibyte character. (See Remarks.)

**Include:** <stdlib.h>

**Prototype:** `int mblen(const char *s, size_t n);`

**Arguments:** *s* points to the multibyte character  
*n* number of bytes to check

**Return Value:** Returns zero if *s* points to a null character; otherwise, returns 1.

**Remarks:** The 16-bit compiler does not support multibyte characters with length greater than 1 byte.

---

---

## mbstowcs

---

**Description:** Converts a multibyte string to a wide character string. (See Remarks.)

**Include:** <stdlib.h>

**Prototype:** `size_t mbstowcs(wchar_t *wcs, const char *s, size_t n);`

**Arguments:** *wcs* points to the wide character string  
*s* points to the multibyte string  
*n* the number of wide characters to convert.

**Return Value:** Returns the number of wide characters stored excluding the null character.

**Remarks:** `mbstowcs` converts *n* number of wide characters unless it encounters a null wide character first. The 16-bit compiler does not support multibyte characters with length greater than 1 byte.

---

---

## mbtowc

---

**Description:** Converts a multibyte character to a wide character. (See Remarks.)

**Include:** <stdlib.h>

**Prototype:** `int mbtowc(wchar_t *pwc, const char *s, size_t n);`

**Arguments:** *pwc* points to the wide character  
*s* points to the multibyte character  
*n* number of bytes to check

**Return Value:** Returns zero if *s* points to a null character; otherwise, returns 1.

**Remarks:** The resulting wide character will be stored at *pwc*. The 16-bit compiler does not support multibyte characters with length greater than 1 byte.

---

# 16-Bit Language Tools Libraries

---

---

## qsort

---

**Description:** Performs a quick sort.

**Include:** <stdlib.h>

**Prototype:** void qsort(void \*base, size\_t nelem, size\_t size, int (\*cmp)(const void \*e1, const void \*e2));

**Arguments:**

- base* pointer to the start of the array
- nelem* number of elements
- size* size of the elements
- cmp* pointer to the comparison function
- e1* pointer to the key for the search
- e2* pointer to the element being compared with the key

**Remarks:** qsort overwrites the array with the sorted array. The comparison function is supplied by the user. In the following example, the list is sorted according to the comparison function. This `comp` uses ascending order. This function requires a heap.

**Example:**

```
#include <stdlib.h> /* for qsort */
#include <stdio.h> /* for printf */

#define NUM 7

int comp(const void *e1, const void *e2);

int main(void)
{
    int list[NUM] = {35, 47, 63, 25, 93, 16, 52};
    int x;

    printf("Unsorted List: ");
    for (x = 0; x < NUM; x++)
        printf("%d ", list[x]);

    qsort(list, NUM, sizeof(int), comp);

    printf("\n");
    printf("Sorted List: ");
    for (x = 0; x < NUM; x++)
        printf("%d ", list[x]);
}

int comp(const void *e1, const void *e2)
{
    const int * a1 = e1;
    const int * a2 = e2;

    if (*a1 < *a2)
        return -1;
    else if (*a1 == *a2)
        return 0;
    else
        return 1;
}
```

---

## qsort (Continued)

---

**Output:**

```
Unsorted List: 35 47 63 25 93 16 52
Sorted List:   16 25 35 47 52 63 93
```

---

## rand

---

**Description:** Generates a pseudo-random integer.

**Include:** <stdlib.h>

**Prototype:** int rand(void);

**Return Value:** Returns an integer between 0 and RAND\_MAX.

**Remarks:** Calls to this function return pseudo-random integer values in the range [0,RAND\_MAX]. To use this function effectively, you must seed the random number generator using the `srand` function. This function will always return the same sequence of integers when no seeds are used (as in the example below) or when identical seed values are used. (See `srand` for seed example.)

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for rand */

int main(void)
{
    int x;

    for (x = 0; x < 5; x++)
        printf("Number = %d\n", rand());
}
```

**Output:**

```
Number = 21422
Number = 2061
Number = 16443
Number = 11617
Number = 9125
```

Notice if the program is run a second time, the numbers are the same. See the example for `srand` to seed the random number generator.

---

## realloc

---

**Description:** Reallocates memory to allow a size change.

**Include:** <stdlib.h>

**Prototype:** void \*realloc(void \*ptr, size\_t size);

**Arguments:** *ptr* points to previously allocated memory  
*size* new size to allocate to

**Return Value:** Returns a pointer to the allocated space if successful; otherwise, returns a null pointer. This function requires a heap.

**Remarks:** If the existing object is smaller than the new object, the entire existing object is copied to the new object and the remainder of the new object is indeterminate. If the existing object is larger than the new object, the function copies as much of the existing object as will fit in the new object. If `realloc` succeeds in allocating a new object, the existing object will be deallocated; otherwise, the existing object is left unchanged. Keep a temporary pointer to the existing object since `realloc` will return a null pointer on failure.

---

# 16-Bit Language Tools Libraries

---

---

---

## realloc (Continued)

---

**Example:**

```
#include <stdio.h> /* for printf, sizeof, NULL */
#include <stdlib.h> /* for realloc, malloc, free */

int main(void)
{
    long *i, *j;

    if ((i = (long *)malloc(50 * sizeof(long)))
        == NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        /* Temp pointer in case realloc() fails */
        j = i;

        if ((i = (long *)realloc(i, 25 * sizeof(long)))
            == NULL)
        {
            printf("Cannot reallocate memory\n");
            /* j pointed to allocated memory */
            free(j);
        }
        else
        {
            printf("Memory reallocated\n");
            free(i);
        }
    }
}
```

**Output:**

```
Memory allocated
Memory reallocated
```

---

## srand

---

**Description:** Set the starting seed for the pseudo-random number sequence.

**Include:** <stdlib.h>

**Prototype:** void srand(unsigned int seed);

**Argument:** seed starting value for the pseudo-random number sequence

**Return Value:** None.

**Remarks:** This function sets the starting seed for the pseudo-random number sequence generated by the rand function. The rand function will always return the same sequence of integers when identical seed values are used. If rand is called with a seed value of 1, the sequence of numbers generated will be the same as if rand had been called without srand having been called first.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for rand, srand */

int main(void)
{
    int x;

    srand(7);
    for (x = 0; x < 5; x++)
        printf("Number = %d\n", rand());
}
```

**Output:**

```
Number = 16327
Number = 5931
Number = 23117
Number = 30985
Number = 29612
```

---

## strdup

---

**Description:** Duplicates a string.

**Include:** <stdlib.h>

**Prototype:** char \*strdup(const char \*s1);

**Arguments:** s1 pointer to the string

**Return Value:** A pointer to a new string on success. Otherwise, it shall return a null pointer.

**Remarks:** The strdup() function shall return a pointer to a new string, which is a duplicate of the string pointed to by s1. The returned pointer can be passed to free(). A null pointer is returned if the new string cannot be created.  
This function required a heap.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for strdup */

int main(void)
{
    char a[] = "MyString";
    char *b;
```

---

## strdup (Continued)

---

```
b = strdup(a);
printf("String a = \"%s\\n\" , a );
printf("String b = \"%s\\n\" , b );
```

### Output:

```
String a = "MyString"
String b = "MyString"
```

---

## strtod

---

**Description:** Converts a partial string to a floating-point number of type double.

**Include:** <stdlib.h>

**Prototype:** double strtod(const char \*s, char \*\*endptr);

**Arguments:** s string to be converted  
endptr pointer to the character at which the conversion stopped

**Return Value:** Returns the converted number if successful; otherwise, returns 0.

**Remarks:** The number may consist of the following:  
[whitespace] [sign] digits [.digits]  
[ { e | E } [sign] digits]  
optional whitespace followed by an optional sign, then a sequence of one or more digits with an optional decimal point, followed by one or more optional digits and an optional e or E followed by an optional signed exponent.  
strtod converts the string until it reaches a character that cannot be converted to a number. endptr will point to the remainder of the string starting with the first unconverted character.  
If a range error occurs, errno will be set.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for strtod */

int main(void)
{
    char *end;
    char a[] = "1.28 inches";
    char b[] = "27.835e2i";
    char c[] = "Number1";
    double x;

    x = strtod(a, &end);
    printf("String = \"%s\\n\" float = %f\\n", a, x );
    printf("Stopped at: %s\\n\\n", end );

    x = strtod(b, &end);
    printf("String = \"%s\\n\" float = %f\\n", b, x );
    printf("Stopped at: %s\\n\\n", end );

    x = strtod(c, &end);
    printf("String = \"%s\\n\" float = %f\\n", c, x );
    printf("Stopped at: %s\\n\\n", end );
}
```

---

## strtod (Continued)

---

**Output:**

```
String = "1.28 inches" float = 1.280000
Stopped at: inches
```

```
String = "27.835e2i" float = 2783.500000
Stopped at: i
```

```
String = "Number1" float = 0.000000
Stopped at: Number1
```

---

## strtol

---

**Description:** Converts a partial string to a long integer.

**Include:** <stdlib.h>

**Prototype:** long strtol(const char \*s, char \*\*endptr, int base);

**Arguments:**

<i>s</i>	string to be converted
<i>endptr</i>	pointer to the character at which the conversion stopped
<i>base</i>	number base to use in conversion

**Return Value:** Returns the converted number if successful; otherwise, returns 0.

**Remarks:** If *base* is zero, `strtol` attempts to determine the base automatically. It can be octal, determined by a leading zero; hexadecimal, determined by a leading 0x or 0X; or decimal in any other case. If *base* is specified, `strtol` converts a sequence of digits and letters a-z (case insensitive) where a-z represents the numbers 10-36. Conversion stops when an out of base number is encountered. *endptr* will point to the remainder of the string starting with the first unconverted character. If a range error occurs, `errno` will be set.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for strtol */

int main(void)
{
    char *end;
    char a[] = "-12BGEE";
    char b[] = "1234Number";
    long x;

    x = strtol(a, &end, 16);
    printf("String = \"%s\" long = %ld\n", a, x );
    printf("Stopped at: %s\n\n", end );

    x = strtol(b, &end, 4);
    printf("String = \"%s\" long = %ld\n", b, x );
    printf("Stopped at: %s\n\n", end );
}
```

**Output:**

```
String = "-12BGEE" long = -299
Stopped at: GEE
```

```
String = "1234Number" long = 27
Stopped at: 4Number
```

# 16-Bit Language Tools Libraries

---

---

## strtoul

---

**Description:** Converts a partial string to an unsigned long integer.

**Include:** <stdlib.h>

**Prototype:** unsigned long strtoul(const char \*s, char \*\*endptr, int base);

**Arguments:** *s* string to be converted  
*endptr* pointer to the character at which the conversion stopped  
*base* number base to use in conversion

**Return Value:** Returns the converted number if successful; otherwise, returns 0.

**Remarks:** If *base* is zero, `strtoul` attempts to determine the base automatically. It can be octal, determined by a leading zero, hexadecimal, determined by a leading 0x or 0X, or decimal in any other case. If base is specified `strtoul` converts a sequence of digits and letters a-z (case insensitive), where a-z represents the numbers 10-36. Conversion stops when an out of base number is encountered. *endptr* will point to the remainder of the string starting with the first unconverted character. If a range error occurs, `errno` will be set.

**Example:**

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for strtoul */

int main(void)
{
    char *end;
    char a[] = "12BGET3";
    char b[] = "0x1234Number";
    char c[] = "-123abc";
    unsigned long x;

    x = strtoul(a, &end, 25);
    printf("String = \"%s\" long = %lu\n", a, x );
    printf("Stopped at: %s\n\n", end );

    x = strtoul(b, &end, 0);
    printf("String = \"%s\" long = %lu\n", b, x );
    printf("Stopped at: %s\n\n", end );

    x = strtoul(c, &end, 0);
    printf("String = \"%s\" long = %lu\n", c, x );
    printf("Stopped at: %s\n\n", end );
}
```

**Output:**

```
String = "12BGET3" long = 429164
Stopped at: T3

String = "0x1234Number" long = 4660
Stopped at: Number

String = "-123abc" long = 4294967173
Stopped at: abc
```



---

## system

---

**Description:** Execute a command.

**Include:** <stdlib.h>

**Prototype:** `int system(const char *s);`

**Argument:** *s* command to be executed

**Remarks:** This function must be customized to be used as described (see `pic30-libs`). By default, `system` will cause a Reset if called with anything other than NULL. `system(NULL)` will do nothing.

**Example:**

```
/* This program uses system */
/* to TYPE its source file. */

#include <stdlib.h> /* for system */

int main(void)
{
    system("type sampsystem.c");
}
```

**Output:**

System(type sampsystem.c) called: Aborting

---

## wctomb

---

**Description:** Converts a wide character to a multibyte character. (See Remarks.)

**Include:** <stdlib.h>

**Prototype:** `int wctomb(char *s, wchar_t wchar);`

**Arguments:** *s* points to the multibyte character  
*wchar* the wide character to be converted

**Return Value:** Returns zero if *s* points to a null character; otherwise, returns 1.

**Remarks:** The resulting multibyte character is stored at *s*. The 16-bit compiler does not support multibyte characters with length greater than 1 character.

---

## wcstombs

---

**Description:** Converts a wide character string to a multibyte string. (See Remarks.)

**Include:** <stdlib.h>

**Prototype:** `size_t wcstombs(char *s, const wchar_t *wcs, size_t n);`

**Arguments:** *s* points to the multibyte string  
*wcs* points to the wide character string  
*n* the number of characters to convert

**Return Value:** Returns the number of characters stored excluding the null character.

**Remarks:** `wcstombs` converts *n* number of multibyte characters unless it encounters a null character first. The 16-bit compiler does not support multibyte characters with length greater than 1 character.

---

# 16-Bit Language Tools Libraries

---

## 2.15 <STRING.H> STRING FUNCTIONS

The header file, `string.h`, consists of types, macros and functions that provide tools to manipulate strings.

---

### size\_t

**Description:** The type of the result of the `sizeof` operator.  
**Include:** `<string.h>`

---

### NULL

**Description:** The value of a null pointer constant.  
**Include:** `<string.h>`

---

### memchr

**Description:** Locates a character in a buffer.  
**Include:** `<string.h>`  
**Prototype:** `void *memchr(const void *s, int c, size_t n);`  
**Arguments:** `s` pointer to the buffer  
`c` character to search for  
`n` number of characters to check  
**Return Value:** Returns a pointer to the location of the match if successful; otherwise, returns null.  
**Remarks:** `memchr` stops when it finds the first occurrence of `c`, or after searching `n` number of characters.

**Example:**

```
#include <string.h> /* for memchr, NULL */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'i', ch2 = 'y';
    char *ptr;
    int res;

    printf("buf1 : %s\n\n", buf1);

    ptr = memchr(buf1, ch1, 50);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
        printf("%c not found\n", ch1);
}
```

---

## memchr (Continued)

---

```
printf("\n");

ptr = memchr(buf1, ch2, 50);
if (ptr != NULL)
{
    res = ptr - buf1 + 1;
    printf("%c found at position %d\n", ch2, res);
}
else
    printf("%c not found\n", ch2);
}
```

**Output:**

buf1 : What time is it?

i found at position 7

y not found

---

## memcmp

---

**Description:** Compare the contents of two buffers.

**Include:** <string.h>

**Prototype:** int memcmp(const void \*s1, const void \*s2, size\_t n);

**Arguments:**

s1	first buffer
s2	second buffer
n	number of characters to compare

**Return Value:** Returns a positive number if s1 is greater than s2, zero if s1 is equal to s2 or a negative number if s1 is less than s2.

**Remarks:** This function compares the first n characters in s1 to the first n characters in s2 and returns a value indicating whether the buffers are less than, equal to or greater than each other.

**Example:**

```
#include <string.h> /* memcmp */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n\n", buf3);

    res = memcmp(buf1, buf2, 6);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("6 characters of buf1 and buf2 "
            "are equal\n");
    else
        printf("buf2 comes before buf1\n");
}
```

## memcmp (Continued)

---

```
printf("\n");

res = memcmp(buf1, buf2, 20);
if (res < 0)
    printf("buf1 comes before buf2\n");
else if (res == 0)
    printf("20 characters of buf1 and buf2 "
           "are equal\n");
else
    printf("buf2 comes before buf1\n");

printf("\n");

res = memcmp(buf1, buf3, 20);
if (res < 0)
    printf("buf1 comes before buf3\n");
else if (res == 0)
    printf("20 characters of buf1 and buf3 "
           "are equal\n");
else
    printf("buf3 comes before buf1\n");
}
```

**Output:**

```
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?
```

```
6 characters of buf1 and buf2 are equal
```

```
buf2 comes before buf1
```

```
buf1 comes before buf3
```

---

## memcpy

---

**Description:** Copies characters from one buffer to another.

**Include:** <string.h>

**Prototype:** void \*memcpy(void \*dst , const void \*src , size\_t n);

**Arguments:** *dst* buffer to copy characters to  
*src* buffer to copy characters from  
*n* number of characters to copy

**Return Value:** Returns *dst*.

**Remarks:** memcpy copies *n* characters from the source buffer *src* to the destination buffer *dst*. If the buffers overlap, the behavior is undefined.

**Example:** #include <string.h> /\* memcpy \*/  
#include <stdio.h> /\* for printf \*/

```
int main(void)
{
    char buf1[50] = "";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n\n", buf3);

    memcpy(buf1, buf2, 6);
    printf("buf1 after memcpy of 6 chars of "
           "buf2: \n\t%s\n", buf1);

    printf("\n");

    memcpy(buf1, buf3, 5);
    printf("buf1 after memcpy of 5 chars of "
           "buf3: \n\t%s\n", buf1);
}
```

**Output:**

```
buf1 :
buf2 : Where is the time?
buf3 : Why?
```

```
buf1 after memcpy of 6 chars of buf2:
    Where
```

```
buf1 after memcpy of 5 chars of buf3:
    Why?
```

---

## memmove

---

**Description:** Copies *n* characters of the source buffer into the destination buffer, even if the regions overlap.

**Include:** `<string.h>`

**Prototype:** `void *memmove(void *s1, const void *s2, size_t n);`

**Arguments:**

- s1*            buffer to copy characters to (destination)
- s2*            buffer to copy characters from (source)
- n*             number of characters to copy from *s2* to *s1*

**Return Value:** Returns a pointer to the destination buffer.

**Remarks:** If the buffers overlap, the effect is as if the characters are read first from *s2*, then written to *s1*, so the buffer is not corrupted.

**Example:**

```
#include <string.h> /* for memmove */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "When time marches on";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n\n", buf3);

    memmove(buf1, buf2, 6);
    printf("buf1 after memmove of 6 chars of "
           "buf2: \n\t%s\n", buf1);

    printf("\n");

    memmove(buf1, buf3, 5);
    printf("buf1 after memmove of 5 chars of "
           "buf3: \n\t%s\n", buf1);
}
```

**Output:**

```
buf1 : When time marches on
buf2 : Where is the time?
buf3 : Why?
```

```
buf1 after memmove of 6 chars of buf2:
    Where ime marches on
```

```
buf1 after memmove of 5 chars of buf3:
    Why?
```

## memset

---

**Description:** Copies the specified character into the destination buffer.

**Include:** <string.h>

**Prototype:** void \*memset(void \*s, int c, size\_t n);

**Arguments:**

<i>s</i>	buffer
<i>c</i>	character to put in buffer
<i>n</i>	number of times

**Return Value:** Returns the buffer with characters written to it.

**Remarks:** The character *c* is written to the buffer *n* times.

**Example:**

```
#include <string.h> /* for memset */
#include <stdio.h> /* for printf */
```

```
int main(void)
{
    char buf1[20] = "What time is it?";
    char buf2[20] = "";
    char ch1 = '?', ch2 = 'y';
    char *ptr;
    int res;

    printf("memset(\"%s\", '%c',4);\n", buf1, ch1);
    memset(buf1, ch1, 4);
    printf("buf1 after memset: %s\n", buf1);

    printf("\n");
    printf("memset(\"%s\", '%c',10);\n", buf2, ch2);
    memset(buf2, ch2, 10);
    printf("buf2 after memset: %s\n", buf2);
}
```

**Output:**

```
memset("What time is it?", '?',4);
buf1 after memset: ??? time is it?

memset("", 'y',10);
buf2 after memset: yyyyyyyyyy
```

# 16-Bit Language Tools Libraries

---

---

---

## strcat

---

**Description:** Appends a copy of the source string to the end of the destination string.

**Include:** <string.h>

**Prototype:** char \*strcat(char \*s1, const char \*s2);

**Arguments:** s1 null terminated destination string to copy to  
s2 null terminated source string to be copied

**Return Value:** Returns a pointer to the destination string.

**Remarks:** This function appends the source string (including the terminating null character) to the end of the destination string. The initial character of the source string overwrites the null character at the end of the destination string. If the buffers overlap, the behavior is undefined.

**Example:**

```
#include <string.h> /* for strcat, strlen */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";

    printf("buf1 : %s\n", buf1);
    printf("\t(%d characters)\n\n", strlen(buf1));
    printf("buf2 : %s\n", buf2);
    printf("\t(%d characters)\n\n", strlen(buf2));

    strcat(buf1, buf2);
    printf("buf1 after strcat of buf2: \n\t%s\n",
        buf1);
    printf("\t(%d characters)\n", strlen(buf1));

    printf("\n");

    strcat(buf1, "Why?");
    printf("buf1 after strcat of \"Why?\": \n\t%s\n",
        buf1);
    printf("\t(%d characters)\n", strlen(buf1));
}
```

**Output:**

```
buf1 : We're here
      (10 characters)

buf2 : Where is the time?
      (18 characters)

buf1 after strcat of buf2:
      We're hereWhere is the time?
      (28 characters)

buf1 after strcat of "Why?":
      We're hereWhere is the time?Why?
      (32 characters)
```



## strchr

---

<b>Description:</b>	Locates the first occurrence of a specified character in a string.
<b>Include:</b>	<string.h>
<b>Prototype:</b>	char *strchr(const char *s, int c);
<b>Arguments:</b>	<i>s</i> pointer to the string <i>c</i> character to search for
<b>Return Value:</b>	Returns a pointer to the location of the match if successful; otherwise, returns a null pointer.
<b>Remarks:</b>	This function searches the string <i>s</i> to find the first occurrence of the character, <i>c</i> .
<b>Example:</b>	<pre>#include &lt;string.h&gt; /* for strchr, NULL */ #include &lt;stdio.h&gt;  /* for printf      */  int main(void) {     char buf1[50] = "What time is it?";     char ch1 = 'm', ch2 = 'y';     char *ptr;     int res;      printf("buf1 : %s\n\n", buf1);      ptr = strchr(buf1, ch1);     if (ptr != NULL)     {         res = ptr - buf1 + 1;         printf("%c found at position %d\n", ch1, res);     }     else         printf("%c not found\n", ch1);      printf("\n");      ptr = strchr(buf1, ch2);     if (ptr != NULL)     {         res = ptr - buf1 + 1;         printf("%c found at position %d\n", ch2, res);     }     else         printf("%c not found\n", ch2); }</pre>

**Output:**

buf1 : What time is it?

m found at position 8

y not found

# 16-Bit Language Tools Libraries

---

---

---

## strcmp

---

**Description:** Compares two strings.

**Include:** <string.h>

**Prototype:** int strcmp(const char \*s1, const char \*s2);

**Arguments:** s1 first string  
s2 second string

**Return Value:** Returns a positive number if s1 is greater than s2, zero if s1 is equal to s2 or a negative number if s1 is less than s2.

**Remarks:** This function compares successive characters from s1 and s2 until they are not equal or the null terminator is reached.

**Example:**

```
#include <string.h> /* for strcmp */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n\n", buf3);

    res = strcmp(buf1, buf2);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("buf1 and buf2 are equal\n");
    else
        printf("buf2 comes before buf1\n");

    printf("\n");

    res = strcmp(buf1, buf3);
    if (res < 0)
        printf("buf1 comes before buf3\n");
    else if (res == 0)
        printf("buf1 and buf3 are equal\n");
    else
        printf("buf3 comes before buf1\n");

    printf("\n");

    res = strcmp("Why?", buf3);
    if (res < 0)
        printf("\\"Why?\" comes before buf3\n");
    else if (res == 0)
        printf("\\"Why?\" and buf3 are equal\n");
    else
        printf("buf3 comes before \"Why?\"\\n");
}
```

---

## strcmp (Continued)

---

**Output:**

```
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?
```

```
buf2 comes before buf1
```

```
buf1 comes before buf3
```

```
"Why?" and buf3 are equal
```

---

## strcoll

---

<b>Description:</b>	Compares one string to another. (See Remarks.)
<b>Include:</b>	<string.h>
<b>Prototype:</b>	int strcoll(const char *s1, const char *s2);
<b>Arguments:</b>	s1            first string s2            second string
<b>Return Value:</b>	Using the locale-dependent rules, it returns a positive number if s1 is greater than s2, zero if s1 is equal to s2 or a negative number if s1 is less than s2.
<b>Remarks:</b>	Since the 16-bit compiler does not support alternate locales, this function is equivalent to strcmp.

---

## strcpy

---

<b>Description:</b>	Copy the source string into the destination string.
<b>Include:</b>	<string.h>
<b>Prototype:</b>	char *strcpy(char *s1, const char *s2);
<b>Arguments:</b>	s1            destination string to copy to s2            source string to copy from
<b>Return Value:</b>	Returns a pointer to the destination string.
<b>Remarks:</b>	All characters of s2 are copied, including the null terminating character. If the strings overlap, the behavior is undefined.
<b>Example:</b>	<pre>#include &lt;string.h&gt; /* for strcpy, strlen */ #include &lt;stdio.h&gt;  /* for printf          */  int main(void) {     char buf1[50] = "We're here";     char buf2[50] = "Where is the time?";     char buf3[50] = "Why?";      printf("buf1 : %s\n", buf1);     printf("buf2 : %s\n", buf2);     printf("buf3 : %s\n\n", buf3);      strcpy(buf1, buf2);     printf("buf1 after strcpy of buf2: \n\t%s\n\n",           buf1);</pre>

---

## strcpy (Continued)

---

```
strcpy(buf1, buf3);
printf("buf1 after strcpy of buf3: \n\t%s\n",
      buf1);
}
```

**Output:**

```
buf1 : We're here
buf2 : Where is the time?
buf3 : Why?
```

```
buf1 after strcpy of buf2:
      Where is the time?
```

```
buf1 after strcpy of buf3:
      Why?
```

---

## strcspn

---

**Description:** Calculate the number of consecutive characters at the beginning of a string that are not contained in a set of characters.

**Include:** <string.h>

**Prototype:** size\_t strcspn(const char \*s1, const char \*s2);

**Arguments:** s1 pointer to the string to be searched  
s2 pointer to characters to search for

**Return Value:** Returns the length of the segment in s1 not containing characters found in s2.

**Remarks:** This function will determine the number of consecutive characters from the beginning of s1 that are not contained in s2.

**Example:** #include <string.h> /\* for strcspn \*/  
#include <stdio.h> /\* for printf \*/

```
int main(void)
{
    char str1[20] = "hello";
    char str2[20] = "aeiou";
    char str3[20] = "animal";
    char str4[20] = "xyz";
    int res;

    res = strcspn(str1, str2);
    printf("strcspn(\"%s\", \"%s\") = %d\n",
          str1, str2, res);

    res = strcspn(str3, str2);
    printf("strcspn(\"%s\", \"%s\") = %d\n",
          str3, str2, res);

    res = strcspn(str3, str4);
    printf("strcspn(\"%s\", \"%s\") = %d\n",
          str3, str4, res);
}
```

**Output:**

```
strcspn("hello", "aeiou") = 1
strcspn("animal", "aeiou") = 0
strcspn("animal", "xyz") = 6
```

---

## strcspn (Continued)

---

**Explanation:**

In the first result, *e* is in *s2* so it stops counting after *h*.

In the second result, *a* is in *s2*.

In the third result, none of the characters of *s1* are in *s2* so all characters are counted.

---

## strerror

---

**Description:** Gets an internal error message.

**Include:** <string.h>

**Prototype:** `char *strerror(int errcode);`

**Argument:** *errcode* number of the error code

**Return Value:** Returns a pointer to an internal error message string corresponding to the specified error code *errcode*.

**Remarks:** The array pointed to by `strerror` may be overwritten by a subsequent call to this function.

**Example:**

```
#include <stdio.h> /* for fopen, fclose, */
                    /* printf, FILE, NULL */
#include <string.h> /* for strerror */
#include <errno.h> /* for errno */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samp.fil", "r+")) == NULL)
        printf("Cannot open samp.fil: %s\n",
              strerror(errno));
    else
        printf("Success opening samp.fil\n");
    fclose(myfile);
}
```

**Output:**

Cannot open samp.fil: file open error

---

## strlen

---

**Description:** Finds the length of a string.

**Include:** <string.h>

**Prototype:** `size_t strlen(const char *s);`

**Argument:** *s* the string

**Return Value:** Returns the length of a string.

**Remarks:** This function determines the length of the string, not including the terminating null character.

---

## strlen (Continued)

---

**Example:**

```
#include <string.h> /* for strlen */
#include <stdio.h> /* for printf */

int main(void)
{
    char str1[20] = "We are here";
    char str2[20] = "";
    char str3[20] = "Why me?";

    printf("str1 : %s\n", str1);
    printf("\t(string length = %d characters)\n\n",
        strlen(str1));
    printf("str2 : %s\n", str2);
    printf("\t(string length = %d characters)\n\n",
        strlen(str2));
    printf("str3 : %s\n", str3);
    printf("\t(string length = %d characters)\n\n",
        strlen(str3));
}
```

**Output:**

```
str1 : We are here
      (string length = 11 characters)

str2 :
      (string length = 0 characters)

str3 : Why me?
      (string length = 7 characters)
```

---

## strncat

---

**Description:** Append a specified number of characters from the source string to the destination string.

**Include:** <string.h>

**Prototype:** char \*strncat(char \*s1, const char \*s2, size\_t n);

**Arguments:**

<i>s1</i>	destination string to copy to
<i>s2</i>	source string to copy from
<i>n</i>	number of characters to append

**Return Value:** Returns a pointer to the destination string.

**Remarks:** This function appends up to *n* characters (a null character and characters that follow it are not appended) from the source string to the end of the destination string. If a null character is not encountered, then a terminating null character is appended to the result. If the strings overlap, the behavior is undefined.

**Example:**

```
#include <string.h> /* for strncat, strlen */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";
```

## strncat (Continued)

---

```
printf("buf1 : %s\n", buf1);
printf("\t(%d characters)\n\n", strlen(buf1));
printf("buf2 : %s\n", buf2);
printf("\t(%d characters)\n\n", strlen(buf2));
printf("buf3 : %s\n", buf3);
printf("\t(%d characters)\n\n\n", strlen(buf3));

strncat(buf1, buf2, 6);
printf("buf1 after strncat of 6 characters "
      "of buf2: \n\t%s\n", buf1);
printf("\t(%d characters)\n", strlen(buf1));

printf("\n");

strncat(buf1, buf2, 25);
printf("buf1 after strncat of 25 characters "
      "of buf2: \n\t%s\n", buf1);
printf("\t(%d characters)\n", strlen(buf1));

printf("\n");

strncat(buf1, buf3, 4);
printf("buf1 after strncat of 4 characters "
      "of buf3: \n\t%s\n", buf1);
printf("\t(%d characters)\n", strlen(buf1));
}
```

### Output:

buf1 : We're here  
(10 characters)

buf2 : Where is the time?  
(18 characters)

buf3 : Why?  
(4 characters)

buf1 after strncat of 6 characters of buf2:  
We're hereWhere  
(16 characters)

buf1 after strncat of 25 characters of buf2:  
We're hereWhere Where is the time?  
(34 characters)

buf1 after strncat of 4 characters of buf3:  
We're hereWhere Where is the time?Why?  
(38 characters)

# 16-Bit Language Tools Libraries

---

---

---

## strncmp

---

**Description:** Compare two strings, up to a specified number of characters.

**Include:** <string.h>

**Prototype:** `int strncmp(const char *s1, const char *s2, size_t n);`

**Arguments:**

<i>s1</i>	first string
<i>s2</i>	second string
<i>n</i>	number of characters to compare

**Return Value:** Returns a positive number if *s1* is greater than *s2*, zero if *s1* is equal to *s2* or a negative number if *s1* is less than *s2*.

**Remarks:** `strncmp` returns a value based on the first character that differs between *s1* and *s2*. Characters that follow a null character are not compared.

**Example:**

```
#include <string.h> /* for strncmp */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n\n", buf3);

    res = strncmp(buf1, buf2, 6);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("6 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");

    printf("\n");

    res = strncmp(buf1, buf2, 20);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("20 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");
}
```



---

## strncmp (Continued)

---

```
printf("\n");

res = strncmp(buf1, buf3, 20);
if (res < 0)
    printf("buf1 comes before buf3\n");
else if (res == 0)
    printf("20 characters of buf1 and buf3 "
           "are equal\n");
else
    printf("buf3 comes before buf1\n");
}
```

### Output:

```
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?
```

```
6 characters of buf1 and buf2 are equal
```

```
buf2 comes before buf1
```

```
buf1 comes before buf3
```

---

## strncpy

---

**Description:** Copy characters from the source string into the destination string, up to the specified number of characters.

**Include:** <string.h>

**Prototype:** char \*strncpy(char \*s1, const char \*s2, size\_t n);

**Arguments:**

s1	destination string to copy to
s2	source string to copy from
n	number of characters to copy

**Return Value:** Returns a pointer to the destination string.

**Remarks:** Copies *n* characters from the source string to the destination string. If the source string is less than *n* characters, the destination is filled with null characters to total *n* characters. If *n* characters were copied and no null character was found, then the destination string will not be null-terminated. If the strings overlap, the behavior is undefined.

**Example:**

```
#include <string.h> /* for strncpy, strlen */
#include <stdio.h> /* for printf */
```

```
int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";
    char buf4[7] = "Where?";

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);
    printf("buf4 : %s\n", buf4);
}
```

# 16-Bit Language Tools Libraries

---

---

## strncpy (Continued)

---

```
    strncpy(buf1, buf2, 6);
    printf("buf1 after strncpy of 6 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));

    printf("\n");

    strncpy(buf1, buf2, 18);
    printf("buf1 after strncpy of 18 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));

    printf("\n");

    strncpy(buf1, buf3, 5);
    printf("buf1 after strncpy of 5 characters "
           "of buf3: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));

    printf("\n");

    strncpy(buf1, buf4, 9);
    printf("buf1 after strncpy of 9 characters "
           "of buf4: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));
}
```

### Output:

```
buf1 : We're here
buf2 : Where is the time?
buf3 : Why?
buf4 : Where?
buf1 after strncpy of 6 characters of buf2:
    Where here
    ( 10 characters)

buf1 after strncpy of 18 characters of buf2:
    Where is the time?
    ( 18 characters)

buf1 after strncpy of 5 characters of buf3:
    Why?
    ( 4 characters)

buf1 after strncpy of 9 characters of buf4:
    Where?
    ( 6 characters)
```

---

## strncpy (Continued)

---

**Explanation:**

Each buffer contains the string shown, followed by null characters for a length of 50. Using `strlen` will find the length of the string up to, but not including, the first null character.

In the first example, 6 characters of `buf2` ("Where ") replace the first 6 characters of `buf1` ("We're ") and the rest of `buf1` remains the same ("here" plus null characters).

In the second example, 18 characters replace the first 18 characters of `buf1` and the rest remain null characters.

In the third example, 5 characters of `buf3` ("Why?" plus a null terminating character) replace the first 5 characters of `buf1`. `buf1` now actually contains ("Why?", 1 null character, " is the time?", 32 null characters). `strlen` shows 4 characters because it stops when it reaches the first null character.

In the fourth example, since `buf4` is only 7 characters, `strncpy` uses 2 additional null characters to replace the first 9 characters of `buf1`. The result of `buf1` is 6 characters ("Where?") followed by 3 null characters, followed by 9 characters ("the time?"), followed by 32 null characters.

---

## strpbrk

---

**Description:** Search a string for the first occurrence of a character from a specified set of characters.

**Include:** `<string.h>`

**Prototype:** `char *strpbrk(const char *s1, const char *s2);`

**Arguments:** `s1` pointer to the string to be searched  
`s2` pointer to characters to search for

**Return Value:** Returns a pointer to the matched character in `s1` if found; otherwise, returns a null pointer.

**Remarks:** This function will search `s1` for the first occurrence of a character contained in `s2`.

**Example:**

```
#include <string.h> /* for strpbrk, NULL */
#include <stdio.h> /* for printf */

int main(void)
{
    char str1[20] = "What time is it?";
    char str2[20] = "xyz";
    char str3[20] = "eou?";
    char *ptr;
    int res;

    printf("strpbrk(\"%s\", \"%s\")\n", str1, str2);
    ptr = strpbrk(str1, str2);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("match found at position %d\n", res);
    }
    else
        printf("match not found\n");
}
```

---

## strpbrk (Continued)

---

```
printf("\n");

printf("strpbrk(\"%s\", \"%s\")\n", str1, str3);
ptr = strpbrk(str1, str3);
if (ptr != NULL)
{
    res = ptr - str1 + 1;
    printf("match found at position %d\n", res);
}
else
    printf("match not found\n");
}
```

### Output:

```
strpbrk("What time is it?", "xyz")
match not found
```

```
strpbrk("What time is it?", "eou?")
match found at position 9
```

---

## strrchr

---

**Description:** Search for the last occurrence of a specified character in a string.

**Include:** <string.h>

**Prototype:** char \*strrchr(const char \*s, int c);

**Arguments:** s pointer to the string to be searched  
c character to search for

**Return Value:** Returns a pointer to the character if found; otherwise, returns a null pointer.

**Remarks:** The function searches the string s, including the terminating null character, to find the last occurrence of character c.

**Example:**

```
#include <string.h> /* for strrchr, NULL */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'm', ch2 = 'y';
    char *ptr;
    int res;

    printf("buf1 : %s\n\n", buf1);

    ptr = strrchr(buf1, ch1);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
        printf("%c not found\n", ch1);
}
```

---

## strchr (Continued)

---

```
printf("\n");

ptr = strchr(buf1, ch2);
if (ptr != NULL)
{
    res = ptr - buf1 + 1;
    printf("%c found at position %d\n", ch2, res);
}
else
    printf("%c not found\n", ch2);
}
```

**Output:**

buf1 : What time is it?

m found at position 8

y not found

---

## strspn

---

**Description:** Calculate the number of consecutive characters at the beginning of a string that are contained in a set of characters.

**Include:** <string.h>

**Prototype:** `size_t strspn(const char *s1, const char *s2);`

**Arguments:**

<i>s1</i>	pointer to the string to be searched
<i>s2</i>	pointer to characters to search for

**Return Value:** Returns the number of consecutive characters from the beginning of *s1* that are contained in *s2*.

**Remarks:** This function stops searching when a character from *s1* is not in *s2*.

**Example:**

```
#include <string.h> /* for strspn */
#include <stdio.h> /* for printf */

int main(void)
{
    char str1[20] = "animal";
    char str2[20] = "aeioum";
    char str3[20] = "aimnl";
    char str4[20] = "xyz";
    int res;

    res = strspn(str1, str2);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str2, res);

    res = strspn(str1, str3);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str3, res);

    res = strspn(str1, str4);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str4, res);
}
```

---

## strspn (Continued)

---

**Output:**

```
strspn("animal", "aeioum") = 5
strspn("animal", "aimnl") = 6
strspn("animal", "xyz") = 0
```

**Explanation:**

In the first result, l is not in *s2*.

In the second result, the terminating null is not in *s2*.

In the third result, a is not in *s2*, so the comparison stops.

---

## strstr

---

**Description:** Search for the first occurrence of a string inside another string.

**Include:** <string.h>

**Prototype:** char \*strstr(const char \*s1, const char \*s2);

**Arguments:** *s1* pointer to the string to be searched  
*s2* pointer to substring to be searched for

**Return Value:** Returns the address of the first element that matches the substring if found; otherwise, returns a null pointer.

**Remarks:** This function will find the first occurrence of the string *s2* (excluding the null terminator) within the string *s1*. If *s2* points to a zero length string, *s1* is returned.

**Example:** #include <string.h> /\* for strstr, NULL \*/  
#include <stdio.h> /\* for printf \*/

```
int main(void)
{
    char str1[20] = "What time is it?";
    char str2[20] = "is";
    char str3[20] = "xyz";
    char *ptr;
    int res;

    printf("str1 : %s\n", str1);
    printf("str2 : %s\n", str2);
    printf("str3 : %s\n\n", str3);

    ptr = strstr(str1, str2);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("\n%s\n" found at position %d\n",
            str2, res);
    }
    else
        printf("\n%s\n" not found\n", str2);
}
```

---

## strstr (Continued)

---

```
printf("\n");

ptr = strstr(str1, str3);
if (ptr != NULL)
{
    res = ptr - str1 + 1;
    printf("\n%s\" found at position %d\n",
          str3, res);
}
else
    printf("\n%s\" not found\n", str3);
}
```

### Output:

```
str1 : What time is it?
str2 : is
str3 : xyz
```

```
"is" found at position 11
```

```
"xyz" not found
```

---

## strtok

---

**Description:** Break a string into substrings, or tokens, by inserting null characters in place of specified delimiters.

**Include:** <string.h>

**Prototype:** char \*strtok(char \*s1, const char \*s2);

**Arguments:** *s1* pointer to the null terminated string to be searched  
*s2* pointer to characters to be searched for (used as delimiters)

**Return Value:** Returns a pointer to the first character of a token (the first character in *s1* that does not appear in the set of characters of *s2*). If no token is found, the null pointer is returned.

**Remarks:** A sequence of calls to this function can be used to split up a string into substrings (or tokens) by replacing specified characters with null characters. The first time this function is invoked on a particular string, that string should be passed in *s1*. After the first time, this function can continue parsing the string from the last delimiter by invoking it with a null value passed in *s1*.

It skips all leading characters that appear in the string *s2* (delimiters), then skips all characters not appearing in *s2* (this segment of characters is the token), and then overwrites the next character with a null character, terminating the current token. The function, *strtok*, then saves a pointer to the character that follows, from which the next search will start. If *strtok* finds the end of the string before it finds a delimiter, the current token extends to the end of the string pointed to by *s1*. If this is the first call to *strtok*, it does not modify the string (no null characters are written to *s1*). The set of characters that is passed in *s2* need not be the same for each call to *strtok*.

If *strtok* is called with a non-null parameter for *s1* after the initial call, the string becomes the new string to search. The old string previously searched will be lost.

# 16-Bit Language Tools Libraries

---

---

## strtok (Continued)

---

**Example:**

```
#include <string.h> /* for strtok, NULL */
#include <stdio.h> /* for printf */

int main(void)
{
    char str1[30] = "Here, on top of the world!";
    char delim[5] = ", .";
    char *word;
    int x;

    printf("str1 : %s\n", str1);
    x = 1;
    word = strtok(str1,delim);
    while (word != NULL)
    {
        printf("word %d: %s\n", x++, word);
        word = strtok(NULL, delim);
    }
}
```

**Output:**

```
str1 : Here, on top of the world!
word 1: Here
word 2: on
word 3: top
word 4: of
word 5: the
word 6: world!
```

---

## strxfrm

---

**Description:** Transforms a string using the locale-dependent rules (see Remarks).

**Include:** <string.h>

**Prototype:** `size_t strxfrm(char *s1, const char *s2, size_t n);`

**Arguments:**

<i>s1</i>	destination string
<i>s2</i>	source string to be transformed
<i>n</i>	number of characters to transform

**Return Value:** Returns the length of the transformed string not including the terminating null character. If *n* is zero, the string is not transformed (*s1* may be a point null in this case) and the length of *s2* is returned.

**Remarks:** If the return value is greater than or equal to *n*, the content of *s1* is indeterminate. Since the 16-bit compiler does not support alternate locales, the transformation is equivalent to `strcpy`, except that the length of the destination string is bounded by *n*-1.



## 2.16 <TIME.H> DATE AND TIME FUNCTIONS

The header file `time.h` consists of types, macros and functions that manipulate time.

---

### clock\_t

---

**Description:** Stores processor time values.  
**Include:** `<time.h>`  
**Prototype:** `typedef long clock_t`

---

---

### size\_t

---

**Description:** The type of the result of the `sizeof` operator.  
**Include:** `<time.h>`

---

---

### struct tm

---

**Description:** Structure used to hold the time and date (calendar time).  
**Include:** `<time.h>`  
**Prototype:**

```
struct tm {
    int tm_sec; /*seconds after the minute ( 0 to 61 )*/
                /*allows for up to two leap seconds*/
    int tm_min; /*minutes after the hour ( 0 to 59 )*/
    int tm_hour; /*hours since midnight ( 0 to 23 )*/
    int tm_mday; /*day of month ( 1 to 31 )*/
    int tm_mon; /*month ( 0 to 11 where January = 0 )*/
    int tm_year; /*years since 1900*/
    int tm_wday; /*day of week ( 0 to 6 where Sunday = 0
)*/
    int tm_yday; /*day of year ( 0 to 365 where January 1
= 0 )*/
    int tm_isdst; /*Daylight Savings Time flag*/
}
```

**Remarks:** If `tm_isdst` is a positive value, Daylight Savings is in effect. If it is zero, Daylight Saving Time is not in effect. If it is a negative value, the status of Daylight Saving Time is not known.

---

---

### time\_t

---

**Description:** Represents calendar time values.  
**Include:** `<time.h>`  
**Prototype:** `typedef long time_t`

---

---

### CLOCKS\_PER\_SEC

---

**Description:** Number of processor clocks per second.  
**Include:** `<time.h>`  
**Prototype:** `#define CLOCKS_PER_SEC`  
**Value:** 1  
**Remarks:** The compiler returns clock ticks (instruction cycles) not actual time.

---

# 16-Bit Language Tools Libraries

---

---

---

## NULL

---

**Description:** The value of a null pointer constant.  
**Include:** <time.h>

---

---

## asctime

---

**Description:** Converts the time structure to a character string.  
**Include:** <time.h>  
**Prototype:** `char *asctime(const struct tm *tptr);`  
**Argument:** `tptr` time/date structure  
**Return Value:** Returns a pointer to a character string of the following format:  
DDD MMM dd hh:mm:ss YYYY  
DDD is day of the week  
MMM is month of the year  
dd is day of the month  
hh is hour  
mm is minute  
ss is second  
YYYY is year

**Example:**

```
#include <time.h> /* for asctime, tm */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    struct tm when;
    time_t whattime;

    when.tm_sec = 30;
    when.tm_min = 30;
    when.tm_hour = 2;
    when.tm_mday = 1;
    when.tm_mon = 1;
    when.tm_year = 103;

    whattime = mktime(&when);
    printf("Day and time is %s\n", asctime(&when));
}
```

**Output:**

Day and time is Sat Feb 1 02:30:30 2003

---

---

## clock

---

**Description:** Calculates the processor time.  
**Include:** <time.h>  
**Prototype:** `clock_t clock(void);`  
**Return Value:** Returns the number of clock ticks of elapsed processor time.  
**Remarks:** If the target environment cannot measure elapsed processor time, the function returns -1 cast as a `clock_t` (i.e. `(clock_t) -1`). By default, the 16-bit compiler returns the time as instruction cycles.

---

---

## clock (Continued)

---

**Example:**

```
#include <time.h> /* for clock */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    clock_t start, stop;
    int ct;

    start = clock();
    for (i = 0; i < 10; i++)
        stop = clock();
    printf("start = %ld\n", start);
    printf("stop = %ld\n", stop);
}

Output:
start = 0
stop = 317
```

---

## ctime

---

**Description:** Converts calendar time to a string representation of local time.

**Include:** <time.h>

**Prototype:** char \*ctime(const time\_t \*tod);

**Argument:** *tod* pointer to stored time

**Return Value:** Returns the address of a string that represents the local time of the parameter passed.

**Remarks:** This function is equivalent to asctime(localtime(*tod*)).

**Example:**

```
#include <time.h> /* for mktime, tm, ctime */
#include <stdio.h> /* for printf */

int main(void)
{
    time_t whattime;
    struct tm nowtime;

    nowtime.tm_sec = 30;
    nowtime.tm_min = 30;
    nowtime.tm_hour = 2;
    nowtime.tm_mday = 1;
    nowtime.tm_mon = 1;
    nowtime.tm_year = 103;

    whattime = mktime(&nowtime);
    printf("Day and time %s\n", ctime(&whattime));
}

Output:
Day and time Sat Feb 1 02:30:30 2003
```

# 16-Bit Language Tools Libraries

---

---

---

## difftime

---

<b>Description:</b>	Find the difference between two times.
<b>Include:</b>	<time.h>
<b>Prototype:</b>	double difftime(time_t t1, time_t t0);
<b>Arguments:</b>	<i>t1</i> ending time <i>t0</i> beginning time
<b>Return Value:</b>	Returns the number of seconds between <i>t1</i> and <i>t0</i> .
<b>Remarks:</b>	By default, the 16-bit compiler returns the time as instruction cycles so difftime returns the number of ticks between <i>t1</i> and <i>t0</i> .
<b>Example:</b>	<pre>#include &lt;time.h&gt; /* for clock, difftime */ #include &lt;stdio.h&gt; /* for printf */  volatile int i;  int main(void) {     clock_t start, stop;     double elapsed;      start = clock();     for (i = 0; i &lt; 10; i++)         stop = clock();     printf("start = %ld\n", start);     printf("stop = %ld\n", stop);     elapsed = difftime(stop, start);     printf("Elapsed time = %.0f\n", elapsed); }</pre> <p><b>Output:</b> start = 0 stop = 317 Elapsed time = 317</p>

---

## gmtime

---

<b>Description:</b>	Converts calendar time to time structure expressed as Universal Time Coordinated (UTC) also known as Greenwich Mean Time (GMT).
<b>Include:</b>	<time.h>
<b>Prototype:</b>	struct tm *gmtime(const time_t *tod);
<b>Argument:</b>	<i>tod</i> pointer to stored time
<b>Return Value:</b>	Returns the address of the time structure.
<b>Remarks:</b>	This function breaks down the <i>tod</i> value into the time structure of type tm. By default, the 16-bit compiler returns the time as instruction cycles. With this default, gmtime and localtime will be equivalent, except gmtime will return tm_isdst (Daylight Savings Time flag) as zero to indicate that Daylight Savings Time is not in effect.

---

## gmtime (Continued)

---

**Example:**

```
#include <time.h> /* for gmtime, asctime, */
                    /* time_t, tm          */
#include <stdio.h> /* for printf          */

int main(void)
{
    time_t timer;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */

    newtime = gmtime(&timer);
    printf("UTC time = %s\n", asctime(newtime));
}
```

**Output:**

UTC time = Mon Oct 20 16:43:02 2003

---

## localtime

---

**Description:** Converts a value to the local time.

**Include:** <time.h>

**Prototype:** struct tm \*localtime(const time\_t \*tod);

**Argument:** tod pointer to stored time

**Return Value:** Returns the address of the time structure.

**Remarks:** By default, the 16-bit compiler returns the time as instruction cycles. With this default, localtime and gmtime will be equivalent, except localtime will return tm\_isdst (Daylight Savings Time flag) as -1 to indicate that the status of Daylight Savings Time is not known.

**Example:**

```
#include <time.h> /* for localtime,      */
                    /* asctime, time_t, tm */
#include <stdio.h> /* for printf          */

int main(void)
{
    time_t timer;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */

    newtime = localtime(&timer);
    printf("Local time = %s\n", asctime(newtime));
}
```

**Output:**

Local time = Mon Oct 20 16:43:02 2003

---

# 16-Bit Language Tools Libraries

---

---

---

## mktime

---

**Description:** Converts local time to a calendar value.

**Include:** `<time.h>`

**Prototype:** `time_t mktime(struct tm *tptr);`

**Argument:** `tptr` a pointer to the time structure

**Return Value:** Returns the calendar time encoded as a value of `time_t`.

**Remarks:** If the calendar time cannot be represented, the function returns -1 cast as a `time_t` (i.e. `(time_t) -1`).

**Example:**

```
#include <time.h> /* for localtime, */
                    /* asctime, mktime, */
                    /* time_t, tm */
#include <stdio.h> /* for printf */

int main(void)
{
    time_t timer, whattime;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */
    /* localtime allocates space for struct tm */
    newtime = localtime(&timer);
    printf("Local time = %s", asctime(newtime));

    whattime = mktime(newtime);
    printf("Calendar time as time_t = %ld\n",
           whattime);
}
```

**Output:**

```
Local time = Mon Oct 20 16:43:02 2003
Calendar time as time_t = 1066668182
```

---

## strftime

---

**Description:** Formats the time structure to a string based on the format parameter.

**Include:** `<time.h>`

**Prototype:** `size_t strftime(char *s, size_t n, const char *format, const struct tm *tptr);`

**Arguments:** `s` output string  
`n` maximum length of string  
`format` format-control string  
`tptr` pointer to tm data structure

**Return Value:** Returns the number of characters placed in the array, `s`, if the total, including the terminating null, is not greater than `n`. Otherwise, the function returns 0 and the contents of array `s` are indeterminate.

**Remarks:** The format parameters follow:

- %a** abbreviated weekday name
- %A** full weekday name
- %b** abbreviated month name
- %B** full month name
- %c** appropriate date and time representation
- %d** day of the month (01-31)
- %H** hour of the day (00-23)

## strftime (Continued)

---

**%l** hour of the day (01-12)  
**%j** day of the year (001-366)  
**%m** month of the year (01-12)  
**%M** minute of the hour (00-59)  
**%p** AM/PM designator  
**%S** second of the minute (00-61)  
allowing for up to two leap seconds  
**%U** week number of the year where Sunday is the first day of week 1  
(00-53)  
**%w** weekday where Sunday is day 0 (0-6)  
**%W** week number of the year where Monday is the first day of week 1  
(00-53)  
**%x** appropriate date representation  
**%X** appropriate time representation  
**%y** year without century (00-99)  
**%Y** year with century  
**%Z** time zone (possibly abbreviated) or no characters if time zone is  
unavailable  
**%%** percent character %

**Example:**

```
#include <time.h> /* for strftime, */
                    /* localtime, */
                    /* time_t, tm */
#include <stdio.h> /* for printf */

int main(void)
{
    time_t timer, whattime;
    struct tm *newtime;
    char buf[128];

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */
    /* localtime allocates space for structure */
    newtime = localtime(&timer);

    strftime(buf, 128, "It was a %A, %d days into the "
               "month of %B in the year %Y.\n", newtime);
    printf(buf);

    strftime(buf, 128, "It was %W weeks into the year "
               "or %j days into the year.\n", newtime);
    printf(buf);
}
```

**Output:**

```
It was a Monday, 20 days into the month of October in
the year 2003.
It was 42 weeks into the year or 293 days into the
year.
```

# 16-Bit Language Tools Libraries

---

---

## time

---

**Description:** Calculates the current calendar time.

**Include:** <time.h>

**Prototype:** `time_t time(time_t *tod);`

**Argument:** `tod` pointer to storage location for time

**Return Value:** Returns the calendar time encoded as a value of `time_t`.

**Remarks:** If the target environment cannot determine the time, the function returns -1 cast as a `time_t`. By default, the 16-bit compiler returns the time as instruction cycles. This function is customizable (see `pic30-libs`).

**Example:**

```
#include <time.h> /* for time */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    time_t ticks;

    time(0); /* start time */
    for (i = 0; i < 10; i++) /* waste time */
        time(&ticks); /* get time */
    printf("Time = %ld\n", ticks);
}
```

**Output:**

Time = 256



---

---

## Chapter 3. Standard C Libraries - Math Functions

---

---

### 3.1 INTRODUCTION

Standard ANSI C library math functions are contained in the file, `libm-omf.a`, where `omf` will be `coff` or `elf` depending upon the selected object module format.

#### 3.1.1 Assembly Code Applications

A free version of the math functions library and header file is available from the Microchip web site. No source code is available with this free version.

#### 3.1.2 C Code Applications

The MPLAB XC16 C Compiler install directory (**Section 1.1.1 “Compiler Installation Locations”**) contains the following subdirectories with library-related files:

- `lib` – standard C library files
- `src\libm` – source code for math library functions, batch file to rebuild the library
- `support\h` – header files for libraries

In addition, there is a file, `ResourceGraphs.pdf`, which contains diagrams of resources used by each function, located in `lib`.

#### 3.1.3 Chapter Organization

This chapter is organized as follows:

- Using the Standard C Libraries
- `<math.h>` mathematical functions

### 3.2 USING THE STANDARD C LIBRARIES

Building an application which utilizes the standard C libraries requires two types of files: header files and library files.

#### 3.2.1 Header Files

All standard C library entities are declared or defined in one or more standard headers. (See list in **Section 3.1.3 “Chapter Organization”**.) To make use of a library entity in a program, write an include directive that names the relevant standard header.

The contents of a standard header are included by naming them in an include directive, as in:

```
#include <stdio.h> /* include I/O facilities */
```

The standard headers can be included in any order. Do not include a standard header within a declaration. Do not define macros that have the same names as keywords before including a standard header.

A standard header never includes another standard header.

# 16-Bit Language Tools Libraries

---

## 3.2.2 Library Files

The archived library files contain all the individual object files for each library function. When linking an application, the library file must be provided as an input to the linker (using the `--library` or `-l` linker option), such that the functions used by the application may be linked into the application.

A typical C application will require three library files: `libc-omf.a`, `libm-omf.a` and `libpic30-omf.a` (see **Section 1.2 “OMF-Specific Libraries/Start-up Modules”** for more on OMF-specific libraries). These libraries will be included automatically if linking is performed using the compiler.

**Note:** Some standard library functions require a heap. These include the standard I/O functions that open files and the memory allocation functions. See the “*MPLAB<sup>®</sup> XC16 Assembler, Linker and Utilities User’s Guide*” (DS50002106) and “*MPLAB<sup>®</sup> XC16 C Compiler User’s Guide*” (DS50002071) for more information on the heap.

## 3.3 <MATH.H> MATHEMATICAL FUNCTIONS

The header file `math.h` consists of a macro and various functions that calculate common mathematical operations. Error conditions may be handled with a domain error or range error (see `errno.h`).

A domain error occurs when the input argument is outside the domain over which the function is defined. The error is reported by storing the value of `EDOM` in `errno` and returning a particular value defined for each function.

A range error occurs when the result is too large or too small to be represented in the target precision. The error is reported by storing the value of `ERANGE` in `errno` and returning `HUGE_VAL` if the result overflowed (return value was too large) or a zero if the result underflowed (return value is too small).

Responses to special values, such as NaNs, zeros and infinities, may vary depending upon the function. Each function description includes a definition of the function's response to such values.

---

### HUGE\_VAL

---

<b>Description:</b>	<code>HUGE_VAL</code> is returned by a function on a range error (e.g., the function tries to return a value too large to be represented in the target precision).
<b>Include:</b>	<code>&lt;math.h&gt;</code>
<b>Remarks:</b>	<code>-HUGE_VAL</code> is returned if a function result is negative and is too large (in magnitude) to be represented in the target precision. When the printed result is <code>+/- HUGE_VAL</code> , it will be represented by <code>+/- inf</code> .

---

### acos

---

<b>Description:</b>	Calculates the trigonometric arc cosine function of a double precision floating-point value.
<b>Include:</b>	<code>&lt;math.h&gt;</code>
<b>Prototype:</b>	<code>double acos (double x);</code>
<b>Argument:</b>	<code>x</code> value between -1 and 1 for which to return the arc cosine
<b>Return Value:</b>	Returns the arc cosine in radians in the range of 0 to pi (inclusive).
<b>Remarks:</b>	A domain error occurs if <code>x</code> is less than -1 or greater than 1.
<b>Example:</b>	<pre>#include &lt;math.h&gt; /* for acos          */ #include &lt;stdio.h&gt; /* for printf, perror */ #include &lt;errno.h&gt; /* for errno        */  int main(void) {     double x,y;      errno = 0;     x = -2.0;     y = acos (x);     if (errno)         perror("Error");     printf("The arccosine of %f is %f\n\n", x, y);</pre>

---

## acos (Continued)

---

```
    errno = 0;
    x = 0.10;
    y = acos (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n\n", x, y);
}
```

**Output:**

```
Error: domain error
The arccosine of -2.000000 is nan
```

```
The arccosine of 0.100000 is 1.470629
```

---

## acosf

---

**Description:** Calculates the trigonometric arc cosine function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float acosf (float x);

**Argument:** x value between -1 and 1

**Return Value:** Returns the arc cosine in radians in the range of 0 to pi (inclusive).

**Remarks:** A domain error occurs if x is less than -1 or greater than 1.

**Example:**

```
#include <math.h> /* for acosf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = acosf (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0F;
    y = acosf (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n", x, y);
}
```

**Output:**

```
Error: domain error
The arccosine of 2.000000 is nan
```

```
The arccosine of 0.000000 is 1.570796
```

# Standard C Libraries - Math Functions

---

## asin

---

**Description:** Calculates the trigonometric arc sine function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double asin (double x);

**Argument:** x value between -1 and 1 for which to return the arc sine

**Return Value:** Returns the arc sine in radians in the range of -pi/2 to +pi/2 (inclusive).

**Remarks:** A domain error occurs if x is less than -1 or greater than 1.

**Example:**

```
#include <math.h> /* for asin          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = asin (x);
    if (errno)
        perror("Error");
    printf("The arcsine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0;
    y = asin (x);
    if (errno)
        perror("Error");
    printf("The arcsine of %f is %f\n\n", x, y);
}
```

**Output:**

```
Error: domain error
The arcsine of 2.000000 is nan

The arcsine of 0.000000 is 0.000000
```

---

## asinf

---

**Description:** Calculates the trigonometric arc sine function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float asinf (float x);

**Argument:** x value between -1 and 1

**Return Value:** Returns the arc sine in radians in the range of -pi/2 to +pi/2 (inclusive).

**Remarks:** A domain error occurs if x is less than -1 or greater than 1.

**Example:**

```
#include <math.h> /* for asinf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */

int main(void)
{
    float x, y;
```

---

## asin (Continued)

---

```
    errno = 0;
    x = 2.0F;
    y = asinf(x);
    if (errno)
        perror("Error");
    printf("The arcsine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0F;
    y = asinf(x);
    if (errno)
        perror("Error");
    printf("The arcsine of %f is %f\n\n", x, y);
}
```

**Output:**

```
Error: domain error
The arcsine of 2.000000 is nan

The arcsine of 0.000000 is 0.000000
```

---

## atan

---

**Description:** Calculates the trigonometric arc tangent function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double atan (double x);

**Argument:** x value for which to return the arc tangent

**Return Value:** Returns the arc tangent in radians in the range of  $-\pi/2$  to  $+\pi/2$  (inclusive).

**Remarks:** No domain or range error will occur.

**Example:** #include <math.h> /\* for atan \*/  
#include <stdio.h> /\* for printf \*/

```
int main(void)
{
    double x, y;

    x = 2.0;
    y = atan (x);
    printf("The arctangent of %f is %f\n\n", x, y);

    x = -1.0;
    y = atan (x);
    printf("The arctangent of %f is %f\n\n", x, y);
}
```

**Output:**

```
The arctangent of 2.000000 is 1.107149

The arctangent of -1.000000 is -0.785398
```

# Standard C Libraries - Math Functions

---

## atanf

---

**Description:** Calculates the trigonometric arc tangent function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float atanf (float x);

**Argument:** x value for which to return the arc tangent

**Return Value:** Returns the arc tangent in radians in the range of  $-\pi/2$  to  $+\pi/2$  (inclusive).

**Remarks:** No domain or range error will occur.

**Example:**

```
#include <math.h> /* for atanf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x, y;

    x = 2.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n\n", x, y);

    x = -1.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n\n", x, y);
}
```

**Output:**

The arctangent of 2.000000 is 1.107149

The arctangent of -1.000000 is -0.785398

---

## atan2

---

**Description:** Calculates the trigonometric arc tangent function of  $y/x$ .

**Include:** <math.h>

**Prototype:** double atan2 (double y, double x);

**Arguments:** y y value for which to return the arc tangent  
x x value for which to return the arc tangent

**Return Value:** Returns the arc tangent in radians in the range of  $-\pi$  to  $\pi$  (inclusive) with the quadrant determined by the signs of both parameters.

**Remarks:** A domain error occurs if both  $x$  and  $y$  are zero or both  $x$  and  $y$  are  $\pm$  infinity.

**Example:**

```
#include <math.h> /* for atan2 */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x, y, z;
```

# 16-Bit Language Tools Libraries

---

---

## atan2 (Continued)

---

```
    errno = 0;
    x = 0.0;
    y = 2.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);

    errno = 0;
    x = -1.0;
    y = 0.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);

    errno = 0;
    x = 0.0;
    y = 0.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);
}
```

### Output:

The arctangent of 2.000000/0.000000 is 1.570796

The arctangent of 0.000000/-1.000000 is 3.141593

Error: domain error

The arctangent of 0.000000/0.000000 is nan



# Standard C Libraries - Math Functions

---

## atan2f

---

**Description:** Calculates the trigonometric arc tangent function of  $y/x$ .

**Include:** `<math.h>`

**Prototype:** `float atan2f (float y, float x);`

**Arguments:**  $y$   $y$  value for which to return the arc tangent  
 $x$   $x$  value for which to return the arc tangent

**Return Value:** Returns the arc tangent in radians in the range of  $-\pi$  to  $\pi$  with the quadrant determined by the signs of both parameters.

**Remarks:** A domain error occurs if both  $x$  and  $y$  are zero or both  $x$  and  $y$  are  $\pm$  infinity.

**Example:**

```
#include <math.h> /* for atan2f          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y, z;

    errno = 0;
    x = 2.0F;
    y = 0.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);

    errno = 0;
    x = 0.0F;
    y = -1.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);

    errno = 0;
    x = 0.0F;
    y = 0.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);
}
```

**Output:**

The arctangent of 2.000000/0.000000 is 1.570796

The arctangent of 0.000000/-1.000000 is 3.141593

Error: domain error

The arctangent of 0.000000/0.000000 is nan

# 16-Bit Language Tools Libraries

---

---

## ceil

---

**Description:** Calculates the ceiling of a value.

**Include:** <math.h>

**Prototype:** double ceil(double x);

**Argument:** x a floating-point value for which to return the ceiling

**Return Value:** Returns the smallest integer value greater than or equal to x.

**Remarks:** No domain or range error will occur. See floor.

**Example:**

```
#include <math.h> /* for ceil */
#include <stdio.h> /* for printf */

int main(void)
{
    double x[8] = {2.0, 1.75, 1.5, 1.25, -2.0,
                  -1.75, -1.5, -1.25};
    double y;
    int i;

    for (i=0; i<8; i++)
    {
        y = ceil (x[i]);
        printf("The ceiling for %f is %f\n", x[i], y);
    }
}
```

**Output:**

```
The ceiling for 2.000000 is 2.000000
The ceiling for 1.750000 is 2.000000
The ceiling for 1.500000 is 2.000000
The ceiling for 1.250000 is 2.000000
The ceiling for -2.000000 is -2.000000
The ceiling for -1.750000 is -1.000000
The ceiling for -1.500000 is -1.000000
The ceiling for -1.250000 is -1.000000
```

# Standard C Libraries - Math Functions

---

## ceilf

---

**Description:** Calculates the ceiling of a value.

**Include:** <math.h>

**Prototype:** float ceilf(float x);

**Argument:** x floating-point value

**Return Value:** Returns the smallest integer value greater than or equal to x.

**Remarks:** No domain or range error will occur (see floorf).

**Example:**

```
#include <math.h> /* for ceilf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x[8] = {2.0F, 1.75F, 1.5F, 1.25F,
                 -2.0F, -1.75F, -1.5F, -1.25F};
    float y;
    int i;

    for (i=0; i<8; i++)
    {
        y = ceilf (x[i]);
        printf("The ceiling for %f is %f\n", x[i], y);
    }
}
```

**Output:**

```
The ceiling for 2.000000 is 2.000000
The ceiling for 1.750000 is 2.000000
The ceiling for 1.500000 is 2.000000
The ceiling for 1.250000 is 2.000000
The ceiling for -2.000000 is -2.000000
The ceiling for -1.750000 is -1.000000
The ceiling for -1.500000 is -1.000000
The ceiling for -1.250000 is -1.000000
```

---

## COS

---

**Description:** Calculates the trigonometric cosine function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double cos (double x);

**Argument:** x value for which to return the cosine

**Return Value:** Returns the cosine of x in radians in the ranges of -1 to 1 inclusive.

**Remarks:** A domain error will occur if x is a NaN or infinity.

**Example:**

```
#include <math.h> /* for cos */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x,y;
```

---

## cos (Continued)

---

```
    errno = 0;
    x = -1.0;
    y = cos (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n\n", x, y);
    errno = 0;
    x = 0.0;
    y = cos (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n\n", x, y);
}
```

### Output:

The cosine of -1.000000 is 0.540302

The cosine of 0.000000 is 1.000000

---

## cosf

---

**Description:** Calculates the trigonometric cosine function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float cosf (float x);

**Argument:** x value for which to return the cosine

**Return Value:** Returns the cosine of x in radians in the ranges of -1 to 1 inclusive.

**Remarks:** A domain error will occur if x is a NaN or infinity.

**Example:**

```
#include <math.h> /* for cosf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = cosf (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0F;
    y = cosf (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n\n", x, y);
}
```

# Standard C Libraries - Math Functions

---

---

## cosf (Continued)

---

**Output:**

The cosine of -1.000000 is 0.540302

The cosine of 0.000000 is 1.000000

---

## cosh

---

**Description:** Calculates the hyperbolic cosine function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double cosh (double x);

**Argument:** x value for which to return the hyperbolic cosine

**Return Value:** Returns the hyperbolic cosine of x.

**Remarks:** A range error will occur if the magnitude of x is too large.

**Example:**

```
#include <math.h> /* for cosh          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno       */

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.5;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\n",
          x, y);

    errno = 0;
    x = 0.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\n",
          x, y);

    errno = 0;
    x = 720.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\n",
          x, y);
}
```

**Output:**

The hyperbolic cosine of -1.500000 is 2.352410

The hyperbolic cosine of 0.000000 is 1.000000

Error: range error

The hyperbolic cosine of 720.000000 is inf

---

---

## coshf

---

**Description:** Calculates the hyperbolic cosine function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float coshf (float x);

**Argument:** x value for which to return the hyperbolic cosine

**Return Value:** Returns the hyperbolic cosine of x.

**Remarks:** A range error will occur if the magnitude of x is too large.

**Example:**

```
#include <math.h> /* for coshf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */
```

```
int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 720.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\n",
           x, y);
}
```

**Output:**

The hyperbolic cosine of -1.000000 is 1.543081

The hyperbolic cosine of 0.000000 is 1.000000

Error: range error

The hyperbolic cosine of 720.000000 is inf

# Standard C Libraries - Math Functions

---

## exp

---

**Description:** Calculates the exponential function of  $x$  ( $e$  raised to the power  $x$  where  $x$  is a double precision floating-point value).

**Include:** <math.h>

**Prototype:** double exp (double x);

**Argument:**  $x$  value for which to return the exponential

**Return Value:** Returns the exponential of  $x$ . On an overflow, exp returns inf and on an underflow exp returns 0.

**Remarks:** A range error occurs if the magnitude of  $x$  is too large.

**Example:**

```
#include <math.h> /* for exp          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno       */

int main(void)
{
    double x, y;

    errno = 0;
    x = 1.0;
    y = exp (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);

    errno = 0;
    x = 1E3;
    y = exp (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);

    errno = 0;
    x = -1E3;
    y = exp (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);
}
```

**Output:**

The exponential of 1.000000 is 2.718282

Error: range error

The exponential of 1000.000000 is inf

Error: range error

The exponential of -1000.000000 is 0.000000

# 16-Bit Language Tools Libraries

---

---

## expf

---

**Description:** Calculates the exponential function of  $x$  (e raised to the power  $x$  where  $x$  is a single precision floating-point value).

**Include:** <math.h>

**Prototype:** float expf (float x);

**Argument:**  $x$  floating-point value for which to return the exponential

**Return Value:** Returns the exponential of  $x$ . On an overflow, expf returns inf and on an underflow exp returns 0.

**Remarks:** A range error occurs if the magnitude of  $x$  is too large.

**Example:**

```
#include <math.h> /* for expf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno       */
```

```
int main(void)
{
    float x, y;

    errno = 0;
    x = 1.0F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);

    errno = 0;
    x = 1.0E3F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);

    errno = 0;
    x = -1.0E3F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);
}
```

**Output:**

The exponential of 1.000000 is 2.718282

Error: range error

The exponential of 1000.000000 is inf

Error: range error

The exponential of -1000.000000 is 0.000000



# Standard C Libraries - Math Functions

---

## fabs

---

**Description:** Calculates the absolute value of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double fabs(double x);

**Argument:** x floating-point value for which to return the absolute value

**Return Value:** Returns the absolute value of x. A negative number is returned as positive, a positive number is unchanged.

**Remarks:** No domain or range error will occur.

**Example:**

```
#include <math.h> /* for fabs */
#include <stdio.h> /* for printf */

int main(void)
{
    double x, y;

    x = 1.75;
    y = fabs (x);
    printf("The absolute value of %f is %f\n", x, y);

    x = -1.5;
    y = fabs (x);
    printf("The absolute value of %f is %f\n", x, y);
}
```

**Output:**

```
The absolute value of 1.750000 is 1.750000
The absolute value of -1.500000 is 1.500000
```

---

## fabsf

---

**Description:** Calculates the absolute value of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float fabsf(float x);

**Argument:** x floating-point value for which to return the absolute value

**Return Value:** Returns the absolute value of x. A negative number is returned as positive, a positive number is unchanged.

**Remarks:** No domain or range error will occur.

**Example:**

```
#include <math.h> /* for fabsf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x,y;

    x = 1.75F;
    y = fabsf (x);
    printf("The absolute value of %f is %f\n", x, y);

    x = -1.5F;
    y = fabsf (x);
    printf("The absolute value of %f is %f\n", x, y);
}
```

**Output:**

```
The absolute value of 1.750000 is 1.750000
The absolute value of -1.500000 is 1.500000
```

# 16-Bit Language Tools Libraries

---

---

---

## floor

---

**Description:** Calculates the floor of a double precision floating-point value.

**Include:** `<math.h>`

**Prototype:** `double floor (double x);`

**Argument:** `x` floating-point value for which to return the floor

**Return Value:** Returns the largest integer value less than or equal to `x`.

**Remarks:** No domain or range error will occur. See `ceil`.

**Example:**

```
#include <math.h> /* for floor */
#include <stdio.h> /* for printf */

int main(void)
{
    double x[8] = {2.0, 1.75, 1.5, 1.25, -2.0,
                  -1.75, -1.5, -1.25};

    double y;
    int i;

    for (i=0; i<8; i++)
    {
        y = floor (x[i]);
        printf("The ceiling for %f is %f\n", x[i], y);
    }
}
```

**Output:**

```
The floor for 2.000000 is 2.000000
The floor for 1.750000 is 1.000000
The floor for 1.500000 is 1.000000
The floor for 1.250000 is 1.000000
The floor for -2.000000 is -2.000000
The floor for -1.750000 is -2.000000
The floor for -1.500000 is -2.000000
The floor for -1.250000 is -2.000000
```

---

## floorf

---

**Description:** Calculates the floor of a single precision floating-point value.

**Include:** `<math.h>`

**Prototype:** `float floorf(float x);`

**Argument:** `x` floating-point value

**Return Value:** Returns the largest integer value less than or equal to `x`.

**Remarks:** No domain or range error will occur. See `ceilf`.

# Standard C Libraries - Math Functions

---

## floorf (Continued)

---

**Example:**

```
#include <math.h> /* for floorf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x[8] = {2.0F, 1.75F, 1.5F, 1.25F,
                 -2.0F, -1.75F, -1.5F, -1.25F};
    float y;
    int i;

    for (i=0; i<8; i++)
    {
        y = floorf (x[i]);
        printf("The floor for %f is %f\n", x[i], y);
    }
}
```

**Output:**

```
The floor for 2.000000 is 2.000000
The floor for 1.750000 is 1.000000
The floor for 1.500000 is 1.000000
The floor for 1.250000 is 1.000000
The floor for -2.000000 is -2.000000
The floor for -1.750000 is -2.000000
The floor for -1.500000 is -2.000000
The floor for -1.250000 is -2.000000
```

---

## fmod

---

**Description:** Calculates the remainder of  $x/y$  as a double precision value.

**Include:** `<math.h>`

**Prototype:** `double fmod(double x, double y);`

**Arguments:**

- `x` a double precision floating-point value
- `y` a double precision floating-point value

**Return Value:** Returns the remainder of  $x$  divided by  $y$ .

**Remarks:** If  $y = 0$ , a domain error occurs. If  $y$  is non-zero, the result will have the same sign as  $x$  and the magnitude of the result will be less than the magnitude of  $y$ .

**Example:**

```
#include <math.h> /* for fmod */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x,y,z;

    errno = 0;
    x = 7.0;
    y = 3.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n\n",
          x, y, z);
}
```

# 16-Bit Language Tools Libraries

---

---

## fmod (Continued)

---

```
    errno = 0;
    x = 7.0;
    y = 7.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n\n",
           x, y, z);

    errno = 0;
    x = -5.0;
    y = 3.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n\n",
           x, y, z);

    errno = 0;
    x = 5.0;
    y = -3.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n\n",
           x, y, z);

    errno = 0;
    x = -5.0;
    y = -5.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n\n",
           x, y, z);

    errno = 0;
    x = 7.0;
    y = 0.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n\n",
           x, y, z);
}
```

# Standard C Libraries - Math Functions

---

---

## fmod (Continued)

---

**Output:**

For fmod(7.000000, 3.000000) the remainder is  
1.000000

For fmod(7.000000, 7.000000) the remainder is  
0.000000

For fmod(-5.000000, 3.000000) the remainder is  
-2.000000

For fmod(5.000000, -3.000000) the remainder is  
2.000000

For fmod(-5.000000, -5.000000) the remainder is  
-0.000000

Error: domain error

For fmod(7.000000, 0.000000) the remainder is nan

---

## fmodf

---

**Description:** Calculates the remainder of  $x/y$  as a single precision value.

**Include:** <math.h>

**Prototype:** float fmodf(float x, float y);

**Arguments:** x a single precision floating-point value

y a single precision floating-point value

**Return Value:** Returns the remainder of  $x$  divided by  $y$ .

**Remarks:** If  $y = 0$ , a domain error occurs. If  $y$  is non-zero, the result will have the same sign as  $x$  and the magnitude of the result will be less than the magnitude of  $y$ .

**Example:**

```
#include <math.h> /* for fmodf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x,y,z;

    errno = 0;
    x = 7.0F;
    y = 3.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
           " %f\n\n", x, y, z);

    errno = 0;
    x = -5.0F;
    y = 3.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
           " %f\n\n", x, y, z);
```

## fmodf (Continued)

---

```
    errno = 0;
    x = 5.0F;
    y = -3.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
        " %f\n\n", x, y, z);

    errno = 0;
    x = 5.0F;
    y = -5.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
        " %f\n\n", x, y, z);

    errno = 0;
    x = 7.0F;
    y = 0.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
        " %f\n\n", x, y, z);

    errno = 0;
    x = 7.0F;
    y = 7.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
        " %f\n\n", x, y, z);
}
Output:
For fmodf (7.000000, 3.000000) the remainder is
1.000000

For fmodf (-5.000000, 3.000000) the remainder is
-2.000000

For fmodf (5.000000, -3.000000) the remainder is
2.000000

For fmodf (5.000000, -5.000000) the remainder is
0.000000

Error: domain error
For fmodf (7.000000, 0.000000) the remainder is nan

For fmodf (7.000000, 7.000000) the remainder is
0.000000
```

# Standard C Libraries - Math Functions

---

---

## frexp

---

**Description:** Gets the fraction and the exponent of a double precision floating-point number.

**Include:** <math.h>

**Prototype:** double frexp (double x, int \*exp);

**Arguments:** x floating-point value for which to return the fraction and exponent  
exp pointer to a stored integer exponent

**Return Value:** Returns the fraction, *exp* points to the exponent. If *x* is 0, the function returns 0 for both the fraction and exponent.

**Remarks:** The absolute value of the fraction is in the range of 1/2 (inclusive) to 1 (exclusive). No domain or range error will occur.

**Example:**

```
#include <math.h> /* for frexp */
#include <stdio.h> /* for printf */

int main(void)
{
    double x,y;
    int n;

    x = 50.0;
    y = frexp (x, &n);
    printf("For frexp of %f\n the fraction is %f\n ",
           x, y);
    printf(" and the exponent is %d\n\n", n);

    x = -2.5;
    y = frexp (x, &n);
    printf("For frexp of %f\n the fraction is %f\n ",
           x, y);
    printf(" and the exponent is %d\n\n", n);

    x = 0.0;
    y = frexp (x, &n);
    printf("For frexp of %f\n the fraction is %f\n ",
           x, y);
    printf(" and the exponent is %d\n\n", n);
}
```

**Output:**

```
For frexp of 50.000000
  the fraction is 0.781250
  and the exponent is 6

For frexp of -2.500000
  the fraction is -0.625000
  and the exponent is 2

For frexp of 0.000000
  the fraction is 0.000000
  and the exponent is 0
```

# 16-Bit Language Tools Libraries

---

---

## frexpf

---

**Description:** Gets the fraction and the exponent of a single precision floating-point number.

**Include:** <math.h>

**Prototype:** float frexpf (float x, int \*exp);

**Arguments:** x floating-point value for which to return the fraction and exponent  
exp pointer to a stored integer exponent

**Return Value:** Returns the fraction, *exp* points to the exponent. If *x* is 0, the function returns 0 for both the fraction and exponent.

**Remarks:** The absolute value of the fraction is in the range of 1/2 (inclusive) to 1 (exclusive). No domain or range error will occur.

**Example:**

```
#include <math.h> /* for frexpf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x,y;
    int n;

    x = 0.15F;
    y = frexpf (x, &n);
    printf("For frexpf of %f\n the fraction is %f\n ",
           x, y);
    printf(" and the exponent is %d\n\n", n);

    x = -2.5F;
    y = frexpf (x, &n);
    printf("For frexpf of %f\n the fraction is %f\n ",
           x, y);
    printf(" and the exponent is %d\n\n", n);

    x = 0.0F;
    y = frexpf (x, &n);
    printf("For frexpf of %f\n the fraction is %f\n ",
           x, y);
    printf(" and the exponent is %d\n\n", n);
}
```

**Output:**

```
For frexpf of 0.150000
  the fraction is 0.600000
  and the exponent is -2
```

```
For frexpf of -2.500000
  the fraction is -0.625000
  and the exponent is 2
```

```
For frexpf of 0.000000
  the fraction is 0.000000
  and the exponent is 0
```



# Standard C Libraries - Math Functions

---

## ldexp

---

**Description:** Calculates the result of a double precision floating-point number multiplied by an exponent of 2.

**Include:** <math.h>

**Prototype:** double ldexp(double x, int ex);

**Arguments:** x floating-point value

ex integer exponent

**Return Value:** Returns  $x * 2^{ex}$ . On an overflow, ldexp returns inf and on an underflow, ldexp returns 0.

**Remarks:** A range error will occur on overflow or underflow.

**Example:**

```
#include <math.h> /* for ldexp */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x,y;
    int n;

    errno = 0;
    x = -0.625;
    n = 2;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexp(%f, %d) = %f\n\n",
           x, n, y);

    errno = 0;
    x = 2.5;
    n = 3;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexp(%f, %d) = %f\n\n",
           x, n, y);

    errno = 0;
    x = 15.0;
    n = 10000;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexp(%f, %d) = %f\n\n",
           x, n, y);
}
```

---

## ldexp (Continued)

---

### Output:

For a number = -0.625000 and an exponent = 2  
ldexp(-0.625000, 2) = -2.500000

For a number = 2.500000 and an exponent = 3  
ldexp(2.500000, 3) = 20.000000

Error: range error

For a number = 15.000000 and an exponent = 10000  
ldexp(15.000000, 10000) = inf

---

## ldexpf

---

**Description:** Calculates the result of a single precision floating-point number multiplied by an exponent of 2.

**Include:** <math.h>

**Prototype:** float ldexpf(float x, int ex);

**Arguments:** x floating-point value  
ex integer exponent

**Return Value:** Returns  $x * 2^{ex}$ . On an overflow, ldexp returns inf and on an underflow, ldexp returns 0.

**Remarks:** A range error will occur on overflow or underflow.

**Example:**

```
#include <math.h> /* for ldexpf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x,y;
    int n;

    errno = 0;
    x = -0.625F;
    n = 2;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexpf(%f, %d) = %f\n\n",
           x, n, y);

    errno = 0;
    x = 2.5F;
    n = 3;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexpf(%f, %d) = %f\n\n",
           x, n, y);
}
```

## ldexpf (Continued)

---

```
    errno = 0;
    x = 15.0F;
    n = 10000;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf("  ldexpf(%f, %d) = %f\n\n",
           x, n, y);
}
```

### Output:

```
For a number = -0.625000 and an exponent = 2
  ldexpf(-0.625000, 2) = -2.500000
```

```
For a number = 2.500000 and an exponent = 3
  ldexpf(2.500000, 3) = 20.000000
```

```
Error: range error
```

```
For a number = 15.000000 and an exponent = 10000
  ldexpf(15.000000, 10000) = inf
```

# 16-Bit Language Tools Libraries

---

---

## log

---

**Description:** Calculates the natural logarithm of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double log(double x);

**Argument:** x any positive value for which to return the log

**Return Value:** Returns the natural logarithm of x. -inf is returned if x is 0 and NaN is returned if x is a negative number.

**Remarks:** A domain error occurs if  $x \leq 0$ .

**Example:**

```
#include <math.h> /* for log          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno       */

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = log (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0;
    y = log (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = -2.0;
    y = log (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n\n",
           x, y);
}
```

**Output:**

The natural logarithm of 2.000000 is 0.693147

The natural logarithm of 0.000000 is -inf

Error: domain error

The natural logarithm of -2.000000 is nan

# Standard C Libraries - Math Functions

---

---

## log10

---

**Description:** Calculates the base-10 logarithm of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double log10(double x);

**Argument:** x any double precision floating-point positive number

**Return Value:** Returns the base-10 logarithm of x. -inf is returned if x is 0 and NaN is returned if x is a negative number.

**Remarks:** A domain error occurs if  $x \leq 0$ .

**Example:**

```
#include <math.h> /* for log10          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno         */

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = log10 (x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0;
    y = log10 (x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = -2.0;
    y = log10 (x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n",
           x, y);
}
```

**Output:**

```
The base-10 logarithm of 2.000000 is 0.301030

The base-10 logarithm of 0.000000 is -inf

Error: domain error
The base-10 logarithm of -2.000000 is nan
```

# 16-Bit Language Tools Libraries

---

---

## log10f

---

**Description:** Calculates the base-10 logarithm of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float log10f(float x);

**Argument:** x any single precision floating-point positive number

**Return Value:** Returns the base-10 logarithm of x. -inf is returned if x is 0 and NaN is returned if x is a negative number.

**Remarks:** A domain error occurs if  $x \leq 0$ .

**Example:**

```
#include <math.h> /* for log10f      */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno      */

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = -2.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n",
           x, y);
}
```

**Output:**

The base-10 logarithm of 2.000000 is 0.301030

Error: domain error  
The base-10 logarithm of 0.000000 is -inf

Error: domain error  
The base-10 logarithm of -2.000000 is nan

# Standard C Libraries - Math Functions

---

---

## logf

---

**Description:** Calculates the natural logarithm of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float logf(float x);

**Argument:** x any positive value for which to return the log

**Return Value:** Returns the natural logarithm of x. -inf is returned if x is 0 and NaN is returned if x is a negative number.

**Remarks:** A domain error occurs if  $x \leq 0$ .

**Example:**

```
#include <math.h> /* for logf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = logf (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0F;
    y = logf (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n\n",
           x, y);

    errno = 0;
    x = -2.0F;
    y = logf (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n\n",
           x, y);
}
```

**Output:**

```
The natural logarithm of 2.000000 is 0.693147

The natural logarithm of 0.000000 is -inf

Error: domain error
The natural logarithm of -2.000000 is nan
```

# 16-Bit Language Tools Libraries

---

---

---

## modf

---

**Description:** Splits a double precision floating-point value into fractional and integer parts.

**Include:** <math.h>

**Prototype:** double modf(double *x*, double \**pint*);

**Arguments:** *x* double precision floating-point value  
*pint* pointer to a stored the integer part

**Return Value:** Returns the signed fractional part and *pint* points to the integer part.

**Remarks:** The absolute value of the fractional part is in the range of 0 (inclusive) to 1 (exclusive). No domain or range error will occur.

**Example:**

```
#include <math.h> /* for modf */
#include <stdio.h> /* for printf */

int main(void)
{
    double x,y,n;

    x = 0.707;
    y = modf (x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf(" and the integer is %0.f\n\n", n);

    x = -15.2121;
    y = modf (x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf(" and the integer is %0.f\n\n", n);
}
```

**Output:**

```
For 0.707000 the fraction is 0.707000
and the integer is 0
```

```
For -15.212100 the fraction is -0.212100
and the integer is -15
```



# Standard C Libraries - Math Functions

---

## modff

---

**Description:** Splits a single precision floating-point value into fractional and integer parts.

**Include:** <math.h>

**Prototype:** float modff(float x, float \*pint);

**Arguments:** x single precision floating-point value  
pint pointer to stored integer part

**Return Value:** Returns the signed fractional part and *pint* points to the integer part.

**Remarks:** The absolute value of the fractional part is in the range of 0 (inclusive) to 1 (exclusive). No domain or range error will occur.

**Example:**

```
#include <math.h> /* for modff */
#include <stdio.h> /* for printf */

int main(void)
{
    float x,y,n;

    x = 0.707F;
    y = modff (x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf(" and the integer is %0.f\n\n", n);

    x = -15.2121F;
    y = modff (x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf(" and the integer is %0.f\n\n", n);
}
```

**Output:**

```
For 0.707000 the fraction is 0.707000
and the integer is 0
```

```
For -15.212100 the fraction is -0.212100
and the integer is -15
```

# 16-Bit Language Tools Libraries

---

---

## pow

---

**Description:** Calculates  $x$  raised to the power  $y$ .

**Include:** `<math.h>`

**Prototype:** `double pow(double x, double y);`

**Arguments:**  $x$  the base  
 $y$  the exponent

**Return Value:** Returns  $x$  raised to the power  $y$  ( $x^y$ ).

**Remarks:** If  $y$  is 0, `pow` returns 1. If  $x$  is 0.0 and  $y$  is less than 0, `pow` returns `inf` and a domain error occurs. If the result overflows or underflows, a range error occurs.

**Example:**

```
#include <math.h> /* for pow          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */

int main(void)
{
    double x,y,z;

    errno = 0;
    x = -2.0;
    y = 3.0;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n ", x, y, z);

    errno = 0;
    x = 3.0;
    y = -0.5;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n ", x, y, z);

    errno = 0;
    x = 4.0;
    y = 0.0;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n ", x, y, z);

    errno = 0;
    x = 0.0;
    y = -3.0;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n ", x, y, z);
}
```

# Standard C Libraries - Math Functions

---

---

## pow (Continued)

---

**Output:**

-2.000000 raised to 3.000000 is -8.000000

3.000000 raised to -0.500000 is 0.577350

4.000000 raised to 0.000000 is 1.000000

Error: domain error

0.000000 raised to -3.000000 is inf

---

## powf

---

**Description:** Calculates  $x$  raised to the power  $y$ .

**Include:** <math.h>

**Prototype:** float powf(float  $x$ , float  $y$ );

**Arguments:**  
 $x$  base  
 $y$  exponent

**Return Value:** Returns  $x$  raised to the power  $y$  ( $x^y$ ).

**Remarks:** If  $y$  is 0, powf returns 1. If  $x$  is 0.0 and  $y$  is less than 0, powf returns inf and a domain error occurs. If the result overflows or underflows, a range error occurs.

**Example:**

```
#include <math.h> /* for powf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x,y,z;

    errno = 0;
    x = -2.0F;
    y = 3.0F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n ", x, y, z);

    errno = 0;
    x = 3.0F;
    y = -0.5F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n ", x, y, z);

    errno = 0;
    x = 0.0F;
    y = -3.0F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n ", x, y, z);
}
```

---

## powf (Continued)

---

**Output:**

-2.000000 raised to 3.000000 is -8.000000

3.000000 raised to -0.500000 is 0.577350

Error: domain error

0.000000 raised to -3.000000 is inf

---

## sin

---

**Description:** Calculates the trigonometric sine function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double sin (double x);

**Argument:** x value for which to return the sine

**Return Value:** Returns the sine of x in radians in the ranges of -1 to 1 inclusive.

**Remarks:** A domain error will occur if x is a NaN or infinity.

**Example:**

```
#include <math.h> /* for sin */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */
```

```
int main(void)
{
    double x, y;

    errno = 0;
    x = -1.0;
    y = sin (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0;
    y = sin (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n\n", x, y);
}
```

**Output:**

The sine of -1.000000 is -0.841471

The sine of 0.000000 is 0.000000

---

---

# Standard C Libraries - Math Functions

---

## **sinf**

---

**Description:** Calculates the trigonometric sine function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float sinf (float x);

**Argument:** x value for which to return the sine

**Return Value:** Returns the sin of x in radians in the ranges of -1 to 1 inclusive.

**Remarks:** A domain error will occur if x is a NaN or infinity.

**Example:**

```
#include <math.h> /* for sinf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = sinf (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0F;
    y = sinf (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n\n", x, y);
}
```

**Output:**

The sine of -1.000000 is -0.841471

The sine of 0.000000 is 0.000000

## sinh

---

**Description:** Calculates the hyperbolic sine function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double sinh (double x);

**Argument:** x value for which to return the hyperbolic sine

**Return Value:** Returns the hyperbolic sine of x

**Remarks:** A range error will occur if the magnitude of x is too large.

**Example:**

```
#include <math.h> /* for sinh          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.5;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 720.0;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n",
           x, y);
}
```

**Output:**

The hyperbolic sine of -1.500000 is -2.129279

The hyperbolic sine of 0.000000 is 0.000000

Error: range error

The hyperbolic sine of 720.000000 is inf

## sinhf

---

**Description:** Calculates the hyperbolic sine function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float sinhf (float x);

**Argument:** x value for which to return the hyperbolic sine

**Return Value:** Returns the hyperbolic sine of x .

**Remarks:** A range error will occur if the magnitude of x is too large.

```
Example: #include <math.h> /* for sinhf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = sinh(x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0F;
    y = sinh(x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n",
           x, y);
}
```

**Output:**

The hyperbolic sine of -1.000000 is -1.175201

The hyperbolic sine of 0.000000 is 0.000000

# 16-Bit Language Tools Libraries

---

---

---

## sqrt

---

**Description:** Calculates the square root of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double sqrt(double x);

**Argument:** x a non-negative floating-point value

**Return Value:** Returns the non-negative square root of x.

**Remarks:** If x is negative, a domain error occurs.

**Example:**

```
#include <math.h> /* for sqrt          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno        */

int main(void)
{
    double x, y;

    errno = 0;
    x = 0.0;
    y = sqrt (x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);

    errno = 0;
    x = 9.5;
    y = sqrt (x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);

    errno = 0;
    x = -25.0;
    y = sqrt (x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);
}
```

**Output:**

The square root of 0.000000 is 0.000000

The square root of 9.500000 is 3.082207

Error: domain error

The square root of -25.000000 is nan



## sqrtf

---

**Description:** Calculates the square root of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float sqrtf(float x);

**Argument:** x non-negative floating-point value

**Return Value:** Returns the non-negative square root of x.

**Remarks:** If x is negative, a domain error occurs.

**Example:**

```
#include <math.h> /* for sqrtf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x;

    errno = 0;
    x = sqrtf (0.0F);
    if (errno)
        perror("Error");
    printf("The square root of 0.0F is %f\n\n", x);

    errno = 0;
    x = sqrtf (9.5F);
    if (errno)
        perror("Error");
    printf("The square root of 9.5F is %f\n\n", x);

    errno = 0;
    x = sqrtf (-25.0F);
    if (errno)
        perror("Error");
    printf("The square root of -25F is %f\n", x);
}
```

**Output:**

The square root of 0.0F is 0.000000

The square root of 9.5F is 3.082207

Error: domain error  
The square root of -25F is nan

# 16-Bit Language Tools Libraries

---

---

---

## tan

---

**Description:** Calculates the trigonometric tangent function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double tan (double x);

**Argument:** x value for which to return the tangent

**Return Value:** Returns the tangent of x in radians.

**Remarks:** A domain error will occur if x is a NaN or infinity.

**Example:**

```
#include <math.h> /* for tan */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */
```

```
int main(void)
{
    double x, y;

    errno = 0;
    x = -1.0;
    y = tan (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0;
    y = tan (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n\n", x, y);
}
```

**Output:**

The tangent of -1.000000 is -1.557408

The tangent of 0.000000 is 0.000000

---

## tanf

---

**Description:** Calculates the trigonometric tangent function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float tanf (float x);

**Argument:** x value for which to return the tangent

**Return Value:** Returns the tangent of x.

**Remarks:** A domain error will occur if x is a NaN or infinity.

**Example:**

```
#include <math.h> /* for tanf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */
```

```
int main(void)
{
    float x, y;
```

## **tanf (Continued)**

---

```
    errno = 0;
    x = -1.0F;
    y = tanf (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0F;
    y = tanf (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n", x, y);
}
```

### **Output:**

The tangent of -1.000000 is -1.557408

The tangent of 0.000000 is 0.000000

---

## **tanh**

---

**Description:** Calculates the hyperbolic tangent function of a double precision floating-point value.

**Include:** <math.h>

**Prototype:** double tanh (double x);

**Argument:** x value for which to return the hyperbolic tangent

**Return Value:** Returns the hyperbolic tangent of x in the ranges of -1 to 1 inclusive.

**Remarks:** No domain or range error will occur.

**Example:**

```
#include <math.h> /* for tanh */
#include <stdio.h> /* for printf */

int main(void)
{
    double x, y;

    x = -1.0;
    y = tanh (x);
    printf("The hyperbolic tangent of %f is %f\n\n",
           x, y);

    x = 2.0;
    y = tanh (x);
    printf("The hyperbolic tangent of %f is %f\n\n",
           x, y);
}
```

### **Output:**

The hyperbolic tangent of -1.000000 is -0.761594

The hyperbolic tangent of 2.000000 is 0.964028

## **tanhf**

---

**Description:** Calculates the hyperbolic tangent function of a single precision floating-point value.

**Include:** <math.h>

**Prototype:** float tanhf (float x);

**Argument:** x value for which to return the hyperbolic tangent

**Return Value:** Returns the hyperbolic tangent of x in the ranges of -1 to 1 inclusive.

**Remarks:** No domain or range error will occur.

**Example:**

```
#include <math.h> /* for tanhf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x, y;

    x = -1.0F;
    y = tanhf (x);
    printf("The hyperbolic tangent of %f is %f\n\n",
           x, y);

    x = 0.0F;
    y = tanhf (x);
    printf("The hyperbolic tangent of %f is %f\n\n",
           x, y);
}
```

**Output:**

The hyperbolic tangent of -1.000000 is -0.761594

The hyperbolic tangent of 0.000000 is 0.000000

---

---

**Chapter 4. Standard C Libraries - Support Functions**

---

---

**4.1 INTRODUCTION**

This chapter describes support functions that either must be customized for correct operation of the Standard C Library in your target environment or are already customized for a Microchip target environment. The default behavior section describes what the function does as it is distributed. The description and remarks describe what it typically should do.

The corresponding object modules are distributed in the `libpic30-omf.a` archive, and the source code (for the compiler) is available in the `src\pic30` folder.

**4.1.1 Assembly Code Applications**

A free version of this library and its associated header file is available from the Microchip web site. Source code is included.

**4.1.2 C Code Applications**

The MPLAB XC16 C Compiler install directory (**Section 1.1.1 “Compiler Installation Locations”**) contains the following subdirectories with library-related files:

- `lib` – standard C library files
- `src\pic30` – source code for library functions, batch file to rebuild the library
- `support\h` – header files for libraries

In addition, there is a file, `ResourceGraphs.pdf`, which contains diagrams of resources used by each function, located in `lib`.

**4.1.3 Chapter Organization**

This chapter is organized as follows:

- Using the Support Functions
- Standard C Library Helper Functions
- Standard C Library Functions That Require Modification
- Functions/Constants to Support A Simulated UART
- Functions for Erasing and Writing EEDATA Memory
- Functions for Erasing and Writing Flash Memory
- Functions for Specialized Copying and Initialization

# 16-Bit Language Tools Libraries

---

## 4.2 USING THE SUPPORT FUNCTIONS

Building an application which utilizes the support functions requires two types of files: header files and library files.

- Rebuilding the `libpic30-omf.a` Library

### 4.2.1 Header Files

All standard C library entities are declared or defined in one or more standard headers (See list in **Section 4.1.3 “Chapter Organization”**.) To make use of a library entity in a program, write an include directive that names the relevant standard header.

The contents of a standard header is included by naming it in an include directive, as in:

```
#include <libpic30.h> /* include dsPIC30F facilities */
```

The standard headers can be included in any order. Do not include a standard header within a declaration. Do not define macros that have the same names as keywords before including a standard header.

A standard header never includes another standard header.

### 4.2.2 Library Files

The archived library files contain all the individual object files for each library function.

When linking an application, the library file must be provided as an input to the linker (using the `--library` or `-l` linker option) such that the functions used by the application may be linked into the application.

A typical C application will require three library files: `libc-omf.a`, `libm-omf.a` and `libpic30-omf.a`. (See **Section 1.2 “OMF-Specific Libraries/Start-up Modules”** for more on OMF-specific libraries.) These libraries will be included automatically if linking is performed using the compiler.

<p><b>Note:</b> Some standard library functions require a heap. These include the standard I/O functions that open files and the memory allocation functions. See the “MPLAB® XC16 Assembler, Linker and Utilities User’s Guide” (DS50002106) and “MPLAB® XC16 C Compiler User’s Guide” (DS50002071) for more information on the heap.</p>
--

### 4.2.3 Rebuilding the `libpic30-omf.a` Library

By default, the helper functions listed in this chapter were written to work with the `sim30` simulator. The header file `simio.h` defines the interface between the library and the simulator. It is provided so you can rebuild the libraries and continue to use the simulator. However, your application should not use this interface since the simulator will not be available to an embedded application.

The helper functions must be modified and rebuilt for your target application. The `libpic30-omf.a` library can be rebuilt with the batch file named `makeLib.bat` which has been provided with the sources in `src\pic30`. Execute the batch file from a command window. Be sure you are in the `src\pic30` directory. Then, copy the newly compiled file (`libpic30-omf.a`) into the `lib` directory.

# Standard C Libraries - Support Functions

---

## 4.3 STANDARD C LIBRARY HELPER FUNCTIONS

These functions are called by other functions in the standard C library and must be modified for the target application. The corresponding object modules are distributed in the `libpic30-omf.a` archive, and the source code (for the compiler) is available in the `src\pic30` folder.

---

### **`_dump_heap_info`**

---

**Description:** Displays the current use/free state of the heap.

**Include:** None

**Prototype:** `void _dump_heap_info();`

**Argument:** None

**Remarks:** This helper function can be used to profile memory allocation in the heap. It is useful for memory allocation functions.

**Default Behavior:** This function prints information about the heap to `stderr`. Information includes the memory region where heap is located, the different sections of the heap, whether a section is currently busy or free, and a summary of how much heap is free.

For example, below, a heap of 1000 has been allocated. The heap begins and ends at `d0c` and `10f4`, respectively. The `malloc` function has been executed 3 different times, resulting in 3 different sections of BUSY memory and 1 FREE memory section of heap that has not been used. If the free function were to be used on any of these 3 allocated sections, the status would turn from BUSY to FREE as well.

```
*** Unused Heap status:
***  start: 0x00000d0c   end: 0x000010f4
***  0d0c      8  BUSY
***  0d14     16  BUSY
***  0d24     20  BUSY
***  0d38    956  FREE
***  44 used, 956 free, 010f0 end
```

**File:** `_dump_heap_info.c`

---

### **`_exit`**

---

**Description:** Terminate program execution.

**Include:** None

**Prototype:** `void _exit (int status);`

**Argument:** `status` exit status

**Remarks:** This is a helper function called by the `exit()` Standard C Library function.

**Default Behavior:** As distributed, this function flushes `stdout` and terminates. The parameter `status` is the same as that passed to the `exit()` standard C library function.

**File:** `_exit.c`

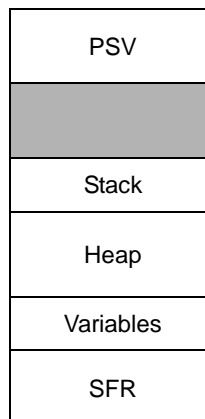
# 16-Bit Language Tools Libraries

---

## brk

---

<b>Description:</b>	Set the end of the process's data space.
<b>Include:</b>	None
<b>Prototype:</b>	<code>int brk(void *endds);</code>
<b>Argument:</b>	<code>endds</code> pointer to the end of the data segment
<b>Return Value:</b>	Returns '0' if successful; otherwise, returns '-1'.
<b>Remarks:</b>	<p><code>brk()</code> is used to dynamically change the amount of space allocated for the calling process's data segment. The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.</p> <p>Newly allocated space is uninitialized.</p> <p>This helper function is used by the Standard C Library function <code>malloc()</code>.</p>
<b>Default Behavior:</b>	<p>If the argument <code>endds</code> is zero, the function sets the global variable <code>__curbrk</code> to the address of the start of the heap and returns zero.</p> <p>If the argument <code>endds</code> is non-zero and has a value less than the address of the end of the heap, the function sets the global variable <code>__curbrk</code> to the value of <code>endds</code> and returns zero.</p> <p>Otherwise, the global variable <code>__curbrk</code> is unchanged and the function returns -1.</p> <p>The argument <code>endds</code> must be within the heap range (see data space memory map below).</p>



Notice that, since the stack is located immediately above the heap, using `brk()` or `sbrk()` has little effect on the size of the dynamic memory pool. The `brk()` and `sbrk()` functions are primarily intended for use in run-time environments where the stack grows downward and the heap grows upward.

The linker allocates a block of memory for the heap if the `-Wl,--heap=n` option is specified, where `n` is the desired heap size in characters. The starting and ending addresses of the heap are reported in variables: `_heap` and `_eheap`, respectively.

For the 16-bit compiler, using the linker's heap size option is the standard way of controlling heap size, rather than relying on `brk()` and `sbrk()`.

**File:** `brk.c`



# Standard C Libraries - Support Functions

---

---

---

## close

---

<b>Description:</b>	Close a file.
<b>Include:</b>	None
<b>Prototype:</b>	<code>int close(int handle);</code>
<b>Argument:</b>	<i>handle</i> handle referring to an opened file
<b>Return Value:</b>	Returns '0' if the file is successfully closed. A return value of '-1' indicates an error.
<b>Remarks:</b>	This helper function is called by the <code>fclose()</code> Standard C Library function.
<b>Default Behavior:</b>	As distributed, this function passes the file handle to the simulator, which issues a close in the host file system.
<b>File:</b>	<code>close.c</code>

---

## lseek

---

<b>Description:</b>	Move a file pointer to a specified location.
<b>Include:</b>	None
<b>Prototype:</b>	<code>long lseek(int handle, long offset, int origin);</code>
<b>Argument:</b>	<i>handle</i> refers to an opened file <i>offset</i> the number of characters from the origin <i>origin</i> the position from which to start the seek. <i>origin</i> may be one of the following values (as defined in <code>stdio.h</code> ): SEEK_SET – Beginning of file. SEEK_CUR – Current position of file pointer. SEEK_END – End-of-file.
<b>Return Value:</b>	Returns the offset, in characters, of the new position from the beginning of the file. A return value of '-1L' indicates an error.
<b>Remarks:</b>	This helper function is called by the Standard C Library functions <code>fgetpos()</code> , <code>ftell()</code> , <code>fseek()</code> , <code>fsetpos</code> and <code>rewind()</code> .
<b>Default Behavior:</b>	As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.
<b>File:</b>	<code>lseek.c</code>

---

## open

---

<b>Description:</b>	Open a file.
<b>Include:</b>	None
<b>Prototype:</b>	<code>int open(const char *name, int access, int mode);</code>
<b>Argument:</b>	<i>name</i> name of the file to be opened <i>access</i> access method to open file <i>mode</i> type of access permitted
<b>Return Value:</b>	If successful, the function returns a file handle: a small positive integer. This handle is then used on subsequent low-level file I/O operations. A return value of '-1' indicates an error.

# 16-Bit Language Tools Libraries

---

---

---

## open (Continued)

---

<b>Remarks:</b>	The access flag is a union of one of the following access methods and zero or more access qualifiers: 0 – Open a file for reading. 1 – Open a file for writing. 2 – Open a file for both reading and writing. The following access qualifiers must be supported: 0x0008 – Move file pointer to end-of-file before every write operation. 0x0100 – Create and open a new file for writing. 0x0200 – Open the file and truncate it to zero length. 0x4000 – Open the file in text (translated) mode. 0x8000 – Open the file in binary (untranslated) mode. The mode parameter may be one of the following: 0x0100 – Reading only permitted. 0x0080 – Writing permitted (implies reading permitted). This helper function is called by the Standard C Library functions <code>fopen()</code> and <code>freopen()</code> .
<b>Default Behavior:</b>	As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system. If the host system returns a value of '-1', the global variable <code>errno</code> is set to the value of the symbolic constant, <code>EFOPEN</code> , defined in <code>&lt;errno.h&gt;</code> .
<b>File:</b>	<code>open.c</code>

---

## read

---

<b>Description:</b>	Read data from a file.						
<b>Include:</b>	None						
<b>Prototype:</b>	<pre>int read(int handle, void *buffer,          unsigned int len);</pre>						
<b>Argument:</b>	<table><tr><td><i>handle</i></td><td>handle referring to an opened file</td></tr><tr><td><i>buffer</i></td><td>points to the storage location for read data</td></tr><tr><td><i>len</i></td><td>the maximum number of characters to read</td></tr></table>	<i>handle</i>	handle referring to an opened file	<i>buffer</i>	points to the storage location for read data	<i>len</i>	the maximum number of characters to read
<i>handle</i>	handle referring to an opened file						
<i>buffer</i>	points to the storage location for read data						
<i>len</i>	the maximum number of characters to read						
<b>Return Value:</b>	Returns the number of characters read, which may be less than <i>len</i> if there are fewer than <i>len</i> characters left in the file or if the file was opened in text mode, in which case, each carriage return-linefeed (CR-LF) pair is replaced with a single linefeed character. Only the single linefeed character is counted in the return value. The replacement does not affect the file pointer. If the function tries to read at end-of-file, it returns '0'. If the handle is invalid, or the file is not open for reading or the file is locked, the function returns '-1'.						
<b>Remarks:</b>	This helper function is called by the Standard C Library functions <code>fgetc()</code> , <code>fgets()</code> , <code>fread()</code> and <code>gets()</code> .						
<b>Default Behavior:</b>	As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.						
<b>File:</b>	<code>read.c</code>						

---

## sbrk

---

<b>Description:</b>	Extend the process' data space by a given increment.		
<b>Include:</b>	None		
<b>Prototype:</b>	<pre>void * sbrk(int incr);</pre>		
<b>Argument:</b>	<table><tr><td><i>incr</i></td><td>number of characters to increment/decrement</td></tr></table>	<i>incr</i>	number of characters to increment/decrement
<i>incr</i>	number of characters to increment/decrement		

# Standard C Libraries - Support Functions

---

---

## sbrk (Continued)

---

<b>Return Value:</b>	Return the start of the new space allocated or '-1' for errors.
<b>Remarks:</b>	<p><code>sbrk()</code> adds <i>incr</i> characters to the break value and changes the allocated space accordingly. <i>incr</i> can be negative, in which case the amount of allocated space is decreased.</p> <p><code>sbrk()</code> is used to dynamically change the amount of space allocated for the calling process's data segment. The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.</p> <p>This is a helper function called by the Standard C Library function <code>malloc()</code>.</p>
<b>Default Behavior:</b>	<p>If the global variable <code>__curbrk</code> is zero, the function calls <code>brk()</code> to initialize the break value. If <code>brk()</code> returns -1, so does this function.</p> <p>If the <i>incr</i> is zero, the current value of the global variable <code>__curbrk</code> is returned.</p> <p>If the <i>incr</i> is non-zero, the function checks that the address (<code>__curbrk + incr</code>) is less than the end address of the heap. If it is less, the global variable <code>__curbrk</code> is updated to that value and the function returns the unsigned value of <code>__curbrk</code>.</p> <p>Otherwise, the function returns -1.</p> <p>See the description of <code>brk()</code>.</p>
<b>File:</b>	<code>sbrk.c</code>

---

## write

---

<b>Description:</b>	Write data to a file.
<b>Include:</b>	None
<b>Prototype:</b>	<pre>int __attribute__((__section__(".libc.write"))) write(int handle, void *buffer, unsigned int count);</pre>
<b>Argument:</b>	<p><i>handle</i> refers to an opened file</p> <p><i>buffer</i> points to the storage location of data to be written</p> <p><i>count</i> the number of characters to write.</p>
<b>Return Value:</b>	If successful, <code>write</code> returns the number of characters actually written. A return value of '-1' indicates an error.
<b>Remarks:</b>	<p>If the actual space remaining on the disk is less than the size of the buffer, the function trying to write to the disk write fails and does not flush any of the buffer's contents to the disk. If the file is opened in text mode, each linefeed character is replaced with a carriage return – linefeed pair in the output. The replacement does not affect the return value.</p> <p>This is a helper function called by the Standard C Library function <code>fflush()</code>.</p>
<b>Default Behavior:</b>	As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.
<b>File:</b>	<code>write.c</code>

# 16-Bit Language Tools Libraries

---

## 4.4 STANDARD C LIBRARY FUNCTIONS THAT REQUIRE MODIFICATION

Although these functions are part of the Standard C Library, the object modules are distributed in the `libpic30-omf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder. These modules are not distributed as part of `libc-omf.a`.

---

### getenv

---

**Description:** Get a value for an environment variable.

**Include:** `<stdlib.h>`

**Prototype:** `char *getenv(const char *s);`

**Argument:** `s` name of environment variable

**Return Value:** Returns a pointer to the value of the environment variable if successful; otherwise, returns a null pointer.

**Default Behavior:** As distributed, this function returns a null pointer. There is no support for environment variables.

**File:** `getenv.c`

---

### remove

---

**Description:** Remove a file.

**Include:** `<stdio.h>`

**Prototype:** `int remove(const char *filename);`

**Argument:** `filename` file to be removed

**Return Value:** Returns '0' if successful, '-1' if unsuccessful.

**Default Behavior:** As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.

**File:** `remove.c`

---

### rename

---

**Description:** Rename a file or directory.

**Include:** `<stdio.h>`

**Prototype:** `int rename(const char *oldname, const char *newname);`

**Argument:** `oldname` pointer to the old name  
`newname` pointer to the new name

**Return Value:** Returns '0' if it is successful. On an error, the function returns a non-zero value.

**Default Behavior:** As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.

**File:** `rename.c`

---

### system

---

**Description:** Execute a command.

**Include:** `<stdlib.h>`

**Prototype:** `int system(const char *s);`

**Argument:** `s` command to be executed

---

# Standard C Libraries - Support Functions

---

---

---

## system (Continued)

---

**Default Behavior:** As distributed, this function acts as a stub or placeholder for your function. If *s* is not NULL, an error message is written to *stdout* and the program will reset; otherwise, a value of -1 is returned.

**File:** `system.c`

---

## time

---

**Description:** Get the system time.

**Include:** `<time.h>`

**Prototype:** `time_t time(time_t *timer);`

**Argument:** *timer* points to a storage location for time

**Return Value:** Returns the elapse time in seconds. There is no error return.

**Default Behavior:** As distributed, if Timer2 is not enabled, it is enabled in 32-bit mode. The return value is the current value of the 32-bit Timer2 register. Except in very rare cases, this return value is not the elapsed time in seconds.

**File:** `time.c`

---

## 4.5 FUNCTIONS/CONSTANTS TO SUPPORT A SIMULATED UART

These functions and constants support UART functionality in the MPLAB SIM simulator.

---

### \_\_attach\_input\_file

---

**Description:** Attach a hosted file to the standard input stream.

**Include:** `<libpic30.h>`

**Prototype:** `int __attach_input_file(const char *p);`

**Argument:** *p* pointer to file

**Remarks:** This function differs from the MPLAB IDE or MPLAB X IDE mechanism of providing an input file because it provides “on-demand” access to the file. That is, data will only be read from the file upon request and the asynchronous nature of the UART is not simulated. This function may be called more than once; any opened file will be closed. It is only appropriate to call this function in a simulated environment.

**Default Behavior:** Allows the programmer to attach a hosted file to the standard input stream, *stdin*. The function will return 0 to indicate failure. If the file cannot be opened for whatever reason, standard in will remain connected (or be re-connected) to the simulated UART.

**File:** `attach.c`

---

### \_\_close\_input\_file

---

**Description:** Close a previously attached file.

**Include:** `<libpic30.h>`

**Prototype:** `void __close_input_file(void);`

**Argument:** None

**Remarks:** None.

**Default Behavior:** This function will close a previously attached file and re-attach *stdin* to the simulated UART. This should occur before a Reset to ensure that the file can be re-opened.

**File:** `close.c`

---

# 16-Bit Language Tools Libraries

---

---

---

## \_\_delay32

---

**Description:** Produce a delay of a specified number of clock cycles.

**Include:** <libpic30.h>

**Prototype:** void \_\_delay32(unsigned long *cycles*);

**Argument:** *cycles* number of cycles to delay

**Remarks:** None.

**Default Behavior:** This function will effect a delay of the requested number of cycles. The minimum supported delay is 12 cycles (an argument of less than or equal to 12 will result in 12 cycles). The delay includes the `call` and `return` statements, but not any cycles required to set up the argument (typically this would be two for a literal value).

**File:** delay32.s

---

---

## \_\_delay\_ms

---

**Description:** Produce a delay of a specified number of milliseconds (ms).

**Include:** <libpic30.h>

**Prototype:** void \_\_delay\_ms(unsigned int *time*);

**Argument:** *time* number of ms to delay

**Remarks:** This function is implemented as a macro.

**Default Behavior:** This function relies on a user-supplied definition of `FCY` to represent the instruction clock frequency. `FCY` must be defined before header file `libpic30.h` is included. The specified delay is converted to the equivalent number of instruction cycles and passed to `__delay32()`. If `FCY` is not defined, then `__delay_ms()` is declared external, causing the link to fail unless the user provides a function with that name.

**File:** delay32.s

---

---

## \_\_delay\_us

---

**Description:** Produce a delay of a specified number of microseconds (us).

**Include:** <libpic30.h>

**Prototype:** void \_\_delay\_us(unsigned int *time*);

**Argument:** *time* number of us to delay

**Remarks:** This function is implemented as a macro. The minimum delay is equivalent to 12 instruction cycles.

**Default Behavior:** This function relies on a user-supplied definition of `FCY` to represent the instruction clock frequency. `FCY` must be defined before header file, `libpic30.h`, is included. The specified delay is converted to the equivalent number of instruction cycles and passed to `__delay32()`. If `FCY` is not defined, then `__delay_us()` is declared external, causing the link to fail unless the user provides a function with that name.

**File:** delay32.s

---

---

## \_\_C30\_UART

---

**Description:** Constant that defines the default UART.

**Include:** N/A

**Prototype:** int \_\_C30\_UART;

**Argument:** N/A

---

# Standard C Libraries - Support Functions

---

---

## \_\_C30\_UART (Continued)

---

<b>Return Value:</b>	N/A
<b>Remarks:</b>	Defines the default UART that <code>read()</code> and <code>write()</code> will use for <code>stdin</code> (unless a file has been attached) and <code>stdout</code> .
<b>Default Behavior:</b>	By default, or with a value of 1, UART 1 will be used. Otherwise, UART 2 will be used. <code>read()</code> and <code>write()</code> are the eventual destinations of the C standard I/O functions.
<b>File:</b>	N/A

### Examples of Use

#### EXAMPLE 4-1: UART1 I/O

```
#include <libpic30.h>          /* a new header file for
                               these defintions */
#include <stdio.h>

void main() {
    if (__attach_input_file("foo.txt")) {
        while (!feof(stdin)) {
            putchar(getchar());
        }
        __close_input_file();
    }
}
```

#### EXAMPLE 4-2: USING UART2

```
/* This program flashes a light and transmits a lot of messages at
   9600 8n1 through uart 2 using the default stdio provided
   by the 16-bit compiler. This is for a dsPIC33F DSC
   on an Explorer 16(tm) board (and isn't very pretty) */

#include <libpic30.h>          /* a new header file for these
                               defintions */
#include <stdio.h>

#ifdef __dsPIC33F__
#error this is a 33F demo for the explorer 16(tm) board
#endif
#include <p33Fxxxx.h>

_FOSCSEL(FNOSC_PRI );
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_XT);
_FWDT(FWDTEN_OFF);

main() {
    ODCA = 0;
    TRISAbits.TRISA6 = 0;
    __C30_UART=2;
    U2BRG = 38;
    U2MODEbits.UARTEN = 1;
    while (1) {
        __builtin_btg(&LATA,6);
        printf("Hello world %d\n",U2BRG);
    }
}
```

# 16-Bit Language Tools Libraries

---

## EXAMPLE 4-3: MILLISECOND DELAY

```
#define FCY 1000000UL
#include <libpic30.h>

int main()
{
    /* at 1MHz, these are equivalent */
    __delay_ms(1);
    __delay32(1000);
}
```

## EXAMPLE 4-4: MICROSECOND DELAY

```
#define FCY 1000000UL
#include <libpic30.h>

int main()
{
    /* at 1MHz, these are equivalent */
    __delay_us(1000);
    __delay32(1000);
}
```

## 4.6 FUNCTIONS FOR ERASING AND WRITING EEDATA MEMORY

These functions support the erasing and writing of EEDATA memory for devices that have this type of memory.

---

### **\_erase\_eedata**

---

<b>Description:</b>	Erase EEDATA memory on dsPIC30F and PIC24FXXKA devices.
<b>Include:</b>	<libpic30.h>
<b>Prototype:</b>	void _erase_eedata(_prog_addressT dst, int len);
<b>Argument:</b>	<i>dst</i> destination memory address <i>len</i> dsPIC30F: length may be <code>_EE_WORD</code> or <code>_EE_ROW</code> (bytes) PIC24FxxKA: length may be <code>_EE_WORD</code> , <code>_EE_4WORDS</code> or <code>_EE_8WORDS</code> (bytes)
<b>Return Value:</b>	None.
<b>Remarks:</b>	None.
<b>Default Behavior:</b>	Erase EEDATA memory as specified by parameters.
<b>File:</b>	eedata_helper.c

---

### **\_erase\_eedata\_all**

---

<b>Description:</b>	Erase the entire range of EEDATA memory on dsPIC30F and PIC24FXXKA devices.
<b>Include:</b>	<libpic30.h>
<b>Prototype:</b>	void _erase_eedata_all(void);
<b>Argument:</b>	None
<b>Return Value:</b>	None.
<b>Remarks:</b>	None.
<b>Default Behavior:</b>	Erase all EEDATA memory for the selected device.
<b>File:</b>	eedata_helper.c



# Standard C Libraries - Support Functions

---

---

---

## **\_wait\_eedata**

---

**Description:** Wait for an erase or write operation to complete on dsPIC30F and PIC24FXXKA devices.

**Include:** <libpic30.h>

**Prototype:** void \_wait\_eedata(void);

**Argument:** None

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Wait for an erase or write operation to complete.

**File:** eedata\_helper.c

---

---

## **\_write\_eedata\_word**

---

**Description:** Write 16 bits of EEDATA memory on dsPIC30F and PIC24FXXKA devices.

**Include:** <libpic30.h>

**Prototype:** void \_write\_eedata\_word(\_prog\_addressT *dst*, int *dat*);

**Argument:** *dst* destination memory address  
*dat* integer data to be written

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Write one word of EEDATA memory for dsPIC30F devices.

**File:** eedata\_helper.c

---

---

## **\_write\_eedata\_row**

---

**Description:** Write `_EE_ROW` bytes of EEDATA memory on dsPIC30F devices.

**Include:** <libpic30.h>

**Prototype:** void \_write\_eedata\_row(\_prog\_addressT *dst*, int \**src*);

**Argument:** *dst* destination memory address  
*\*src* points to the storage location of data to be written

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Write specified bytes of EEDATA memory.

**File:** eedata\_helper.c

---

# 16-Bit Language Tools Libraries

---

## Example of Use – dsPIC30F DSCs

```
#include "libpic30.h"
#include "p30fxxxx.h"

char __attribute__((space(eedata), aligned(_EE_ROW))) dat[_EE_ROW];

int main()
{
    char i,source[_EE_ROW];
    _prog_addressT p;

    for (i = 0; i < _EE_ROW; )
        source[i] = i++;           /* initialize some data */

    _init_prog_address(p, dat);    /* get address in program space */

    _erase_eedata(p, _EE_ROW);    /* erase a row */

    _wait_eedata();               /* wait for operation to complete */

    _write_eedata_row(p, source); /* write a row */
}
```

## Example of Use – PIC24FXXKA MCUs

```
#include "libpic30.h" /* should use <> here */
#include "p24Fxxxx.h"

int __attribute__((space(eedata), aligned(_EE_4WORDS)))
    dat[_EE_4WORDS/2];

int main()
{
    _prog_addressT p;

    _init_prog_address(p, dat);    /* get address in program
                                     space */
    _erase_eedata(p, _EE_4WORDS); /* erase the dat[] array */
    _wait_eedata();               /* wait to complete */
    _write_eedata_word(p, 0x1234); /* write a word to dat[0] */
    _wait_eedata();

    p += 2;
    _write_eedata_word(p, 0x5678); /* write a word to dat[1] */
    _wait_eedata();
}
```

# Standard C Libraries - Support Functions

---

## 4.7 FUNCTIONS FOR ERASING AND WRITING FLASH MEMORY

These functions support the erasing and writing of Flash memory for devices that have this type of memory.

---

### **\_erase\_flash**

---

**Description:** Erase a page of Flash memory. The length of a page is `_FLASH_PAGE` words (1 word = 3 bytes = 2 PC address units).

**Include:** `<libpic30.h>`

**Prototype:** `void _erase_flash(_prog_addressT dst);`

**Argument:** `dst` destination memory address

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Erase a page of Flash memory.

**File:** `flash_helper.s`

---

### **\_erase\_flash (PIC24FXXKA Only)**

---

**Description:** Erase rows of Flash memory, either one, two or four rows.

**Include:** `<libpic30.h>`

**Prototype:** `void _erase_flash(_prog_addressT dst, int len);`

**Argument:** `dst` destination memory address  
`len` length may be `_FLASH_ROW`, `_FLASH_2ROWS` or `_FLASH_4ROWS` (bytes)

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Erase rows of Flash memory.

**File:** `flash_helper2.s`

---

### **\_write\_flash16**

---

**Description:** Write a row of Flash memory with 16-bit data. The length of a row is `_FLASH_ROW` words. The upper byte of each destination word is filled with `0xFF`. Note that the row must be erased before any write can be successful.

**Include:** `<libpic30.h>`

**Prototype:** `void _write_flash16(_prog_addressT dst, int *src);`

**Argument:** `dst` destination memory address  
`*src` points to the storage location of data to be written

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Write a row of Flash memory with 16-bit data.

**File:** `flash_helper.s`

# 16-Bit Language Tools Libraries

---

---

---

## \_write\_flash24

---

<b>Description:</b>	Write a row of Flash memory with 24-bit data. The length of a row is <code>_FLASH_ROW</code> words. Note that the row must be erased before any write can be successful.
<b>Include:</b>	<code>&lt;libpic30.h&gt;</code>
<b>Prototype:</b>	<code>void _write_flash24(_prog_addressT dst, long *src);</code>
<b>Argument:</b>	<i>dst</i> destination memory address <i>*src</i> points to the storage location of data to be written
<b>Return Value:</b>	None.
<b>Remarks:</b>	None.
<b>Default Behavior:</b>	Write a row of Flash memory with 24-bit data.
<b>File:</b>	<code>flash_helper.s</code>

---

## \_write\_flash\_word16

---

<b>Description:</b>	Write a word of Flash memory with 16-bit data. The upper byte of the destination word is filled with 0xFF. Note that the word must be erased before any write can be successful. This function is currently available only for PIC24F devices (excluding PIC24FXXKA MCUs).
<b>Include:</b>	<code>&lt;libpic30.h&gt;</code>
<b>Prototype:</b>	<code>void _write_flash_word16(_prog_addressT dst, int dat);</code>
<b>Argument:</b>	<i>dst</i> destination memory address <i>dat</i> integer data to be written
<b>Return Value:</b>	None.
<b>Remarks:</b>	None.
<b>Default Behavior:</b>	Write a word of Flash memory with 16-bit data for most PIC24 devices.
<b>File:</b>	<code>flash_helper.s</code>

---

## \_write\_flash\_word24

---

<b>Description:</b>	Write a word of Flash memory with 24-bit data. Note that the word must be erased before any write can be successful. This function is currently available only for PIC24F devices (excluding PIC24FXXKA MCUs).
<b>Include:</b>	<code>&lt;libpic30.h&gt;</code>
<b>Prototype:</b>	<code>void _write_flash_word24(_prog_addressT dst, long dat);</code>
<b>Argument:</b>	<i>dst</i> destination memory address <i>dat</i> integer data to be written
<b>Return Value:</b>	None.
<b>Remarks:</b>	None.
<b>Default Behavior:</b>	Write a word of Flash memory with 24-bit data for most PIC24 devices.
<b>File:</b>	<code>flash_helper.s</code>

# Standard C Libraries - Support Functions

---

---

---

## **`_write_flash_word32`**

---

**Description:** Write two words of FLASH memory with 16 bits of data per word. The 16 bits are written to the low 16 bits of the word. Word writes are supported dsPIC33E and PIC24E devices. The row address is specified with type `_prog_addressT`. Note that the location must be erased before any write can be successful.  
This function is currently disabled for devices subject to the Device ID errata as described in DS-80444, DS-80446, or DS-80447 (#32).

**Include:** `<libpic30.h>`

**Prototype:** `void _write_flash_word32(_prog_addressT dst, int dat1, int dat2);`

**Argument:** `dst` destination memory address  
`dat1, dat2` integer data to be written

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Write two words of Flash memory with 16-bit data for most dsPIC33E/PIC24E devices.

**File:** `flash_helper.s`

---

## **`_write_flash_word48`**

---

**Description:** Write two words of FLASH memory with 24 bits of data per word. Word writes are supported 33E and 24E devices. The row address is specified with type `_prog_addressT`. Note that the location must be erased before any write can be successful.  
This function is currently disabled for devices subject to the Device ID errata as described in DS-80444, DS-80446, or DS-80447 (#32).

**Include:** `<libpic30.h>`

**Prototype:** `void _write_flash_word48(_prog_addressT dst, int dat1, int dat2);`

**Argument:** `dst` destination memory address  
`dat1, dat2` integer data to be written

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Write two words of Flash memory with 48-bit data for most dsPIC33E/PIC24E devices.

**File:** `flash_helper.s`

# 16-Bit Language Tools Libraries

---

## Example of Use

```
#include "libpic30.h"
#include "p24Fxxxx.h"

int __attribute__((space(prog),aligned(_FLASH_PAGE*2)))
dat[_FLASH_PAGE];

int main()
{
    int i;
    int  source1[_FLASH_ROW];
    long source2[_FLASH_ROW];
    _prog_addressT p;

    for (i = 0; i < _FLASH_ROW; ) {
        source1[i] = i;
        source2[i] = i++;
    }                                /* initialize some data */

    _init_prog_address(p, dat); /* get address in program space */

    _erase_flash(p);            /* erase a page */

    _write_flash16(p, source1); /* write first row with 16-bit data */

#ifdef __dsPIC30F__
    _erase_flash(p);            /* on dsPIC30F, only 1 row per page */
#else
    p += (_FLASH_ROW * 2);      /* advance to next row */
#endif

    _write_flash24(p, source2); /* write second row with 24-bit data */
}
```

# Standard C Libraries - Support Functions

---

## Example of Use – PIC24FXXKA MCUs

```
#include "libpic30.h" /* should use <> here */
#include "p24Fxxxx.h"

int __attribute__((space(prog),aligned(_FLASH_2ROWS*2)))
    dat[_FLASH_2ROWS];

int main()
{
    int i;
    int  source1[_FLASH_ROW];
    long source2[_FLASH_ROW];
    _prog_addressT p;

    for (i = 0; i < _FLASH_ROW; ) {
        source1[i] = i;
        source2[i] = i++;
    }

    _init_prog_address(p, dat); /* get address in program
                                space */
    _erase_flash(p, _FLASH_2ROWS); /* erase two rows */

    _write_flash16(p, source1); /* write first row with
                                16-bit data */
    p += (_FLASH_ROW * 2); /* advance to next row */

    _write_flash24(p, source2); /* write second row with
                                24-bit data */
}
```

# 16-Bit Language Tools Libraries

---

## 4.8 FUNCTIONS FOR SPECIALIZED COPYING AND INITIALIZATION

These functions support specialized data copying and initialization.

---

### **`_memcpy_p2d16`**

---

**Description:** Copy 16 bits of data from each address in program memory to data memory. The next unused source address is returned.

**Include:** `<libpic30.h>`

**Prototype:** `_prog_addressT _memcpy_p2d16(char *dest, _prog_addressT src, unsigned int len);`

**Argument:** *dest* pointer to destination memory address  
*src* address of data to be written  
*len* length of program memory

**Return Value:** The next unused source address.

**Remarks:** None.

**Default Behavior:** Copy 16 bits of data from each address in program memory to data memory.

**File:** `memcpy_helper.s`

---

### **`_memcpy_p2d24`**

---

**Description:** Copy 24 bits of data from each address in program memory to data memory. The next unused source address is returned.

**Include:** `<libpic30.h>`

**Prototype:** `_prog_addressT _memcpy_p2d24(char *dest, _prog_addressT src, unsigned int len);`

**Argument:** *dest* pointer to destination memory address  
*src* address of data to be written  
*len* length of program memory

**Return Value:** The next unused source address.

**Remarks:** None.

**Default Behavior:** Copy 24 bits of data from each address in program memory to data memory.

**File:** `memcpy_helper.s`

---

### **`_strncpy_p2d16`**

---

**Description:** Copy 16 bits of data from each address in program memory to data memory. The operation terminates early if a NULL char is copied. The next unused source address is returned.

**Include:** `<libpic30.h>`

**Prototype:** `_prog_addressT _strncpy_p2d16(char *dest, _prog_addressT src, unsigned int len);`

**Argument:** *dest* pointer to destination memory address  
*src* address of data to be written  
*len* length of program memory

**Return Value:** The next unused source address.

**Remarks:** None.

**Default Behavior:** Copy 16 bits of data from each address in program memory to data memory.

**File:** `memcpy_helper.s`



# Standard C Libraries - Support Functions

---

---

---

## **`_strncpy_p2d24`**

---

<b>Description:</b>	Copy 24 bits of data from each address in program memory to data memory. The operation terminates early if a NULL char is copied. The next unused source address is returned.
<b>Include:</b>	<code>&lt;libpic30.h&gt;</code>
<b>Prototype:</b>	<code>_prog_addressT _strncpy_p2d24(char *dest, _prog_addressT src, unsigned int len);</code>
<b>Argument:</b>	<i>dest</i> pointer to destination memory address <i>src</i> address of data to be written <i>len</i> length of program memory
<b>Return Value:</b>	The next unused source address.
<b>Remarks:</b>	None.
<b>Default Behavior:</b>	Copy 24 bits of data from each address in program memory to data memory.
<b>File:</b>	<code>memcpy_helper.s</code>

---

## **`_init_prog_address`**

---

<b>Description:</b>	A macro that is used to initialize variables of type, <code>_prog_addressT</code> . These variables are not equivalent to C pointers.
<b>Include:</b>	<code>&lt;libpic30.h&gt;</code>
<b>Prototype:</b>	<code>_init_prog_address(a,b);</code>
<b>Argument:</b>	<i>a</i> variable of type <code>_prog_addressT</code> <i>b</i> initialization value for variable <i>a</i>
<b>Return Value:</b>	N/A
<b>Remarks:</b>	None.
<b>Default Behavior:</b>	Initialize variable to specified value.
<b>File:</b>	<code>libpic30.c</code>

# 16-Bit Language Tools Libraries

---

## Example of Use

```
#include "stdio.h"
#include "libpic30.h"

void display_mem(char *p, unsigned int len) {
    int i;
    for (i = 0; i < len; i++) {
        printf(" %d", *p++);
    }
    printf("\n");
}

char __attribute__((space(prog))) dat[] =
{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

char buf[10];

int main() {
    int i;
    _prog_addressT p;

    /* method 1 */
    _init_prog_address(p, dat);
    (void) _memcpy_p2d16(buf, p, 10);

    display_mem(buf,10);

    /* method 2 */
    _init_prog_address(p, dat);
    p = _memcpy_p2d16(buf, p, 4);
    p = _memcpy_p2d16(&buf[4], p, 6);

    display_mem(buf,10);
}
```

---

---

## Chapter 5. Fixed-Point Math Functions

---

---

### 5.1 INTRODUCTION

Fixed-point library math functions are contained in the files `libq-omf.a` (standard) and `libq-dsp-omf.a` (DSP), where `omf` will be `coff` or `elf` depending upon the selected object module format. The header file is named `libq.h`, and is the same for standard or DSP versions of the library. Linker options `-lq` (standard and DSP) and `-lq-dsp` (DSP only) must be used when linking the respective libraries.

#### 5.1.1 Assembly Code Applications

A free version of the math functions library and header file is available from the Microchip web site.

#### 5.1.2 C Code Applications

The MPLAB XC16 C Compiler install directory (**Section 1.1.1 “Compiler Installation Locations”**) contains the following subdirectories with library-related files:

- `lib` – standard C library files
- `support\h` – header files for libraries

In addition, there is a file, `ResourceGraphs.pdf`, which contains diagrams of resources used by each function, located in `lib`.

#### 5.1.3 Chapter Organization

This chapter is organized as follows:

- Overview of Fixed-Point Data Formats
- Using the Fixed-Point Libraries
- `<libq.h>` mathematical functions

# 16-Bit Language Tools Libraries

## 5.2 OVERVIEW OF FIXED-POINT DATA FORMATS

The integer data is encoded as its two's complement to accommodate both positive and negative numbers in binary format. The two's complement can be represented using integer format or the fractional format.

### 5.2.1 Integer Format

The integer format data is represented as a signed two's complement value, where the Most Significant bit is defined as a sign bit. The range of an N-bit two's complement integer is  $-2^{N-1}$  to  $2^{N-1}-1$  with a resolution of 1. For a 16-bit integer, the data range is -32768 (0x8000) to +32767 (0x7FFF) with a resolution of 1. For a 32-bit integer, the data range is -2,147,483,648 (0x8000 0000) to +2,147,483,647 (0x7FFF FFFF) with a resolution of 1.

### 5.2.2 Fractional Format

The fractional data format (Qn.m) has integral part (n) and fractional part (m) and the Most Significant bit represents the sign, thus consisting of (m+n+1) bits. It represents a signed two's complement value. Qn.m format data has a range of  $[-2^n, (2^n-2^{-m})]$  with  $2^{-m}$  resolution.

The binary representation of an N-bit (m+n+1 bits) number in Qn.m is shown in Figure 5-1. The value is given by the equation shown in Figure 5-2.

**FIGURE 5-1: BINARY REPRESENTATION**

$$b_{\underbrace{m+n}_{N-1}} b_{m+n-1} \dots b_m . b_{m-1} \dots b_1 b_0$$

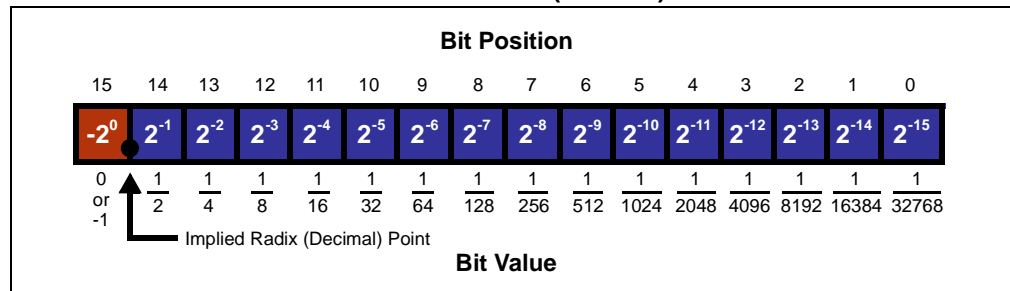
**FIGURE 5-2: EQUATION VALUE**

$$Value = -b_{N-1} 2^n + \sum_{l=0}^{N-2} b_l 2^{l-m}$$

#### 5.2.2.1 Q15 (1.15) FORMAT

In Q15 format, the Most Significant bit is defined as a sign bit and the radix point is implied to lie just after the sign bit followed by the fractional value. This format is commonly referred to as 1.15 or Q15 format, where 1 is the number of bits used to represent the integer portion of the number, and 15 is the number of bits used to represent the fractional portion. The range of an N-bit two's complement fraction with this implied radix point is  $-1.0$  to  $(1 - 2^{1-N})$ . For a 16-bit fraction, the 1.15 data range is  $-1.0$  (0x8000) to  $+0.999969482$  (0x7FFF) with a precision of  $3.05176 \times 10^{-5}$ .

**FIGURE 5-3: FRACTIONAL FORMAT (16 BITS)**



# Fixed-Point Math Functions

The following table shows the conversion of a two's complement 16-bit integer +24576 to Q15 value +0.75.

**TABLE 5-1: CONVERSION OF A TWO'S COMPLEMENT 16-BIT INTEGER TO Q15**

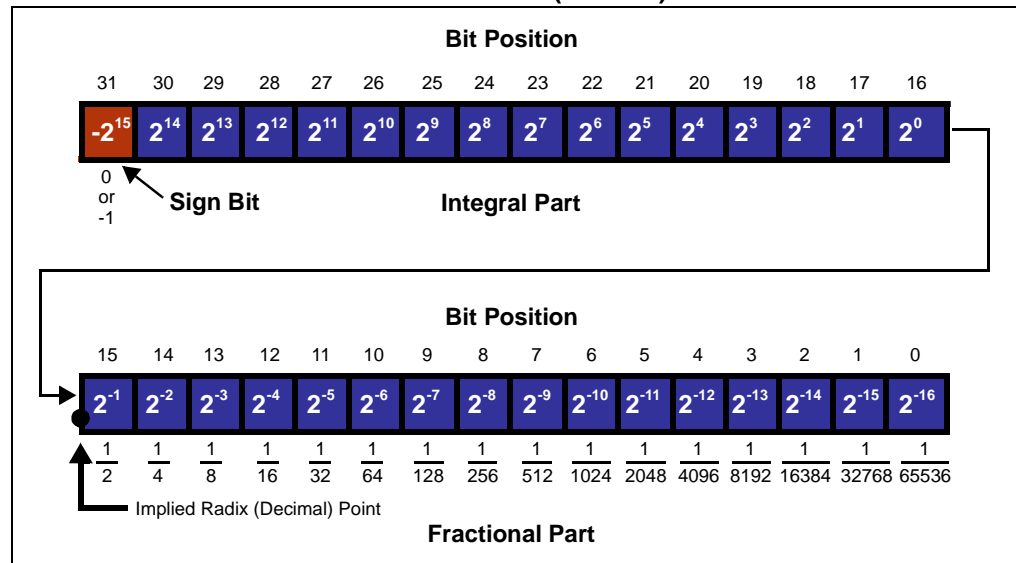
Binary		Dec		Q15
0	$0 \times (-2^{15})$	0	$0 \times (-2^0)$	0
			●	●
1	$1 \times 2^{14}$	16384	$1 \times 2^{-1}$	0.5
1	$1 \times 2^{13}$	8192	$1 \times 2^{-2}$	0.25
0	$0 \times 2^{12}$	0	$0 \times 2^{-3}$	0
0	$0 \times 2^{11}$	0	$0 \times 2^{-4}$	0
0	$0 \times 2^{10}$	0	$0 \times 2^{-5}$	0
0	$0 \times 2^9$	0	$0 \times 2^{-6}$	0
0	$0 \times 2^8$	0	$0 \times 2^{-7}$	0
0	$0 \times 2^7$	0	$0 \times 2^{-8}$	0
0	$0 \times 2^6$	0	$0 \times 2^{-9}$	0
0	$0 \times 2^5$	0	$0 \times 2^{-10}$	0
0	$0 \times 2^4$	0	$0 \times 2^{-11}$	0
0	$0 \times 2^3$	0	$0 \times 2^{-12}$	0
0	$0 \times 2^2$	0	$0 \times 2^{-13}$	0
0	$0 \times 2^1$	0	$0 \times 2^{-14}$	0
0	$0 \times 2^0$	0	$0 \times 2^{-15}$	0
	<b>SUM</b>	<b>+24576</b>	<b>SUM</b>	<b>+0.75</b>

● - Radix Point

## 5.2.2.2 Q15.16 FORMAT

In the Q15.16 format, the Most Significant bit is defined as a sign bit followed by 16 bits of the integral part. The radix point is implied to lie just after the integral part, followed by 16 bits of the fractional value. This format is commonly referred to as Q15.16 format. The range of Q15.16 numbers is from -32768.0 (0x8000 0000) to +32767.9999847412109375 (0x7FFF FFFF) and has a precision of  $2^{-16}$ .

**FIGURE 5-4: FRACTIONAL FORMAT (32 BITS)**



# 16-Bit Language Tools Libraries

The following table shows the conversion of a two's complement 32-bit integer, -715827882 to Q15.16 value -10922.6666564941.

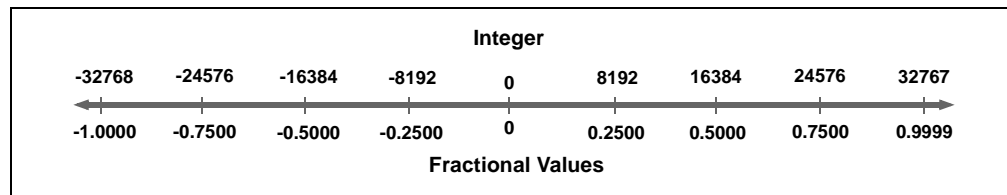
**TABLE 5-2: CONVERSION OF A TWO'S COMPLEMENT 32-BIT INTEGER TO Q15.16**

Binary		Dec		Q15.16
1	$1 \times (-2^{31})$	-2147483648	$1 \times (-2^{15})$	-32768
1	$1 \times 2^{30}$	1073741824	$1 \times 2^{14}$	16384
0	$0 \times 2^{29}$	0	$0 \times 2^{13}$	0
1	$1 \times 2^{28}$	268435456	$1 \times 2^{12}$	4096
0	$0 \times 2^{27}$	0	$0 \times 2^{11}$	0
1	$1 \times 2^{26}$	67108864	$1 \times 2^{10}$	1024
0	$0 \times 2^{25}$	0	$0 \times 2^9$	0
1	$1 \times 2^{24}$	16777216	$1 \times 2^8$	256
0	$0 \times 2^{23}$	0	$0 \times 2^7$	0
1	$1 \times 2^{22}$	4194304	$1 \times 2^6$	64
0	$0 \times 2^{21}$	0	$0 \times 2^5$	0
1	$1 \times 2^{20}$	1048576	$1 \times 2^4$	16
0	$0 \times 2^{19}$	0	$0 \times 2^3$	0
1	$1 \times 2^{18}$	262144	$1 \times 2^2$	4
0	$0 \times 2^{17}$	0	$0 \times 2^1$	0
1	$1 \times 2^{16}$	65536	$1 \times 2^0$	1
			●	●
0	$0 \times 2^{15}$	0	$0 \times 2^{-1}$	0
1	$1 \times 2^{14}$	16384	$1 \times 2^{-2}$	0.25
0	$0 \times 2^{13}$	0	$0 \times 2^{-3}$	0
1	$1 \times 2^{12}$	4096	$1 \times 2^{-4}$	0.0625
0	$0 \times 2^{11}$	0	$0 \times 2^{-5}$	0
1	$1 \times 2^{10}$	1024	$1 \times 2^{-6}$	0.015625
0	$0 \times 2^9$	0	$0 \times 2^{-7}$	0
1	$1 \times 2^8$	256	$1 \times 2^{-8}$	0.00390625
0	$0 \times 2^7$	0	$0 \times 2^{-9}$	0
1	$1 \times 2^6$	64	$1 \times 2^{-10}$	0.000976563
0	$0 \times 2^5$	0	$0 \times 2^{-11}$	0
1	$1 \times 2^4$	16	$1 \times 2^{-12}$	0.000244141
0	$0 \times 2^3$	0	$0 \times 2^{-13}$	0
1	$1 \times 2^2$	4	$1 \times 2^{-14}$	6.10352E-05
1	$1 \times 2^1$	2	$1 \times 2^{-15}$	3.01576E-05
0	$0 \times 2^0$	0	$0 \times 2^{-16}$	0
	<b>SUM</b>	<b>-715827882</b>	<b>SUM</b>	<b>-10922.6666564941</b>
● - Radix Point				

## 5.2.3 Integer - Fractional Format Mapping

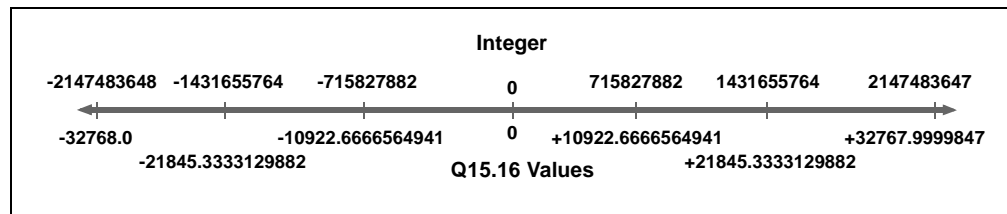
The same twos complement N-bit word may represent an integer format value or a fractional format value. For example., the 16-bit integer range [-32768, +32767] format maps to Q15 range of [-1.0, +0.999969482]. Figure 5-5 shows the mapping between these formats.

**FIGURE 5-5: MAPPING BETWEEN 16-BIT INTEGER FORMAT AND Q15 FRACTIONAL FORMAT**



A similar relationship exists between the 32-bit integer format and the Q15.16 format, where the integer range [-2147483648, +2147483647] is mapped to the Q15.16 range [-32768.0, +32767.9999847412109375].

**FIGURE 5-6: MAPPING BETWEEN 32-BIT INTEGER FORMAT AND Q15.16 FORMAT**



## 5.2.4 libq Library and Fixed-Point Data Format

The functions in the `libq` library use the fixed-point data format. The parameters passed and the results generated by the functions are fractional in nature. There are two similar sets of math functions which perform the same math operations. One set supports Q15 operands and the other supports Q15.16 operands. Q15.16 operand functions, naturally, have better precision and range compared to Q15 operand functions.

## 5.3 USING THE FIXED-POINT LIBRARIES

Building an application which utilizes the fixed-point libraries requires two types of files: header files and library files.

### 5.3.1 Header Files

All standard C library entities are declared or defined in one or more standard headers (see list in **Section 5.1.3 “Chapter Organization”**). To make use of a library entity in a program, write an include directive that names the relevant standard header.

The contents of a standard header are included by naming them in an include directive, as in:

```
#include <libq.h> /* include fixed-point library */
```

The standard headers can be included in any order. Do not include a standard header within a declaration. Do not define macros that have the same names as keywords before including a standard header.

A standard header never includes another standard header.

# 16-Bit Language Tools Libraries

## 5.3.2 Library Files

The archived library files contain all the individual object files for each library function. When linking an application, the library file (`libq-omf.a` or `libq-dsp-omf.a`) must be provided as an input to the linker (using the `--library` or `-l` linker option), such that the functions used by the application may be linked into the application. Also, linker options `-lq` and `-lq-dsp` must be used when linking the respective libraries.

A typical C application will require three library files: `libc-omf.a`, `libm-omf.a` and `libpic30-omf.a` (see **Section 1.2 “OMF-Specific Libraries/Start-up Modules”** for more on OMF-specific libraries). These libraries will be included automatically if linking is performed using the compiler.

## 5.3.3 Function Naming Conventions

Signed fixed-point types are defined as follows:

$Qn_m$

where:

- $n$  is the number of data bits to the left of the radix point
- $m$  is the number of data bits to the right of the radix point

**Note:** A sign bit is implied.

For convenience, short names are also defined:

Exact Name	# Bits Required	Short Name
<code>_Q0_15</code>	16	<code>_Q15</code>
<code>_Q15_16</code>	32	<code>_Q16</code>

In this document, the terms `Q15.16` and `Q16` are used interchangeably; however, both imply `Q15.16` format. Functions in the library are prefixed with the type of the return value. For example, `_Q15acos` returns a `Q15` value equal to the arc cosine of its argument.

Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments.

In cases where the return value is not a fixed-point type, the argument type is appended to the function name. For example, function `_itoaQ15` accepts a type `Q15` argument.

In cases where two versions of a function are provided with the same return type but different argument types, the argument type is appended to the function name. For example:

Function Name	Return Type	Argument Type
<code>_Q16reciprocalQ15</code>	<code>_Q16</code>	<code>_Q15</code>
<code>_Q16reciprocalQ16</code>	<code>_Q16</code>	<code>_Q16</code>



## 5.4 <LIBQ.H> MATHEMATICAL FUNCTIONS

The header file `libq.h` consists of macro definitions and various functions that calculate fixed-point mathematical operations.

### 5.4.1 Q15 Functions

Many functions in this section use fixed-point Q15 (1.15) format. For each function, the entire range may not be used.

---

#### **`_Q15abs`**

---

**Description:** The function finds the absolute value of a Q15 value.  
**Include:** `<libq.h>`  
**Prototype:** `_Q15 _Q15abs(_Q15 x);`  
**Argument:** `x` a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
**Return Value:** This function returns the absolute value of `x` in Q15 format. The value ranges from 0 to 32767.

---

---

#### **`_Q15acos`**

---

**Description:** This function finds the arc cosine of a Q15 value.  
**Include:** `<libq.h>`  
**Prototype:** `_Q15 _Q15acos(_Q15 x);`  
**Argument:** `x` a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from 17705 to 32767.  
**Return Value:** This function returns the arc cosine of `x` in Q15 format. The value ranges from 256 to 32767.

---

---

#### **`_Q15acosByPI`**

---

**Description:** This function finds the arc cosine of a Q15 value and then divides by PI ( $\pi$ ).  
**Include:** `<libq.h>`  
**Prototype:** `_Q15 _Q15acosByPI(_Q15 x);`  
**Argument:** `x` a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
**Return Value:** This function returns the arc cosine of `x`, divided by PI, in Q15 format. The value ranges from 82 to 32767.

---

# 16-Bit Language Tools Libraries

---

---

---

## **\_Q15add**

---

**Description:** This function finds the sum value of two Q15 values. The function takes care of saturation during overflow and underflow occurrences.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15add(_Q15 x, _Q15 y);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
 $y$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the sum of  $x$  and  $y$  in Q15 format. The value ranges from -32768 to 32767.

---

---

## **\_Q15asin**

---

**Description:** This function finds the arc sine of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15asin(_Q15 x);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -27573 to 27573.

**Return Value:** This function returns the arc sine of  $x$  in Q15 format. The value ranges from -32768 to 32767.

---

---

## **\_Q15asinByPI**

---

**Description:** This function finds the arc sine of a Q15 value and then divides by PI ( $\pi$ ).

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15asinByPI(_Q15 x);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the arc sine of  $x$ , divided by PI, in Q15 format. The value ranges from -16384 to 16303.

---

---

## **\_Q15atan**

---

**Description:** This function finds the arc tangent of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15atan(_Q15 x);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the arc tangent of  $x$  in Q15 format. The value ranges from -25736 to 25735.

---

---

## **\_Q15atanByPI**

---

**Description:** This function finds the arc tangent of a Q15 value and then divides by PI ( $\pi$ ).

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15atanByPI(_Q15 x);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the arc tangent of  $x$ , divided by PI, in Q15 format. The value ranges from -8192 to 8192.

---

---

## **\_Q15atanYByX**

---

**Description:** This function finds the arc tangent of a Q15 value divided by a second Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15atanYByX(_Q15 x, _Q15 y);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
 $y$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the arc tangent of  $y$  divided by  $x$  in Q15 format. The value ranges from -25736 to 25735.

---

---

## **\_Q15atanYByXByPI**

---

**Description:** This function finds the arc tangent of a Q15 value divided by a second Q15 value and then divides the result by PI ( $\pi$ ).

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15atanYByXByPI(_Q15 x, _Q15 y);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
 $y$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the arc tangent of  $y$  divided by  $x$ , divided by PI, in Q15 format. The value ranges from -8192 to 8192.

---

---

## **\_Q15atoi**

---

**Description:** This function takes a string which holds the ASCII representation of decimal digits and converts it into a single Q15 number.  
**Note:** The decimal digit should not be beyond the range: -32768 to 32767.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15atoi(const char *s);`

**Argument:**  $s$  a buffer holding the ASCII values of each decimal digit.

**Return Value:** This function returns the integer equivalent of  $s$  in Q15 format, which range is from -32768 to 32767.

---

# 16-Bit Language Tools Libraries

---

---

---

## **\_Q15cos**

---

**Description:** This function finds the cosine of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15cos(_Q15 x);`

**Argument:** `x` a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the cosine of `x` in Q15 format. The value ranges from 17705 to 32767.

---

---

## **\_Q15cosPI**

---

**Description:** This function finds the cosine of PI ( $\pi$ ) times a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15cosPI(_Q15 x);`

**Argument:** `x` a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the cosine of PI times `x` in Q15 format. The value ranges from -32768 to 32767.

---

---

## **\_Q15exp**

---

**Description:** This function finds the exponential value of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15exp(_Q15 x);`

**Argument:** `x` a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 0.

**Return Value:** This function returns the exponent value of `x` in Q15 format. The value ranges from 12055 to 32767.

---

---

## **\_Q15ftoi**

---

**Description:** This function converts a single precision floating-point value into its corresponding Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15ftoi(float x);`

**Argument:** `x` a floating-point equivalent number. The corresponding floating-point range is -1 to 0.99996.

**Return Value:** This function returns a fixed-point number in Q15 format. The value ranges from -32768 to 32767.

---

---

## **\_itoaQ15**

---

**Description:** This function converts each decimal digit of a Q15 value to its representation in ASCII. For example, 1 is converted to 0x31, which is the ASCII representation of 1.

**Include:** `<libq.h>`

**Prototype:** `void _itoaQ15(_Q15 x, char *s);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
*s* a buffer holding values in ASCII, at least 8 characters long.

**Return Value:** None.

---

---

## **\_itofQ15**

---

**Description:** This function converts a Q15 value into its corresponding floating-point value.

**Include:** `<libq.h>`

**Prototype:** `float _itofQ15(_Q15 x);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns a floating-point equivalent number. The corresponding floating-point range is -1 to 0.99996.

---

---

## **\_Q15log**

---

**Description:** This function finds the natural log of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15log(_Q15 x);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from 12055 to 32767.

**Return Value:** This function returns the natural log of *x* in Q15 format. The value ranges from -32768 to -1.

---

---

## **\_Q15log10**

---

**Description:** This function finds the log (base 10) of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15log10(_Q15 x);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from 3277 to 32767.

**Return Value:** This function returns the log of *x* in Q15 format. The value ranges from -32768 to 0.

---

# 16-Bit Language Tools Libraries

---

---

---

## **\_Q15neg**

---

**Description:** This function negates a Q15 value with saturation. The value is saturated in the case where the input is -32768.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15neg(_Q15 x);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns  $-x$  in Q15 format. The value ranges from -32768 to 32767.

---

---

## **\_Q15norm**

---

**Description:** This function finds the normalized value of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15norm(_Q15 x);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the square root of *x* in Q15 format. The value ranges from 16384 to -32767 for a positive number and -32768 to -16384 for a negative number.

---

---

## **\_Q15power**

---

**Description:** This function finds the power result given the base value and the power value in Q15 format.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15power(_Q15 x, _Q15 y);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from 1 to  $2^{15}-1$ . The value of this argument ranges from 1 to 32767.  
*y* a fixed-point number in Q15 format, which ranges from 1 to  $2^{15}-1$ . The value of this argument ranges from 1 to 32767.

**Return Value:** This function returns *x* to the power of *y* in Q15 format. The value ranges from 1 to 32767.

---

---

## **\_Q15random**

---

**Description:** This function generates a random number in the range from -32768 to 32767. The random number generation is periodic with period 65536. The function uses the `_Q15randomSeed` variable as a random seed value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15random(void);`

**Argument:** None.

**Return Value:** This function returns a random number in Q15 format. The value ranges from -32768 to 32767.

---

---

## \_Q15shl

---

**Description:** This function shifts a Q15 value by *num* bits, to the left if *num* is positive or to the right if *num* is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15shl(_Q15 x, short num);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
*num* an integer number, which ranges from -15 to 15.

**Return Value:** This function returns the shifted value of *x* in Q15 format. The value ranges from -32768 to 32767.

---

---

## \_Q15shlNoSat

---

**Description:** This function shifts a Q15 value by *num* bits, to the left if *num* is positive or to the right if *num* is negative. The function sets the `_Q15shlSatFlag` variable in case of underflow or overflow but does not take care of saturation.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15shlNoSat(_Q15 x, short num);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
*num* an integer number, which ranges from -15 to 15.

**Return Value:** This function returns the shifted value of *x* in Q15 format. The value ranges from -32768 to 32767.

---

---

## \_Q15shr

---

**Description:** This function shifts a Q15 value by *num* bits, to the right if *num* is positive or to the left if *num* is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15shr(_Q15 x, short num);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
*num* an integer number, which ranges from -15 to 15.

**Return Value:** This function returns the shifted value of *x* in Q15 format. The value ranges from -32768 to 32767.

---

---

## \_Q15shrNoSat

---

**Description:** This function shifts a Q15 value by *num* bits, to the right if *num* is positive or to the left if *num* is negative. The function sets the `_Q15shrSatFlag` variable in case of underflow or overflow but does not take care of saturation.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15shrNoSat(_Q15 x, short num);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
*num* an integer number, which ranges from -15 to 15.

**Return Value:** This function returns the shifted value of *x* in Q15 format. The value ranges from -32768 to 32767.

---

---

# 16-Bit Language Tools Libraries

---

---

---

## \_Q15sin

---

**Description:** This function finds the sine of a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15sin(_Q15 x);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the sine of *x* in Q15 format. The value ranges from -27573 to 27573.

---

---

## \_Q15sinPI

---

**Description:** This function finds the sine of PI ( $\pi$ ) times a Q15 value.

**Include:** `<libq.h>`

**Prototype:** `_Q15 _Q15sinPI(_Q15 x);`

**Argument:** *x* a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the sine of PI times *x* in Q15 format. The value ranges from -32768 to 32767.

---

---

## \_Q15sinSeries

---

**Description:** Generates the sine series with the given normalizing frequency, *f*, and the given number of samples, *num*, starting from *start*. Stores the result in buffer, *buf*.

**Include:** `<libq.h>`

**Prototype:** `short _Q15sinSeries(_Q15 f, short start, short num, _Q15 *buf);`

**Argument:** *f* a fixed-point number in Q15 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from -16384 to 16384. The argument represents the Normalizing frequency.  
*start* a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Starting Sample number in the Sine Series.  
*num* a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Number of Sine Samples the function is called to generate.  
**Note:** *num* should not be more than 16383 for dsPIC and 32767 for PIC devices.  
*buf* a pointer to the buffer where the generated sine samples would get copied into.

**Return Value:** This function returns *num*, the number of generated sine samples.

---



---

## **\_Q15sqrt**

---

**Description:** This function finds the square root of a Q15 value.

**Include:** <libq.h>

**Prototype:** `_Q15 _Q15sqrt(_Q15 x);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from 1 to 32767.

**Return Value:** This function returns the square root of  $x$  in Q15 format. The value ranges from 1 to 32767.

---

---

## **\_Q15sub**

---

**Description:** This function finds the difference of two Q15 values. The function takes care of saturation during overflow and underflow occurrences.

**Include:** <libq.h>

**Prototype:** `_Q15 _Q15sub(_Q15 x, _Q15 y);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.  
 $y$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns  $x$  minus  $y$  in Q15 format. The value ranges from -32768 to 32767.

---

---

## **\_Q15tan**

---

**Description:** This function finds the tangent of a Q15 value.

**Include:** <libq.h>

**Prototype:** `_Q15 _Q15tan(_Q15 x);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -25736 to 25735.

**Return Value:** This function returns the tangent of  $x$  in Q15 format. The value ranges from -32768 to 32767.

---

---

## **\_Q15tanPI**

---

**Description:** This function finds the tangent of  $\pi$  times a Q15 value.

**Include:** <libq.h>

**Prototype:** `_Q15 _Q15tanPI(_Q15 x);`

**Argument:**  $x$  a fixed-point number in Q15 format, which ranges from  $-2^{15}$  to  $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value:** This function returns the tangent of  $\pi$  times  $x$  in Q15 format. The value ranges from -32768 to 32767.

---

# 16-Bit Language Tools Libraries

---

## 5.4.2 Q16 Functions

Many functions in this section use fixed-point Q16 (15.16) format. For each function, the entire range may not be used.

---

### \_Q16acos

---

**Description:** This function finds the arc cosine of a Q16 value.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16acos(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -65566 to 65536.

**Return Value:** This function returns the arc cosine of *x* in Q16 format. The value ranges from -205887 to 205887.

---

### \_Q16acosByPI

---

**Description:** This function finds the arc cosine of a Q16 value and then divides by PI ( $\pi$ ).

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16acosByPI(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65536.

**Return Value:** This function returns the arc cosine of *x*, divided by PI, in Q16 format. The value ranges from -65536 to 65536.

---

### \_Q16asin

---

**Description:** This function finds the arc sine of a Q16 value.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16asin(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -65566 to 65536.

**Return Value:** This function returns the arc sine of *x* in Q16 format. The value ranges from -102944 to 102944.

---

### \_Q16asinByPI

---

**Description:** This function finds the arc sine of a Q16 value and then divides by PI ( $\pi$ ).

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16asinByPI(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65536.

**Return Value:** This function returns the arc sine of *x*, divided by PI, in Q16 format. The value ranges from -65536 to 65536.

---

## **\_Q16atan**

---

<b>Description:</b>	This function finds the arc tangent of a Q16 value.
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16atan(_Q16 x);</code>
<b>Argument:</b>	<code>x</code> a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
<b>Return Value:</b>	This function returns the arc tangent of <code>x</code> in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## **\_Q16atanByPI**

---

<b>Description:</b>	This function finds the arc tangent of a Q16 value and then divides by PI ( $\pi$ ).
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16atanByPI(_Q16 x);</code>
<b>Argument:</b>	<code>x</code> a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
<b>Return Value:</b>	This function returns the arc tangent of <code>x</code> , divided by PI, in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## **\_Q16atanYByX**

---

<b>Description:</b>	This function finds the arc tangent of <code>y</code> divided by <code>x</code> .
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16atanYByX(_Q16 x, _Q16 y);</code>
<b>Argument:</b>	<code>x</code> a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the x input. <code>y</code> a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the y input.
<b>Return Value:</b>	This function returns the arc tangent of <code>y</code> , divided by <code>x</code> , in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## **\_Q16atanYByXByPI**

---

<b>Description:</b>	This function finds the arc tangent of the 32-bit input, <code>y</code> , divided by <code>x</code> , and then divides by PI ( $\pi$ ).
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16atanYByXByPI(_Q16 x, _Q16 y);</code>
<b>Argument:</b>	<code>x</code> a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the x input. <code>y</code> a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the y input.
<b>Return Value:</b>	This function returns the arc tangent <code>y</code> divided by <code>x</code> , divided by PI, in Q16 format. The value ranges from -2147483648 to 2147483647.

---

# 16-Bit Language Tools Libraries

---

---

---

## **\_Q16cos**

---

**Description:** This function finds the cosine of a Q16 value.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16cos(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns the cosine of *x* in Q16 format. The value ranges from -65566 to 65536.

---

---

## **\_Q16cosPI**

---

**Description:** This function finds the cosine of PI ( $\pi$ ) times a Q16 value.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16cosPI(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns the cosine of PI times *x* in Q16 format. The value ranges from -65536 to 65536.

---

---

## **\_Q16div**

---

**Description:** This function returns the quotient of its arguments.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16mpy(_Q16 dividend, _Q16 divisor);`

**Argument:** *dividend* a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
*divisor* a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value:** This function returns the quotient of its arguments. The value ranges from 0 to 2147483647..

---

---

## **\_Q16divmod**

---

**Description:** This function returns the quotient and remainder of its arguments.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16mpy(_Q16 dividend, _Q16 divisor, _Q16 *remainder);`

**Argument:** *dividend* a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
*divisor* a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
*remainder* a pointer to an object large enough to hold the remainder. This must be provided by the caller.

**Return Value:** This function returns the quotient and remainder of its arguments. The values range from 0 to 2147483647.

---

---

## **\_Q16exp**

---

**Description:** This function finds the exponential value of a Q16 value.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16exp(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -772244 to 681391.

**Return Value:** This function returns the exponent value of *x* in Q16 format. The value ranges from 0 to 2147483647.

---

---

## **\_Q16log**

---

**Description:** This function finds the natural log of a Q16 value.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16log(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from 1 to 2147483647.

**Return Value:** This function returns the natural log of *x* in Q16 format. The value ranges from -726817 to 681391.

---

---

## **\_Q16log10**

---

**Description:** This function finds the log (base 10) of a Q16 value.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16log10(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from 1 to 2147483647.

**Return Value:** This function returns the log of *x* in Q16 format. The value ranges from -315653 to 295925.

---

---

## **\_Q16mac**

---

**Description:** This function multiplies the two 32-bit inputs, *x* and *y*, and accumulates the product with *prod*. The function takes care of saturating the result in case of underflow or overflow.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16mac(_Q16x, _Q16 y, _Q16 prod);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
*y* a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
*prod* a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value:** This function returns the multiplied and accumulated value *prod* in Q16 format. The value ranges from 0 to 2147483647.

---

# 16-Bit Language Tools Libraries

---

---

---

## **\_Q16macNoSat**

---

**Description:** This function multiplies the two 32 bit inputs,  $x$  and  $y$  and accumulates the product with  $prod$ . This function only sets the `_Q16macSatFlag` variable in case of an overflow or underflow and does not take care of saturation.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16macNoSat(_Q16 x, _Q16 y, _Q16 prod);`

**Argument:**  $x$  a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
 $y$  a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
 $prod$  a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value:** This function returns the multiplied and accumulated value  $prod$  in Q16 format. The value ranges from 0 to 2147483647.

---

---

## **\_Q16mpy**

---

**Description:** This function returns the product of its arguments.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16mpy(_Q16 a, _Q16 b);`

**Argument:**  $a$  a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.  
 $b$  a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value:** This function returns the product of its arguments. The value ranges from 0 to 2147483647.

---

---

## **\_Q16neg**

---

**Description:** This function negates  $x$  with saturation. The value is saturated in the case where the input is -2147483648.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16neg(_Q16 x);`

**Argument:**  $x$  a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns the negated value of  $x$  in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## **\_Q16norm**

---

**Description:** This function finds the normalized value of a Q16 value.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16norm(_Q16 x);`

**Argument:**  $x$  a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns the square root value of  $x$  in Q16 format. The value ranges from 1073741824 to 2147483647 for a positive number and -2147483648 to -1073741824 for a negative number.

---

---

## **\_Q16power**

---

<b>Description:</b>	This function finds the power result, given the base value, $x$ , and the power value, $y$ .
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16power(_Q16 x, _Q16 y);</code>
<b>Argument:</b>	$x$ a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647. $y$ a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>Return Value:</b>	This function returns the value of $x$ to the power of $y$ in Q16 format. The value ranges from 0 to 2147483647.

---

## **\_Q16random**

---

<b>Description:</b>	This function generates a pseudo random number with a period of 2147483648. The function uses the <code>_Q16randomSeed</code> variable as a random seed value.
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16random(void);</code>
<b>Argument:</b>	None.
<b>Return Value:</b>	This function returns the generated random number in Q16 format. The value of this output ranges from -2147483648 to 2147483647.
<b>Remarks:</b>	$\text{RndNum}(n) = (\text{RndNum}(n-1) * \text{RAN\_MULT}) + \text{RAN\_INC}$ SEED VALUE = 21845, RAN_MULT = 1664525 and RAN_INC = 1013904223.

---

## **\_Q16reciprocal**

---

<b>Description:</b>	This function returns the reciprocal of a Q16 value.
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16reciprocal(_Q16 x);</code>
<b>Argument:</b>	$x$ a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
<b>Return Value:</b>	This function returns the reciprocal of $x$ in Q16 format. The value of this output ranges from -2147483648 to 2147483647.

---

## **\_Q16reciprocalQ15**

---

<b>Description:</b>	This function returns the reciprocal of a Q15 value. Since the input range lies in the -1 to +1 region, the output is always greater than the -1 or +1 region. So, Q16 format is used to represent the output.
<b>Include:</b>	<libq.h>
<b>Prototype:</b>	<code>_Q16 _Q16reciprocalQ15(_Q15 x);</code>
<b>Argument:</b>	$x$ a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
<b>Return Value:</b>	This function returns the reciprocal of $x$ in Q16 format. The value ranges from -2147483648 to 2147418112.

# 16-Bit Language Tools Libraries

---

---

---

## **\_Q16reciprocalQ16**

---

**Description:** This function returns the reciprocal value of the input.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16reciprocalQ16(_Q16 x);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns the reciprocal of *x* in Q16 format. The value of this output ranges from -2147483648 to 2147483647.

---

---

## **\_Q16shl**

---

**Description:** This function shifts the input argument, *x*, by *y* number of bits, to the left if *y* is positive or to the right if *y* is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16shl(_Q16 x, short y);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.  
*y* an integer number, which ranges from -32 to +32.

**Return Value:** This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## **\_Q16shlNoSat**

---

**Description:** This function shifts the input argument, *x*, by *y* number of bits, to the left if *y* is positive or to the right if *y* is negative. The function sets the `_Q16shlSatFlag` variable in case of underflow or overflow, but does not take care of saturation.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16shlNoSat(_Q16 x, short y);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.  
*y* an integer number, which ranges from -32 to +32.

**Return Value:** This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## **\_Q16shr**

---

**Description:** This function shifts the input argument, *x*, by *y* number of bits, to the right if *y* is positive or to the left if *y* is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include:** <libq.h>

**Prototype:** `_Q16 _Q16shr(_Q16 x, short y);`

**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.  
*y* an integer number, which ranges from -32 to +32.

**Return Value:** This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

---



---

## \_Q16shrNoSat

---

**Description:** This function shifts the input argument,  $x$ , by  $y$  number of bits, to the right if  $y$  is positive or to the left if  $y$  is negative. The function sets the `_Q16shrSatFlag` variable in case of underflow or overflow, but does not take care of saturation.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16shrNoSat(_Q16 x, short y);`

**Argument:**  $x$  a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.  
 $y$  an integer number, which ranges from -32 to +32.

**Return Value:** This function returns the shifted value of  $x$  in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## \_Q16sin

---

**Description:** This function finds the sine of a Q16 value.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16sin(_Q16 x);`

**Argument:**  $x$  a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns the sine of  $x$  in Q16 format. The value ranges from -65566 to 65536.

---

---

## \_Q16sinPI

---

**Description:** This function finds the sine of  $\text{PI} (\pi)$  times a Q16 value.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16sinPI(_Q16 x);`

**Argument:**  $x$  a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns the sine of  $\text{PI}$  times  $x$  in Q16 format. The value ranges from -65536 to 65536.

---

---

## \_Q16sinSeries

---

**Description:** Generates the sine series with the given normalizing frequency,  $f$ , and the given number of samples,  $num$ , starting from  $start$ . Stores the result in buffer,  $buf$ .

**Include:** `<libq.h>`

**Prototype:** `short _Q16sinSeries(_Q16 f, short start, short num, _Q16 *buf);`

---

---

## \_Q16sinSeries (Continued)

---

**Argument:** *f* a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from -32768 to 32768. This argument represents the Normalizing frequency.  
*start* a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Starting Sample number in the Sine Series.  
*num* a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Number of Sine Samples the function is called to generate.  
**Note:** *num* should not be more than 16383 for dsPIC and 32767 for PIC devices.  
*buf* a pointer to the buffer where the generated sine samples would get copied into.

**Return Value:** This function returns *num*, the number of generated sine samples.

---

---

## \_Q16tan

---

**Description:** This function finds the tangent of a Q16 value.  
**Include:** `<libq.h>`  
**Prototype:** `_Q16 _Q16tan(_Q16 x);`  
**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.  
**Return Value:** This function returns the tangent of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## \_Q16tanPI

---

**Description:** This function finds the tangent of PI ( $\pi$ ) times a Q16 value.  
**Include:** `<libq.h>`  
**Prototype:** `_Q16 _Q16tanPI(_Q16 x);`  
**Argument:** *x* a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.  
**Return Value:** This function returns the tangent of PI times *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

---

---

## **\_Q16ftoi**

---

**Description:** This function converts a `float` value to a Q16 fixed-point fractional value.

**Include:** `<libq.h>`

**Prototype:** `_Q16 _Q16ftoi(float f);`

**Argument:** `f` a floating-point number. The value of this argument ranges from -32768 to 32768.

**Return Value:** This function returns a Q16 fixed-point fractional value. The value ranges from -2147483648 to 2147483647.

---

---

## **\_itofQ16**

---

**Description:** This function converts a Q16 fixed-point fractional value to a `float` value.

**Include:** `<libq.h>`

**Prototype:** `float _Q16ftoi(_Q16 q);`

**Argument:** `q` a fixed-point number. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value:** This function returns a floating-point value. The value ranges from -32768 to 32768.

---

# 16-Bit Language Tools Libraries

---

NOTES:

**Appendix A. ASCII Character Set**

This table shows the ASCII character set in nibbles.

		Most Significant Nibbles							
		Hex	0	1	2	3	4	5	6
<b>Least Significant Nibbles</b>	<b>0</b>	NUL	DLE	Space	0	@	P	`	p
	<b>1</b>	SOH	DC1	!	1	A	Q	a	q
	<b>2</b>	STX	DC2	"	2	B	R	b	r
	<b>3</b>	ETX	DC3	#	3	C	S	c	s
	<b>4</b>	EOT	DC4	\$	4	D	T	d	t
	<b>5</b>	ENQ	NAK	%	5	E	U	e	u
	<b>6</b>	ACK	SYN	&	6	F	V	f	v
	<b>7</b>	Bell	ETB	'	7	G	W	g	w
	<b>8</b>	BS	CAN	(	8	H	X	h	x
	<b>9</b>	HT	EM	)	9	I	Y	i	y
	<b>A</b>	LF	SUB	*	:	J	Z	j	z
	<b>B</b>	VT	ESC	+	;	K	[	k	{
	<b>C</b>	FF	FS	,	<	L	\	l	
	<b>D</b>	CR	GS	-	=	M	]	m	}
	<b>E</b>	SO	RS	.	>	N	^	n	~
	<b>F</b>	SI	US	/	?	O	_	o	DEL

# 16-Bit Language Tools Libraries

---

NOTES:

---

---

## Appendix B. Document Revision History

---

---

### Revision J (December 2014)

- **Section 2.14 “<stdlib.h> utility functions”** - updated `malloc` function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for `_erase_flash` (PIC24FXXKA Only) and `_write_flash_word24()`.
- **Section 4.8 “Functions for Specialized Copying and Initialization”** - the following function descriptions have been updated:
  - `_memcpy_p2d24`
  - `_memcpy_p2d16`
  - `_strncpy_p2d24`
  - `_strncpy_p2d16`

### Revision H

- **Section 1.1 “Introduction”** - added Compiler Installation Locations
- **Section 2.10 “<signal.h> signal handling”** - updated `signal` function.
- **Section 2.13 “<stdio.h> input and output”** - updated Customizing STDIO handles and descriptions for `fdopen` and `vasprintf`.
- **Section 4.3 “Standard C Library Helper Functions”** - updated `_dump_heap_info` function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for `_write_flash_word32` and `_write_flash_word48`.
- **Section 5.2 “Overview of Fixed-Point Data Formats”** - inserted Table 5-1 and Table 5-2.
- **Section 5.4 “<libq.h> mathematical functions”** - the following function descriptions have been updated:
  - `_Q16div`
  - `_Q16divmod`
  - `_Q16mpy`
- Minor typographical changes.

### Revision G

- **Section 2.13 “<stdio.h> input and output”** - updated `scanf` function
- **Section 2.15 “<string.h> string functions”** - updated `memchr` function
- Inserted **Section 5.2 “Overview of Fixed-Point Data Formats”**
- Minor typographical changes.

## Revision F

- **Section 4.5 “Functions/Constants to Support A Simulated UART”** - the following function descriptions have been updated:
  - `_delay_ms`
  - `_delay_us`
  - `_C30UART`
- **Section 4.6 “Functions for Erasing and Writing EEDATA Memory”**- updated `_write_eedata_row` function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for `_erase_flash` and `_write_flash_word24`.
- **Section 5.3 “Using the Fixed-Point Libraries”** - updated descriptions for `Q16ftoi` and `itofQ16`.
- Minor typographical changes.

## Revision E

- **Section 2.4 “<ctype.h> character handling”** - the following function descriptions have been updated:
  - `islower`
  - `isxdigit`
  - `tolower`
  - `toupper`
- **Section 2.6 “<float.h> floating-point characteristics”**- the following function descriptions have been updated:
  - `FLT_MAX_EXP`
  - `FLT_MIN`
  - `FLT_MIN_EXP`
  - `LDBL_MIN`
  - `LDBL_MIN_EXP`
  - `UCHAR_MAX`
  - `UINT_MAX`
  - `ULLONG_MAX`
  - `ULONG_MAX`
  - `USHRT_MAX`
- **Section 2.9 “<setjmp.h> non-local jumps”** - the following function descriptions have been updated:
  - `jmp_buf`
  - `longjmp`
  - `SIGILL`
  - `SIGSEGV`
  - `raise`
  - `signal`
- Divided **Chapter 3. “Standard C Libraries - Math Functions”** into **Chapter 4. “Standard C Libraries - Support Functions”** and **Chapter 5. “Fixed-Point Math Functions”**
- Minor typographical changes.



## Revision D

- **Section 3.7 “<limits.h> implementation-defined limits”** - the following function descriptions have been updated:
  - LLONG\_MIN
  - LONG\_MAX
  - LONG\_MIN
  - MB\_LEN\_MAX
  - SCHAR\_MAX
  - SCHAR\_MIN
  - SHRT\_MAX
  - SHRT\_MIN
  - UCHAR\_MAX
- **Section 3.9 “<setjmp.h> non-local jumps”** - the following function descriptions have been updated:
  - jmp\_buf
  - setjmp
  - longjmp
- **Section 3.10 “<signal.h> signal handling”** - the following function descriptions have been updated:
  - SIGFPE
  - SIGILL
  - SIGINT
  - raise
- **Section 3.13 “<stdio.h> input and output”** - the following function descriptions have been updated:
  - FILE
  - fpos\_t
  - \_IOFBF
  - \_IOLBF
  - \_IONBF
  - BUFSIZ
  - FOPEN\_MAX
  - L\_tmpnam
  - SEEK\_CUR
  - stderr
  - stdout
  - TMP\_MAX
  - clearerr
  - feof
  - fgetpos
  - fopen
  - fprintf
  - fread
- Minor typographical changes

## Revision C

- Updated **Section 2.8 “<locale.h> localization”** and the following function descriptions:
  - PIDInit
  - PIDCoeffCalc
  - PID
- Updated **Section 2.9 “<setjmp.h> non-local jumps”** and the following function descriptions:
  - Fract2Float
  - Float2Fract
- Updated **Section 3.2 “Using the Standard C Libraries”**
- **Section 3.3 “<math.h> mathematical functions”** - the following function descriptions have been updated:
  - OpenXLCD
  - putsXLCD
  - SetDDRamAddr
  - WriteDataXLCDWriteCmdXLCD
- Divided **Section 3.3 “<math.h> mathematical functions”** into **Section 3.4 “CAN Functions”**. The following function descriptions have been updated:
  - CAN1AbortAll
  - CAN1GetRXErrorCount
  - CAN1GetTXErrorCount
  - CAN1IsBusOff
  - CAN1IsRXReady
  - CAN1IsRXPassive
  - CAN1IsTXPassive
- **Section 5.2 “Overview of Fixed-Point Data Formats”** - the following function descriptions have been updated:
  - \_builtin\_addab
  - \_builtin\_add
  - \_builtin\_btg
  - \_builtin\_clr
  - \_builtin\_clr\_prefetch
  - \_builtin\_dmaoffset
  - \_builtin\_ed
  - \_builtin\_edac
  - \_builtin\_fbcl
  - \_builtin\_mac
  - \_builtin\_movsac
  - \_builtin\_mpy
  - \_builtin\_mpy\_n
  - \_builtin\_msc
  - \_builtin\_mulss
  - \_builtin\_mulsu
  - \_builtin\_mulus
  - \_builtin\_muluu
  - \_builtin\_nop
  - \_builtin\_psvpage
- Minor typographical changes

# Document Revision History

---

## Revision B

- Added "NOTICE TO CUSTOMERS"
- Divided **Chapter 4. "Standard C Libraries - Support Functions"** into **Chapter 5. "Fixed-Point Math Functions"**
- Added **Section 1.2 "OMF-Specific Libraries/Start-up Modules"**
- Minor typographical changes

## Revision A

- Initial release of this document.

**Index**

**Symbols**

^, Caret.....	86	_Q15sin.....	232
_attach_input_file.....	205	_Q15sinPI.....	232
_C30_UART.....	206	_Q15sinSeries.....	232
_close_input_file.....	205	_Q15sqrt.....	233
_delay_ms.....	206	_Q15sub.....	233
_delay_us.....	206	_Q15tan.....	233
_delay32.....	206	_Q15tanPI.....	233
_FILE.....	17	_Q16acos.....	234
_LINE.....	17	_Q16acosByPI.....	234
_dump_heap_info.....	199	_Q16asin.....	234
_erase_eedata.....	208	_Q16asinByPI.....	234
_erase_eedata_all.....	208	_Q16atan.....	235
_erase_flash.....	211	_Q16atanByPI.....	235
_exit.....	199	_Q16atanYByX.....	235
_init_prog_address.....	217	_Q16atanYByXByPI.....	235
_IOFBF.....	50, 87, 88	_Q16cos.....	236
_IOLBF.....	50, 88	_Q16cosPI.....	236
_IONBF.....	50, 87, 88	_Q16div.....	236
_itoaQ15.....	229	_Q16divmod.....	236
_itofQ15.....	229	_Q16exp.....	237
_itofQ16.....	243	_Q16ftoi.....	243
_MathError.....	41	_Q16log.....	237
_memcpy_p2d16.....	216	_Q16log10.....	237
_memcpy_p2d24.....	216	_Q16mac (Continued).....	237
_NSETJMP.....	36	_Q16macNoSat.....	238
_Q15abs.....	225	_Q16mpy.....	238
_Q15acos.....	225	_Q16neg.....	238
_Q15acosByPI.....	225	_Q16norm.....	238
_Q15add.....	226	_Q16power.....	239
_Q15asin.....	226	_Q16random.....	239
_Q15asinByPI.....	226	_Q16reciprocal.....	239
_Q15atan.....	226	_Q16reciprocalQ15.....	239
_Q15atanByPI.....	227	_Q16reciprocalQ16.....	240
_Q15atanYByX.....	227	_Q16shl.....	240
_Q15atanYByXByPI.....	227	_Q16shlNoSat.....	240
_Q15atoi.....	227	_Q16shr.....	240
_Q15cos.....	228	_Q16shrNoSat.....	241
_Q15cosPI.....	228	_Q16sin.....	241
_Q15exp.....	228	_Q16sinPI.....	241
_Q15ftoi.....	228	_Q16sinSeries.....	241
_Q15log.....	229	_Q16tan.....	242
_Q15log10.....	229	_Q16tanPI.....	242
_Q15neg.....	230	_strncpy_p2d16.....	216
_Q15norm.....	230	_strncpy_p2d24.....	217
_Q15power.....	230	_VERBOSE_DEBUGGING.....	17
_Q15random.....	230	_wait_eedata.....	209
_Q15shl.....	231	_write_eedata_row.....	209
_Q15shlNoSat.....	231	_write_eedata_word.....	209
_Q15shr.....	231	_write_flash_word16.....	212
_Q15shrNoSat.....	231	_write_flash_word24.....	212
		_write_flash_word32.....	213

_write_flash_word48 .....	213
_write_flash16 .....	211
_write_flash24 .....	212
-, Dash .....	86
\f, Form Feed .....	23
\n, Newline .....	23, 47, 61, 66, 77, 78, 82
\r, Carriage Return .....	23
\t, Horizontal Tab .....	23
\v, Vertical Tab .....	23
#if .....	33
#include .....	16, 153, 198, 223
%, Percent .....	80, 85, 86, 151
<b>Numerics</b>	
0x .....	24, 79, 119, 120
<b>A</b>	
Abnormal Termination Signal .....	38
abort .....	17, 98
abs .....	99
Absolute Value	
Double Floating Point .....	169
Integer .....	99
Long Integer .....	109
Single Floating Point .....	169
Absolute Value Function	
abs .....	99
fabs .....	169
fabsf .....	169
labs .....	109
Access Mode	
Binary .....	55, 62
Text .....	55, 62
acos .....	155
acosf .....	156
Allocate Memory .....	111
calloc .....	105
Free .....	108
realloc .....	115
Alphabetic Character	
Defined .....	18
Test for .....	18
Alphanumeric Character	
Defined .....	18
Test for .....	18
AM/PM .....	151
Append .....	128, 134
arccosine	
Double Floating Point .....	155
Single Floating Point .....	156
arcsine	
Double Floating Point .....	157
Single Floating Point .....	157
arctangent	
Double Floating Point .....	158
Single Floating Point .....	159
arctangent of y/x	
Double Floating Point .....	159
Single Floating Point .....	161
Argument List .....	43, 93, 94, 95, 96
Arithmetic Error Message .....	38

ASCII Character Set .....	245
asctime .....	146
asin .....	157
asinf .....	157
assert .....	17
assert.h .....	17
Assignment Suppression .....	85
Asterisk .....	79, 85
atan .....	158
atan2 .....	159
atan2f .....	161
atanf .....	159
atexit .....	99, 107
atof .....	101
atoi .....	102
atol .....	102
attach_input_file .....	205
<b>B</b>	
Base .....	119, 120
10 .....	29, 30, 31, 32, 181, 182
2 .....	31
e .....	180, 183
FLT_RADIX .....	28, 29, 30, 31, 32, 33
Binary	
Base .....	31
Mode .....	55, 62, 90
Search .....	103
Streams .....	47
Bitfields .....	46
brk .....	200, 203
bsearch .....	103
Buffer Size .....	50, 88
Buffering Modes .....	88
Buffering, See File Buffering	
BUFSIZ .....	50, 87
Built-in Functions .....	13
<b>C</b>	
C Locale .....	18, 35
C30_UART .....	206
Calendar Time .....	145, 147, 148, 150, 152
calloc .....	105, 108
Caret (^) .....	86
Carriage Return .....	23
ceil .....	162
ceilf .....	163
ceiling	
Double Floating Point .....	162
Single Floating Point .....	163
char	
Maximum Value .....	33
Minimum Value .....	33
Number of Bits .....	33
CHAR_BIT .....	33
CHAR_MAX .....	33
CHAR_MIN .....	33
Character Array .....	86
Character Case Mapping	
Lower Case Alphabetic Character .....	25
Upper Case Alphabetic Character .....	26

# 16-Bit Language Tools Libraries

---

Character Case Mapping Functions		
tolower .....	25	
toupper .....	26	
Character Handling, See ctype.h		
Character Input/Output Functions		
fgetc .....	59	
fgets .....	61	
fputc .....	65	
fputs .....	65	
getc .....	76	
getchar .....	77	
gets .....	77	
putc .....	81	
putchar .....	82	
puts .....	82	
ungetc .....	91	
Character Testing		
Alphabetic Character .....	18	
Alphanumeric Character .....	18	
Control Character .....	19	
Decimal Digit .....	20	
Graphical Character .....	20	
Hexadecimal Digit .....	24	
Lower Case Alphabetic Character .....	21	
Printable Character .....	22	
Punctuation Character .....	22	
Upper Case Alphabetic Character .....	24	
White-Space Character .....	23	
Character Testing Functions		
isalnum .....	18	
isalpha .....	18	
iscntrl .....	19	
isdigit .....	20	
isgraph .....	20	
islower .....	21	
isprint .....	22	
ispunct .....	22	
isspace .....	23	
isupper .....	24	
isxdigit .....	24	
Characters		
Alphabetic .....	18	
Alphanumeric .....	18	
Control .....	19	
Convert to Lower Case Alphabetic .....	25	
Convert to Upper Case Alphabetic .....	26	
Decimal Digit .....	20	
Graphical .....	20	
Hexadecimal Digit .....	24	
Lower Case Alphabetic .....	21	
Printable .....	22	
Punctuation .....	22	
Upper Case Alphabetic .....	24	
White-Space .....	23	
Classifying Characters .....	18	
clearerr .....	53	
Clearing Error Indicator .....	53	
clock .....	146	
clock_t .....	145, 146	
CLOCKS_PER_SEC .....	145	
close .....	201	
close_input_file .....	205	
COFF .....	12	
Common Definitions, See stddef.h		
Compare Strings .....	130	
Comparison Function .....	103, 114	
Comparison Functions		
memcmp .....	123	
strcmp .....	130	
strcoll .....	131	
strncmp .....	136	
strxfrm .....	144	
Compiler Options		
-fno-short-double .....	48	
-msmart-io .....	47	
Concatenation Functions		
strcat .....	128	
strncat .....	134	
Control Character		
Defined .....	19	
Test for .....	19	
Control Transfers .....	36	
Conversion .....	79, 85, 89	
Convert		
Character to Multibyte Character .....	121	
Multibyte Character to Wide Character .....	113	
Multibyte String to Wide Character String .....	113	
String to Double Floating Point .....	101, 118	
String to Integer .....	102	
String to Long Integer .....	102, 119	
String to Unsigned Long Integer .....	120	
To Lower Case Alphabetic Character .....	25	
To Upper Case Alphabetic Character .....	26	
Wide Character String to Multibyte String .....	121	
Copying Functions		
memcpy .....	125	
memmove .....	126	
memset .....	127	
strcpy .....	131	
strncpy .....	137	
cos .....	163	
cosf .....	164	
cosh .....	165	
coshf .....	166	
cosine		
Double Floating Point .....	163	
Single Floating Point .....	164	
crt0, crt1 .....	12	
ctime .....	147	
ctype.h .....	18	
isalnum .....	18	
iscntrl .....	19	
isdigit .....	20	
isgraph .....	20	
isalpha .....	18	
islower .....	21	
isprint .....	22	
ispunct .....	22	
isspace .....	23	
isupper .....	24	

isxdigit .....	24	Minimum Exponent (base 10) .....	29
tolower .....	25	Minimum Exponent (base 2) .....	30
toupper .....	26	Minimum Value .....	29
Current Argument .....	43	Number of Binary Digits .....	28
Customer Notification Service .....	8	Number of Decimal Digits .....	28
Customer Support .....	9	double Type .....	48
Customized Function .....	108	Dream Function .....	79
<b>D</b>		dump heap info .....	199
Dash (-) .....	86	Duplicate a string .....	117
Date and Time .....	150	DWARF .....	12
Date and Time Functions, <i>See</i> time.h		<b>E</b>	
Day of the Month .....	145, 146, 150	EDOM .....	27
Day of the Week .....	145, 146, 150	edom .....	155
Day of the Year .....	145, 151	ELF .....	12
Daylight Savings Time .....	145, 148, 149	Ellipses (...) .....	43, 86
DBL_DIG .....	28	Empty Binary File .....	55, 62
DBL_EPSILON .....	28	Empty Text File .....	55, 62
DBL_MANT_DIG .....	28	End Of File .....	50
DBL_MAX .....	28	Indicator .....	47
DBL_MAX_10_EXP .....	29	Seek .....	70
DBL_MAX_EXP .....	29	Test For .....	57
DBL_MIN .....	29	Environment Function	
DBL_MIN_10_EXP .....	29	getenv .....	108
DBL_MIN_EXP .....	30	Environment Variable .....	204
Deallocate Memory .....	108, 115	EOF .....	50
Debugging Logic Errors .....	17	ERANGE .....	27
Decimal .....	80, 86, 119, 120	erange .....	155
Decimal Digit		erase_eedata .....	208
Defined .....	20	erase_eedata_all .....	208
Number Of .....	28, 30, 31	erase_flash .....	211
Test for .....	20	errno .....	27, 155
Decimal Point .....	79	errno.h .....	27, 155, 202
Default Handler .....	37	EDOM .....	27
delay_ms .....	206	ERANGE .....	27
delay_us .....	206	errno .....	27
delay32 .....	206	Error Codes .....	27, 133
Diagnostics, <i>See</i> assert.h		Error Conditions .....	155
difftime .....	148	Error Handler .....	105
Digit, Decimal, <i>See</i> Decimal Digit		Error Handling Functions	
Digit, Hexadecimal, <i>See</i> Hexadecimal Digit		clearerr .....	53
Direct Input/Output Functions		feof .....	57
fread .....	66	ferror .....	58
fwrite .....	74	perror .....	78
div .....	97, 105	Error Indicator .....	47
div_t .....	97	Error Indicators	
Divide		Clearing .....	53, 84
Integer .....	105	End Of File .....	53, 61
Long Integer .....	110	Error .....	53, 61
Divide by Zero .....	38, 41, 105	Test For .....	58
Documentation		Error Signal .....	37
Conventions .....	6	Errors, <i>See</i> errno.h	
Layout .....	5	Errors, Testing For .....	27
Domain Error.....	27, 155, 156, 157, 159, 161, 163, 164, 171, 173, 180, 181, 182, 183, 188, 189, 192, 193, 194	Exception Error .....	105
dot .....	79	exit .....	90, 97, 99, 107, 199
Double Precision Floating Point		EXIT_FAILURE .....	97
Machine Epsilon .....	28	EXIT_SUCCESS .....	97
Maximum Exponent (base 10) .....	29	exp .....	167
Maximum Exponent (base 2) .....	29	expf .....	168
Maximum Value .....	28	Exponential and Logarithmic Functions	
		exp .....	167

# 16-Bit Language Tools Libraries

---

expf .....	168	DBL_MAX_EXP .....	29
frexp .....	175	DBL_MIN .....	29
frexpf .....	176	DBL_MIN_10_EXP .....	29
ldexp .....	177	DBL_MIN_EXP .....	30
ldexpf .....	178	FLT_DIG .....	30
log .....	180	FLT_EPSILON .....	30
log10 .....	181	FLT_MANT_DIG .....	30
log10f .....	182	FLT_MAX .....	30
logf .....	183	FLT_MAX_10_EXP .....	30
modf .....	184	FLT_MAX_EXP .....	31
modff .....	185	FLT_MIN .....	31
Exponential Function		FLT_MIN_10_EXP .....	31
Double Floating Point .....	167	FLT_MIN_EXP .....	31
Single Floating Point .....	168	FLT_RADIX .....	31
<b>F</b>		FLT_ROUNDS .....	31
fabs .....	169	LDBL_DIG .....	31
fabsf .....	169	LDBL_EPSILON .....	32
fclose .....	54, 201	LDBL_MANT_DIG .....	32
fdopen .....	55	LDBL_MAX .....	32
feof .....	53, 57	LDBL_MAX_10_EXP .....	32
ferror .....	53, 58	LDBL_MAX_EXP .....	32
fflush .....	59, 203	LDBL_MIN .....	32
fgetc .....	59, 202	LDBL_MIN_10_EXP .....	32
fgetpos .....	60, 201	LDBL_MIN_EXP .....	33
fgets .....	61, 202	Floating Point	
Field Width .....	79	Limits .....	28
FILE .....	17, 47, 50	No Conversion .....	47
File Access Functions		Types, Properties Of .....	28
fclose .....	54	Floating Point, See float.h	
fdopen .....	55	Floating-Point Error Signal .....	38
fflush .....	59	floor .....	170
fopen .....	62	Double Floating Point .....	170
freopen .....	68	Single Floating Point .....	170
setbuf .....	87	floorf .....	170
setvbuf .....	88	FLT_DIG .....	30
File Access Modes .....	47, 55, 62	FLT_EPSILON .....	30
File Buffering		FLT_MANT_DIG .....	30
Fully Buffered .....	47, 50	FLT_MAX .....	30
Line Buffered .....	47, 50	FLT_MAX_10_EXP .....	30
Unbuffered .....	47, 50	FLT_MAX_EXP .....	31
File Operations		FLT_MIN .....	31
Remove .....	83	FLT_MIN_10_EXP .....	31
Rename .....	83	FLT_MIN_EXP .....	31
File Positioning Functions		FLT_RADIX .....	31
fgetpos .....	60	FLT_RADIX Digit	
fseek .....	70	Number Of .....	28, 30, 32
fsetpos .....	71	FLT_ROUNDS .....	31
ftell .....	73	Flush .....	59, 107
rewind .....	84	fmod .....	171
FILENAME_MAX .....	51	fmodf .....	173
File-Position Indicator .....	47, 50, 59, 60, 65, 66, 71, 74	-fno-short-double .....	28, 29, 30, 48
Files, Maximum Number Open .....	51	fopen .....	47, 62, 88, 202
Fixed Point Math Library .....	219	FOPEN_MAX .....	51
flags .....	79	Form Feed .....	23
float.h .....	28	Format Specifiers .....	79, 85
DBL_DIG .....	28	Formatted I/O Routines .....	47
DBL_EPSILON .....	28	Formatted Input/Output Functions	
DBL_MANT_DIG .....	28	fprintf .....	64
DBL_MAX .....	28	fscanf .....	68
DBL_MAX_10_EXP .....	29	printf .....	79



scanf .....	85	limits.h .....	33
sprintf .....	89	locale.h .....	35
sscanf .....	89	math.h .....	155
vasprintf .....	93	setjmp.h .....	36
vfprintf .....	94	signal.h .....	37
vprintf .....	95	stdarg.h .....	43
vsprintf .....	96	stddef.h .....	45
Formatted Text		stdio.h .....	47, 204
Printing .....	89	stdlib.h .....	97, 204
Scanning .....	89	string.h .....	122
fpos_t .....	50	time.h .....	145, 205
fprintf .....	47, 64	Heap .....	200
fputc .....	65	Hexadecimal .....	80, 86, 119, 120
fputs .....	65	Hexadecimal Conversion .....	79
fraction and exponent function		Hexadecimal Digit	
Double Floating Point .....	175	Defined .....	24
Single Floating Point .....	176	Test for .....	24
Fraction Digits .....	79	Horizontal Tab .....	23
fread .....	66, 202	Hour .....	145, 146, 150
free .....	108	HUGE_VAL .....	155
Free Memory .....	108	Hyperbolic Cosine	
freopen .....	47, 68, 202	Double Floating Point .....	165
frexp .....	175	Single Floating Point .....	166
frexpf .....	176	Hyperbolic Functions	
fscanf .....	47, 68	cosh .....	165
fseek .....	70, 91, 201	coshf .....	166
fsetpos .....	71, 91, 201	sinh .....	190
ftell .....	73, 201	sinhf .....	191
Full Buffering .....	87, 88	tanh .....	195
Fully Buffered .....	47, 50	tanhf .....	196
fwrite .....	74	Hyperbolic Sine	
<b>G</b>		Double Floating Point .....	190
getc .....	76	Single Floating Point .....	191
getchar .....	77	Hyperbolic Tangent	
getenv .....	108, 204	Double Floating Point .....	195
gets .....	77, 202	hyperbolic tangent	
GMT .....	148	Single Floating Point .....	196
gmtime .....	148, 149	<b>I</b>	
Graphical Character		Ignore Signal .....	37
Defined .....	20	Illegal Instruction Signal .....	39
Test for .....	20	Implementation-Defined Limits, See limits.h	
Greenwich Mean Time .....	148	Indicator	
<b>H</b>		End Of File .....	47, 50
h modifier .....	80, 85	Error .....	47, 58
Handler		File Position .....	47, 59, 60, 65, 66, 71, 74
Default .....	37	Infinity .....	155
Error .....	105	init_prog_address .....	217
Interrupt .....	41	Input and Output, See stdio.h	
Nested .....	36	Input Formats .....	47
Signal .....	37, 42	Instruction Cycles .....	148, 149, 152
Signal Type .....	37	int	
Handling		Maximum Value .....	33
Interrupt Signal .....	42	Minimum Value .....	33
Header Files		INT_MAX .....	33
assert.h .....	17	INT_MIN .....	33
ctype.h .....	18	Integer Limits .....	33
errno.h .....	27, 155, 202	Internal Error Message .....	133
float.h .....	28	Internet Address, Microchip .....	8
libq.h .....	225	Interrupt Handler .....	41
		Interrupt Signal .....	39

# 16-Bit Language Tools Libraries

---

Interrupt Signal Handling .....	42	Fixed Point Math .....	219
Interruption Message .....	39	Math .....	153
Invalid Executable Code Message .....	39	Standard C .....	15
Invalid Storage Request Message .....	40	Standard C Math .....	155
Inverse Cosine, See arccosine		Support .....	197
Inverse Sine, See arcsine		Limits	
Inverse Tangent, See arctangent		Floating Point .....	28
IOFBF .....	50, 87, 88	Integer .....	33
IOLBF .....	50, 88	limits.h .....	33
IONBF .....	50, 87, 88	CHAR_BITS .....	33
isalnum .....	18	CHAR_MAX .....	33
iscntrl .....	19	CHAR_MIN .....	33
isdigit .....	20	INT_MAX .....	33
isgraph .....	20	INT_MIN .....	33
isalpha .....	18	LLONG_MAX .....	33
islower .....	21	LLONG_MIN .....	34
isprint .....	22	LONG_MAX .....	34
ispunct .....	22	LONG_MIN .....	34
isspace .....	23	MB_LEN_MAX .....	34
isupper .....	24	SCHAR_MAX .....	34
isxdigit .....	24	SCHAR_MIN .....	34
<b>J</b>		SHRT_MAX .....	34
jmp_buf .....	36	SHRT_MIN .....	34
Justify .....	79	UCHAR_MAX .....	35
<b>L</b>		UINT_MAX .....	35
L modifier .....	80, 85	ULLONG_MAX .....	35
l modifier .....	80, 85	ULONG_MAX .....	35
L_tmpnam .....	51, 91	USHRT_MAX .....	35
labs .....	109	LINE .....	17
LC_ALL .....	35	Line Buffered .....	47, 50
LC_COLLATE .....	35	Line Buffering .....	88
LC_CTYPE .....	35	ll modifier .....	80, 85
LC_MONETARY .....	35	LLONG_MAX .....	33
LC_NUMERIC .....	35	LLONG_MIN .....	34
LC_TIME .....	35	Load Exponent Function	
lconv, struct .....	35	Double Floating Point .....	177
LDBL_DIG .....	31	Single Floating Point .....	178
LDBL_EPSILON .....	32	Local Time .....	147, 149, 150
LDBL_MANT_DIG .....	32	Locale, C .....	18, 35
LDBL_MAX .....	32	Locale, Other .....	35
LDBL_MAX_10_EXP .....	32	locale.h .....	35
LDBL_MAX_EXP .....	32	localeconv .....	35
LDBL_MIN .....	32	Localization, See locale.h	
LDBL_MIN_10_EXP .....	32	localtime .....	147, 148, 149
LDBL_MIN_EXP .....	33	Locate Character .....	129
ldexp .....	177	log .....	180
ldexpf .....	178	log10 .....	181
ldiv .....	97, 110	log10f .....	182
ldiv_t .....	97	Logarithm Function	
Leap Second .....	145, 151	Double Floating Point .....	181
Left Justify .....	79	Single Floating Point .....	182
libc .....	13	Logarithm Function, Natural	
libdsp .....	12	Double Floating Point .....	180
libm .....	13	Single Floating Point .....	183
libp .....	12	logf .....	183
libpic30 .....	13	Logic Errors, Debugging .....	17
libpic30, Rebuilding .....	198	Long Double Precision Floating Point	
libq.h .....	225	Machine Epsilon .....	32
Libraries		Maximum Exponent (base 10) .....	32
		Maximum Exponent (base 2) .....	32

Maximum Value .....	32	HUGE_VAL.....	155
Minimum Exponent (base 10).....	32	ldexp .....	177
Minimum Exponent (base 2).....	33	ldexpf .....	178
Minimum Value .....	32	log .....	180
Number of Binary Digits.....	32	log10 .....	181
Number of Decimal Digits.....	31	log10f .....	182
long double Type .....	48	logf .....	183
long int		modf .....	184
Maximum Value .....	34	modff .....	185
Minimum Value .....	34	pow .....	186
long long int		powf .....	187
Maximum Value .....	33	sin .....	188
Minimum Value .....	34	sinf .....	189
long long unsigned int		sinh .....	190
Maximum Value .....	35	sinhf .....	191
long unsigned int		sqrt .....	192
Maximum Value .....	35	sqrtf .....	193
LONG_MAX.....	34	tan .....	194
LONG_MIN .....	34	tanf .....	194
longjmp .....	36	tanh .....	195
Lower Case Alphabetic Character		tanhf .....	196
Convert To .....	25	Mathematical Functions, See libq.h	
Defined .....	21	Mathematical Functions, See math.h	
Test for.....	21	MathError .....	41
lseek.....	201	Maximum	
<b>M</b>		Multibyte Character .....	98
Machine Epsilon		Maximum Value	
Double Floating Point .....	28	Double Floating-Point Exponent (base 10) .....	29
Long Double Floating Point .....	32	Double Floating-Point Exponent (base 2) .....	29
Single Floating Point .....	30	Long Double Floating-Point Exponent (base 10) ..	32
Magnitude .....	155, 167, 168, 171, 173, 190, 191	Long Double Floating-Point Exponent (base 2) ..	32
malloc.....	108, 111, 200, 203	Multibyte Character .....	34
Mapping Characters.....	18	rand.....	98
Math Exception Error .....	105	Single Floating-Point Exponent (base 10) .....	30
Math Library .....	153	Single Floating-Point Exponent (base 2) .....	31
math.h .....	155	Type char .....	33
acos. ....	155	Type Double.....	28
acosf. ....	156	Type int .....	33
asin. ....	157	Type Long Double.....	32
asinf .....	157	Type long int .....	34
atan.....	158	Type long long int.....	33
atan2.....	159	Type long long unsigned int.....	35
atan2f.....	161	Type long unsigned int.....	35
atanf.....	159	Type short int .....	34
ceil .....	162	Type signed char .....	34
ceilf .....	163	Type Single .....	30
cos .....	163	Type unsigned char .....	35
cosf .....	164	Type unsigned int.....	35
cosh .....	165	Type unsigned short int.....	35
coshf .....	166	MB_CUR_MAX .....	98
exp .....	167	MB_LEN_MAX .....	34
expf .....	168	mblen .....	113
fabs .....	169	mbstowcs .....	113
fabsf.....	169	mbtowc.....	113
floor.....	170	memchr .....	122
floorf.....	170	memcmp .....	123
fmod.....	171	memcpy.....	125
fmodf.....	173	memcpy_p2d16.....	216
frexp.....	175	memcpy_p2d24.....	216
frexpf.....	176		

# 16-Bit Language Tools Libraries

---

memmove .....	126
Memory	
Allocate .....	105, 111
Deallocate .....	108
Free .....	108
Reallocate .....	115
memset .....	127
Message	
Arithmetic Error .....	38
Interrupt .....	39
Invalid Executable Code .....	39
Invalid Storage Request .....	40
Termination Request .....	40
Minimum Value	
Double Floating-Point Exponent (base 10) .....	29
Double Floating-Point Exponent (base 2) .....	30
Long Double Floating-Point Exponent (base 10) ..	32
Long Double Floating-Point Exponent (base 2) ..	33
Single Floating-Point Exponent (base 10) .....	31
Single Floating-Point Exponent (base 2) .....	31
Type char .....	33
Type Double .....	29
Type int .....	33
Type Long Double .....	32
Type long int .....	34
Type long long int .....	34
Type short int .....	34
Type signed char .....	34
Type Single .....	31
Minute .....	145, 146, 151
mktime .....	150
modf .....	184
modff .....	185
modulus function	
Double Floating Point .....	184
Single Floating Point .....	185
Month .....	145, 146, 150, 151
-msmart-io .....	47
Multibyte Character .....	98, 113, 121
Maximum Number of Bytes .....	34
Multibyte String .....	113, 121
<b>N</b>	
NaN .....	155
Natural Logarithm	
Double Floating Point .....	180
Single Floating Point .....	183
NDEBUG .....	17
Nearest Integer Functions	
ceil .....	162
ceilf .....	163
floor .....	170
floorf .....	170
Nested Signal Handler .....	36
Newline .....	23, 47, 61, 66, 77, 78, 82
No Buffering .....	47, 50, 87, 88
Non-Local Jumps, See setjmp.h	
NSETJMP .....	36
NULL .....	35, 45, 51, 98, 122, 146

## O

Object Module Format .....	12
Octal .....	80, 86, 119, 120
Octal Conversion .....	79
offsetof .....	46
OMF .....	12
open .....	201
Output Formats .....	47
Overflow Errors .....	27, 155, 167, 168, 177, 178, 186, 187
Overlap .....	125, 126, 128, 131, 134, 137

## P

Pad Characters .....	79
Percent .....	80, 85, 86, 151
Peripheral Libraries .....	12
perror .....	78
pic30-libs	
__attach_input_file .....	205
__C30_UART .....	206
__close_input_file .....	205
__delay_ms .....	206
__delay_us .....	206
__delay32 .....	206
__dump_heap_info .....	199
__erase_eedata .....	208
__erase_eedata_all .....	208
__erase_flash .....	211
__exit .....	199
__init_prog_address .....	217
__memcpy_p2d16 .....	216
__memcpy_p2d24 .....	216
__strncpy_p2d16 .....	216
__strncpy_p2d24 .....	217
__wait_eedata .....	209
__write_eedata_row .....	209
__write_eedata_word .....	209
__write_flash_word16 .....	212
__write_flash_word24 .....	212
__write_flash_word32 .....	213
__write_flash_word48 .....	213
__write_flash16 .....	211
__write_flash24 .....	212
brk .....	200
close .....	201
getenv .....	204
lseek .....	201
open .....	201
read .....	202
remove .....	204
rename .....	204
sbrk .....	202
system .....	204
time .....	205
write .....	203
Plus Sign .....	79
Pointer, Temporary .....	115
pow .....	186
Power Function	
Double Floating Point .....	186
Single Floating Point .....	187
Power Functions	

pow .....	186	Search Functions	
powf .....	187	memchr .....	122
powf .....	187	strchr .....	129
precision .....	79	strcspn .....	132
Prefix .....	24, 79	strpbrk .....	139
Print Formats .....	47	strrchr .....	140
Printable Character		strspn .....	141
Defined .....	22	strstr .....	142
Test for .....	22	strtok .....	143
printf .....	47, 79	Second .....	145, 146, 148, 151
Processor Clocks per Second .....	145	Seed .....	115, 117
Processor Time .....	145, 146	Seek	
Pseudo-Random Number .....	115, 117	From Beginning of File .....	70
ptrdiff_t .....	45	From Current Position .....	70
Punctuation Character		From End Of File .....	70
Defined .....	22	SEEK_CUR .....	51, 70
Test for .....	22	SEEK_END .....	51, 70
Pushed Back .....	91	SEEK_SET .....	51, 70
putc .....	81	setbuf .....	47, 50, 87
putchar .....	82	setjmp .....	36
puts .....	82	setjmp.h .....	36
<b>Q</b>		jmp_buf .....	36
Q15 Functions .....	225	longjmp .....	36
Q16 Functions .....	234	setjmp .....	36
qsort .....	103, 114	setlocale .....	35
Quick Sort .....	114	setvbuf .....	47, 50, 88
<b>R</b>		short int	
Radix .....	31	Maximum Value .....	34
raise .....	37, 38, 39, 40, 41, 42	Minimum Value .....	34
rand .....	115, 117	SHRT_MAX .....	34
RAND_MAX .....	98, 115	SHRT_MIN .....	34
Range .....	86	sig_atomic_t .....	37
Range Error 27, 119, 120, 165, 166, 167, 168, 177, 178,		SIG_DFL .....	37
..... 186, 187, 190, 191		SIG_ERR .....	37
read .....	202	SIG_IGN .....	37
Reading, Recommended .....	7	SIGABRT .....	38
realloc .....	108, 115	SIGFPE .....	38
Reallocate Memory .....	115	SIGILL .....	39
Rebuilding the libpic30 library .....	198	SIGINT .....	39
Registered Functions .....	99, 107	Signal	
Remainder		Abnormal Termination .....	38
Double Floating Point .....	171	Error .....	37
Single Floating Point .....	173	Floating-Point Error .....	38
Remainder Functions		Ignore .....	37
fmod .....	171	Illegal Instruction .....	39
fmodf .....	173	Interrupt .....	39
remove .....	83, 204	Reporting .....	41
rename .....	83, 204	Termination Request .....	40
Reset .....	98, 121	signal .....	38, 39, 40, 42
Reset File Pointer .....	84	Signal Handler .....	37, 42
rewind .....	84, 91, 201	Signal Handler Type .....	37
Rounding Mode .....	31	Signal Handling, See signal.h	
<b>S</b>		signal.h .....	37
sbrk .....	200, 202	raise .....	41
Scan Formats .....	47	sig_atomic_t .....	37
scanf .....	47, 85	SIG_DFL .....	37
SCHAR_MAX .....	34	SIG_ERR .....	37
SCHAR_MIN .....	34	SIG_IGN .....	37
		SIGABRT .....	38
		SIGFPE .....	38

# 16-Bit Language Tools Libraries

---

SIGILL.....	39	va_arg.....	43
SIGINT.....	39	va_end.....	45
signal.....	42	va_list.....	43
SIGSEGV.....	40	va_start.....	45
SIGTERM.....	40	stddef.h.....	45
signed char		NULL.....	45
Maximum Value.....	34	offsetof.....	46
Minimum Value.....	34	ptrdiff_t.....	45
SIGSEGV.....	40	size_t.....	45
SIGTERM.....	40	wchar_t.....	45
sim30 simulator.....	198	stderr.....	17, 47, 51, 78
sin.....	188	stdin.....	47, 51, 52, 77, 85
sine		stdio.h.....	47, 204
Double Floating Point.....	188	_IIOBF.....	50
Single Floating Point.....	189	_ILOBF.....	50
sinf.....	189	_IONBF.....	50
Single Precision Floating Point		BUFSIZ.....	50
Machine Epsilon.....	30	clearerr.....	53
Maximum Exponent (base 10).....	30	EOF.....	50
Maximum Exponent (base 2).....	31	fclose.....	54
Maximum Value.....	30	fdopen.....	55
Minimum Exponent (base 10).....	31	feof.....	57
Minimum Exponent (base 2).....	31	ferror.....	58
Minimum Value.....	31	fflush.....	59
Number of Binary Digits.....	30	fgetc.....	59
Number of Decimal Digits.....	30	fgetpos.....	60
sinh.....	190	fgets.....	61
sinhf.....	191	FILE.....	50
size.....	80	FILENAME_MAX.....	51
size_t.....	45, 50, 97, 122, 145	fopen.....	62
sizeof.....	45, 50, 97, 122, 145	FOPEN_MAX.....	51
Sort, Quick.....	114	fpos_t.....	50
Source File Name.....	17	fprintf.....	64
Source Line Number.....	17	fputc.....	65
Space.....	79	fputs.....	65
Space Character		fread.....	66
Defined.....	23	freopen.....	68
Test for.....	23	fscanf.....	68
Specifiers.....	79, 85	fseek.....	70
sprintf.....	47, 89	fsetpos.....	71
sqrt.....	192	ftell.....	73
sqrtf.....	193	fwrite.....	74
Square Root Function		getc.....	76
Double Floating Point.....	192	getchar.....	77
Single Floating Point.....	193	gets.....	77
Square Root Functions		L_tmpnam.....	51
sqrt.....	192	NULL.....	51
sqrtf.....	193	perror.....	78
srand.....	117	printf.....	79
sscanf.....	47, 89	putc.....	81
Stack.....	200	putchar.....	82
Standard C Libraries.....	15	puts.....	82
Standard C Locale.....	18	remove.....	83
Standard Error.....	47, 51	rename.....	83
Standard Input.....	47, 52	rewind.....	84
Standard Output.....	47, 52	scanf.....	85
Start-up.....	47	SEEK_CUR.....	51
Module, Alternate.....	12	SEEK_END.....	51
Module, Primary.....	12	SEEK_SET.....	51
stdarg.h.....	43	setbuf.....	87

setvbuf .....	88	strcspn .....	132
size_t .....	50	strdup .....	117
sprintf .....	89	Streams .....	47
sscanf .....	89	Associate .....	55
stderr .....	51	Binary .....	47
stdin .....	52	Buffering .....	88
stdout .....	52	Closing .....	54, 107
TMP_MAX .....	52	Opening .....	62
tmpfile .....	90	Reading From .....	76
tmpnam .....	91	Text .....	47
ungetc .....	91	Writing To .....	74, 81
vasprintf .....	93	sterror .....	133
vfprintf .....	94	strftime .....	150
vprintf .....	95	String .....	
vsprintf .....	96	Length .....	133
stdlib.h .....	97, 204	Search .....	142
abort .....	98	Transform .....	144
abs .....	99	String Functions, See string.h .....	
atexit .....	99	string.h .....	122
atof .....	101	memchr .....	122
atoi .....	102	memcmp .....	123
atol .....	102	memcpy .....	125
bsearch .....	103	memmove .....	126
calloc .....	105	memset .....	127
div .....	105	NULL .....	122
div_t .....	97	size_t .....	122
exit .....	107	strcat .....	128
EXIT_FAILURE .....	97	strchr .....	129
EXIT_SUCCESS .....	97	strcmp .....	130
free .....	108	strcoll .....	131
getenv .....	108	strcpy .....	131
labs .....	109	strcspn .....	132
ldiv .....	110	sterror .....	133
ldiv_t .....	97	strlen .....	133
malloc .....	111	strncat .....	134
MB_CUR_MAX .....	98	strncmp .....	136
mblen .....	113	strncpy .....	137
mbstowcs .....	113	strpbrk .....	139
mbtowc .....	113	strchr .....	140
NULL .....	98	strspn .....	141
qsort .....	114	strstr .....	142
rand .....	115	strtok .....	143
RAND_MAX .....	98	strxfrm .....	144
realloc .....	115	strlen .....	133
size_t .....	97	strncat .....	134
srand .....	117	strncmp .....	136
strdup .....	117	strncpy .....	137
strtod .....	118	strncpy_p2d16 .....	216
strtol .....	119	strncpy_p2d24 .....	217
strtoul .....	120	strpbrk .....	139
system .....	121	strchr .....	140
wchar_t .....	97	strspn .....	141
wctomb .....	121	strstr .....	142
wxstombs .....	121	strtod .....	101, 118
stdout .....	47, 51, 52, 79, 82	strtok .....	143
strcat .....	128	strtol .....	102, 119
strchr .....	129	strtoul .....	120
strcmp .....	130	struct lconv .....	35
strcoll .....	131	struct tm .....	145
strcpy .....	131	strxfrm .....	144

# 16-Bit Language Tools Libraries

---

Substrings .....	143	atan .....	158
Subtracting Pointers .....	45	atan2 .....	159
Successful Termination .....	97	atan2f .....	161
Support Library .....	197	atanf .....	159
system .....	121, 204	cos .....	163
<b>T</b>		cosf .....	164
Tab .....	23	sin .....	188
tan .....	194	sinf .....	189
tanf .....	194	tan .....	194
tangent		tanf .....	194
Double Floating Point .....	194	type .....	80, 86
Single Floating Point .....	194	<b>U</b>	
tanh .....	195	UCHAR_MAX .....	35
tanhf .....	196	UINT_MAX .....	35
Temporary		ULLONG_MAX .....	35
File .....	90, 107	ULONG_MAX .....	35
Filename .....	51, 91	Underflow Errors 27, 155, 167, 168, 177, 178, 186, 187	
Pointer .....	115	ungetc .....	91
Termination		Universal Time Coordinated .....	148
Request Message .....	40	unsigned char	
Request Signal .....	40	Maximum Value .....	35
Successful .....	97	unsigned int	
Unsuccessful .....	97	Maximum Value .....	35
Text Mode .....	55, 62	unsigned short int	
Text Streams .....	47	Maximum Value .....	35
Ticks .....	145, 146, 148	Unsuccessful Termination .....	97
time .....	152, 205	Upper Case Alphabetic Character	
Time Difference .....	148	Convert To .....	26
Time Structure .....	145, 150	Defined .....	24
Time Zone .....	151	Test for .....	24
time_t .....	145, 150, 152	USHRT_MAX .....	35
time.h .....	145, 205	UTC .....	148
asctime .....	146	Utility Functions, See stdlib.h	
clock .....	146	<b>V</b>	
clock_t .....	145	va_arg .....	43, 45, 93, 94, 95, 96
CLOCKS_PER_SEC .....	145	va_end .....	45, 93, 94, 95, 96
ctime .....	147	va_list .....	43
difftime .....	148	va_start .....	45, 93, 94, 95, 96
gmtime .....	148	Variable Argument Lists, See stdarg.h	
localtime .....	149	Variable Length Argument List ...	43, 45, 93, 94, 95, 96
mktime .....	150	vasprintf .....	93
NULL .....	146	VERBOSE_DEBUGGING .....	17
size_t .....	145	Vertical Tab .....	23
strftime .....	150	vfprintf .....	47, 94
struct tm .....	145	vprintf .....	47, 95
time .....	152	vsprintf .....	47, 96
time_t .....	145	<b>W</b>	
TMP_MAX .....	52	wait_eeedata .....	209
tmpfile .....	90	wchar_t .....	45, 97
tmpnam .....	91	wcstombs .....	121
Tokens .....	143	wctomb .....	121
tolower .....	25	Web Site, Microchip .....	8
toupper .....	26	Week .....	151
Transferring Control .....	36	White Space .....	85, 101, 102, 118
Transform String .....	144	White-Space Character	
Trigonometric Functions		Defined .....	23
acos .....	155	Test for .....	23
acosf .....	156	wide .....	97
asin .....	157		
asinf .....	157		



Wide Character .....	113, 121
Wide Character String.....	113, 121
Wide Character Value .....	45
Width .....	79
width.....	79, 85
write .....	203
write_eedata_row.....	209
write_eedata_word.....	209
write_flash_word16 .....	212
write_flash_word24 .....	212
write_flash_word32 .....	213
write_flash_word48 .....	213
write_flash16.....	211
write_flash24 .....	212

## Y

Year .....	145, 146, 151
------------	---------------

## Z

Zero.....	155
Zero, divide by .....	38, 41, 105



# MICROCHIP

## Worldwide Sales and Service

### AMERICAS

**Corporate Office**  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
<http://www.microchip.com/support>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

**Atlanta**  
Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455

**Austin, TX**  
Tel: 512-257-3370

**Boston**  
Westborough, MA  
Tel: 774-760-0087  
Fax: 774-760-0088

**Chicago**  
Itasca, IL  
Tel: 630-285-0071  
Fax: 630-285-0075

**Cleveland**  
Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

**Dallas**  
Addison, TX  
Tel: 972-818-7423  
Fax: 972-818-2924

**Detroit**  
Novi, MI  
Tel: 248-848-4000

**Houston, TX**  
Tel: 281-894-5983

**Indianapolis**  
Noblesville, IN  
Tel: 317-773-8323  
Fax: 317-773-5453

**Los Angeles**  
Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608

**New York, NY**  
Tel: 631-435-6000

**San Jose, CA**  
Tel: 408-735-9110

**Canada - Toronto**  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

**Asia Pacific Office**  
Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2943-5100  
Fax: 852-2401-3431

**Australia - Sydney**  
Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

**China - Beijing**  
Tel: 86-10-8569-7000  
Fax: 86-10-8528-2104

**China - Chengdu**  
Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

**China - Chongqing**  
Tel: 86-23-8980-9588  
Fax: 86-23-8980-9500

**China - Hangzhou**  
Tel: 86-571-8792-8115  
Fax: 86-571-8792-8116

**China - Hong Kong SAR**  
Tel: 852-2943-5100  
Fax: 852-2401-3431

**China - Nanjing**  
Tel: 86-25-8473-2460  
Fax: 86-25-8473-2470

**China - Qingdao**  
Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205

**China - Shanghai**  
Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066

**China - Shenyang**  
Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

**China - Shenzhen**  
Tel: 86-755-8864-2200  
Fax: 86-755-8203-1760

**China - Wuhan**  
Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

**China - Xian**  
Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

**China - Xiamen**  
Tel: 86-592-2388138  
Fax: 86-592-2388130

**China - Zhuhai**  
Tel: 86-756-3210040  
Fax: 86-756-3210049

### ASIA/PACIFIC

**India - Bangalore**  
Tel: 91-80-3090-4444  
Fax: 91-80-3090-4123

**India - New Delhi**  
Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

**India - Pune**  
Tel: 91-20-3019-1500

**Japan - Osaka**  
Tel: 81-6-6152-7160  
Fax: 81-6-6152-9310

**Japan - Tokyo**  
Tel: 81-3-6880-3770  
Fax: 81-3-6880-3771

**Korea - Daegu**  
Tel: 82-53-744-4301  
Fax: 82-53-744-4302

**Korea - Seoul**  
Tel: 82-2-554-7200  
Fax: 82-2-558-5932 or  
82-2-558-5934

**Malaysia - Kuala Lumpur**  
Tel: 60-3-6201-9857  
Fax: 60-3-6201-9859

**Malaysia - Penang**  
Tel: 60-4-227-8870  
Fax: 60-4-227-4068

**Philippines - Manila**  
Tel: 63-2-634-9065  
Fax: 63-2-634-9069

**Singapore**  
Tel: 65-6334-8870  
Fax: 65-6334-8850

**Taiwan - Hsin Chu**  
Tel: 886-3-5778-366  
Fax: 886-3-5770-955

**Taiwan - Kaohsiung**  
Tel: 886-7-213-7830

**Taiwan - Taipei**  
Tel: 886-2-2508-8600  
Fax: 886-2-2508-0102

**Thailand - Bangkok**  
Tel: 66-2-694-1351  
Fax: 66-2-694-1350

### EUROPE

**Austria - Wels**  
Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393

**Denmark - Copenhagen**  
Tel: 45-4450-2828  
Fax: 45-4485-2829

**France - Paris**  
Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

**Germany - Dusseldorf**  
Tel: 49-2129-3766400

**Germany - Munich**  
Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

**Germany - Pforzheim**  
Tel: 49-7231-424750

**Italy - Milan**  
Tel: 39-0331-742611  
Fax: 39-0331-466781

**Italy - Venice**  
Tel: 39-049-7625286

**Netherlands - Drunen**  
Tel: 31-416-690399  
Fax: 31-416-690340

**Poland - Warsaw**  
Tel: 48-22-3325737

**Spain - Madrid**  
Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

**Sweden - Stockholm**  
Tel: 46-8-5090-4654

**UK - Wokingham**  
Tel: 44-118-921-5800  
Fax: 44-118-921-5820

03/25/14