16-Bit Language Tools Libraries
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in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates.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, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC32 logo, rPIC and Uni/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXMLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, M Platinum Certified logo, MPLIB, MPLINK, mTouch, Omnicient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock 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.

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

© 2010, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

ISBN: 978-1-60932-612-8

Microchip received ISO/TS-16949:2002 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

Preface .................................................................................................................................................. 5

Chapter 1. Library Overview
  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

Chapter 2. Standard C Libraries
  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 ......................................................................................................... 94
  2.15 <string.h> string functions .......................................................................................................... 118
  2.16 <time.h> date and time functions ............................................................................................... 141

  3.1 Introduction .................................................................................................................................. 149
  3.2 Using the Standard C Libraries ...................................................................................................... 149
  3.3 <math.h> mathematical functions ................................................................................................. 151
Chapter 4. Standard C Libraries - Support Functions

4.1 Introduction ................................................................................................. 193
4.2 Using the Support Functions ...................................................................... 194
4.3 Standard C Library Helper Functions ......................................................... 195
4.4 Standard C Library Functions That Require Modification ....................... 200
4.5 Functions/Constants to Support A Simulated UART ............................... 201
4.6 Functions for Erasing and Writing EEDATA Memory ............................. 204
4.7 Functions for Erasing and Writing Flash Memory ................................... 206
4.8 Functions for Specialized Copying and Initialization ............................... 209

Chapter 5. Fixed Point Math Functions

5.1 Introduction ................................................................................................. 213
5.2 Overview of Fixed Point Data Formats .................................................... 214
5.3 Using the Fixed Point Libraries ................................................................. 217
5.4 <libq.h> mathematical functions ............................................................. 219

Appendix A. ASCII Character Set ................................................................. 237

Index ............................................................................................................... 239

Worldwide Sales and Service ........................................................................... 252
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 layout is as follows:

- **Chapter 1: Library Overview** – gives an overview of libraries. Some are described further in this document, while others are described in other documents or on-line 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**
CONVENTIONS USED IN THIS GUIDE

The following conventions may appear in this documentation:

### DOCUMENTATION CONVENTIONS

<table>
<thead>
<tr>
<th>Description</th>
<th>Represents</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arial font:</td>
<td>Referenced books</td>
<td><em>MPLAB® IDE User’s Guide</em></td>
</tr>
<tr>
<td>Italic</td>
<td>Emphasized text</td>
<td><em>...is the only compiler...</em></td>
</tr>
<tr>
<td>Initial caps</td>
<td>A window</td>
<td>the Output window</td>
</tr>
<tr>
<td></td>
<td>A dialog</td>
<td>the Settings dialog</td>
</tr>
<tr>
<td></td>
<td>A menu selection</td>
<td>select Enable Programmer</td>
</tr>
<tr>
<td>Quotes</td>
<td>A field name in a window or dialog</td>
<td>“Save project before build”</td>
</tr>
<tr>
<td>Underlined, italic with right angle bracket</td>
<td>A menu path</td>
<td>File&gt;Save</td>
</tr>
<tr>
<td>Bold</td>
<td>A dialog button</td>
<td>Click OK</td>
</tr>
<tr>
<td></td>
<td>A tab</td>
<td>Click the Power tab</td>
</tr>
<tr>
<td>Text in angle brackets &lt; &gt;</td>
<td>A key on the keyboard</td>
<td>Press &lt;Enter&gt;, &lt;F1&gt;</td>
</tr>
<tr>
<td>Courier New font:</td>
<td>Sample source code</td>
<td><code>#define START</code></td>
</tr>
<tr>
<td></td>
<td>Filenames</td>
<td>autoexec.bat</td>
</tr>
<tr>
<td></td>
<td>File paths</td>
<td>c:\mcc18\h</td>
</tr>
<tr>
<td></td>
<td>Keywords</td>
<td>_asm, _endasm, static</td>
</tr>
<tr>
<td></td>
<td>Command-line options</td>
<td>-Opa+, -Opa-</td>
</tr>
<tr>
<td></td>
<td>Bit values</td>
<td>0, 1</td>
</tr>
<tr>
<td></td>
<td>Constants</td>
<td>0xFF, 'A'</td>
</tr>
<tr>
<td>Italic</td>
<td>A variable argument</td>
<td>file.o, where file can be any valid filename</td>
</tr>
<tr>
<td>Square brackets [ ]</td>
<td>Optional arguments</td>
<td>mpasmwin [options] file [options]</td>
</tr>
<tr>
<td>Curly brackets and pipe character: {</td>
<td>Choice of mutually exclusive arguments; an OR selection</td>
<td>errorlevel {0</td>
</tr>
<tr>
<td>}</td>
<td>Ellipses...</td>
<td>Replaces repeated text</td>
</tr>
</tbody>
</table>
| Ellipses... | 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® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® 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 C Compiler for PIC24 MCUs and dsPIC® 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.

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


THE MICROCHIP WEB SITE

Microchip provides online support via our web site at 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, 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. These include the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators
- **In-Circuit Debuggers** – The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD 2 and 3 in-circuit debuggers and PICkit™ 2 and 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 the device (production) programmers MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger, MPLAB PM3, and PRO MATE II and development (nonproduction) programmers MPLAB ICD 2 in-circuit debugger, PICSTART® Plus and PICkit 1, 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://support.microchip.com
1.1 INTRODUCTION

A library is a collection of functions grouped for reference and ease of linking. See the “MPLAB® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51317) for more information about making and using libraries.

1.1.1 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 C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30).

1.1.2 C Code Applications

The 16-bit language tool libraries are included in the lib subdirectory of the MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs install directory, which is by default:

C:\Program Files\Microchip\MPLAB C30\lib

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

1.1.3 Chapter Organization

This chapter is organized as follows:
• 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
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 `PIC30_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® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51317) and, for C applications, the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284).

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 found. By default, the documentation is found in:

```
C:\Program Files\Microchip\MPLAB C30\docs\dsp_lib
```

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:

```
C:\Program Files\Microchip\MPLAB C30\docs\periph_lib
```
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 \texttt{libc-	extasciitilde omf.a} (written by Dinkumware, an industry leader) and \texttt{libm-	extasciitilde 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 \texttt{libpic30-	extasciitilde 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, \texttt{libq-	extasciitilde omf.a}. Documentation for these library functions is contained in this manual.

1.8  COMPILER BUILT-IN FUNCTIONS

The MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs contains built-in functions that, to the developer, work like library functions. These functions are listed in the “\textit{MPLAB\textsuperscript{®} C Compiler for PIC24 MCUs and dsPIC\textsuperscript{®} DSCs Users’ Guide}” (DS51284).
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 C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) install directory (c:\Program Files\Microchip\MPLAB C30) 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
- <float.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
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® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51317) and “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284) 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:**

```c
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:**

```c
#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
### 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

<table>
<thead>
<tr>
<th>Description:</th>
<th>Test for an alphanumeric character.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;ctype.h&gt;</code></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>int isalnum(int c);</code></td>
</tr>
<tr>
<td>Argument:</td>
<td><code>c</code> The character to test.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns a non-zero integer value if the character is alphanumeric; otherwise, returns zero.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Alphanumeric characters are included within the ranges A-Z, a-z or 0-9.</td>
</tr>
</tbody>
</table>
| 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

<table>
<thead>
<tr>
<th>Description:</th>
<th>Test for an alphabetic character.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;ctype.h&gt;</code></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>int isalpha(int c);</code></td>
</tr>
<tr>
<td>Argument:</td>
<td><code>c</code> The character to test.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns a non-zero integer value if the character is alphabetic; otherwise, returns zero.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Alphabetic characters are included within the ranges A-Z or a-z.</td>
</tr>
</tbody>
</table>
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

iscntrl

Description: Test for a control character.

Include: `<ctype.h>`

Prototype: `int iscntrl(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 iscntrl */
#include <stdio.h> /* for printf */

int main(void)
{
    char ch;

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

    ch = '\t';
    if (iscntrl(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
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:
```c
#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:
```c
#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 lower case 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 lower case alphabetic character; otherwise, returns zero.
Remarks: A character is considered to be a lower case 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 lower case\n");
    else
        printf("B is NOT lower case\n");

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

islower (Continued)

Output:
B is NOT lower case
b is lower case

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:
!"#$%&'()\*+,-./:;<=>?@[^`]{|}~
Example:
```c
#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:**
```c
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:**
```c
#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");
}
```
### isspace (Continued)

**Output:**

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

### isupper

**Description:** Test for an upper case 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 upper case alphabetic character; otherwise, returns zero.

**Remarks:** A character is considered to be an upper case alphabetic character if it is in the range of ‘A’-‘Z’.

**Example:**

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

int main(void)
{
    int ch;

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

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

**Output:**

```
B is upper case
b is NOT upper case
```

### 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:
```c
#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 lower case alphabetical character.
Include: `<ctype.h>`
Prototype: `int tolower (int c);`
Argument: `c` The character to convert to lower case.
Return Value: Returns the corresponding lower case alphabetical character if the argument was originally upper case; otherwise, returns the original character.
Remarks: Only upper case alphabetical characters may be converted to lower case.
Example:
```c
#include <ctype.h> /* for tolower */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

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

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

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


### toupper

**Output:**
- B changes to lower case b
- b remains lower case b
- @ has no lower case, so @ is returned

**Description:** Convert a character to an upper case alphabetical character.

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

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

**Argument:**
- `c` The character to convert to upper case.

**Return Value:** Returns the corresponding upper case alphabetical character if the argument was originally lower case; otherwise, returns the original character.

**Remarks:** Only lower case alphabetical characters may be converted to upper case.

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

int main(void)
{
    int ch;

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

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

    ch = '@';
    printf("@ has no upper case, ", toupper(ch));
}
```

**Output:**
- b changes to upper case B
- B remains upper case B
- @ has no upper case, 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**

| Description: | Represents a domain error. |
| Include: | `<errno.h>` |
| Remarks: | EDOM represents a domain error, which occurs when an input argument is outside the domain in which the function is defined. |

**ERANGE**

| Description: | Represents an overflow or underflow error. |
| Include: | `<errno.h>` |
| Remarks: | ERANGE represents an overflow or underflow error, which occurs when a result is too large or too small to be stored. |

**errno**

| Description: | Contains the value of an error when an error occurs in a function. |
| Include: | `<errno.h>` |
| Remarks: | The variable, `errno`, is set to a non-zero integer value by a library function when an error occurs. At program start-up, `errno` is set to zero. `Errno` should be reset to zero prior to calling a function that sets it. |
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.
### 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
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Include</th>
<th>Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT_MAX_EXP</td>
<td>Maximum integer value for a single precision floating-point exponent in base FLT_RADIX.</td>
<td>&lt;float.h&gt;</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>FLT_MIN</td>
<td>Minimum single precision floating-point value.</td>
<td>&lt;float.h&gt;</td>
<td>1.175494e-38</td>
<td></td>
</tr>
<tr>
<td>FLT_MIN_10_EXP</td>
<td>Minimum negative integer value for a single precision floating-point exponent in base 10.</td>
<td>&lt;float.h&gt;</td>
<td>-37</td>
<td></td>
</tr>
<tr>
<td>FLT_MIN_EXP</td>
<td>Minimum negative integer value for a single precision floating-point exponent in base FLT_RADIX.</td>
<td>&lt;float.h&gt;</td>
<td>-125</td>
<td></td>
</tr>
<tr>
<td>FLT_RADIX</td>
<td>Radix of exponent representation.</td>
<td>&lt;float.h&gt;</td>
<td>2</td>
<td>The base representation of the exponent is base-2 or binary.</td>
</tr>
<tr>
<td>FLT_ROUNDS</td>
<td>Represents the rounding mode for floating-point operations.</td>
<td>&lt;float.h&gt;</td>
<td>1</td>
<td>Rounds to the nearest representable value.</td>
</tr>
<tr>
<td>LDBL_DIG</td>
<td>Number of decimal digits of precision in a long double precision floating-point value.</td>
<td>&lt;float.h&gt;</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Include</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------</td>
<td>-------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>LDBL_EPSILON</td>
<td>The difference between 1.0 and the next larger representable long double precision floating-point value.</td>
<td>&lt;float.h&gt;</td>
<td>2.220446e-16</td>
<td></td>
</tr>
<tr>
<td>LDBL_MANT_DIG</td>
<td>Number of base-FLT_RADIX digits in a long double precision floating-point significand.</td>
<td>&lt;float.h&gt;</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>LDBL_MAX</td>
<td>Maximum finite long double precision floating-point value.</td>
<td>&lt;float.h&gt;</td>
<td>1.797693e+308</td>
<td></td>
</tr>
<tr>
<td>LDBL_MAX_10_EXP</td>
<td>Maximum integer value for a long double precision floating-point exponent in base 10.</td>
<td>&lt;float.h&gt;</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>LDBL_MAX_EXP</td>
<td>Maximum integer value for a long double precision floating-point exponent in base FLT_RADIX.</td>
<td>&lt;float.h&gt;</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>LDBL_MIN</td>
<td>Minimum long double precision floating-point value.</td>
<td>&lt;float.h&gt;</td>
<td>2.225074e-308</td>
<td></td>
</tr>
<tr>
<td>LDBL_MIN_10_EXP</td>
<td>Minimum negative integer value for a long double precision floating-point exponent in base 10.</td>
<td>&lt;float.h&gt;</td>
<td>-307</td>
<td></td>
</tr>
</tbody>
</table>
## 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Number of bits to represent type <code>char</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;limits.h&gt;</code></td>
</tr>
<tr>
<td>Value</td>
<td>8</td>
</tr>
</tbody>
</table>

### CHAR_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of a <code>char</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;limits.h&gt;</code></td>
</tr>
<tr>
<td>Value</td>
<td>127</td>
</tr>
</tbody>
</table>

### CHAR_MIN

<table>
<thead>
<tr>
<th>Description</th>
<th>Minimum value of a <code>char</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;limits.h&gt;</code></td>
</tr>
<tr>
<td>Value</td>
<td>-128</td>
</tr>
</tbody>
</table>

### INT_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of an <code>int</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;limits.h&gt;</code></td>
</tr>
<tr>
<td>Value</td>
<td>32767</td>
</tr>
</tbody>
</table>

### INT_MIN

<table>
<thead>
<tr>
<th>Description</th>
<th>Minimum value of an <code>int</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;limits.h&gt;</code></td>
</tr>
<tr>
<td>Value</td>
<td>-32768</td>
</tr>
</tbody>
</table>

### LLONG_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of a <code>long long int</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;limits.h&gt;</code></td>
</tr>
<tr>
<td>Value</td>
<td>9223372036854775807</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>LLONG_MIN</td>
<td>Minimum value of a long long int.</td>
</tr>
<tr>
<td>LONG_MAX</td>
<td>Maximum value of a long int.</td>
</tr>
<tr>
<td>LONG_MIN</td>
<td>Minimum value of a long int.</td>
</tr>
<tr>
<td>MB_LEN_MAX</td>
<td>Maximum number of bytes in a multibyte character.</td>
</tr>
<tr>
<td>SCHAR_MAX</td>
<td>Maximum value of a signed char.</td>
</tr>
<tr>
<td>SCHAR_MIN</td>
<td>Minimum value of a signed char.</td>
</tr>
<tr>
<td>SHRT_MAX</td>
<td>Maximum value of a short int.</td>
</tr>
<tr>
<td>SHRT_MIN</td>
<td>Minimum value of a short int.</td>
</tr>
</tbody>
</table>
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

UCHAR_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of an unsigned char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;limits.h&gt;</td>
</tr>
<tr>
<td>Value:</td>
<td>255</td>
</tr>
</tbody>
</table>

UINT_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of an unsigned int.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;limits.h&gt;</td>
</tr>
<tr>
<td>Value:</td>
<td>65535</td>
</tr>
</tbody>
</table>

ULLONG_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of a long long unsigned int.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;limits.h&gt;</td>
</tr>
<tr>
<td>Value:</td>
<td>18446744073709551615</td>
</tr>
</tbody>
</table>

ULONG_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of a long unsigned int.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;limits.h&gt;</td>
</tr>
<tr>
<td>Value:</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

USHRT_MAX

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum value of an unsigned short int.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;limits.h&gt;</td>
</tr>
<tr>
<td>Value:</td>
<td>65535</td>
</tr>
</tbody>
</table>
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 access 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:**

```c
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>
### SIGABRT

**Description:** Name for the abnormal termination signal.

**Include:**

```c
#include <signal.h>
```

**Prototype:**

```c
#define SIGABRT
```

**Remarks:** SIGABRT represents an abnormal termination signal and is used in conjunction with `raise` or `signal`. The default raise behavior (action identified by `SIG_DFL`) is to output to the standard error stream:

```
abort - terminating
```

See the example accompanying `signal` to see general usage of signal names and signal handling.

**Example:**

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

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

**Output:**

```
ABRT
```

**Explanation:**

ABRT stands for “abort”.

---

### SIGFPE

**Description:** Signals floating-point, error such as for division by zero or result out of range.

**Include:**

```c
#include <signal.h>
```

**Prototype:**

```c
#define SIGFPE
```

**Remarks:** SIGFPE is used as an argument for `raise` and/or `signal`. 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 `signal` for an example of a user-defined function.

**Example:**

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

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

**Output:**

```
FPE
```

**Explanation:**

FPE stands for “floating-point error”.
### SIGILL

**Description:** Signals illegal instruction.

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

**Prototype:**
```c
#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:**
```c
#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:**
```c
#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:**
```c
#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”.
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:
```c
#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:
```c
#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, which is the function, illegalinsn.
Thus, error messages are printed and the program is terminated.
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.\n");
}

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 indi-
cate 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)
        {


va_arg (Continued)

```c
    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
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>
### 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 */

```c
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.
-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.

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:

```c
/* 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:

```c
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");

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.
### FOPEN_MAX (Continued)

<table>
<thead>
<tr>
<th>Include</th>
<th>&lt;stdio.h&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
</tr>
<tr>
<td>Remarks</td>
<td>stderr, stdin and stdout are included in the FOPEN_MAX count.</td>
</tr>
</tbody>
</table>

### L_tmpnam

<table>
<thead>
<tr>
<th>Description</th>
<th>Defines the number of characters for the longest temporary filename created by the function, tmpnam.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>&lt;stdio.h&gt;</td>
</tr>
<tr>
<td>Value</td>
<td>16</td>
</tr>
<tr>
<td>Remarks</td>
<td>L_tmpnam is used to define the size of the array used by tmpnam.</td>
</tr>
</tbody>
</table>

### NULL

<table>
<thead>
<tr>
<th>Description</th>
<th>The value of a null pointer constant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>&lt;stdio.h&gt;</td>
</tr>
</tbody>
</table>

### SEEK_CUR

<table>
<thead>
<tr>
<th>Description</th>
<th>Indicates that fseek should seek from the current position of the file pointer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>&lt;stdio.h&gt;</td>
</tr>
<tr>
<td>Example</td>
<td>See example for fseek.</td>
</tr>
</tbody>
</table>
### 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>`

### 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:**

```
#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:**

```c
#include <stdio.h> /* for ferror, clearerr, */
#include <cstdio> /* printf, fprintf, fopen, */
#include <cerrno> /* 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.

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
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:
```c
#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.
ferror

Description: Tests if error indicator is set.

Include:  <stdio.h>

Prototype: int ferror(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 ferror 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 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
**fflush**

**Description:** Flashes 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|
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:
```c
/* 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\n", s);
        }
        fclose(buf);
    }
}

Input:
Contents of afile.txt (used as input):
Short
Longer string

Output:
Short
|Longer string
|
**fopen**

**Description:**
Opens a file.

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

**Prototype:**
```c
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:**
Following are the 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.)

**Example:**
```c
#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.
**fprintf**

Description: Prints formatted data to a stream.

Include: `<stdio.h>`

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

Arguments: `stream` pointer to the stream in which to output data

`format` 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 `printf`.

Example:

```c
#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:**

```c
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:**

```c
#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:**

```c
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:**

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

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

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

**Output:**

This is text

|
# fread

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

**Include:**  
<stdio.h>

**Prototype:**  
```c
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:**
```c
#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");
}
```
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.
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.

Example: 
```c
#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.

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

Output:
Print this string 100 times
## fseek

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

**Include:**

```c
#include <stdio.h>
```

**Prototype:**

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

**Arguments:**

- `stream`: stream in which to move the file pointer.
- `offset`: value to add to the current position
- `mode`: type of seek to perform

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

```c
#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("\"%s\"\n\n", s);
        }

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

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

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

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

**Prototype:**
```c
int fsetpos(FILE *stream, const fpos_t *pos);
```

**Arguments:**
- `stream` target stream
- `pos` position-indicator storage as returned by an earlier call to `fgetpos`

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

**Remarks:**
The function sets the file-position indicator for the given stream in `*pos` if successful; otherwise, `fsetpos` sets `errno`. 
**Example:**

```c
/* 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:**

```
#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:**

```c
#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
```
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)

```c
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.
### getc

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

**Include:**
```c
#include <stdio.h>
```

**Prototype:**
```c
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:**
```c
#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:**

```c
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:**

```c
#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);
    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:**

```c
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.
gets (Continued)

Example:

```c
#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 a newline.

Example:

```c
#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**

**Description:** Prints formatted text to stdout.

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

**Prototype:**

```c
int printf(const char *format, ...);
```

**Arguments:**

- `format` format control string
- `...` optional arguments

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

**Remarks:** 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:

```
%[flags][width][.precision][size]type
```

**flags**

- `-` left justify the value within a given field width
- `0` Use 0 for the pad character instead of space (which is the default)
- `+` generate a plus sign for positive signed values
- `space` generate a space or signed values that have neither a plus nor a minus sign
- `#` 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

**width**

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 `int`) 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.

**precision**

The field width can be followed with dot (.) and a decimal integer representing the precision that specifies one of the following:

- minimum number of digits to generate on an integer conversion
- number of fraction digits to generate on an e, E, or f conversion
- maximum number of significant digits to generate on a g or G conversion
- maximum number of characters to generate from a C string on an s conversion

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 `int`) will be used for the precision.
printf (Continued)

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Used with type c, s; specifies a string</td>
</tr>
<tr>
<td>h modifier –</td>
<td>Used with type d, i, o, u, x, X; converts the value to a short int or unsigned short int</td>
</tr>
<tr>
<td>h modifier –</td>
<td>Used with n; specifies that the pointer points to a short int</td>
</tr>
<tr>
<td>l modifier –</td>
<td>Used with type d, i, o, u, x, X; converts the value to a long int or unsigned long int</td>
</tr>
<tr>
<td>l modifier –</td>
<td>Used with n; specifies that the pointer points to a long int</td>
</tr>
<tr>
<td>l modifier –</td>
<td>Used with c; specifies a wide character</td>
</tr>
<tr>
<td>ll modifier –</td>
<td>Used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int</td>
</tr>
<tr>
<td>ll modifier –</td>
<td>Used with n; specifies that the pointer points to a long long int</td>
</tr>
<tr>
<td>L modifier –</td>
<td>Used with e, E, f, F, g, G; converts the value to a long double</td>
</tr>
<tr>
<td>type</td>
<td></td>
</tr>
<tr>
<td>d, i</td>
<td>Signed int</td>
</tr>
<tr>
<td>o</td>
<td>Unsigned int in octal</td>
</tr>
<tr>
<td>u</td>
<td>Unsigned int in decimal</td>
</tr>
<tr>
<td>x</td>
<td>Unsigned int in lowercase hexadecimal</td>
</tr>
<tr>
<td>X</td>
<td>Unsigned int in uppercase hexadecimal</td>
</tr>
<tr>
<td>e, E</td>
<td>Double in scientific notation</td>
</tr>
<tr>
<td>f</td>
<td>Double decimal notation</td>
</tr>
<tr>
<td>g, G</td>
<td>Double (takes the form of e, E or f as appropriate)</td>
</tr>
<tr>
<td>c</td>
<td>Char - a single character</td>
</tr>
<tr>
<td>s</td>
<td>String</td>
</tr>
<tr>
<td>p</td>
<td>Value of a pointer</td>
</tr>
<tr>
<td>n</td>
<td>The associated argument shall be an integer pointer into which is placed the number of characters written so far. No characters are printed.</td>
</tr>
<tr>
<td>%</td>
<td>A % character is printed</td>
</tr>
</tbody>
</table>

Example:
```c
#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) /* for putc, EOF, stdout */
{
    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:
This is text
|
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:
```c
#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:
```c
#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:**

```c
int remove(const char *filename);
```

**Argument:**

- `filename` name of file to be deleted

**Return Value:** Returns 0 if successful, -1 if not.

**Remarks:** If `filename` does not exist or is open, `remove` will fail.

**Example:**

```c
#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:**

```c
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, non-zero if not.

**Remarks:** The new name must not already exist in the current working directory, the old name must exist in the current working directory.

**Example:**

```c
#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
**Rewind**

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

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

**Prototype:**

```c
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:**

```c
#include <stdio.h> /* for rewind, fopen, */
/* fscanf, 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);
        fscanf(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**

**Description:** Scans formatted text from stdin.

**Include:**

```c
#include <stdio.h>
```

**Prototype:**

```c
int scanf(const char *format, ...);
```

**Argument:**

- `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 an input failure is encountered before the first.

**Remarks:** Each format specifier begins with a percent sign followed by optional fields and a required type as shown here:

```c
%[*][width][modifier]type
```

- `*` indicates assignment suppression. This will cause the input field to be skipped and no assignment made.
- `width` specify the maximum number of input characters to match for the conversion not including white space that can be skipped.
- `modifier`
  - `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
  - `I` 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
  - `I` modifier – used with type e, E, f, F, g, G; converts the value to a double
  - `I` modifier – used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int
  - `l` 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 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.

Example: For MPLAB SIM simulator.

```c
#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, "
           "favorite color desired salary "
           "and SSN:\n\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.

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\n");
        fclose(myfile2);
    }
}
```

Output:

myfile1 has no buffering
myfile2 has full buffering
### setvbuf

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

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

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

**Arguments:**
- `stream` pointer to the open stream
- `buf` user allocated buffer
- `mode` type of buffering
- `size` 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

**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:**

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

**Arguments:**

- `s` storage string for output
- `format` 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:**

```c
#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:**

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

**Arguments:**

- `s` storage string for input
- `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 an input error is encountered before the first conversion.

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

```c
```

---

© 2010 Microchip Technology Inc.
Example:

```c
#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
```

### tmpfile

**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.

**Example:**

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

int main(void)
{
    FILE *mytempfile;

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

**Output:**

```
Temporary file was created
```
### tmpnam

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

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

**Prototype:**

```c
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 cannot 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:**

```c
#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:**

```c
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.
# 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
vfprintf

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

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

**Prototype:**
```c
int vfprintf(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 `vfprintf` function is called. Invoke `va_end` after the function returns. For more details, see `stdarg.h`.

**Example:**
```c
#include <stdio.h> /* for vfprintf, 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);
    vfprintf(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:

```c
<stdio.h>
<stdio.h>
```

Prototype:

```c
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:

```c
#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.
vsprintf

Description: Prints formatted text to a string using a variable length argument list.

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

Prototype:
int vsprintf(char *s, const char *format, va_list ap);

Arguments:
  s           storage string for output
  format      format control string
  ap          pointer to a list of arguments

Return Value: Returns 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.
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 vsprintf function is called. Invoke va_end after the function returns. For more details, see stdarg.h

Example:
#include <stdio.h>   /* for vsprintf, printf */
#include <stdarg.h>  /* for va_start, va_list, va_end */

void errmsg(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;

    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.
## 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:**
```c
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:**
```c
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.
### 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:**

```c
void abort(void);
```

**Remarks:** `abort` will cause the processor to reset.

**Example:**

```c
#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:**

```c
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:**

```c
#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:**

```c
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:**

```c
#include <stdio.h> /* for scanf, printf */
#include <stdlib.h> /* for atexit, exit */

void good_msg(void);
void bad_msg(void);
void end_msg(void);
```
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 = "27.835e2" float = 2783.500000
String = "Number1" float = 0.000000
# atoi

**Description:** Converts a string to an integer.

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

**Prototype:**

```c
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:

- 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:**

```c
#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"	int = %d
", a, x);

    x = atoi(b);
    printf("String = "%s"	int = %d
", b, x);
}
```

**Output:**

```
String = " -127"        int = -127
String = "Number1"      int = 0
```

# atol

**Description:** Converts a string to a long integer.

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

**Prototype:**

```c
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:

- 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:
key object to search for
base pointer to the start of the search data
nelem number of elements
size size of elements
cmp pointer to the comparison function
ck pointer to the key for the search
ce 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 cmp uses ascending order.
Example:

```c
#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

bsearch (Continued)
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.

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.
Example:

```c
#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", 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", 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", 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", 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", 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
### free

**Description:** Frees memory.

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

**Prototype:**
```c
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.

**Example:**
```c
#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:**
```c
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.
Example:
```c
#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:**
```c
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:
```c
#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
#Idiv

**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:**

```c
#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);
}
```
Idiv (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.
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.
Example:

```c
#include <stdio.h>  /* for printf, sizeof, */
#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
**mblen**

**Description:** Gets the length of a multibyte character. (See Remarks.)

**Include:**

```c
<stdlib.h>
```

**Prototype:**

```c
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:**

```c
<stdlib.h>
```

**Prototype:**

```c
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:**

```c
<stdlib.h>
```

**Prototype:**

```c
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.
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.

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;
}

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:**
```c
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:**
```c
#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:**
```c
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.

**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.
realloc (Continued)

Example:

```c
#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)
            {  // j pointed to allocated memory
                printf("Cannot reallocate memory\n");
                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:**

```c
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:**

```c
#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

### strtod

**Description:** Converts a partial string to a floating-point number of type double.

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

**Prototype:**

```c
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:

- 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:

```c
#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\" float = %f\n", a, x);
    printf("Stopped at: %s\n", end);

    x = strtod(b, &end);
    printf("String = "s\" float = %f\n", b, x);
    printf("Stopped at: %s\n", end);

    x = strtod(c, &end);
    printf("String = "s\" float = %f\n", c, x);
    printf("Stopped at: %s\n", end);
}
```

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:**

```c
long strtol(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, `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:**

```c
#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", end);

    x = strtol(b, &end, 4);
    printf("String = \"%s\"  long = %ld\n", b, x);
    printf("Stopped at: %s\n", end);
}
```

**Output:**

```
String = "-12BGEE"  long = -299
Stopped at: GEE

String = "1234Number"  long = 27
Stopped at: 4Number
```
**strtoul**

**Description:** Converts a partial string to an unsigned long integer.

**Include:**

```c
#include <stdlib.h>
```

**Prototype:**

```c
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, `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:**

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

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:**

```c
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:**

```c
/* 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:**

```c
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:**

```c
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.
2.15  <STRING.H> STRING FUNCTIONS

The header file, string.h, consists of types, macros and functions that provide tools
to manipulate strings.

<table>
<thead>
<tr>
<th><strong>size_t</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>NULL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>memchr</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
<tr>
<td><strong>Prototype:</strong></td>
</tr>
<tr>
<td><strong>Arguments:</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
</tr>
</tbody>
</table>
| **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)

```c
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:**

```c
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:**

```c
#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", 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");

    return 0;
}
```
memcmp (Continued)

```c
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:
```c
#include <string.h> /* memcpy */
#include <stdio.h>  /* for printf */

double 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", 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:
```c
#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", 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:

- `s`  buffer
- `c`  character to put in buffer
- `n`  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:

```c
#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
```
strcat

**Description:** Appends a copy of the source string to the end of the destination string.

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

**Prototype:**

```c
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:**

```c
#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("buf2 : %s\n", buf2);
    printf("\n");

    strcat(buf1, buf2);
    printf("buf1 after strcat of buf2: %s\n", buf1);

    printf("\n");

    strcat(buf1, "Why?");
    printf("buf1 after strcat of "Why?: %s\n", buf1);

    printf("\n");

    printf("buf1 : %s\n", buf1);
    printf("(10 characters)\n");

    printf("buf2 : %s\n", buf2);
    printf("(18 characters)\n");

    printf("buf1 after strcat of buf2:
    We're hereWhere is the time?\n    (28 characters)\n");

    printf("buf1 after strcat of "Why?:
    We're hereWhere is the time?Why?\n    (32 characters)\n");
}
```

**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**

<table>
<thead>
<tr>
<th>Description:</th>
<th>Locates the first occurrence of a specified character in a string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;string.h&gt;</code></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>char *strchr(const char *s, int c);</code></td>
</tr>
<tr>
<td>Arguments:</td>
<td>s pointer to the string</td>
</tr>
<tr>
<td></td>
<td>c character to search for</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns a pointer to the location of the match if successful; otherwise, returns a null pointer.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function searches the string s to find the first occurrence of the character, c.</td>
</tr>
<tr>
<td>Example:</td>
<td>```c</td>
</tr>
</tbody>
</table>
|             | #include <string.h> /* for strchr, 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 = 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);                         |
|             | }                                                          |
| Output:     | buf1 : What time is it?                                    |
|             | m found at position 8                                       |
|             | y not found                                                |```
**strcmp**

**Description:** Compares two strings.

**Include:**
```
#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 <stdio.h>  /* for printf */
#include <string.h> /* for strcmp */

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", 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");
}
```
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
Description: Compares one string to another. (See Remarks.)
Include: <string.h>
Prototype: int strcoll(const char *s1, const char *s2);
Arguments: s1 first string
s2 second string
Return Value: 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.
Remarks: Since the 16-bit compiler does not support alternate locales, this
function is equivalent to strcmp.

strcpy
Description: Copy the source string into the destination string.
Include: <string.h>
Prototype: char *strcpy(char *s1, const char *s2);
Arguments: s1 destination string to copy to
s2 source string to copy from
Return Value: Returns a pointer to the destination string.
Remarks: All characters of s2 are copied, including the null terminating character.
If the strings overlap, the behavior is undefined.
Example:
#include <string.h> /* for strcpy, 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?";

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);

    strcpy(buf1, buf2);
    printf("buf1 after strcpy of buf2: \n\t%s\n\n", buf1);
}
**strncpy (Continued)**

```c
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:**

```c
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:**

```c
#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.

strftime
Description:  Formats a date and time string.
Include:  <time.h>
Prototype:  char *strftime(time_t *t, const char *fmt, char *s);
Argument:  t  the time
fmt  format string
s  output buffer
Return Value:  Returns the number of characters in the result.
Remarks:  This function formats a date and time according to
the specified format.
Example:  #include <time.h>
          #include <stdio.h>

          int main(void)
          {
            struct tm *t;
            char buf[80];

            t = localtime(time(NULL));
            strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", t);
            printf("Now is: %s\n", buf);  // Outputs a formatted date and time
          }

        Output:
        Now is: 2023-11-15 14:30:00

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.
Example:

```c
#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", strlen(str1));
    printf("str2 : %s\n", str2);
    printf("\t(string length = %d characters)\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:**

```c
char *strncat(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 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:**

```c
#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?"
    char buf3[50] = "Why";
```
strncat (Continued)

```c
printf("buf1 : We\'re here\n" , buf1);
printf("\t(10 characters)\n", strlen(buf1));
printf("buf2 : Where is the time?\n" , buf2);
printf("\t(18 characters)\n", strlen(buf2));
printf("buf3 : Why?\n" , buf3);
printf("\t(4 characters)\n", strlen(buf3));
```

```c
strncat(buf1, buf2, 6);
printf("buf1 after strncat of 6 characters 
" "of buf2: \n\t\s\n", buf1);
printf("\"\t(16 characters)\n", strlen(buf1));
```

```c
strncat(buf1, buf2, 25);
printf("buf1 after strncat of 25 characters 
" "of buf2: \n\t\s\n", buf1);
printf("\"\t(34 characters)\n", strlen(buf1));
```

```c
strncat(buf1, buf3, 4);
printf("buf1 after strncat of 4 characters 
" "of buf3: \n\t\s\n", buf1);
printf("\"\t(38 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 here
  (16 characters)

buf1 after strncat of 25 characters of buf2:
  We're here
  Where is the time?
  (34 characters)

buf1 after strncat of 4 characters of buf3:
  We're here
  Where is the time? Why?
  (38 characters)
**strncmp**

**Description:**
Compare two strings, up to a specified number of characters.

**Include:**
```
#include <string.h>
```

**Prototype:**
```
int strncmp(const char *s1, const char *s2, size_t n);
```

**Arguments:**
- `s1` : first string
- `s2` : second string
- `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:**
`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
", buf1);
    printf("buf2 : %s
", buf2);
    printf("buf3 : %s
", 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");
}
```
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          */
#include <stdlib.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);
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:**
```c
#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

<table>
<thead>
<tr>
<th>Description:</th>
<th>Search for the last occurrence of a specified character in a string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;string.h&gt;</td>
</tr>
<tr>
<td>Prototype:</td>
<td>char *strrchr(const char *s, int c);</td>
</tr>
<tr>
<td>Arguments:</td>
<td>s pointer to the string to be searched</td>
</tr>
<tr>
<td></td>
<td>c character to search for</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns a pointer to the character if found; otherwise, returns a null</td>
</tr>
<tr>
<td></td>
<td>pointer.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>The function searches the string s, including the terminating null</td>
</tr>
<tr>
<td></td>
<td>character, to find the last occurrence of character c.</td>
</tr>
<tr>
<td>Example:</td>
<td>#include &lt;string.h&gt; /* for strrchr, NULL */</td>
</tr>
<tr>
<td></td>
<td>#include &lt;stdio.h&gt; /* for printf */</td>
</tr>
<tr>
<td></td>
<td>int main(void)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>char buf1[50] = &quot;What time is it?&quot;;</td>
</tr>
<tr>
<td></td>
<td>char ch1 = 'm', ch2 = 'y';</td>
</tr>
<tr>
<td></td>
<td>char *ptr;</td>
</tr>
<tr>
<td></td>
<td>int res;</td>
</tr>
<tr>
<td></td>
<td>printf(&quot;buf1 : %s\n\n&quot;, buf1);</td>
</tr>
<tr>
<td></td>
<td>ptr = strrchr(buf1, ch1);</td>
</tr>
<tr>
<td></td>
<td>if (ptr != NULL)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>res = ptr - buf1 + 1;</td>
</tr>
<tr>
<td></td>
<td>printf(&quot;%c found at position %d\n&quot;, ch1, res);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>printf(&quot;%c not found\n&quot;, ch1);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
strrchr (Continued)

printf("\n");

ptr = strrchr(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: s1 pointer to the string to be searched
            s2 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] = "aeiou
    char str3[20] = "aimn";
    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", "aeioumn") = 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:
```c
#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", str3);

    ptr = strstr(str1, str2);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("\"%s\" found at position %d\n", str2, res);
    }
    else
        printf("\"%s\" not found\n", str2);

    return 0;
}
```
### strstr (Continued)

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

ptr = strstr(str1, str3);
if (ptr != NULL)
{
    res = ptr - str1 + 1;
    printf("\"%s\" found at position %d\n", str3, res);
}
else
    printf("\"%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:**
```c
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.
strtok (Continued)

Example:

```c
#include <string.h> /* for strtok, NULL */
#include <stdio.h> /* for printf */

int main(void)
{
    char str1[30] = "Here, on top of the world!";
    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
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: 
- `s1`: destination string
- `s2`: source string to be transformed
- `n`: 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>clock_t</strong></td>
<td>Stores processor time values.</td>
<td><code>&lt;time.h&gt;</code></td>
<td><code>typedef long clock_t</code></td>
</tr>
<tr>
<td><strong>size_t</strong></td>
<td>The type of the result of the <code>sizeof</code> operator.</td>
<td><code>&lt;time.h&gt;</code></td>
<td><code>typedef size_t</code></td>
</tr>
<tr>
<td><strong>struct tm</strong></td>
<td>Structure used to hold the time and date (calendar time).</td>
<td><code>&lt;time.h&gt;</code></td>
<td><code>struct tm {</code>&lt;br&gt;<code>   int tm_sec;/*seconds after the minute ( 0 to 61 )*/</code>&lt;br&gt;<code>   /*allows for up to two leap seconds*/</code>&lt;br&gt;<code>   int tm_min;/*minutes after the hour ( 0 to 59 )*/</code>&lt;br&gt;<code>   int tm_hour;/*hours since midnight ( 0 to 23 )*/</code>&lt;br&gt;<code>   int tm_mday;/*day of month ( 1 to 31 )*/</code>&lt;br&gt;<code>   int tm_mon;/*month ( 0 to 11 where January = 0 )*/</code>&lt;br&gt;<code>   int tm_year;/*years since 1900*/</code>&lt;br&gt;<code>   int tm_wday;/*day of week ( 0 to 6 where Sunday = 0 )*/</code>&lt;br&gt;<code>   int tm_yday;/*day of year ( 0 to 365 where January 1 = 0 )*/</code>&lt;br&gt;<code>   int tm_isdst;/*Daylight Savings Time flag*/</code>&lt;br&gt;<code> </code>}`</td>
</tr>
<tr>
<td>Remarks:</td>
<td>If <code>tm_isdst</code> 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>time_t</strong></td>
<td>Represents calendar time values.</td>
<td><code>&lt;time.h&gt;</code></td>
<td><code>typedef long time_t</code></td>
</tr>
<tr>
<td><strong>CLOCKS_PER_SEC</strong></td>
<td>Number of processor clocks per second.</td>
<td><code>&lt;time.h&gt;</code></td>
<td><code>#define CLOCKS_PER_SEC 1</code></td>
</tr>
<tr>
<td>Remarks:</td>
<td>The compiler returns clock ticks (instruction cycles) not actual time.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 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);`n```

**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:
```c
#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:
```c
#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```
### difftime

**Description:** Find the difference between two times.

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

**Prototype:**

```c
double difftime(time_t t1, time_t t0);
```

**Arguments:**

- `t1` ending time
- `t0` beginning time

**Return Value:** Returns the number of seconds between `t1` and `t0`.

**Remarks:** By default, the 16-bit compiler returns the time as instruction cycles so `difftime` returns the number of ticks between `t1` and `t0`.

**Example:**

```c
#include <time.h>  /* for clock, difftime */
#include <stdio.h> /* for printf          */
volatile int i;

int main(void)
{
    clock_t start, stop;
    double elapsed;

    start = clock();
    for (i = 0; i < 10; i++)
        stop = clock();
    printf("start = %ld\n", start);
    printf("stop = %ld\n", stop);
    elapsed = difftime(stop, start);
    printf("Elapsed time = %.0f\n", elapsed);
}
```

**Output:**

```
start = 0
stop = 317
Elapsed time = 317
```

### gmtime

**Description:** Converts calendar time to time structure expressed as Universal Time Coordinated (UTC) also known as Greenwich Mean Time (GMT).

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

**Prototype:**

```c
struct tm *gmtime(const time_t *tod);
```

**Argument:** `tod` pointer to stored time

**Return Value:** Returns the address of the time structure.

**Remarks:** This function breaks down the `tod` 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.
### gmt ime (Continued)

**Example:**

```c
#include <time.h> /* for gmt ime, 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:**

```c
#include <time.h>
```

**Prototype:**

```c
struct tm *localtime(const time_t *t o d);
```

**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:**

```c
#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
### 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, */
#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)**

- `%I` 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:**

```c
#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.
**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 */  

```c
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

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 C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) install directory (`c:\Program Files\Microchip\MPLAB C30`) 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.
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® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51317) and “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284) for more information on the heap.
### <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

**Description:** `HUGE_VAL` 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).

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

**Remarks:** `-HUGE_VAL` 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 +/- `HUGE_VAL`, it will be represented by +/- `inf`.

### acos

**Description:** Calculates the trigonometric arc cosine function of a double precision floating-point value.

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

**Prototype:**

```c
double acos (double x);
```

**Argument:** `x` value between -1 and 1 for which to return the arc cosine

**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:**

```c
#include <math.h>  /* for acos           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* 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);
}
```
acos (Continued)

```c
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:**

```c
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:**

```c
#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
asinh

Description: Calculates the trigonometric arc sine function of a double precision floating-point value.

Include: <math.h>

Prototype: double asinh (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:
```c
#include <math.h> /* for asinh */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
  double x, y;

  errno = 0;
  x = 2.0;
  y = asinh (x);
  if (errno)
    perror("Error");
  printf("The arcsine of %.1f is %.10f\n", x, y);

  errno = 0;
  x = 0.0;
  y = asinh (x);
  if (errno)
    perror("Error");
  printf("The arcsine of %.1f is %.10f\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:
```c
#include <math.h> /* for asinf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
  float x, y;
```
asinf (Continued)

```c
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:
```c
#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
### atanf

**Description:** Calculates the trigonometric arc tangent function of a single precision floating-point value.

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

**Prototype:**

```c
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:**

```c
#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:**

```c
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 +/- infinity.

**Example:**

```c
#include <math.h>  /* for atan2 */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x, y, z;
```
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
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
+/- infinity.

Example:
```c
#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
# ceil

**Description:** Calculates the ceiling of a value.

**Include:**

```c
#include <math.h>
```

**Prototype:**

```c
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:**

```c
#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
```
ceilf

**Description:** Calculates the ceiling of a value.

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

**Prototype:**
```c
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:**
```c
#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:**
```c
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:**
```c
#include <math.h> /* for cos */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x,y;
```
### cos (Continued)

```c
errno = 0;
x = -1.0;
y = cosf(x);
if (errno)
    perror("Error");
    printf("The cosine of %f is %f\n\n", x, y);
errno = 0;
x = 0.0;
y = cosf(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:**

```c
#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);
}
```
cos (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:

```c
#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:
```c
#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
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
expf

Description: Calculates the exponential function of x (e raised to the power x where x is a single precision floating-point value).

Include:

```
#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
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:
```c
#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:
```c
#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
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:
```c
#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 ceil.
Example:
```c
#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:**
```c
#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);
}
fmod (Continued)

```c
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);
```

```c
}
```
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 is 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);
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
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:

```c
#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
 the fraction is %f
", x, y);
    printf(" and the exponent is %d\n", n);

    x = -2.5;
    y = frexp (x, &n);
    printf("For frexp of %f
 the fraction is %f
", x, y);
    printf(" and the exponent is %d\n", n);

    x = 0.0;
    y = frexp (x, &n);
    printf("For frexp of %f
 the fraction is %f
", x, y);
    printf(" and the exponent is %d\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
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:**
```c
#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);

    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);

    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);
}
```

**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
# ldexp

**Description:** Calculates the result of a double precision floating-point number multiplied by an exponent of 2.

**Include:**

```c
#include <math.h>
```

** Prototype:**

```c
double ldexp(double x, int ex);
```

** Arguments:**

- `x` floating-point value
- `ex` integer exponent

** Return Value:** Returns \( x \times 2^ex \). On an overflow, `ldexp` returns `inf` and on an underflow, `ldexp` returns `0`.

** Remarks:** A range error will occur on overflow or underflow.

** Example:**

```c
#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
",
            x, n);
    printf("  ldexp(%f, %d) = %f
",
            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
",
            x, n);
    printf("  ldexp(%f, %d) = %f
",
            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
",
            x, n);
    printf("  ldexp(%f, %d) = %f
",
            x, n, y);
}
```
## ldexp (Continued)

Output:
For a number = -0.625000 and an exponent = 2
\[ \text{ldexp}(-0.625000, 2) = -2.500000 \]

For a number = 2.500000 and an exponent = 3
\[ \text{ldexp}(2.500000, 3) = 20.000000 \]

Error: range error
For a number = 15.000000 and an exponent = 10000
\[ \text{ldexp}(15.000000, 10000) = \text{inf} \]

### 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 \( x \cdot 2^e \). On an overflow, `ldexp` returns `inf` and on an underflow, `ldexp` returns 0. |
| Remarks: | A range error will occur on overflow or underflow. |
| Example: | 
```c
#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)

```c
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
**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 ≤ 0.

**Example:**
```c
#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", x, y);

    errno = 0;
    x = 0.0;
    y = log(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n", x, y);

    errno = 0;
    x = -2.0;
    y = log(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\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
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 ≤ 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
**log10f**

**Description:**
Calculates the base-10 logarithm of a single precision floating-point value.

**Include:**
<math.h>

**Prototype:**
float log10f(float x);

**Argument:**
any single precision floating-point positive number

**Return Value:**
Returns the base-10 logarithm of \(x\). \(-\infty\) 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
**logf**

**Description:** Calculates the natural logarithm of a single precision floating-point value.

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

**Prototype:**

```c
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 ≤ 0`.

**Example:**

```c
#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 %.16f is %.16f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = logf(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %.16f is %.16f\n", x, y);

    errno = 0;
    x = -2.0F;
    y = logf(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %.16f is %.16f\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
```
**modf**

**Description:** Splits a double precision floating-point value into fractional and integer parts.

**Include:**

```c
<math.h>
```

**Prototype:**

```c
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:**

```c
#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
", x, y);
    printf(" and the integer is %0.f\n", n);

    x = -15.2121;
    y = modf (x, &n);
    printf("For %f the fraction is %f
", x, y);
    printf(" and the integer is %0.f\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
### modff

**Description:** Splits a single precision floating-point value into fractional and integer parts.

**Include:**
```
#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
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", 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", 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", 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", x, y, z);
}

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);
}

pow (Continued)
### 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:**

```c
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:**

```c
#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", x, y);
    
    errno = 0;
    x = 0.0;
    y = sin (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n", x, y);
}
```

**Output:**

The sine of -1.000000 is -0.841471

The sine of 0.000000 is 0.000000
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:
```c
#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

<table>
<thead>
<tr>
<th>Description:</th>
<th>Calculates the hyperbolic sine function of a double precision floating-point value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;math.h&gt;</code></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>double sinh (double x);</code></td>
</tr>
<tr>
<td>Argument:</td>
<td><code>x</code> value for which to return the hyperbolic sine</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns the hyperbolic sine of <code>x</code></td>
</tr>
<tr>
<td>Remarks:</td>
<td>A range error will occur if the magnitude of <code>x</code> is too large.</td>
</tr>
<tr>
<td>Example:</td>
<td>#include <code>&lt;math.h&gt;</code> /* for sinh */</td>
</tr>
<tr>
<td></td>
<td>#include <code>&lt;stdio.h&gt;</code> /* for printf, perror */</td>
</tr>
<tr>
<td></td>
<td>#include <code>&lt;errno.h&gt;</code> /* for errno */</td>
</tr>
<tr>
<td></td>
<td>int main(void)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>double x, y;</td>
</tr>
<tr>
<td></td>
<td>errno = 0;</td>
</tr>
<tr>
<td></td>
<td>x = -1.5;</td>
</tr>
<tr>
<td></td>
<td>y = sinh (x);</td>
</tr>
<tr>
<td></td>
<td>if (errno)</td>
</tr>
<tr>
<td></td>
<td>perror(&quot;Error&quot;);</td>
</tr>
<tr>
<td></td>
<td>printf(&quot;The hyperbolic sine of %f is %f\n\n&quot;,</td>
</tr>
<tr>
<td></td>
<td>x, y);</td>
</tr>
<tr>
<td></td>
<td>errno = 0;</td>
</tr>
<tr>
<td></td>
<td>x = 0.0;</td>
</tr>
<tr>
<td></td>
<td>y = sinh (x);</td>
</tr>
<tr>
<td></td>
<td>if (errno)</td>
</tr>
<tr>
<td></td>
<td>perror(&quot;Error&quot;);</td>
</tr>
<tr>
<td></td>
<td>printf(&quot;The hyperbolic sine of %f is %f\n\n&quot;,</td>
</tr>
<tr>
<td></td>
<td>x, y);</td>
</tr>
<tr>
<td></td>
<td>errno = 0;</td>
</tr>
<tr>
<td></td>
<td>x = 720.0;</td>
</tr>
<tr>
<td></td>
<td>y = sinh (x);</td>
</tr>
<tr>
<td></td>
<td>if (errno)</td>
</tr>
<tr>
<td></td>
<td>perror(&quot;Error&quot;);</td>
</tr>
<tr>
<td></td>
<td>printf(&quot;The hyperbolic sine of %f is %f\n\n&quot;,</td>
</tr>
<tr>
<td></td>
<td>x, y);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Output:</td>
<td>The hyperbolic sine of -1.500000 is -2.129279</td>
</tr>
<tr>
<td></td>
<td>The hyperbolic sine of 0.000000 is 0.000000</td>
</tr>
<tr>
<td></td>
<td>Error: range error</td>
</tr>
<tr>
<td></td>
<td>The hyperbolic sine of 720.000000 is inf</td>
</tr>
</tbody>
</table>
### sinh

**Description:** Calculates the hyperbolic sine function of a single precision floating-point value.

**Include:**

```c
#include <math.h>
```

**Prototype:**

```c
float sinh (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:**

```c
#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
### 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:**

```c
#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**

<table>
<thead>
<tr>
<th>Description:</th>
<th>Calculates the square root of a single precision floating-point value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;math.h&gt;</code></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>float sqrtf(float x);</code></td>
</tr>
<tr>
<td>Argument:</td>
<td><code>x</code>  non-negative floating-point value</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns the non-negative square root of <code>x</code>.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>If <code>x</code> is negative, a domain error occurs.</td>
</tr>
<tr>
<td>Example:</td>
<td>#include <code>&lt;math.h&gt;</code>  /* for sqrtf */</td>
</tr>
<tr>
<td></td>
<td>#include <code>&lt;stdio.h&gt;</code> /* for printf, perror */</td>
</tr>
<tr>
<td></td>
<td>#include <code>&lt;errno.h&gt;</code> /* for errno */</td>
</tr>
</tbody>
</table>
|             | 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
### tan

**Description:** Calculates the trigonometric tangent function of a double precision floating-point value.

**Include:**
```c
#include <math.h>
```

**Prototype:**
```c
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:*
```c
#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:**
```c
#include <math.h>
```

**Prototype:**
```c
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:*
```c
#include <math.h>  /* for tanf           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;
```
Standard C Libraries - Math Functions

**tanf (Continued)**

```c
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:**
```c
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:**
```c
#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 C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) install directory \( \text{(c:\Program Files\Microchip\MPLAB C30)} \) 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
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.

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® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51317) and “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284) for more information on the heap.

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.
### 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
<th>Argument</th>
<th>Return Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>_exit</td>
<td>Terminate program execution.</td>
<td>None</td>
<td>void _exit (int status);</td>
<td>status exit status</td>
<td></td>
<td>This is a helper function called by the <code>exit()</code> Standard C Library function.</td>
</tr>
<tr>
<td>brk</td>
<td>Set the end of the process’s data space.</td>
<td>None</td>
<td>int brk(void *endds);</td>
<td>endds pointer to the end of the data segment</td>
<td>Returns '0' if successful, '-1' if not.</td>
<td>brk() 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. Newly allocated space is uninitialized. This helper function is used by the Standard C Library function, <code>malloc()</code>.</td>
</tr>
</tbody>
</table>
brk (Continued)

Default Behavior:
If the argument, \textit{endds}, is zero, the function sets the global variable, \_\textunderscore curbrk, to the address of the start of the heap and returns zero.
If the argument, \textit{endds}, is non-zero, and has a value less than the address of the end of the heap, the function sets the global variable, \_\textunderscore curbrk, to the value of \textit{endds} and returns zero.
Otherwise, the global variable, \_\textunderscore curbrk, is unchanged and the function returns -1.
The argument, \textit{endds}, must be within the heap range (see data space memory map below).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{heap_memory_map.png}
\caption{Data Space Memory Map}
\end{figure}

Notice that, since the stack is located immediately above the heap, using \texttt{brk()} or \texttt{sbrk()} has little effect on the size of the dynamic memory pool. The \texttt{brk()} and \texttt{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\texttt{-Wl,--heap=n} option is specified, where \textit{n} is the desired heap size in characters. The starting and ending addresses of the heap are reported in variables, \_\textunderscore heap and \_\textunderscore 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 \texttt{brk()} and \texttt{sbrk()}.

File:
\texttt{brk.c}

close

Description: Close a file.
Include: None
Prototype: int close(int \texttt{handle});
Argument: \texttt{handle} \quad \texttt{handle} referring to an opened file
Return Value: Returns '0' if the file is successfully closed. A return value of '-1' indicates an error.
Remarks: This helper function is called by the \texttt{fclose()} Standard C Library function.
Default Behavior: As distributed, this function passes the file handle to the simulator, which issues a close in the host file system.

File: \texttt{close.c}
Iseek

Description: Move a file pointer to a specified location.

Include: None

Prototype: long lseek(int handle, long offset, int origin);

Argument:

- handle refers to an opened file
- offset the number of characters from the origin
- origin the position from which to start the seek. origin may be one of the following values (as defined in stdio.h):
  - SEEK_SET – Beginning of file.
  - SEEK_CUR – Current position of file pointer.

Return Value: Returns the offset, in characters, of the new position from the beginning of the file. A return value of -1L indicates an error.

Remarks: This helper function is called by the Standard C Library functions, fgetpos(), ftell(), fseek(), fsetpos and rewind().

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: lseek.c

open

Description: Open a file.

Include: None

Prototype: int open(const char *name, int access, int mode);

Argument:

- name name of the file to be opened
- access access method to open file
- mode type of access permitted

Return Value: 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.

Remarks: 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, fopen() and freopen().

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. If the host system returns a value of -1, the global variable, errno, is set to the value of the symbolic constant, EOPEN, defined in <errno.h>.

File: open.c
### read

**Description:** Read data from a file.

**Include:** None

**Prototype:**
```c
int read(int handle, void *buffer, unsigned int len);
```

**Argument:**
- `handle` handle referring to an opened file
- `buffer` points to the storage location for read data
- `len` the maximum number of characters to read

**Return Value:** Returns the number of characters read, which may be less than `len` if there are fewer than `len` 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`.

**Remarks:** This helper function is called by the Standard C Library functions `fgetc()`, `fgets()`, `fread()` and `gets()`.

**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:** `read.c`

### sbrk

**Description:** Extend the process’ data space by a given increment.

**Include:** None

**Prototype:**
```c
void * sbrk(int incr);
```

**Argument:**
- `incr` number of characters to increment/decrement

**Return Value:** Return the start of the new space allocated or `-1` for errors.

**Remarks:** `sbrk()` adds `incr` characters to the break value and changes the allocated space accordingly. `incr` can be negative, in which case, the amount of allocated space is decreased.

`sbrk()` 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.

This is a helper function called by the Standard C Library function, `malloc()`.

**Default Behavior:** If the global variable, `__curbrk`, is zero, the function calls `brk()` to initialize the break value. If `brk()` returns -1, so does this function. If the `incr` is zero, the current value of the global variable, `__curbrk`, is returned.

If the `incr` is non-zero, the function checks that the address, `(__curbrk + incr)`, is less than the end address of the heap. If it is less, the global variable, `__curbrk`, is updated to that value and the function returns the unsigned value of `__curbrk`. Otherwise, the function returns -1.

See the description of `brk()`.

**File:** `sbrk.c`
### write

**Description:** Write data to a file.

**Include:** None

**Prototype:**

```c
int write(int handle, void *buffer,
          unsigned int count);
```

**Argument:**
- `handle` refers to an opened file
- `buffer` points to the storage location of data to be written
- `count` the number of characters to write.

**Return Value:** If successful, `write` returns the number of characters actually written. A return value of `-1` indicates an error.

**Remarks:** If the actual space remaining on the disk is less than the size of the buffer, the function is 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.

This is a helper function called by the Standard C Library function, `fflush()`.

**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:** `write.c`
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
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 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
### __delay32

**Description:** Produce a delay of a specified number of clock cycles.

**Include:** `<libpic30.h>`

**Prototype:**
```c
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 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:**
```c
void __delay_ms(unsigned int time);
```

**Argument:**
- `time`: number of ms to delay

**Remarks:** This function is implemented as a macro. The 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:**
```c
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:**
```c
int __C30_UART;
```
### EXAMPLE 4-1: UART1 I/O

```c
#include <libpic30.h>      /* a new header file for
    these definitions */
#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) */

```c
#include <libpic30.h>      /* a new header file for these
definitions */
#include <stdio.h>

#ifndef __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);
    }
}
```
EXAMPLE 4-3: MILLISECOND DELAY

```c
#define FCY 1000000UL
#include <libpic30.h>

int main()
{
    /* at 1MHz, these are equivalent */
    __delay_ms(1);
    __delay32(1000);
}
```

EXAMPLE 4-4: MICROSECOND DELAY

```c
#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**

<table>
<thead>
<tr>
<th>Description:</th>
<th>Erase EEDATA memory on dsPIC30F and PIC24FXXKA devices.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;libpic30.h&gt;</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void _erase_eedata(_prog_addressT dst, int len);</td>
</tr>
<tr>
<td>Argument:</td>
<td>dst destination memory address</td>
</tr>
<tr>
<td></td>
<td>len dsPIC30F: length may be _EE_WORD or _EE_ROW (bytes)</td>
</tr>
<tr>
<td></td>
<td>PIC24FXXKA: length may be _EE_WORD, _EE_4WORDS or _EE_8WORDS (bytes)</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>None.</td>
</tr>
<tr>
<td>Default Behavior:</td>
<td>Erase EEDATA memory as specified by parameters.</td>
</tr>
<tr>
<td>File:</td>
<td>eedata_helper.c</td>
</tr>
</tbody>
</table>

**_erase_eedata_all**

<table>
<thead>
<tr>
<th>Description:</th>
<th>Erase the entire range of EEDATA memory on dsPIC30F and PIC24FXXKA devices.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>&lt;libpic30.h&gt;</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void _erase_eedata_all(void);</td>
</tr>
<tr>
<td>Argument:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>None.</td>
</tr>
<tr>
<td>Default Behavior:</td>
<td>Erase all EEDATA memory for the selected device.</td>
</tr>
<tr>
<td>File:</td>
<td>eedata_helper.c</td>
</tr>
</tbody>
</table>
Example of Use – dsPIC30F DSCs

```c
#include "libpic30.h"
#include "p30fxxxx.h"

char __attribute__((space(eedata), aligned(_EE_ROW))) data[_EE_ROW];

int main()
{
    char i, source[_EE_ROW];
    _prog_addressT p;

    for (i = 0; i < _EE_ROW;)
        source[i] = i++;             /* initialize some data */

    // Use the functions...

    #wait_eedata();
    #write_eedata_word(dst, dat);
    #write_eedata_row(dst, *src);
}
```
Example of Use – PIC24FXXKA MCUs

```c
#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_4ROWS);    /* 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();
}
```

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.

```c
void _erase_flash(_prog_addressT dst);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Erase a page of Flash memory. The length of a page is _FLASH_PAGE words (1 word = 3 bytes = 2 PC address units).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>&lt;libpic30.h&gt;</td>
</tr>
<tr>
<td>Prototype</td>
<td>void _erase_flash(_prog_addressT dst);</td>
</tr>
<tr>
<td>Argument</td>
<td>dst destination memory address</td>
</tr>
<tr>
<td>Return Value</td>
<td>None.</td>
</tr>
<tr>
<td>Remarks</td>
<td>None.</td>
</tr>
<tr>
<td>Default Behavior</td>
<td>Erase a page of Flash memory.</td>
</tr>
<tr>
<td>File</td>
<td>flash_helper.c</td>
</tr>
</tbody>
</table>
_erase_flash (PIC24FXXKA Only)

**Description:** Erase rows of Flash memory, either one, two or four rows.

**Include:** <libpic30.h>

**Prototype:**

```c
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_helper.c

_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:**

```c
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.c

_write_flash24

**Description:** Write a row of Flash memory with 24-bit data. The length of a row is _FLASH_ROW words. Note that the row must be erased before any write can be successful.

**Include:** <libpic30.h>

**Prototype:**

```c
void _write_flash24(_prog_addressT dst, long *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 24-bit data.

**File:** flash_helper.c
**_write_flash_word16**

**Description:** 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).

**Include:** <libpic30.h>

**Prototype:**

```c
void _write_flash_word16(_prog_addressT dst, int dat);
```

**Argument:**
- `dst` destination memory address
- `dat` integer data to be written

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Write a word of Flash memory with 16-bit data for most PIC24 devices.

**File:** flash_helper.c

---

**_write_flash_word24**

**Description:** 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).

**Include:** <libpic30.h>

**Prototype:**

```c
void _write_flash_word24(_prog_addressT dst, int dat);
```

**Argument:**
- `dst` destination memory address
- `dat` integer data to be written

**Return Value:** None.

**Remarks:** None.

**Default Behavior:** Write a word of Flash memory with 24-bit data for most PIC24 devices.

**File:** flash_helper.c

---

**Example of Use**

```c
#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 */
```

---
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 */
}

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.
### _memcpy_p2d16 (Continued)

<table>
<thead>
<tr>
<th>Default Behavior</th>
<th>Copy 16 bits of data from each address in program memory to data memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>memcpy_helper.c</td>
</tr>
</tbody>
</table>

### _memcpy_p2d24

<table>
<thead>
<tr>
<th>Description</th>
<th>Copy 24 bits of data from each address in program memory to data memory. The next unused source address is returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>&lt;libpic30.h&gt;</td>
</tr>
<tr>
<td>Prototype</td>
<td>_prog_addressT _memcpy_p2d24(char *dest, _prog_addressT src, unsigned int len);</td>
</tr>
<tr>
<td>Argument</td>
<td>*dest pointer to destination memory address</td>
</tr>
<tr>
<td></td>
<td>src address of data to be written</td>
</tr>
<tr>
<td></td>
<td>len length of program memory</td>
</tr>
<tr>
<td>Return Value</td>
<td>The next unused source address.</td>
</tr>
<tr>
<td>Remarks</td>
<td>None.</td>
</tr>
<tr>
<td>Default Behavior</td>
<td>Copy 24 bits of data from each address in program memory to data memory.</td>
</tr>
<tr>
<td>File</td>
<td>memcpy_helper.c</td>
</tr>
</tbody>
</table>

### _strncpy_p2d16

<table>
<thead>
<tr>
<th>Description</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>&lt;libpic30.h&gt;</td>
</tr>
<tr>
<td>Prototype</td>
<td>_prog_addressT _strncpy_p2d16(char *dest, _prog_addressT src, unsigned int len);</td>
</tr>
<tr>
<td>Argument</td>
<td>*dest pointer to destination memory address</td>
</tr>
<tr>
<td></td>
<td>src address of data to be written</td>
</tr>
<tr>
<td></td>
<td>len length of program memory</td>
</tr>
<tr>
<td>Return Value</td>
<td>The next unused source address.</td>
</tr>
<tr>
<td>Remarks</td>
<td>None.</td>
</tr>
<tr>
<td>Default Behavior</td>
<td>Copy 16 bits of data from each address in program memory to data memory.</td>
</tr>
<tr>
<td>File</td>
<td>memcpy_helper.c</td>
</tr>
</tbody>
</table>

### _strncpy_p2d24

<table>
<thead>
<tr>
<th>Description</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>&lt;libpic30.h&gt;</td>
</tr>
<tr>
<td>Prototype</td>
<td>_prog_addressT _strncpy_p2d24(char *dest, _prog_addressT src, unsigned int len);</td>
</tr>
<tr>
<td>Argument</td>
<td>*dest pointer to destination memory address</td>
</tr>
<tr>
<td></td>
<td>src address of data to be written</td>
</tr>
<tr>
<td></td>
<td>len length of program memory</td>
</tr>
<tr>
<td>Return Value</td>
<td>The next unused source address.</td>
</tr>
<tr>
<td>Remarks</td>
<td>None.</td>
</tr>
</tbody>
</table>

Remarks: None.
### `_strncpy_p2d24` (Continued)

**Default Behavior:** Copy 24 bits of data from each address in program memory to data memory.

**File:** memcpy_helper.c

### `_init_prog_address`

**Description:** A macro that is used to initialize variables of type `_prog_addressT`. These variables are not equivalent to C pointers.

**Include:** `<libpic30.h>`

**Prototype:**

```
_init_prog_address(a, b);
```

**Argument:**

- `a` variable of type `_prog_addressT`
- `b` initialization value for variable `a`

**Return Value:** N/A

**Remarks:** None.

**Default Behavior:** Initialize variable to specified value.

**File:** libpic30.c
Example of Use

```c
#include "stdio.h"
#include "libpic30.h"

#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");
}

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 C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) install directory (c:\Program Files\Microchip\MPLAB C30) 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
5.2 OVERVIEW OF FIXED POINT DATA FORMATS

The integer data is encoded as its two’s compliment to accommodate both positive and negative numbers in binary format. The two’s compliment 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.

For the binary representation of an N-bit (m+n+1 bits) number in Qn.m:

$$b_{m+n} b_{m+n-1} \ldots b_m \cdot b_{m-1} \ldots b_1 b_0$$

the value is given by the equation:

$$Value = -b_{N-1} 2^n + \sum_{i=0}^{N-2} b_i 2^{i-m}$$

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^{-N}). For a 16-bit fraction, the 1.15 data range is -1.0 (0x8000) to +0.999969482 (0x7FFF) with a precision of 3.05176x10^{-5}.

FIGURE 5-1: FRACTIONAL FORMAT (16 BITS)
The following example shows the conversion of a two’s complement 16-bit integer +24576 to Q15 value +0.75.

**EXAMPLE 5-1: CONVERSION OF A TWO'S COMPLEMENT 16-BIT INTEGER TO Q15**

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Q15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 x (-2^16)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 x 2^14</td>
<td>16384</td>
</tr>
<tr>
<td>1</td>
<td>1 x 2^13</td>
<td>8192</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^12</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^10</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^9</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^8</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^7</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^6</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^0</td>
<td>0</td>
</tr>
</tbody>
</table>

Sum +24576

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 5-2: FRACTIONAL FORMAT (32 BITS)**
The following example shows the conversion of a two’s complement 32-bit integer, -715827882 to Q15.16 value -10922.6666564941.

**EXAMPLE 5-2: CONVERSION OF A TWO’S COMPLEMENT 32-BIT INTEGER TO Q15.16**

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Q15.16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 \times 2^{31}</td>
<td>-2147483648</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{30}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{29}</td>
<td>268435456</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{28}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{27}</td>
<td>67108864</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{26}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{25}</td>
<td>16777216</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{24}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{23}</td>
<td>4194304</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{22}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{21}</td>
<td>1048576</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{20}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{19}</td>
<td>262144</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{18}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{17}</td>
<td>65536</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{16}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{15}</td>
<td>16384</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{14}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{13}</td>
<td>4096</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{12}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{11}</td>
<td>1024</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{10}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{9}</td>
<td>256</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{8}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{7}</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{6}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{5}</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{4}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{3}</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{2}</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0 \times 2^{1}</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 \times 2^{0}</td>
<td>0</td>
</tr>
</tbody>
</table>

**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 eg., the 16-bit integer range [-32768, +32767] format maps to Q15 range of [-1.0, +0.999969482]. Figure 5-3 shows the mapping between these formats.

**FIGURE 5-3: 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 5-4: MAPPING BETWEEN 32-BIT INTEGER FORMAT AND Q15.16 FORMAT**

<table>
<thead>
<tr>
<th>Integer</th>
<th>Q15.16 Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-32768.0</td>
<td>-1845.333129882</td>
</tr>
<tr>
<td>-10922.6666564941</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1431655576</td>
<td>+10922.6666564941</td>
</tr>
<tr>
<td>2147483648</td>
<td>+32767.9999847412109375</td>
</tr>
</tbody>
</table>

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:

```c
#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.

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:

\[ Q_n_m \]

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:

<table>
<thead>
<tr>
<th>Exact Name</th>
<th># Bits Required</th>
<th>Short Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Q0_15</td>
<td>16</td>
<td>_Q15</td>
</tr>
<tr>
<td>_Q15_16</td>
<td>32</td>
<td>_Q16</td>
</tr>
</tbody>
</table>

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, \(_{Q15}acos\) 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 \(_{itoa_{Q15}}\) 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:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Return Type</th>
<th>Argument Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(<em>{Q16}reciprocal</em>{Q15})</td>
<td>(_{Q16})</td>
<td>(_{Q15})</td>
</tr>
<tr>
<td>(<em>{Q16}reciprocal</em>{Q16})</td>
<td>(_{Q16})</td>
<td>(_{Q16})</td>
</tr>
</tbody>
</table>
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 (π).
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.
### _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 (π).

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 (π).

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.
---

**_Q15cos**

**Description:** This function finds the cosine of a Q15 value.

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

**Prototype:**

```c
_Q15 _Q15cos(_Q15 x);
```

**Argument:**

- `x` a fixed point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}$. 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 (π) times a Q15 value.

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

**Prototype:**

```c
_Q15 _Q15cosPI(_Q15 x);
```

**Argument:**

- `x` a fixed point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}$. 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:**

```c
_Q15 _Q15exp(_Q15 x);
```

**Argument:**

- `x` a fixed point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}$. The value of this argument ranges from -32768 to 0.

**Return Value:** This function returns the exponential 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:**

```c
_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:**

```c
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:**

```c
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:**

```c
_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:**

```c
_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.
___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:**

```c
_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:**

```c
_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:**

```c
_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:**

```c
_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.
_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 (π) 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:**
```
#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:**
```
#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:**
```
#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 $-2^{5736}$ to $2^{5735}$.

**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 ($\pi$) times a Q15 value.

**Include:**
```
#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 $32767$. 
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

**Description:** This function finds the arc tangent of a Q16 value.

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

**Prototype:**

```c
_Q16 _Q16atan(_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 arc tangent of `x` in Q16 format. The value ranges from -2147483648 to 2147483647.

## _Q16atanByPI

**Description:** This function finds the arc tangent of a Q16 value and then divides by PI (π).

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

**Prototype:**

```c
_Q16 _Q16atanByPI(_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 arc tangent of `x`, divided by PI, in Q16 format. The value ranges from -2147483648 to 2147483647.

## _Q16atanYByX

**Description:** This function finds the arc tangent of `y` divided by `x`.

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

**Prototype:**

```c
_Q16 _Q16atanYByX(_Q16 x, _Q16 y);
```

**Argument:**

- `x`: a fixed point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the x input.
- `y`: a fixed point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the y input.

**Return Value:**

This function returns the arc tangent of `y`, divided by `x`, in Q16 format. The value ranges from -2147483648 to 2147483647.

## _Q16atanYByXByPI

**Description:** This function finds the arc tangent of the 32-bit input, `y`, divided by `x`, and then divides by PI (π).

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

**Prototype:**

```c
_Q16 _Q16atanYByXByPI(_Q16 x, _Q16 y);
```

**Argument:**

- `x`: a fixed point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the x input.
- `y`: a fixed point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647. This forms the y input.

**Return Value:**

This function returns the arc tangent `y` divided by `x`, divided by PI, in Q16 format. The value ranges from -2147483648 to 2147483647.
_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 (π) 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.

_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 \( \text{prod} \). The function takes care of saturating the result in case of underflow or overflow.

Include: `<libq.h>

Prototype: 

\[
\text{Q16} \_\text{Q16mac}\text{(Q16}_x, \_\text{Q16} y, \_\text{Q16} \text{prod});
\]

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.
- \( \text{prod} \) 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 \( \text{prod} \) in Q16 format. The value ranges from 0 to 2147483647.

_Q16macNoSat

Description: This function multiplies the two 32 bit inputs, \( x \) and \( y \) and accumulates the product with \( \text{prod} \). 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: 

\[
\text{Q16} \_\text{Q16macNoSat}\text{(Q16}_x, \_\text{Q16} y, \_\text{Q16} \text{prod});
\]

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.
- \( \text{prod} \) 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 \( \text{prod} \) in Q16 format. 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: 

\[
\text{Q16} \_\text{Q16neg}\text{(Q16}_x);
\]

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**

Description: This function finds the power result, given the base value, `x`, and the power value, `y`.

Include: `<libq.h>`

Prototype: `_Q16 _Q16power(_Q16 x, _Q16 y);`

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.

Return Value: This function returns the value of `x` to the power of `y` in Q16 format. The value ranges from 0 to 2147483647.

**_Q16random**

Description: This function generates a pseudo random number with a period of 2147483648. The function uses the _Q16randomSeed variable as a random seed value.

Include: `<libq.h>`

Prototype: `_Q16 _Q16random(void);`

Argument: None.

Return Value: This function returns the generated random number in Q16 format. The value of this output ranges from -2147483648 to 2147483648.

Remarks: RndNum(n) = (RndNum(n-1) * RAN_MULT) + RAN_INC
SEED VALUE = 21845, RAN_MULT = 1664525 and RAN_INC = 1013904223.

**_Q16reciprocal**

Description: This function returns the reciprocal of a Q16 value.

Include: `<libq.h>`

Prototype: `_Q16 _Q16reciprocal(_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.
### _Q16reciprocalQ15

**Description:** 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.

**Include:**
```c
#include <libq.h>
```

**Prototype:**
```c
_Q16 _Q16reciprocalQ15(_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 reciprocal of `x` in Q16 format. The value ranges from -2147483648 to 2147483647.

### _Q16reciprocalQ16

**Description:** This function returns the reciprocal value of the input.

**Include:**
```c
#include <libq.h>
```

**Prototype:**
```c
_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:**
```c
#include <libq.h>
```

**Prototype:**
```c
_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:**
```c
#include <libq.h>
```

**Prototype:**
```c
_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 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 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:

```
#include <libq.h>
```

Prototype:

```
short _Q16sinSeries(_Q16 f, short start, short num,
_Q16 *buf);
```

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:

```
#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:

```
#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.
## Appendix A. ASCII Character Set

### TABLE A-1: ASCII CHARACTER SET

<table>
<thead>
<tr>
<th>Hex</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>Space</td>
<td>@</td>
<td>P</td>
<td>’</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>Bell</td>
<td>ETB</td>
<td>’</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>{</td>
<td>k</td>
<td>{</td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
</tr>
</tbody>
</table>
# 16-BIT LANGUAGE TOOLS

## Symbol Index

### Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>^, Caret</td>
<td>84</td>
</tr>
<tr>
<td>_attach_input_file</td>
<td>201</td>
</tr>
<tr>
<td>_C30_UART</td>
<td>202</td>
</tr>
<tr>
<td>_close_input_file</td>
<td>201</td>
</tr>
<tr>
<td>_delay_ms</td>
<td>202</td>
</tr>
<tr>
<td>_delay_us</td>
<td>202</td>
</tr>
<tr>
<td>_delay32</td>
<td>202</td>
</tr>
<tr>
<td>_FILE</td>
<td>17</td>
</tr>
<tr>
<td>_LINE</td>
<td>17</td>
</tr>
<tr>
<td>_erase_eedata</td>
<td>204</td>
</tr>
<tr>
<td>_erase_eedata_all</td>
<td>204</td>
</tr>
<tr>
<td>_erase_flash</td>
<td>206, 207</td>
</tr>
<tr>
<td>exit</td>
<td>195</td>
</tr>
<tr>
<td>_init_prog_address</td>
<td>211</td>
</tr>
<tr>
<td>_IOFBF</td>
<td>50, 58, 86</td>
</tr>
<tr>
<td>_OLBF</td>
<td>50, 86</td>
</tr>
<tr>
<td>_ONBF</td>
<td>50, 58, 86</td>
</tr>
<tr>
<td>_itoaQ15</td>
<td>223</td>
</tr>
<tr>
<td>_itoaQ15</td>
<td>223</td>
</tr>
<tr>
<td>_itoaQ16</td>
<td>236</td>
</tr>
<tr>
<td>_MathError</td>
<td>41</td>
</tr>
<tr>
<td>_memcpy_p2d16</td>
<td>209</td>
</tr>
<tr>
<td>_memcpy_p2d24</td>
<td>210</td>
</tr>
<tr>
<td>_NSETJMP</td>
<td>36</td>
</tr>
<tr>
<td>_Q15abs</td>
<td>219</td>
</tr>
<tr>
<td>_Q15acos</td>
<td>219</td>
</tr>
<tr>
<td>_Q15acosByPI</td>
<td>219</td>
</tr>
<tr>
<td>_Q15add</td>
<td>220</td>
</tr>
<tr>
<td>_Q15asin</td>
<td>220</td>
</tr>
<tr>
<td>_Q15asinByPI</td>
<td>220</td>
</tr>
<tr>
<td>_Q15atan</td>
<td>220</td>
</tr>
<tr>
<td>_Q15atanByPI</td>
<td>221</td>
</tr>
<tr>
<td>_Q15atanYByX</td>
<td>221</td>
</tr>
<tr>
<td>_Q15atanYByXByPI</td>
<td>221</td>
</tr>
<tr>
<td>_Q15atoi</td>
<td>221</td>
</tr>
<tr>
<td>_Q15cos</td>
<td>222</td>
</tr>
<tr>
<td>_Q15cosPI</td>
<td>222</td>
</tr>
<tr>
<td>_Q15exp</td>
<td>222</td>
</tr>
<tr>
<td>_Q15stoi</td>
<td>222</td>
</tr>
<tr>
<td>_Q15log</td>
<td>223</td>
</tr>
<tr>
<td>_Q15log10</td>
<td>223</td>
</tr>
<tr>
<td>_Q15neg</td>
<td>224</td>
</tr>
<tr>
<td>_Q15norm</td>
<td>224</td>
</tr>
<tr>
<td>_Q15power</td>
<td>224</td>
</tr>
<tr>
<td>_Q15random</td>
<td>224</td>
</tr>
<tr>
<td>_Q15shl</td>
<td>225</td>
</tr>
<tr>
<td>_Q15shlNoSat</td>
<td>225</td>
</tr>
<tr>
<td>_Q15shr</td>
<td>225</td>
</tr>
<tr>
<td>_Q15shrNoSat</td>
<td>225</td>
</tr>
<tr>
<td>_Q15sin</td>
<td>226</td>
</tr>
<tr>
<td>_Q15sinPI</td>
<td>226</td>
</tr>
<tr>
<td>_Q15sinSeries</td>
<td>226</td>
</tr>
<tr>
<td>_Q15qrt</td>
<td>227</td>
</tr>
<tr>
<td>_Q15sub</td>
<td>227</td>
</tr>
<tr>
<td>_Q15tan</td>
<td>227</td>
</tr>
<tr>
<td>_Q15tanPI</td>
<td>227</td>
</tr>
<tr>
<td>_Q16acos</td>
<td>228</td>
</tr>
<tr>
<td>_Q16acosByPI</td>
<td>228</td>
</tr>
<tr>
<td>_Q16asin</td>
<td>228</td>
</tr>
<tr>
<td>_Q16asinByPI</td>
<td>228</td>
</tr>
<tr>
<td>_Q16atan</td>
<td>229</td>
</tr>
<tr>
<td>_Q16atanByPI</td>
<td>229</td>
</tr>
<tr>
<td>_Q16atanYByX</td>
<td>229</td>
</tr>
<tr>
<td>_Q16atanYByXByPI</td>
<td>229</td>
</tr>
<tr>
<td>_Q16cos</td>
<td>230</td>
</tr>
<tr>
<td>_Q16cosPI</td>
<td>230</td>
</tr>
<tr>
<td>_Q16exp</td>
<td>230</td>
</tr>
<tr>
<td>_Q16ftoi</td>
<td>236</td>
</tr>
<tr>
<td>_Q16log</td>
<td>230</td>
</tr>
<tr>
<td>_Q16log10</td>
<td>230</td>
</tr>
<tr>
<td>_Q16mac (Continued)</td>
<td>231</td>
</tr>
<tr>
<td>_Q16macNoSat</td>
<td>231</td>
</tr>
<tr>
<td>_Q16neg</td>
<td>231</td>
</tr>
<tr>
<td>_Q16norm</td>
<td>232</td>
</tr>
<tr>
<td>_Q16power</td>
<td>232</td>
</tr>
<tr>
<td>_Q16random</td>
<td>232</td>
</tr>
<tr>
<td>_Q16reciprocal</td>
<td>232</td>
</tr>
<tr>
<td>_Q16reciprocalQ15</td>
<td>233</td>
</tr>
<tr>
<td>_Q16reciprocalQ16</td>
<td>233</td>
</tr>
<tr>
<td>_Q16shl</td>
<td>233</td>
</tr>
<tr>
<td>_Q16shlNoSat</td>
<td>233</td>
</tr>
<tr>
<td>_Q16shr</td>
<td>234</td>
</tr>
<tr>
<td>_Q16shrNoSat</td>
<td>234</td>
</tr>
<tr>
<td>_Q16sin</td>
<td>234</td>
</tr>
<tr>
<td>_Q16sinPI</td>
<td>234</td>
</tr>
<tr>
<td>_Q16sinSeries</td>
<td>235</td>
</tr>
<tr>
<td>_Q16tan</td>
<td>235</td>
</tr>
<tr>
<td>_Q16tanPI</td>
<td>235</td>
</tr>
<tr>
<td>_Q16atanYByXByPI</td>
<td>235</td>
</tr>
<tr>
<td>_strncpy_p2d16</td>
<td>210</td>
</tr>
<tr>
<td>_strncpy_p2d24</td>
<td>210</td>
</tr>
<tr>
<td>_VERBOSE_DEBUGGING</td>
<td>17</td>
</tr>
<tr>
<td>_wait_eedata</td>
<td>205</td>
</tr>
<tr>
<td>_write_eedata_row</td>
<td>205</td>
</tr>
<tr>
<td>_write_eedata_word</td>
<td>205</td>
</tr>
<tr>
<td>_write_flash_word16</td>
<td>208</td>
</tr>
<tr>
<td>_write_flash_word24</td>
<td>208</td>
</tr>
<tr>
<td>_write_flash16</td>
<td>207</td>
</tr>
<tr>
<td>_write_flash24</td>
<td>207</td>
</tr>
<tr>
<td>-, Dash</td>
<td>84</td>
</tr>
<tr>
<td>\l, Form Feed</td>
<td>23</td>
</tr>
<tr>
<td>\n, Newline</td>
<td>23, 47, 59, 64, 75, 76, 80</td>
</tr>
</tbody>
</table>
16-Bit Language Tools Libraries

\r, Carriage Return............................................... 23
\t, Horizontal Tab...................................................... 23
\v, Vertical Tab......................................................... 23
#if .............................................................. 33
#include .................................................. 16, 149, 194, 217
\%, Percent.................................................. 78, 83, 84, 147

Numerics

0x .................................................. 24, 77, 115, 116

A

Abnormal Termination Signal................................. 38
abort ....................................................... 17, 95
abs ......................................................... 96

Absolute Value

Double Floating-Point ........................................ 165
Integer ................................................................ 96
Long Integer .................................................. 106
Single Floating-Point ......................................... 165

Absolute Value Function

abs ......................................................... 96
fabs ....................................................... 151
fabsf ..................................................... 152

Allocate Memory .............................................. 108
calloc ..................................................... 102
Free ........................................................ 105
realloc ..................................................... 111

Alphabetic Character

Defined ...................................................... 18
Test for .................................................... 18

Alphanumeric Character

Defined ...................................................... 18
Test for .................................................... 18

AM/PM ...................................................... 147

Append ...................................................... 124, 130

arccosine

Double Floating-Point ...................................... 151
Single Floating-Point ...................................... 152

arcsine

Double Floating-Point ...................................... 153
Single Floating-Point ...................................... 153

arctangent

Double Floating-Point ...................................... 154
Single Floating-Point ...................................... 155

arctangent of y/x

Double Floating-Point ...................................... 155
Single Floating-Point ...................................... 157

Argument List .................................................. 43, 91, 92, 93

Arithmetic Error Message .................................... 38

ASCII Character Set ........................................ 237
asctime ..................................................... 142
asin ......................................................... 153
asinf ......................................................... 153
assert ......................................................... 17
assert.h .................................................... 17

Assignment Suppression ...................................... 83

Asterisk ..................................................... 77, 83
atan ......................................................... 154
atan2 ....................................................... 155
atan2f ....................................................... 157
atanf ......................................................... 155
atexit ...................................................... 96, 104
atof ......................................................... 98
atol .......................................................... 99
atol ......................................................... 99
attach_input_file ......................................... 201

B

Base ...................................................... 115, 116
10 ......................................................... 29, 30, 31, 32, 177, 178
2 .............................................................. 31
e .......................................................... 176, 179

FLT_RADIX .............................................. 28, 29, 30, 31, 32, 33

Binary

Base ...................................................... 31
Mode ....................................................... 60, 88
Search ..................................................... 100
Streams .................................................... 47

Bitfields ..................................................... 46
brk .......................................................... 195, 198
bsearch .................................................... 100
Buffer Size ............................................ 50, 86
Buffering Modes .......................................... 86
Buffering, See File Buffering
BUFSIZ ......................................................... 50, 85

Built-in Functions ........................................... 13

C

C Locale .................................................... 18, 35
C30_UART .................................................. 202
Calendar Time ........................................... 141, 143, 144, 146, 148
calloc ....................................................... 102, 105
Caret (^) .................................................... 84
Carriage Return ........................................... 23
ceil .......................................................... 158
ceilf ........................................................ 159

ceiling

Double Floating-Point ...................................... 158
Single Floating-Point ...................................... 159

cchar

Maximum Value ............................................. 33
Minimum Value ........................................... 33
Number of Bits ............................................ 33

CHAR_BIT ............................................... 33
CHAR_MAX ............................................. 33
CHAR_MIN .............................................. 33
Character Array ........................................... 84

Character Case Mapping

Lower Case Alphabetic Character ......................... 25
Upper Case Alphabetic Character ......................... 26

Character Case Mapping Functions
tolower .................................................... 25
toupper .................................................... 26

Character Handling, See ctype.h

Character Input/Output Functions
fgetc ...................................................... 57
Index

<table>
<thead>
<tr>
<th>Character Testing</th>
<th>Classifying Characters</th>
<th>Comparison Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic Character</td>
<td>18</td>
<td>100, 110</td>
</tr>
<tr>
<td>Alphanumeric Character</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Control Character</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Decimal Digit</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Graphical Character</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Hexadecimal Digit</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Lower Case Alphabetic Character</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Printable Character</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Punctuation Character</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Upper Case Alphabetic Character</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>White-Space Character</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

| Character Testing Functions | | |
|-----------------------------|---------------------------------|
| isalnum | 18 |
| isalpha | 18 |
| iscntrl | 19 |
| isdigit | 20 |
| isgraph | 20 |
| islower | 21 |
| isprint | 22 |
| ispunct | 22 |
| ispace | 23 |
| ispos | 24 |
| isxdigit | 24 |

<table>
<thead>
<tr>
<th>Characters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic</td>
<td>18</td>
</tr>
<tr>
<td>Alphanumeric</td>
<td>18</td>
</tr>
<tr>
<td>Control</td>
<td>19</td>
</tr>
<tr>
<td>Convert to Lower Case Alphabetic</td>
<td>25</td>
</tr>
<tr>
<td>Convert to Upper Case Alphabetic</td>
<td>26</td>
</tr>
<tr>
<td>Decimal Digit</td>
<td>20</td>
</tr>
<tr>
<td>Graphical</td>
<td>20</td>
</tr>
<tr>
<td>Hexadecimal Digit</td>
<td>24</td>
</tr>
<tr>
<td>Lower Case Alphabetic</td>
<td>21</td>
</tr>
<tr>
<td>Printable</td>
<td>22</td>
</tr>
<tr>
<td>Punctuation</td>
<td>22</td>
</tr>
<tr>
<td>Upper Case Alphabetic</td>
<td>24</td>
</tr>
<tr>
<td>White-Space</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classifying Characters</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearing</td>
<td>53</td>
</tr>
<tr>
<td>Clearing Error Indicator</td>
<td>53</td>
</tr>
<tr>
<td>clock</td>
<td>142</td>
</tr>
<tr>
<td>clock_t</td>
<td>141, 142</td>
</tr>
<tr>
<td>CLOCKS_PER_SEC</td>
<td>141</td>
</tr>
<tr>
<td>close</td>
<td>196</td>
</tr>
<tr>
<td>close_input_file</td>
<td>201</td>
</tr>
<tr>
<td>COFF</td>
<td>12</td>
</tr>
</tbody>
</table>

| Common Definitions, See stddef.h | 8 |
| Compare Strings | 126 |
| Comparison Function | 100, 110 |
| Comparison Functions | |

<table>
<thead>
<tr>
<th>Compiler Options</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-fno-short-double</td>
<td>48</td>
</tr>
<tr>
<td>-msmart-io</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concatenation Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>strcat</td>
<td>124</td>
</tr>
<tr>
<td>strncat</td>
<td>130</td>
</tr>
</tbody>
</table>

| Control Character | 19 |
| Defined | 19 |
| Test for | 19 |

| Control Transfers | 36 |

| Conversion | 77, 83, 87 |

<table>
<thead>
<tr>
<th>Convert</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Character to Multibyte Character</td>
<td>117</td>
</tr>
<tr>
<td>Multibyte Character to Wide Character</td>
<td>109</td>
</tr>
<tr>
<td>Multibyte String to Wide Character String</td>
<td>109</td>
</tr>
<tr>
<td>String to Double Floating-Point</td>
<td>98, 113</td>
</tr>
<tr>
<td>String to Integer</td>
<td>99</td>
</tr>
<tr>
<td>String to Long Integer</td>
<td>99, 115</td>
</tr>
<tr>
<td>String to Unsigned Long Integer</td>
<td>116</td>
</tr>
<tr>
<td>To Lower Case Alphabetic Character</td>
<td>25</td>
</tr>
<tr>
<td>To Upper Case Alphabetic Character</td>
<td>26</td>
</tr>
<tr>
<td>Wide Character String to Multibyte String</td>
<td>117</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Copying Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>memcpy</td>
<td>121</td>
</tr>
<tr>
<td>memmove</td>
<td>122</td>
</tr>
<tr>
<td>memset</td>
<td>123</td>
</tr>
<tr>
<td>strcpy</td>
<td>127</td>
</tr>
<tr>
<td>strncpy</td>
<td>133</td>
</tr>
<tr>
<td>cos</td>
<td>159</td>
</tr>
<tr>
<td>cosf</td>
<td>160</td>
</tr>
<tr>
<td>cosh</td>
<td>161</td>
</tr>
<tr>
<td>coshf</td>
<td>162</td>
</tr>
<tr>
<td>cosine</td>
<td></td>
</tr>
<tr>
<td>Double Floating-Point</td>
<td>159</td>
</tr>
<tr>
<td>Single Floating-Point</td>
<td>160</td>
</tr>
<tr>
<td>crt0, crt1</td>
<td>12</td>
</tr>
<tr>
<td>ctime</td>
<td>143</td>
</tr>
<tr>
<td>ctype.h</td>
<td>18</td>
</tr>
<tr>
<td>isalnum</td>
<td>18</td>
</tr>
<tr>
<td>iscntrl</td>
<td>19</td>
</tr>
<tr>
<td>isdigit</td>
<td>20</td>
</tr>
<tr>
<td>isgraph</td>
<td>20</td>
</tr>
<tr>
<td>islalla</td>
<td>18</td>
</tr>
<tr>
<td>islower</td>
<td>21</td>
</tr>
<tr>
<td>isprint</td>
<td>22</td>
</tr>
<tr>
<td>ispunct</td>
<td>22</td>
</tr>
<tr>
<td>isspace</td>
<td>23</td>
</tr>
<tr>
<td>ispos</td>
<td>24</td>
</tr>
<tr>
<td>isxdigit</td>
<td>24</td>
</tr>
<tr>
<td>islower</td>
<td>26</td>
</tr>
<tr>
<td>isupper</td>
<td>26</td>
</tr>
<tr>
<td>tolower</td>
<td>25</td>
</tr>
<tr>
<td>toupper</td>
<td>26</td>
</tr>
</tbody>
</table>

| Current Argument | 43 |
| Customer Notification Service | 8 |
| Customer Support | 9 |
| Customized Function | 105 |
16-Bit Language Tools Libraries

D
Dash (-) ................................................................. 84
Date and Time .................................................. 146
Date and Time Functions, See time.h .......................... 141, 142, 146
Day of the Month .................................................. 141, 142, 146
Day of the Week .................................................. 141, 142, 146
Day of the Year .................................................. 141, 147
Daylight Savings Time ........................................ 141, 144, 145
DBL_DIG .................................................................. 28
DBL_EPSILON ........................................................ 28
DBL_MANT_DIG ..................................................... 28
DBL_MAX ........................................................... 28
DBL_MAX_10_EXP .................................................. 29
DBL_MAX_EXP ...................................................... 29
DBL_MIN ........................................................... 29
DBL_MIN_10_EXP .................................................. 29
DBL_MIN_EXP ...................................................... 30
Deallocate Memory ................................................ 105, 111
Debugging Logic Errors ............................................. 17
Decimal .........................................................78, 84, 115, 116
Decimal Digit
  Defined .................................................................. 20
  Number Of .......................................................28, 30, 31
  Test for ............................................................ 20
Decimal Point ........................................................ 77
Default Handler .................................................... 37
delay_ms ............................................................ 202
delay_us ............................................................. 202
delay32 ............................................................... 202
Diagnostics, See assert.h ......................................... 144
difftime ............................................................. 144
Digit, Decimal, See Decimal Digit
Digit, Hexadecimal, See Hexadecimal Digit
Direct Input/Output Functions
  fread .............................................................. 72
  fwrite ........................................................... 64
div ................................................................. 94, 102
div_t ............................................................... 94
Divide
  Integer .......................................................... 102
  Long Integer ................................................... 107
Divide by Zero .................................................... 38, 41, 102
Documentation
  Conventions .................................................... 6
  Layout .......................................................... 5
Domain Error .................................................... 27, 151, 152, 153, 155, 157, 159, 160, 167, 169, 176, 177, 178, 179, 184, 185, 186, 188, 189, 190
dot ................................................................. 77
Double Precision Floating-Point
  Machine Epsilon ............................................... 28
  Maximum Exponent (base 10) .......................... 29
  Maximum Exponent (base 2) .......................... 29
  Maximum Value ........................................... 28
  Minimum Exponent (base 10) ........................ 29
  Minimum Exponent (base 2) ......................... 30
  Minimum Value ........................................... 29
  Number of Binary Digits ................................ 28
  Number of Decimal Digits ................................ 28
double Type ..................................................... 48
Dream Function .................................................. 77

DWARF .................................................................. 12

E
EDOM .................................................................. 27
edom ............................................................... 151
ELF ................................................................. 12
Ellipses (…) ........................................................ 43, 84
Empty Binary File ................................................ 60
Empty Text File .................................................... 60
End Of File ....................................................... 50
  Indicator ......................................................... 47
  Seek ............................................................. 68
  Test For ......................................................... 55
Environment Function
getenv ............................................................ 105
Environment Variable ........................................ 200
EOF ................................................................. 50
ERANGE ........................................................... 27
erange .............................................................. 151
erase_eedata .................................................... 204
erase_eedata_all ............................................. 204
erase_flash ....................................................... 206, 207
errno ............................................................... 27, 151
errno.h .......................................................... 27, 151, 197
EDOM .............................................................. 27
ERANGE ........................................................... 27
erro................................................................. 27
Error Codes ....................................................... 27, 129
Error Conditions ................................................ 151
Error Handler .................................................... 102
Error Handling Functions
clearerr .......................................................... 53
feof ............................................................... 55
ferror ............................................................. 56
perror ............................................................ 76
Error Indicator .................................................... 47
Error Indicators
  Clearing ........................................................ 53, 82
  End Of File .................................................... 53, 59
  Error ............................................................ 53, 59
  Test For ......................................................... 56
Error Signal ........................................................ 37
Errors, See errno.h ............................................. 47
Errors, Testing For ............................................. 27
Exception Error ................................................ 102
exit ................................................................. 88, 94, 96, 104, 195
EXIT_FAILURE .................................................. 94
EXIT_SUCCESS .................................................. 94
exp ................................................................. 163
expf ............................................................... 164
Exponential and Logarithmic Functions
  exp ............................................................... 163
  expf ............................................................ 164
  frexp ............................................................ 171
  frexpf .......................................................... 172
  ldexp ............................................................ 173
  ldexpf .......................................................... 174
  log .............................................................. 176
  log10 ........................................................... 177
  log10f .......................................................... 178
  logf ............................................................. 179
fabs .......................................................... 165
fabsf .............................................................. 165
fclose .......................................................... 54, 196
feof ............................................................... 53, 55
ferror .............................................................. 53, 56
fflush .............................................................. 57, 199
fgetc ............................................................... 57, 198
fgetpos .......................................................... 58, 197
fgets .............................................................. 59, 198
Field Width .................................................... 77
FILE .................................................................. 17, 47, 49
File Access Functions
close .......................................................... 54
fopen ............................................................. 60
freopen ........................................................ 66
setbuf ........................................................... 85
setvbuf .......................................................... 86
File Access Modes ........................................... 47, 60
File Buffering
Fully Buffered .................................................. 47, 50
Line Buffered .................................................. 47, 50
Unbuffered ...................................................... 47, 50
File Operations
Remove .......................................................... 81
Rename ........................................................... 81
File Positioning Functions
fgetpos .......................................................... 58
fseek .............................................................. 68
fsetpos .......................................................... 69
ftell ............................................................... 71
rewind ............................................................ 82
FILENAME_MAX .................................................. 50
File-Position Indicator .................................. 47, 49, 57, 58, 63, 64, 69, 72
Files, Maximum Number Open ....................... 50
Fixed Point Math Library .................................. 213
flags .............................................................. 77
float.h ............................................................ 28
DBL_DIG .......................................................... 28
DBL_EPSILON .................................................. 28
DBL_MANT_DIG ................................................. 28
DBL_MAX .......................................................... 28
DBL_MAX_10_EXP ............................................. 29
DBL_MAX_EXP .................................................. 29
DBL_MIN .......................................................... 29
DBL_MIN_10_EXP ............................................. 29
DBL_MIN_EXP .................................................. 30
FLT_DIG .......................................................... 30
FLT_EPSILON .................................................. 30
FLT_INPUT .......................................................... 30
FLT_MAX .......................................................... 30
FLT_MAX_10_EXP ............................................. 30
FLT_MAX_EXP .................................................. 30
FLT_MIN .......................................................... 31
FLT_MIN_10_EXP ............................................. 31
FLT_MIN_EXP .................................................. 31
FLT_ROUNDS .................................................... 31
Floating-Point
Limits ............................................................ 28
No Conversion ................................................ 47
Types, Properties Of ....................................... 28
Floating-Point, See float.h
Floating-Point Error Signal ................................ 38
floor .............................................................. 166
Double Floating-Point ...................................... 166
Single Floating-Point ........................................ 166
floorf ............................................................. 166
FLT_DIG .......................................................... 30
FLT_EPSILON .................................................. 30
FLT_MANT_DIG ................................................. 30
FLT_MAX .......................................................... 30
FLT_MAX_10_EXP ............................................. 30
FLT_MAX_EXP .................................................. 30
FLT_MIN .......................................................... 31
FLT_MIN_10_EXP ............................................. 31
FLT_MIN_EXP .................................................. 31
FLT_RADIX ..................................................... 31
FLT_RADIX Digit
Number Of ..................................................... 28, 30, 32
FILE .................................................................. 17, 47, 49
FILE .................................................................. 30
fmod .............................................................. 167
fmodf ............................................................. 169
-fno-short-double ........................................ 28, 29, 30, 48
fopen ............................................................. 47, 60, 86, 197
FOPEN_MAX ..................................................... 50
Form Feed ...................................................... 23
Format Specifiers ............................................. 77, 83
Formatted I/O Routines ...................................... 47
Formatted Input/Output Functions
fprintf .......................................................... 62
fscanf ........................................................... 66
printf ............................................................ 77
scanf ............................................................. 83
sprintf .......................................................... 87
sscanf ............................................................ 87
vfprintf .......................................................... 91
vprintf .......................................................... 92
vscanf ........................................................... 93
Formatted Text
Printing ....................................................... 87
Scanning ....................................................... 87
fpos_t ............................................................. 49
fprintf .......................................................... 47, 62
16-Bit Language Tools Libraries

fputc ................................................................. 63
fputs ................................................................. 63
fraction and exponent function
  Double Floating-Point ......................................... 171
  Single Floating-Point ......................................... 172
Fraction Digits ........................................... 77
fread ............................................................ 64, 198
free .................................................................. 105
Free Memory .................................................. 105
freopen ............................................................ 47, 66, 197
frexp .................................................................. 171
frexf .................................................................. 172
fscanf ............................................................. 47, 66
fseek ............................................................ 68, 89, 197
fsetpos ........................................................... 69, 89, 197
ftell ............................................................... 71, 197
Full Buffering ............................................... 85, 86
Fully Buffered .................................................. 47, 50
fwrite .............................................................. 72

G
getc .................................................................. 74
getchar ........................................................... 75
getenv ............................................................ 105, 200
gets ............................................................... 75, 198
GMT .............................................................. 144
gmtime ............................................................ 144, 145
Graphical Character
  Defined .......................................................... 20
  Test for .......................................................... 20
Greenwich Mean Time ........................................ 144

H
h modifier .......................................................... 78, 83
Handler
  Default .......................................................... 37
  Error ............................................................. 102
  Interrupt ....................................................... 41
  Nested .......................................................... 36
  Signal ............................................................ 37, 42
  Signal Type .................................................... 37
Handling
  Interrupt Signal ................................................ 42
Header Files
  assert.h .......................................................... 17
  ctype.h .......................................................... 18
  errno.h ......................................................... 27, 151, 197
  float.h .......................................................... 28
  libq.h ........................................................... 219
  limits.h .......................................................... 33
  locale.h .......................................................... 35
  math.h ........................................................... 151
  setjmp.h .......................................................... 36
  signal.h .......................................................... 37
  stdarg.h .......................................................... 43
  stddef.h .......................................................... 45
  stdio.h .......................................................... 47, 200
  stdlib.h .......................................................... 94, 200
  string.h .......................................................... 118
  time.h ............................................................ 141, 201
Heap ................................................................. 196
Hexadecimal .................................................. 78, 84, 115, 116
Hexadecimal Conversion ........................................ 77
Hexadecimal Digit
  Defined .......................................................... 24
  Test for .......................................................... 24
Horizontal Tab .................................................. 23
Hour .............................................................. 141, 142, 146
HUGE_VAL .......................................................... 151
Hyperbolic Cosine
  Double Floating-Point ........................................ 161
  Single Floating-Point ........................................ 162
Hyperbolic Functions
  cosh ............................................................. 161
  coshf ................................................................ 162
  sinh ............................................................. 186
  sinhf ................................................................ 187
  tanh ............................................................. 191
  tanhf ................................................................ 192
Hyperbolic Sine
  Double Floating-Point ........................................ 186
  Single Floating-Point ........................................ 187
Hyperbolic Tangent
  Double Floating-Point ........................................ 191
  hyperbolic tangent ........................................... 192

I
Ignore Signal ........................................................ 37
Illegal Instruction Signal ....................................... 39
Implementation-Defined Limits, See limits.h
  Indicator
    End Of File .................................................. 47, 50
    Error ............................................................ 47, 56
    File Position ................................................ 47, 57, 58, 63, 64, 69, 72
  Infinity ........................................................... 151
  init_prog_address ............................................ 211
  Input and Output, See stdio.h
  Input Formats ................................................ 201
  Instruction Cycles ............................................ 144, 145, 148
int
  Maximum Value ................................................ 33
  Minimum Value ................................................ 33
  INT_MAX .......................................................... 33
  INT_MIN .......................................................... 33
  Integer Limits .................................................. 33
  Internal Error Message ...................................... 129
  Internet Address, Microchip ................................ 8
  Interrupt Handler ............................................. 41
  Interrupt Signal ............................................... 39
  Interrupt Signal Handling ................................... 42
  Interruption Message ......................................... 39
  Invalid Executable Code Message ......................... 39
  Invalid Storage Request Message ......................... 40
  Inverse Cosine, See arcsine
  Inverse Sine, See arcsine
  Inverse Tangent, See arctangent
  IOFBF ............................................................ 50, 85, 86
  IOLBF ............................................................ 50, 86
  IONBF ............................................................ 50, 85, 86
  isalnum .......................................................... 18
  iscntrl ............................................................ 19
index
16-Bit Language Tools Libraries

Minimum Value ....................................................... 34
long long unsigned int
Maximum Value ..................................................... 35
long unsigned int
Maximum Value ..................................................... 35
LONG_MAX ......................................................... 34
LONG_MIN ......................................................... 34
longjmp ................................................................. 36
Lower Case Alphabetic Character
Convert To ......................................................... 25
Defined ............................................................... 21
Test for ............................................................... 21
lseek ........................................................................ 197

M
Machine Epsilon
Double Floating-Point ........................................... 28
Long Double Floating-Point .................................. 32
Single Floating-Point ........................................... 30
Magnitude ......................................................... 151, 163, 164, 167, 169, 186, 187
malloc ............................................................... 105, 108, 195, 198
Mapping Characters ............................................. 18
Math Exception Error .......................................... 102
Math Library ......................................................... 149
math.h ................................................................. 124
acos................................................................. 151
acosf ................................................................. 152
asin ................................................................. 153
asinf ................................................................. 153
atan ................................................................. 154
atan2 ................................................................. 155
atan2f ............................................................... 157
atanf ................................................................. 155
cos ................................................................. 159
cosf ................................................................. 160
cosh ................................................................. 161
coshf ................................................................. 161
exp ................................................................. 163
expf ................................................................. 164
fabs ................................................................. 165
fabsf ................................................................. 165
floor ................................................................. 166
floorf ................................................................. 166
fmod ................................................................. 167
fmodf ................................................................. 169
frexp ................................................................. 171
frexpf ............................................................... 172
HUGE_VAL .......................................................... 151
ldexp ................................................................. 173
ldexpf ............................................................... 174
log ................................................................. 176
log10 ............................................................... 177
log10f .............................................................. 178
logf ................................................................. 179
modf ................................................................. 180
modff ............................................................... 181
pow ................................................................. 182
powf ................................................................. 183
sin ................................................................. 184
sinf ................................................................. 185
sinh ................................................................. 186
sinhf ............................................................... 187
sqrt ................................................................. 188
sqrtf ............................................................... 189
tan ................................................................. 190
tanh ................................................................. 191
tanhf ............................................................... 192
Mathematical Functions, See libq.h
Mathematical Functions, See math.h
MathError .......................................................... 41
Maximum
Multibyte Character ............................................. 95
Maximum Value
Double Floating-Point Exponent (base 10) ............. 29
Double Floating-Point Exponent (base 2) ............... 29
Long Double Floating-Point Exponent (base 10) ... 32
Multibyte Character ............................................. 34
rand ................................................................. 95
Single Floating-Point Exponent (base 10) ............. 30
Single Floating-Point Exponent (base 2) ............... 31
Type char .......................................................... 33
Type Double ....................................................... 28
Type int ........................................................... 33
Type Long Double ............................................... 32
Type long int ..................................................... 34
Type long long int ............................................. 35
Type long unsigned int ..................................... 35
Type short int ................................................... 34
Type signed char ............................................... 34
Type Single ....................................................... 30
Type unsigned char .......................................... 35
Type unsigned int ............................................. 35
Type unsigned short int ................................... 35
MB_CUR_MAX ..................................................... 95
MB_LEN_MAX ..................................................... 34
mblen ............................................................... 109
mbstowcs ......................................................... 109
mbtowc ........................................................... 109
memchr ............................................................ 118
memcmp ............................................................. 119
memcpy ............................................................. 121
memcpy_p2d16 .. ................................................. 209
memcpy_p2d24 .. ................................................. 210
memmove .......................................................... 122
Memory
Allocate ......................................................... 102, 108
Deallocate ....................................................... 105
Free ................................................................. 105
Reallocate ....................................................... 111
memset .............................................................. 123
Message
Arithmetic Error ................................................ 38
Interrupt .......................................................... 39
Invalid Executable Code ...................................... 39
Invalid Storage Request .................................... 40
Punctuation Character
  Defined ................................................. 22
  Test for ................................................ 22
Pushed Back ................................................. 89
putc ............................................................ 79
putchar ....................................................... 80
puts ........................................................... 80

Q
Q15 Functions .................................................. 219
Q16 Functions .................................................. 228
qsort ............................................................ 100, 110
Quick Sort ..................................................... 110

R
Radix .............................................................. 31
raise ............................................................ 37, 38, 39, 40, 41, 42
rand .............................................................. 111, 113
RAND_MAX ...................................................... 95, 111
Range ............................................................ 184
Range Error27, 115, 116, 161, 162, 163, 164, 173, 174,
.................................................. 182, 183, 186, 187
read .............................................................. 198
Reading, Recommended ................................... 7
realloc .......................................................... 105, 111
Reallocate Memory ........................................... 111
Rebuilding the libpic30 library ............................. 194
Registered Functions ........................................ 96, 104
Remainder
  Double Floating-Point ..................................... 167
  Single Floating-Point ..................................... 169
Remainder Functions
  fmod ............................................................ 167
  fmodf .......................................................... 167
remove .......................................................... 81, 200
rename .......................................................... 81, 200
Reset ........................................................... 95, 117
Reset File Pointer .......................................... 82
rewind .......................................................... 82, 89, 197
Rounding Mode .............................................. 31

S
sbbrk ........................................................... 196, 198
Scan Formats .................................................. 47
scanf ........................................................... 47, 83
SCHAR_MAX .................................................... 34
SCHAR_MIN .................................................... 34
Search Functions
  memchr ....................................................... 118
  strchr ......................................................... 125
  strcspn ...................................................... 128
  strpbrk ...................................................... 135
  strchr ......................................................... 136
  strspn ....................................................... 137
  strstr ......................................................... 138
  strtok ......................................................... 139
Second ........................................................ 141, 142, 144, 147
Seed ........................................................... 111, 113
Seek
  From Beginning of File .................................. 68
  From Current Position ................................... 68
  From End Of File ......................................... 68
SEEK_CUR ....................................................... 51, 68
SEEK_END ....................................................... 52, 68
SEEK_SET ....................................................... 52, 68
setbuf .......................................................... 47, 50, 85
setjmp .......................................................... 36
setjmp.h ........................................................ 36
  jmp_buf ...................................................... 36
  longjmp ...................................................... 36
  setjmp ....................................................... 36
setlocale ...................................................... 35
setvbuf ........................................................ 47, 50, 86
short int
  Maximum Value ........................................... 34
  Minimum Value ........................................... 34
SHRT_MAX ..................................................... 34
SHRT_MIN ..................................................... 34
sig_atomic_t .................................................. 37
SIG_DFL ....................................................... 37
SIG_ERR ....................................................... 37
SIG_IGN ....................................................... 37
SIGABRT ...................................................... 38
SIGFPE ....................................................... 38
SIGILL ....................................................... 39
SIGINT ....................................................... 39
Signal
  Abnormal Termination .................................... 38
  Error ......................................................... 37
  Floating-Point Error ..................................... 38
  Ignore ....................................................... 37
  Illegal Instruction ....................................... 39
  Interrupt .................................................. 39
  Reporting .................................................. 41
  Termination Request .................................... 40
signal .......................................................... 38, 39, 40, 42
Signal Handler ................................................ 37, 42
Signal Handler Type ........................................ 37
Signal Handling, See signal.h ............................. 37
signal.h ........................................................ 37
  raise .......................................................... 41
  sig_atomic_t ............................................... 37
  SIG_DFL ..................................................... 37
  SIG_ERR ..................................................... 37
  SIG_IGN ..................................................... 37
  SIGABRT .................................................... 38
  SIGFPE ..................................................... 38
  SIGILL ...................................................... 39
  SIGINT ...................................................... 39
  signal ....................................................... 42
  SIGSEGV ................................................... 40
  SIGTERM ................................................... 40
  signed char
    Maximum Value ........................................... 34
    Minimum Value ........................................... 34
SIGSEGV ..................................................... 40
SIGTERM ..................................................... 40
sim30 simulator ............................................. 194
sin ............................................................ 184
sine

SEEK_CUR
SEEK_END
SEEK_SET
setbuf
setjmp
setjmp.h
jmp_buf
longjmp
setjmp
setlocale
setvbuf
short int
SHRT_MAX
SHRT_MIN
sig_atomic_t
SIG_DFL
SIG_ERR
SIG_IGN
SIGABRT
SIGFPE
SIGILL
SIGINT
Signal
Abnormal Termination
Error
Floating-Point Error
Ignore
Illegal Instruction
Interrupt
Reporting
Termination Request
signal
Signal Handler
Signal Handler Type
Signal Handling, See signal.h
signal.h
raise
sig_atomic_t
SIG_DFL
SIG_ERR
SIG_IGN
SIGABRT
SIGFPE
SIGILL
SIGINT
signal
SIGSEGV
SIGTERM
signed char
Maximum Value
Minimum Value
SIGSEGV
SIGTERM
sim30 simulator
sin
sine
16-Bit Language Tools Libraries

stdlib.h ................................................................. 94, 200
abort ................................................................. 95
abs ................................................................. 96
atexit ................................................................. 96
atof ................................................................. 98
atoi ................................................................. 99
atol ................................................................. 99
bsearch .............................................................. 100
calloc .............................................................. 102
div ................................................................. 102
div_l ................................................................. 94
exit ................................................................. 104
EXIT_FAILURE ..................................................... 94
EXIT_SUCCESS .................................................... 94
free ................................................................. 105
getenv .............................................................. 105
labs ................................................................. 106
ldiv ................................................................. 107
ldiv_t ............................................................... 94
malloc ............................................................ 108
MB_CUR_MAX ....................................................... 95
mblen .............................................................. 109
mbstowcs .......................................................... 109
mbtowc ............................................................ 109
NULL ............................................................... 95
csort ............................................................... 110
rand ............................................................... 111
RAND_MAX ........................................................ 95
realloc ............................................................ 111
size_t ............................................................. 94
srand ............................................................. 113
strtol ............................................................. 113
strtol ............................................................. 115
strtol ............................................................. 116
system ............................................................ 117
wchar_t ........................................................... 94
wctomb ........................................................... 117
wxstombs ........................................................ 117
stdout ............................................................ 47, 51, 52, 77, 80
strcat .............................................................. 124
strchr ............................................................. 125
strcat .............................................................. 126
strcoll ............................................................ 127
strncpy ............................................................ 127
strspn ............................................................. 128
strerror .......................................................... 129
strstr ............................................................. 138
strtok ............................................................. 139
strxfrm .......................................................... 140
temporary ...................................................... 129
strcat .............................................................. 130
strncpy ............................................................ 132
strncpy ............................................................ 133
strncpy ............................................................ 210
strncpy ............................................................ 210
strchr ............................................................. 136
strchr ............................................................. 136
strchr ............................................................. 137
strchr ............................................................. 138
strchr ............................................................. 139
strchr ............................................................. 140
strchr ............................................................. 98, 113
strchr ............................................................. 139
strchr ............................................................. 139
strchr ............................................................. 99, 115
strchr ............................................................. 116
strchr ............................................................. 35
strchr ............................................................. 141
strchr ............................................................. 140
strchr ............................................................. 139
strchr ............................................................. 45
strings ........................................................... 193
Successful Termination ........................................... 94
support library ................................................... 193
system ............................................................ 117, 200
T
Tab ................................................................. 23
tan ................................................................. 190
tan ................................................................. 190
tan ................................................................. 190
tan ................................................................. 190
tan ................................................................. 190
tan ................................................................. 190
tan ................................................................. 190
tan ................................................................. 190
Temporary
File ............................................................. 88, 104
Filename ........................................................ 51, 89
Pointer .......................................................... 111

String Functions, See string.h
Index

Termination
  Request Message ........................................ 40
  Request Signal ........................................... 40
  Successful .................................................. 94
  Unsuccessful ................................................ 94

Text Mode ................................................... 60
Text Streams ................................................ 47
Ticks .............................................................. 141, 142, 144
time .............................................................. 148, 201
Time Difference ............................................ 144
Time Structure .............................................. 141, 146
Time Zone ...................................................... 147
time_t ............................................................. 141, 146, 148
time.h ............................................................. 141, 201
  asctime ......................................................... 142
clock ............................................................. 142
clock_t ........................................................... 141
  CLOCKs_PER_SEC ........................................... 141
cftime ............................................................ 143
difftime ........................................................ 144
gmtime ........................................................... 144
locatime ....................................................... 145
mtime ............................................................. 146
  NULL ............................................................ 142
  size_t ........................................................... 141
strftime ......................................................... 146
struct tm ....................................................... 141
time ............................................................. 148
time_t ........................................................... 141
TMP_MAX .......................................................... 52
tmpfile .......................................................... 88
tmpnam .......................................................... 89
Tokens .......................................................... 139
tolower ........................................................... 25
toupper ........................................................... 26
Transferring Control ....................................... 36
Transform String ............................................ 140
Trigonometric Functions
  acos ............................................................. 151
  acosh ........................................................... 152
  asin ............................................................. 153
  asinh .......................................................... 153
  atan ............................................................ 154
  atan2 ........................................................... 155
  atan2f .......................................................... 157
  atanf ........................................................... 155
  cos .............................................................. 159
  cosh ............................................................ 160
  sin .............................................................. 184
  sinh ............................................................ 185
  tan .............................................................. 190
  tanf ............................................................. 190
type .............................................................. 78, 84

U
  UCHAR_MAX .................................................. 35
  UINT_MAX .................................................... 35
  ULLONG_MAX .................................................. 35
  ULONG_MAX ................................................... 35
  Underflow Errors 27, 151, 163, 164, 173, 174, 182, 183
  ungetc .......................................................... 89

Unsuccessful Termination .................................... 94

Utility Functions, See stdlib.h
  va_arg .......................................................... 43, 45, 91, 92, 93
  va_end .......................................................... 45, 91, 92, 93
  va_list .......................................................... 43
  va_start .......................................................... 45, 91, 92, 93

Variable Argument Lists, See stdarg.h
  Variable Length Argument List ...................... 43, 45, 91, 92, 93
  VERBOSE_DEBUGGING ..................................... 17
  Vertical Tab ................................................... 23
  vfprintf ........................................................... 47, 91
  vsprintf ........................................................... 47, 92

Wait
  wait_eedata .................................................. 205
  wchar_t ......................................................... 45, 94
  wcstombs ...................................................... 117
  wctomb ......................................................... 117
  Week ............................................................. 147
  White Space .................................................... 83, 98, 99, 113
  White-Space Character
    Defined ...................................................... 23
    Test for ..................................................... 23
    Wide .......................................................... 94
    Wide Character ............................................ 109, 117
    Wide Character String .................................... 109, 117
    Wide Character Value ..................................... 45
    Width ........................................................... 77
    width .......................................................... 77, 83
    write .......................................................... 199
    write_eedata .................................................. 205
    write_eedata_row .......................................... 205
    write_eedata_word ........................................ 205
    write_flash .................................................... 208
    write_flash_word .......................................... 208
    write_flash_word16 ....................................... 207
    write_flash_word24 ....................................... 207
    write_flash_word24 ....................................... 207

Year ............................................................ 141, 142, 147

Zero ............................................................. 151
Zero, divide by .............................................. 38, 41, 102
# Worldwide Sales and Service

<table>
<thead>
<tr>
<th>AMERICAS</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>EUROPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455</td>
<td>Australia - Sydney Tel: 61-2-9868-6733 Fax: 61-2-9868-6755</td>
<td>India - Pune Tel: 91-20-2566-1512 Fax: 91-20-2566-1513</td>
<td>Japan - Yokohama Tel: 81-45-471-6166 Fax: 81-45-471-6122</td>
<td>Denmark - Copenhagen Tel: 45-4450-2826 Fax: 45-4485-2829</td>
</tr>
<tr>
<td>Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088</td>
<td>China - Beijing Tel: 86-10-8528-2100 Fax: 86-10-8528-2104</td>
<td>Korea - Daegu Tel: 82-53-744-4301 Fax: 82-53-744-4302</td>
<td>Korea - Seoul Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934</td>
<td>France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79</td>
</tr>
<tr>
<td>Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075</td>
<td>China - Chengdu Tel: 86-28-8665-5511 Fax: 86-28-8665-7889</td>
<td>Malaysia - Kuala Lumpur Tel: 60-3-6201-9857 Fax: 60-3-6201-9859</td>
<td>Malaysia - Penang Tel: 60-4-227-8870 Fax: 60-4-227-4068</td>
<td>Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44</td>
</tr>
<tr>
<td>Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924</td>
<td>China - Hong Kong SAR Tel: 852-2401-1200 Fax: 852-2401-3431</td>
<td>Thailand - Bangkok Tel: 66-2-694-1351 Fax: 66-2-694-1350</td>
<td>Malaysia - Kuala Lumpur Tel: 60-3-6201-9857 Fax: 60-3-6201-9859</td>
<td>Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340</td>
</tr>
<tr>
<td>Detroit Farmington Hills, MI Tel: 248-538-2250 Fax: 248-538-2260</td>
<td>China - Nanjing Tel: 86-25-8473-2460 Fax: 86-25-8473-2470</td>
<td>Taiwan - Hsin Chu Tel: 886-3-6578-300 Fax: 886-3-6578-370</td>
<td>Taiwan - Kaohsiung Tel: 886-7-213-7830 Fax: 886-7-330-9305</td>
<td>Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91</td>
</tr>
<tr>
<td>Kokomo Kokomo, IN Tel: 765-864-8360 Fax: 765-864-8387</td>
<td>China - Qingdao Tel: 86-532-8502-7355 Fax: 86-532-8502-7205</td>
<td>Taiwan - Kaohsiung Tel: 886-7-213-7830 Fax: 886-7-330-9305</td>
<td>Taiwan - Taipei Tel: 886-2-2500-6610 Fax: 886-2-2508-0102</td>
<td>UK - Wokingham Tel: 44-118-921-5869 Fax: 44-118-921-5820</td>
</tr>
<tr>
<td>Toronto Mississauga, Ontario, Canada Tel: 905-673-0699 Fax: 905-673-6509</td>
<td>China - Xian Tel: 86-29-8833-7252 Fax: 86-29-8833-7256</td>
<td>China - Zhuhai Tel: 86-756-3210040 Fax: 86-756-3210049</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2010 Microchip Technology Inc.