Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

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.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

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® MCUs and dsPIC® DSCs, KEELOQ® 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

NOTICE TO CUSTOMERS

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

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

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

INTRODUCTION

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

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

DOCUMENT LAYOUT

This document describes how to use GNU language tools to write code for 16-bit applications. The document 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.
- **Appendix A:** ASCII Character Set

CONVENTIONS USED IN THIS GUIDE

The following conventions may appear in this documentation:

DOCUMENTATION CONVENTIONS

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

RECOMMENDED READING

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

Readme Files

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

16-Bit Language Tools Getting Started (DS70094)

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

MPLAB® 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*. American National Standards Institute (ANSI), 11 West 42nd. Street, New York, New York, 10036.

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

C Reference Manuals

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

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

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

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

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

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™, MPLAB ICE 2000 and MPLAB ICE 4000 in-circuit emulators
- **In-Circuit Debuggers** – The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD 2 in-circuit debugger and PICkit™ 2 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 MPLAB PM3 and PRO MATE® II device programmers and the PICSTART® Plus and PICkit 1 and 2 development programmers.

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

NOTES:

Chapter 1. Library Overview

1.1 INTRODUCTION

A library is a collection of functions grouped for reference and ease of linking. See the "*MPLAB® 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 provided. 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 `libp`*`Device`*`-`*`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 `libc-omf.a` (written by Dinkumware, an industry leader) and `libm-omf.a` (math functions, written by Microchip).

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

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

1.7 FIXED POINT MATH FUNCTIONS

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

1.8 COMPILER BUILT-IN FUNCTIONS

The MPLAB 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 "*MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs Users' Guide*" (DS51284).

NOTES:

Chapter 2. Standard C Libraries

2.1 INTRODUCTION

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

2.1.1 Assembly Code Applications

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

2.1.2 C Code Applications

The MPLAB 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
- <limits.h> implementation-defined limits
- <locale.h> localization
- <setjmp.h> non-local jumps
- <signal.h> signal handling
- <stdarg.h> variable argument lists
- <stddef.h> common definitions
- <stdio.h> input and output
- <stdlib.h> utility functions
- <string.h> string functions
- <time.h> date and time functions

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: `void assert(int expression);`

Argument: *expression* The expression to test.

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

Example:

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

int main(void)
{
  int a;

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

Output:

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

with `_VERBOSE_DEBUGGING` defined:

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

2.4 <CTYPE.H> CHARACTER HANDLING

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

isalnum

Description: Test for an alphanumeric character.

Include: `<ctype.h>`

Prototype: `int isalnum(int c);`

Argument: *c* The character to test.

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

Remarks: Alphanumeric characters are included within the ranges A-Z, a-z or 0-9.

Example:

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

int main(void)
{
  int ch;

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

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

Output:

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

isalpha

Description: Test for an alphabetic character.

Include: <ctype.h>

Prototype: `int isalpha(int c);`

Argument: *c* The character to test.

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

Remarks: Alphabetic characters are included within the ranges A-Z or a-z.

isalpha (Continued)

Example:

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

int main(void)
{
  int ch;

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

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

Output:

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

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:

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

int main(void)
{
  int ch;

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

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

Output:

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

isgraph

Description: Test for a graphical character.

Include: `<ctype.h>`

Prototype: `int isgraph (int c);`

Argument: *c* character to test

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

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

Example:

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

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

isgraph (Continued)

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

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

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

Output:

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

islower

Description: Test for a 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:

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

ispunct (Continued)

Example:

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

int main(void)
{
  int ch;

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

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

Output:

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

isspace

Description: Test for a white-space character.

Include: <ctype.h>

Prototype: `int isspace (int c);`

Argument: *c* character to test

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

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

Example:

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

int main(void)
{

  int ch;

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

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

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:

```
#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:

```
#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:

```
#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));
}
```