



32-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in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates.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, FlashFlex, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC³² logo, rPIC, SST, SST Logo, SuperFlash 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, MTP, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

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


Analog-for-the-Digital Age, Application Maestro, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscent Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rLAB, Select Mode, SQI, Serial Quad I/O, Total Endurance, TSHARC, UniWinDriver, WiperLock, ZENA and Z-Scale 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.

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

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

© 2007-2012, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: ISBN: 978-1-62076-498-5

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

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC[®] 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	5
Chapter 1. Library Overview	
1.1 Introduction	11
1.2 Start-up Code	11
1.3 32-Bit Peripheral Libraries	11
1.4 Standard C Libraries (with Math Functions)	11
Chapter 2. Standard C Libraries with Math Functions	
2.1 Introduction	13
2.2 Using the Standard C Libraries	14
2.3 <assert.h> Diagnostics	14
2.4 <ctype.h> Character Handling	15
2.5 <errno.h> Errors	19
2.6 <float.h> Floating-Point Characteristics	21
2.7 <limits.h> Implementation-Defined Limits	25
2.8 <locale.h> Localization	27
2.9 <setjmp.h> Non-Local Jumps	28
2.10 <signal.h> Signal Handling	29
2.11 <stdarg.h> Variable Argument Lists	32
2.12 <stddef.h> Common Definitions	33
2.13 <stdio.h> Input and Output	34
2.14 <stdlib.h> Utility Functions	56
2.15 <string.h> String Functions	68
2.16 <time.h> Date and Time Functions	76
2.17 <math.h> Mathematical Functions	82
2.18 <unistd.h> Miscellaneous Functions	96
Chapter 3. PIC32 DSP Library	
3.1 Introduction	99
3.2 Vector Math Functions	102
3.3 Filtering Functions	111
3.4 Frequency Domain Transform Functions	116
3.5 Video Processing Functions	120
Chapter 4. PIC32 Debug-Support Library	
4.1 Overview	125
4.2 Configuring Debug Input/Output for the target and tool	125
4.3 <sys/appio.h> PIC32 Debugging Support	126

32-Bit Language Tools Libraries

Appendix A. ASCII Character Set.....	129
Appendix B. Types, Constants, Functions and Macros	131
Appendix C. 16-Bit DSP Wrapper Functions	135
C.1 Introduction	135
C.2 PIC32 DSP Wrapper Functions List	135
C.3 Differences Between Wrapper Functions and dsPIC® DSP Library	136
Index	137
Worldwide Sales and Service	150

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 “DSXXXXA”, where “XXXX” is the document number and “A” is the revision level of the document.

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

INTRODUCTION

This chapter contains general information that will be useful to know before using the 32-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 language tools to write code for 32-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 online Help files.
- **Chapter 2. Standard C Libraries with Math Functions** – lists the library functions and macros for standard C operation.
- **Chapter 3. PIC32 DSP Library** – lists the PIC32 DSP library functions, such as vector operations, filters and transforms.
- **Appendix A. ASCII Character Set** – ASCII Character Set.
- **Appendix B. Types, Constants, Functions and Macros** – an alphabetical list of types, constants, functions and macros.
- **Appendix C. 16-Bit DSP Wrapper Functions** – discusses the PIC32 DSP wrapper functions.

32-Bit Language Tools Libraries

CONVENTIONS USED IN THIS GUIDE

The following conventions may appear in this documentation:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic	Referenced books	<i>MPLAB[®] IDE User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic with right angle bracket	A menu path	<i><u>File</u>>Save</i>
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	<i>file.o</i> , where <i>file</i> can be any valid filename
Square brackets []	Optional arguments	mpasmwin [options] <i>file</i> [options]
Curly brackets and pipe character: { }	Choice of mutually exclusive arguments; an OR selection	errorlevel {0 1}
Ellipses...	Replaces repeated text	var_name [, var_name...]
	Represents code supplied by user	void main (void) { ... }

RECOMMENDED READING

This documentation describes how to use the 32-bit language tools 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.

Device-Specific Documentation

The Microchip web site 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

MPLAB® XC32 C/C++ Compiler User's Guide (DS51686)

Comprehensive guide that describes the operation and features of the Microchip 32-bit C/C++ compiler for PIC32MX devices.

PIC32MX Configuration Settings

Lists the Configuration Bit Settings for the Microchip PIC32MX devices supported by the MPLAB XC32 C/C++ compiler's `#pragma config` directive.

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*, Fifth 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*, Second Edition. LLH Technology
Publishing, Eagle Rock, Virginia 24085.

32-Bit Language Tools Libraries

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 and other language tools. These include all MPLAB[®] C compilers; all MPLAB assemblers (including MPASM[™] assembler); all MPLAB linkers (including MPLINK[™] object linker); and all MPLAB librarians (including MPLIB[™] object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB REAL ICE[™] and MPLAB ICE 2000 in-circuit emulators.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debuggers. These include 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 device programmer and the PICSTART[®] Plus, PICKit[™] 1 and PICKit[™] 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>.

REVISION HISTORY

Revision A (October 2007)

- Initial release of this document.

Revision B (October 2008)

- Added Appendix C. PIC32 DSP Library

Revision C (February 2009)

- Incorporated name changes from MPLAB 32 C Compiler to 32-bit C Compiler.
- Add MIPS and review updates.

Revision D (July 2009)

- Moved PIC32 DSP Library from Appendix C to Chapter 3.
- Added **Chapter 4. PIC32 Debug-Support Library**.

Revision E (October 2012)

- Removed values from the function tables in **Chapter 2. “Standard C Libraries with Math Functions”**.

32-Bit Language Tools Libraries

NOTES:

Chapter 1. Library Overview

1.1 INTRODUCTION

A library is a collection of functions grouped for reference and ease of linking.

1.1.1 C Code Applications

The 32-bit language tool libraries are included in the `pic32mx\lib` subdirectory of the MPLAB® XC32 C/C++ compiler for PIC32MX MCUs (formerly MPLAB C32) install directory, which is by default:

```
C:\Program Files\Microchip\xc32\<version>\pic32mx\lib
```

These libraries can be linked directly into an application with the 32-bit linker.

1.1.2 Chapter Organization

This chapter is organized as follows:

- Start-up Code
- 32-Bit Peripheral Libraries
- Standard C Libraries (with Math Functions)

1.2 START-UP CODE

In order to initialize variables in data memory, the linker creates a data initialization image. This image must be copied into RAM at start-up, before the application proper takes control. Initialization of the runtime environment is performed by start-up code in `crt0.o`. Details of the initialization process are described in Section 5.7 Start-up and Initialization in the *MPLAB® XC32 C/C++ Compiler User's Guide* (DS51686).

1.3 32-BIT PERIPHERAL LIBRARIES

The 32-bit software and hardware peripheral libraries provide functions and macros for setting up and controlling the 32-bit peripherals. These libraries are processor-specific and of the form `libmchp_peripheral_Device.a`, where *Device* is the 32-bit device number.

1.4 STANDARD C LIBRARIES (WITH MATH FUNCTIONS)

A complete set of ANSI-89 conforming libraries are provided. The standard C library files are `libc.a`, `libe.a`, and `libm.a`.

A typical C application will require all three libraries, these are linked in by default and do not need to be specified by the user.

32-Bit Language Tools Libraries

NOTES:

Chapter 2. Standard C Libraries with Math Functions

2.1 INTRODUCTION

Standard ANSI C library functions are contained in the libraries `libc.a` and `libgcc.a`. Multiple versions of these libraries exist, each compiled with different compilation options. They are intended to match closely with a subset of the build options used to compile your application. The compilation environment will select the library that is most appropriate for the selected build options.

The available libraries have been optimized for: speed, size, integer arithmetic only and MIPS16[®] mode.

2.1.1 C Code Applications

The 32-bit C compiler directory contains a library and include file subdirectory that is automatically searched by the tool chain. For a full install of the compiler, the default install directory is `c:\Program Files\Microchip\XC32`.

2.1.2 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
- `<math.h>` Mathematical Functions
- `<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
- `<unistd.h>` Miscellaneous Functions

32-Bit Language Tools Libraries

2.2 USING THE STANDARD C LIBRARIES

Building an application that 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 2.1.2 “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.

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 or by specifying them on the command line) such that the functions used by the application may be linked into the application.

Library linking is order dependent. A library must be required at the inclusion point for it to be used.

A typical C application will require three library files: `libc.a`, `libm.a`, and `libe.a`.

These libraries will be included automatically if linking is performed using the 32-bit compiler.

<p>Note: Some standard library functions require a heap. These include the standard I/O functions that open files and the memory allocation functions. Refer to Section 7.7 of the <i>MPLAB[®] XC32 C/C++ Compiler User's Guide</i> (DS51686).</p>
--

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 a message is printed to `stderr` and `abort()` is called which will terminate execution. The message includes the source file name (`__FILE__`), the source line number (`__LINE__`), the expression being evaluated and the message.

If the macro `NDEBUG` is defined `assert()` will do nothing.
`assert()` is defined as a C macro.

Standard C Libraries with Math Functions

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. Use of any one of these functions will import 257 bytes worth of data.

isalnum

Description: Tests 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.

isalpha

Description: Tests 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.

isascii

Description: Tests for an ascii character.
Include: `<ctype.h>`
Prototype: `int isascii(int c);`
Argument: `c` The character to test.
Return Value: Returns a non-zero integer value if the character is a member of the ascii character set, 0x00 to 0x7F inclusive.

isctrl

Description: Tests for a control character.
Include: `<ctype.h>`
Prototype: `int isctrl(int c);`
Argument: `c` character to test.
Return Value: Returns a non-zero integer value if the character is a control character, otherwise, returns zero.
Remarks: A character is considered to be a control character if its ASCII value is in the range 0x00 to 0x1F inclusive, or 0x7F.

32-Bit Language Tools Libraries

isdigit

Description: Tests 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.

isgraph

Description: Tests 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.

islower

Description: Tests for a lowercase alphabetic character.

Include: `<ctype.h>`

Prototype: `int islower (int c);`

Argument: `c` character to test

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

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

isprint

Description: Tests 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.

Standard C Libraries with Math Functions

ispunct

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

isspace

Description: Tests 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 (lf), newline (ln), carriage return (lr), horizontal tab (lt), or vertical tab (lv).

isupper

Description: Tests for an uppercase letter.

Include: `<ctype.h>`

Prototype: `int isupper (int c);`

Argument: `c` character to test

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

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

isxdigit

Description: Tests 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.

32-Bit Language Tools Libraries

tolower

Description: Converts a character to a lowercase alphabetical character.

Include: `<ctype.h>`

Prototype: `int tolower (int c);`

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

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

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

tolower

Description: Converts a character to a lowercase alphabetical character.

Include: `<ctype.h>`

Prototype: `int tolower (int c);`

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

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

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

toupper

Description: Converts a character to an uppercase alphabetical character.

Include: `<ctype.h>`

Prototype: `int toupper (int c);`

Argument: `c` The character to convert to uppercase.

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

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

Standard C Libraries with Math Functions

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 evaluate to any value greater than zero. To test if a library function encounters an error, the program should store the value zero in `errno` immediately before calling the library function. The value should be checked before another function call which may change the value. At program start-up, `errno` is zero. Library functions will never set `errno` to zero.

The following section identifies error values that are returned by the libraries. The header file defines errors that are not generated by the libraries.

2.5.1 Constants

EBADF

Description: Represents a bad file number.

Include: `<errno.h>`

Remarks: `EBADF` represents a bad file descriptor number. File descriptors are used by low-level IO library functions such as `write()`, which are not provided by default. For more information on library I/O functions, see **Section 2.13.2 “Customizing STDIO”**.

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 for which the function is defined.

EINVAL

Description: Represents an invalid argument.

Include: `<errno.h>`

Remarks: `EINVAL` represents an invalid argument to `fopen()`, which is not provided by default. For more information on library I/O functions, see **Section 2.13.2 “Customizing STDIO”**.

ENOMEM

Description: An error indicating that there is no more memory available.

Include: `<errno.h>`

Remarks: `ENOMEM` is returned from the low-level function when there is no more memory. Typically this in response to a heap allocation request.

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.

2.5.2 Functions and Macros

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.

Standard C Libraries with Math Functions

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 the number of significant figures, digits, size limits and what rounding mode is used. For values, refer to the header file.

DBL_DIG

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

Include: `<float.h>`

DBL_EPSILON

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

Include: `<float.h>`

DBL_MANT_DIG

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

Include: `<float.h>`

DBL_MAX

Description: Maximum finite double precision floating-point value

Include: `<float.h>`

DBL_MAX_10_EXP

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

Include: `<float.h>`

DBL_MAX_EXP

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

Include: `<float.h>`

DBL_MIN

Description: Minimum double precision floating-point value

Include: `<float.h>`

DBL_MIN_10_EXP

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

Include: `<float.h>`

32-Bit Language Tools Libraries

DBL_MIN_EXP

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

Include: `<float.h>`

FLT_DIG

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

Include: `<float.h>`

FLT_EPSILON

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

Include: `<float.h>`

FLT_MANT_DIG

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

Include: `<float.h>`

FLT_MAX

Description: Maximum finite single precision floating-point value

Include: `<float.h>`

FLT_MAX_10_EXP

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

Include: `<float.h>`

FLT_MAX_EXP

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

Include: `<float.h>`

FLT_MIN

Description: Minimum single precision floating-point value

Include: `<float.h>`

FLT_MIN_10_EXP

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

Include: `<float.h>`

Standard C Libraries with Math Functions

FLT_MIN_EXP

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

Include: <float.h>

FLT_RADIX

Description: Radix of exponent representation

Include: <float.h>

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

FLT_ROUNDS

Description: Represents the rounding mode for floating-point operations

Include: <float.h>

Remarks: Rounds to the nearest representable value

LDBL_DIG

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

Include: <float.h>

LDBL_EPSILON

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

Include: <float.h>

LDBL_MANT_DIG

Description: Number of base-FLT_RADIX digits in a long double precision floating-point significand

Include: <float.h>

LDBL_MAX

Description: Maximum finite long double precision floating-point value

Include: <float.h>

LDBL_MAX_10_EXP

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

Include: <float.h>

32-Bit Language Tools Libraries

LDBL_MAX_EXP

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

Include: `<float.h>`

LDBL_MIN

Description: Minimum long double precision floating-point value

Include: `<float.h>`

LDBL_MIN_10_EXP

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

Include: `<float.h>`

LDBL_MIN_EXP

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

Include: `<float.h>`

Standard C Libraries with Math Functions

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. For values, refer to the header file.

CHAR_BIT

Description: Number of bits to represent type `char`
Include: `<limits.h>`

CHAR_MAX

Description: Maximum value of a `char`
Include: `<limits.h>`

CHAR_MIN

Description: Minimum value of a `char`
Include: `<limits.h>`

INT_MAX

Description: Maximum value of an `int`
Include: `<limits.h>`

INT_MIN

Description: Minimum value of an `int`
Include: `<limits.h>`

LLONG_MAX

Description: Maximum value of a long long `int`
Include: `<limits.h>`

LLONG_MIN

Description: Minimum value of a long long `int`
Include: `<limits.h>`

LONG_MAX

Description: Maximum value of a long `int`
Include: `<limits.h>`

LONG_MIN

Description: Minimum value of a long `int`
Include: `<limits.h>`

32-Bit Language Tools Libraries

MB_LEN_MAX

Description: Maximum number of bytes in a multibyte character

Include: <limits.h>

SCHAR_MAX

Description: Maximum value of a signed char

Include: <limits.h>

SCHAR_MIN

Description: Minimum value of a signed char

Include: <limits.h>

SHRT_MAX

Description: Maximum value of a short int

Include: <limits.h>

SHRT_MIN

Description: Minimum value of a short int

Include: <limits.h>

UCHAR_MAX

Description: Maximum value of an unsigned char

Include: <limits.h>

UINT_MAX

Description: Maximum value of an unsigned int

Include: <limits.h>

ULLONG_MAX

Description: Maximum value of a long long unsigned int

Include: <limits.h>

ULONG_MAX

Description: Maximum value of a long unsigned int

Include: <limits.h>

USHRT_MAX

Description: Maximum value of an unsigned short int

Include: <limits.h>

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`
- `LC_ALL`
- `LC_COLLATE`
- `LC_CTYPE`
- `LC_MONETARY`
- `LC_NUMERIC`
- `LC_TIME`
- `localeconv`
- `setlocale`

32-Bit Language Tools Libraries

2.9 <SETJMP.H> NON-LOCAL JUMPS

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

2.9.1 Types

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[_JB_LEN];`

Remarks: `_JB_LEN` is defined as 24.

2.9.2 Functions

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 substituted for the result of the original `setjmp` call.

Remarks: The value parameter `val` should be non-zero, a `val` of zero will cause 1 to be substituted. 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.

setjmp

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

Include: `<setjmp.h>`

Prototype: `int 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.

Standard C Libraries with Math Functions

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. In a hosted environment, a signal may be raised in response to various events (control-C being pressed or resizing an X11 window). In the embedded world, signals are not tied to any specific hardware feature.

By default the 32-bit C compiler does not constitute a hosted environment, and as such there are no signal handling facilities provided. An OS or RTOS may provide these features. Cursory documentation is provided here for information purposes only.

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.

2.10.1 Types

`sig_atomic_t`

Description: A type used by a signal handler

Include: `<signal.h>`

Prototype: `typedef int sig_atomic_t;`

2.10.2 Constants

`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>`

32-Bit Language Tools Libraries

SIGABRT

Description: Name for the abnormal termination signal.

Include: <signal.h>

Prototype: #define SIGABRT

Remarks: SIGABRT represents an abnormal termination signal and is used in conjunction with `raise` or `signal`.

SIGFPE

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

Include: <signal.h>

Prototype: #define SIGFPE

Remarks: SIGFPE is used as an argument for `raise` and/or `signal`.

SIGILL

Description: Signals illegal instruction.

Include: <signal.h>

Prototype: #define SIGILL

Remarks: SIGILL is used as an argument for `raise` and/or `signal`.

SIGINT

Description: Interrupt signal.

Include: <signal.h>

Prototype: #define SIGINT

Remarks: SIGINT is used as an argument for `raise` and/or `signal`.

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`.

SIGTERM

Description: Signals a termination request

Include: <signal.h>

Prototype: #define SIGTERM

Remarks: SIGTERM is used as an argument for `raise` and/or `signal`.

2.10.3 Functions and Macros

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* *should* send the signal identified by *sig* to the executing program, however the default implementation always returns `SIG_ERR`.

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* OR `SIG_ERR`.

Remarks: *signal* should set the signal handler identified by *sig* to the *func* specified, however the default implementation always returns `SIG_ERR`.

32-Bit Language Tools Libraries

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_arg

Description: Gets the current argument.

Include: `<stdarg.h>`

Prototype: `#define va_arg(va_list ap, T)`

Argument: `ap` pointer to list of arguments
`T` type of argument to be retrieved

Return Value: Returns the current argument as type `T`

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

va_end

Description: Ends the use of `ap`.

Include: `<stdarg.h>`

Prototype: `#define va_end(va_list ap)`

Argument: `ap` pointer to list of arguments

Remarks: After a call to `va_end`, the argument list pointer `ap` is considered to be invalid. Further calls to `va_arg` should not be made until the next `va_start`.

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>`

va_start

Description: Sets the argument pointer `ap` to first optional argument in the variable-length argument list.

Include: `<stdarg.h>`

Prototype: `#define va_start(va_list ap, last_arg)`

Argument: `ap` pointer to list of arguments
`last_arg` last named argument before the optional (ellipsis) arguments

Standard C Libraries with Math Functions

2.12 <STDDEF.H> COMMON DEFINITIONS

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

2.12.1 Constants

NULL

Description: The value of a Null Pointer constant.

Include: `<stddef.h>`

2.12.2 Functions and Macros

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 bit fields. An error message will occur if bit fields are used.

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>`

32-Bit Language Tools Libraries

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 stream 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 `-mno-float` option, when enabled, will force linking of standard C libraries that do not support floating-point operations. The functionality is the same as that of the C standard forms, minus the support for floating-point output. Should a floating-point format specifier be used, the floating-point limited versions of the function will consume the value and output the text `:(float)` to the output stream.
- `--msingle-float` will cause the compiler to generate calls to formatted I/O routines that support `double` as if it were a `float` type.

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

Standard C Libraries with Math Functions

2.13.2 Customizing STDIO

The standard I/O relies on helper functions. There are two modes of operation, Simple mode and Full mode. Simple mode supports one character at a time I/O through the standard streams: `stdout`, `stdin`, and `stderr`. Full mode supports the complete set of standard I/O operations, such as files opened via the `fopen()` function.

Simple mode uses four helper functions for I/O. These are: `_mon_puts()`, `_mon_write()`, `_mon_putc()`, and `_mon_getc()`. Default operation for these functions are defined in **Section 2.13.3 “STDIO Functions”**. The default operation may be over-ridden by defining custom versions of these functions.

Full mode uses additional helper functions. These are: `close()`, `link()`, `lseek()`, `open()`, `read()`, `unlink()` and `write()`. Default versions of these functions are not provided, however the required prototypes and operation are discussed in **Section 2.13.3 “STDIO Functions”**.

2.13.3 STDIO Functions

Most of the following prototypes require inclusion of `stdio.h`, however some require `unistd.h` (see **Section 2.18 “<unistd.h> Miscellaneous Functions”**) or `fcntl.h`, particularly those concerned with the low-level implementation of the full STDIO mode. For values, refer to the header file.

2.13.4 Types

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>`

2.13.5 Constants

_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`.

32-Bit Language Tools Libraries

_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>

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>

FOPEN_MAX

Description: Defines the maximum number of files that can be simultaneously open
Include: <stdio.h>
Remarks: `stderr`, `stdin` and `stdout` are included in the `FOPEN_MAX` count.

L_tmpnam

Description: Defines the number of characters for the longest temporary filename created by the function `tmpnam`.
Include: <stdio.h>
Remarks: `L_tmpnam` is used to define the size of the array used by `tmpnam`.

NULL

Description: The value of a Null Pointer constant
Include: <stdio.h>

SEEK_CUR

Description: Indicates that `fseek` should seek from the current position of the file pointer
Include: <stdio.h>

Standard C Libraries with Math Functions

SEEK_END

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

Include: `<stdio.h>`

SEEK_SET

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

Include: `<stdio.h>`

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>`

2.13.6 Functions and Macros

_mon_getc

Description: Reads the next character from `stdin`.

Include: None.

Prototype: `int _mon_getc(int canblock);`

Argument: `canblock` non-zero to indicate that the function should block

Return Value: Returns the next character from the `FILE` associated with `stdin`. -1 is returned to indicate end-of-file.

Remarks: This function as provided always returns -1. This function can be replaced with one that reads from a UART or other input device.

32-Bit Language Tools Libraries

_mon_putc

Description: Writes a character to `stdout`.

Include: None.

Prototype: `void _mon_putc(char c);`

Argument: `c` character to be written

Return Value: Writes a character to the `FILE` associated with `stdout`.

Remarks: This function as provided always writes to UART 2 and assumes that the UART has already been initialized. This function can be replaced with one that writes to another UART or other output device.

asprintf

Description: Prints formatted text to an allocated string.

Prototype: `int asprintf(char **sp, const char *format, ...);`

Arguments: `sp` pointer to the allocated string
`format` format control string
`...` optional arguments

Return Value: Returns the number of characters stored in `s` excluding the terminating null character. A pointer to the allocated string is written to the first argument. If the memory allocation fails, -1 is returned by the function, and null is written to the String Pointer.

Remarks: The String Pointer should be passed to `free` to release the allocated memory when it is no longer needed.

clearerr

Description: Defined as a function-like macro in the header file. Resets the error indicator for the stream.

Include: `<stdio.h>`

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

Argument: `stream` stream to reset error indicators

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

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. `fclose` calls `close`, which is not provided by default.

Standard C Libraries with Math Functions

feof

Description: Defined as a function-like macro in the header file. 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.

ferror

Description: Defined as a function-like macro in the header file. Tests if error indicator is set.

Include: `<stdio.h>`

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

Argument: `stream` stream to check for error indicator
`stream` pointer to `FILE` structure

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

fflush

Description: Flushes the buffer in the specified stream causing all buffer IO to be transferred.

Include: `<stdio.h>`

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

Argument: `stream` 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. This function requires `lseek` in full mode, which is not provided by default.

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`.

32-Bit Language Tools Libraries

fgetpos

Description: Gets the stream's file position.

Include: <stdio.h>

Prototype: `int fgetpos(FILE *stream, fpos_t *pos);`

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

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

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

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.

Standard C Libraries with Math Functions

fopen

Description: Opens a file.

Include: <stdio.h>

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

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

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

Remarks: Following are the modes 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 does not 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 does not 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 does not 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 does not exist.)

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, usually one per format specifier

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`.

32-Bit Language Tools Libraries

fputc

Description: Puts a character to the stream.

Include: `<stdio.h>`

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

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

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

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

fputs

Description: Puts a string to the stream.

Include: `<stdio.h>`

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

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

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

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

fread

Description: Reads data from the stream.

Include: `<stdio.h>`

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

Arguments: *ptr* pointer to the storage buffer
size size of item
nelem maximum number of items to be read
stream pointer to the stream

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

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

Standard C Libraries with Math Functions

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.

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`.

fseek

Description: Moves file pointer to a specific location.

Include: `<stdio.h>`

Prototype: `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
This function requires `lseek`, which is not provided by default.

32-Bit Language Tools Libraries

fsetpos

Description: Sets the stream's file position.

Include: <stdio.h>

Prototype: `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*.

ftell

Description: Gets the current position of a file pointer.

Include: <stdio.h>

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

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

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

Remarks: This function requires `lseek`, which is not provided by default.

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.

getc

Description: Defined as a function-like macro in the header file.
Get a character from the stream.

Include: <stdio.h>

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

Argument: *stream* pointer to the open stream

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

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

Standard C Libraries with Math Functions

getchar

Description: Defined as a function-like macro in the header file. Get a character from `stdin`.

Include: `<stdio.h>`

Prototype: `int getchar(void);`

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

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

gets

Description: Get a string from `stdin`.

Include: `<stdio.h>`

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

Argument: `s` pointer to the storage string

Return Value: Returns a pointer to the string `s` if successful, otherwise, returns a Null pointer

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

open

Description: Open a file for access, returning a file descriptor

Include: `<fcntl.h>`

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

Argument: `name` filename to open
`access` access method used to open file
`mode` access mode to use when creating a file

Return Value: `open` returns the file descriptor for the newly opened file or -1 to signal an error. If an error occurs `errno` is set. Appropriate values might be `ENFILE` or `EACCESS`.

Remarks: This function is not provided by default. This function is required to support `fopen` and `freopen`.

The following values for `access` must be supported at a minimum (others are available, but not documented here):

- `O_APPEND` append mode, the file pointer should initially start at the end of the file
 - `O_BINARY` binary mode, characters are not translated
 - `O_CREAT` create mode, a new file is created if necessary
 - `O_RDONLY` read only mode, file output is not permitted
 - `O_RDWR` read/ write mode
 - `O_WRONLY` write only mode, file input is not permitted
-

32-Bit Language Tools Libraries

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

printf

Description: Prints formatted text to `stdout`. See also **Section 2.13.2 “Customizing STDIO”**.

Include: `<stdio.h>`

Prototype: `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.

Standard C Libraries with Math Functions

printf (Continued)

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

putc

Description:	Defined as a function-like macro in the header file. Puts a character to the stream.
Include:	<stdio.h>
Prototype:	<code>int putc(int c, FILE *stream);</code>
Arguments:	<i>c</i> character to be written <i>stream</i> pointer to FILE structure
Return Value:	Returns the character or EOF if an error occurs or end-of-file is reached.
Remarks:	<code>putc</code> is the same as the function <code>fputc</code> .

32-Bit Language Tools Libraries

putchar

Description: Defined as a function-like macro in the header file. 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.

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.

remove

Description: Deletes the specified file.

Include: `<stdio.h>`

Prototype: `int remove(const char *filename);`

Argument: `filename` name of file to be deleted.

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

Remarks: This function requires a definition of `unlink`. If `filename` does not exist or is open, `remove` will fail.

rename

Description: Renames the specified file.

Include: `<stdio.h>`

Prototype: `int rename(const char *old, const char *new);`

Arguments: `old` pointer to the old name
`new` pointer to the new name.

Return Value: Return 0 if successful, non-zero if not.

Remarks: This function requires definitions of `link` and `unlink`. The new name must not already exist in the current working directory, the old name must exist in the current working directory.

Standard C Libraries with Math Functions

rewind

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

Include: `<stdio.h>`

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

Argument: *stream* stream to reset the file pointer

Remarks: The function calls `fseek(stream, 0L, SEEK_SET)` and then clears the error indicator for the given stream.

scanf

Description: Scans formatted text from `stdin`.

Include: `<stdio.h>`

Prototype: `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 assignment.

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

```
%[*][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
- l modifier – used with type d, i, o, u, x, X; converts the value to a long int or unsigned long int
- l modifier – used with n; specifies that the pointer points to a long int
- l modifier – used with c; specifies a wide character
- l modifier – used with type e, E, f, F, g, G; converts the value to a double
- ll modifier – used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int
- ll modifier – used with n; specifies that the pointer points to a long long int
- L modifier – used with e, E, f, g, G; converts the value to a long double

32-Bit Language Tools Libraries

scanf (Continued)

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

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.

setvbuf

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

Include: <stdio.h>

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

Arguments: *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

Standard C Libraries with Math Functions

snprintf

Description: Prints formatted text to a string with maximum length.

Prototype: `int snprintf(char *s, size_t n, const char *format, ...);`

Arguments: *s* storage string for input
n number of characters to print
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`.

sprintf

Description: Prints formatted text to a string

Include: `<stdio.h>`

Prototype: `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`.

sscanf

Description: Scans formatted text from a string

Include: `<stdio.h>`

Prototype: `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`.

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.

32-Bit Language Tools Libraries

tmpnam

Description: Creates a unique temporary filename

Include: `<stdio.h>`

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

Argument: *s* pointer to the temporary name

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

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

ungetc

Description: Pushes character back onto stream.

Include: `<stdio.h>`

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

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

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

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

vfprintf

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

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

Prototype: `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 **Section 2.11 “<stdarg.h> Variable Argument Lists”**.

Standard C Libraries with Math Functions

vfscanf

- Description:** Scans formatted text using variable length argument list.
- Prototype:** `int vfscanf(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 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 assignment.
- Remarks:** The format argument has the same syntax and use that it has in `scanf`. 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 `vfscanf` function is called. Invoke `va_end` after the function returns. For more details see **Section 2.11 “<stdarg.h> Variable Argument Lists”**.
-

vprintf

- Description:** Prints formatted text to `stdout` using a variable length argument list
- Include:** `<stdio.h>`
`<stdarg.h>`
- Prototype:** `int vprintf(const char *format, va_list ap);`
- Arguments:** *format* format control string
ap pointer to a list of arguments
- Return Value:** Returns number of characters generated or a negative number if an error occurs.
- Remarks:** The format argument has the same syntax and use that it has in `printf`. To access the variable length argument list, the *ap* variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vprintf` function is called. Invoke `va_end` after the function returns. For more details see **Section 2.11 “<stdarg.h> Variable Argument Lists”**.
-

vscanf

- Description:** Scans formatted text from `stdin` using variable length argument list.
- Prototype:** `int vscanf(const char *format, va_list ap);`
- Arguments:** *format* format control string
ap pointer to a list of 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 assignment.
- Remarks:** The format argument has the same syntax and use that it has in `scanf`. 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 `vscanf` function is called. Invoke `va_end` after the function returns. For more details see **Section 2.11 “<stdarg.h> Variable Argument Lists”**.
-

32-Bit Language Tools Libraries

vsnprintf

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

Prototype: `int vsnprintf(char *s, size_t n, const char *format, va_list ap);`

Arguments:

<i>s</i>	storage string for input
<i>n</i>	number of characters to print
<i>format</i>	format control string
<i>ap</i>	pointer to a list of 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`. 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 `vsnprintf` function is called. Invoke `va_end` after the function returns. For more details see **Section 2.11 “<stdarg.h> Variable Argument Lists”**

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:

<i>s</i>	storage string for output
<i>format</i>	format control string
<i>ap</i>	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 **Section 2.11 “<stdarg.h> Variable Argument Lists”**.

Standard C Libraries with Math Functions

vsscanf

Description: Scans formatted text from a string using variable length argument list.

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

Arguments:

<i>s</i>	storage string for input
<i>format</i>	format control string
<i>ap</i>	pointer to a list of 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 assignment.

Remarks: The format argument has the same syntax and use that it has in `scanf`. 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 `vsscanf` function is called. Invoke `va_end` after the function returns. For more details see **Section 2.11 “<stdarg.h> Variable Argument Lists”**.

32-Bit Language Tools Libraries

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. For values, refer to the header file.

2.14.1 Types

div_t

Description: A type that holds a quotient and remainder of a signed integer division with operands of type `int`.

Include: `<stdlib.h>`

Prototype: `typedef struct { int quot, rem; } div_t;`

Remarks: This is the structure type returned by the function `div`.

ldiv_t

Description: A type that holds a quotient and remainder of a signed integer division with operands of type `long`.

Include: `<stdlib.h>`

Prototype: `typedef struct { long quot, rem; } ldiv_t;`

Remarks: This is the structure type returned by the function `ldiv`.

lldiv_t

Description: A type that holds a quotient and remainder of a signed integer division with operands of type `long`.

Include: `<stdlib.h>`

Prototype: `typedef struct { long long quot, rem; } lldiv_t;`

Remarks: This is the structure type returned by the function `lldiv`.

2.14.2 Constants

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

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.

MB_CUR_MAX

Description: Maximum number of characters in a multibyte character

Include: `<stdlib.h>`

Standard C Libraries with Math Functions

RAND_MAX

Description: Maximum value capable of being returned by the `rand` function

Include: `<stdlib.h>`

2.14.3 Functions and Macros

abort

Description: Aborts the current process.

Include: `<stdlib.h>`

Prototype: `void abort(void);`

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

abs

Description: Calculates the absolute value.

Include: `<stdlib.h>`

Prototype: `int abs(int i);`

Argument: `i` integer value

Return Value: Returns the absolute value of `i`.

Remarks: A negative number is returned as positive. A positive number is unchanged.

atexit

Description: Registers the specified function to be called when the program terminates normally.

Include: `<stdlib.h>`

Prototype: `int atexit(void(*func)(void));`

Argument: `func` function to be called

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

Remarks: For the registered functions to be called, the program must terminate with the `exit` function call.

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, NULL)`.

32-Bit Language Tools Libraries

atoi

Description: Converts a string to an integer.

Include: `<stdlib.h>`

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

Argument: `s` string to be converted

Return Value: Returns the converted integer if successful, otherwise, returns 0.

Remarks: The number may consist of the following:
[whitespace] [sign] digits
optional whitespace, followed by an optional sign then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to `(int) strtol(s, NULL, 10)`.

atol

Description: Converts a string to a long integer.

Include: `<stdlib.h>`

Prototype: `long atol(const char *s);`

Argument: `s` string to be converted

Return Value: Returns the converted long integer if successful, otherwise, returns 0

Remarks: The number may consist of the following:
[whitespace] [sign] digits
optional whitespace, followed by an optional sign then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to `strtol(s, NULL, 10)`.

atoll

Description: Converts a string to a long long integer.

Include: `<stdlib.h>`

Prototype: `long long atoll(const char *s);`

Argument: `s` string to be converted

Return Value: Returns the converted long long integer if successful, otherwise, returns 0

Remarks: The number may consist of the following:
[whitespace] [sign] digits
optional whitespace, followed by an optional sign then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to `strtoll(s, NULL, 10)`.

Standard C Libraries with Math Functions

bsearch

Description: Performs a binary search

Include: <stdlib.h>

Prototype:

```
void *bsearch(const void *key, const void *base,
             size_t nelem, size_t size,
             int (*cmp)(const void *ck, const void *ce));
```

Arguments:

<i>key</i>	object to search for
<i>base</i>	pointer to the start of the search data
<i>nelem</i>	number of elements
<i>size</i>	size of elements
<i>cmp</i>	pointer to the comparison function
<i>ck</i>	pointer to the key for the search
<i>ce</i>	pointer to the element being compared with the key.

Return Value: Returns a pointer to the object being searched for if found, otherwise, returns null.

Remarks: The value returned by the compare function is <0 if *ck* is less than *ce*, 0 if *ck* is equal to *ce*, or >0 if *ck* is greater than *ce*. *bsearch* requires the list to be sorted in increasing order according to the compare function pointed to by *cmp*.

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:

<i>nelem</i>	number of elements
<i>size</i>	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. In order to allocate memory using *calloc*, a heap must be created by specifying a linker command option. See Section 7.7 in the *MPLAB[®] XC32 C/C++ Compiler User's Guide* for more information.

div

Description: Calculates the quotient and remainder of two numbers

Include: <stdlib.h>

Prototype:

```
div_t div(int numer, int denom);
```

Arguments:

<i>numer</i>	numerator
<i>denom</i>	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 an infinite loop. Write a math error handler to take another application-specific action.

32-Bit Language Tools Libraries

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 enters an infinite loop.

free

Description: Frees memory.

Include: `<stdlib.h>`

Prototype: `void free(void *ptr);`

Argument: *ptr* points to memory to be freed

Remarks: Frees memory previously allocated with `calloc`, `malloc`, or `realloc`. If `free` is used on space that has already been deallocated (by a previous call to `free` or by `realloc`) or on space not allocated with `calloc`, `malloc`, or `realloc`, the behavior is undefined.

getenv

Description: Get a value for an environment variable.

Include: `<stdlib.h>`

Prototype: `char *getenv(const char *name);`

Argument: *name* name of environment variable

Return Value: Returns a pointer to the value of the environment variable if successful, otherwise, returns a Null Pointer.

Remarks: In a hosted environment, this function can be used to access environment variables defined by the host operating system. By default the 32-bit C compiler does not constitute a hosted environment, and as such this function always returns `NULL`.

labs

Description: Calculates the absolute value of a long integer.

Include: `<stdlib.h>`

Prototype: `long labs(long i);`

Argument: *i* long integer value

Return Value: Returns the absolute value of *i*.

Remarks: A negative number is returned as positive. A positive number is unchanged.

Standard C Libraries with Math Functions

ldiv

Description: Calculates the quotient and remainder of two long integers.

Include: `<stdlib.h>`

Prototype: `ldiv_t ldiv(long numer, long denom);`

Arguments: *numer* numerator
denom denominator

Return Value: Returns the quotient and the remainder.

Remarks: The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator (quot * denom + rem = numer). If the denominator is zero, the behavior is undefined.

labs

Description: Calculates the absolute value of a long long integer.

Include: `<stdlib.h>`

Prototype: `long long labs(long long i);`

Arguments: *i* long long integer value

Return Value: Returns the absolute value of *i*.

Remarks: A negative number is returned as positive. A positive number is unchanged.

lldiv

Description: Calculates the quotient and remainder of two long long integers.

Include: `<stdlib.h>`

Prototype: `lldiv_t lldiv(long long num, long long denom);`

Arguments: *numer* numerator
denom denominator

Return Value: Returns the quotient and 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.

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. In order to allocate memory using `malloc`, a heap must be created by specifying a linker command option. See Section 7.7 in the MPLAB® XC32 C/C++ Compiler User's Guide for more information.

32-Bit Language Tools Libraries

mblen

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

Include: `<stdlib.h>`

Prototype: `int mblen(const char *s, size_t n);`

Arguments: *s* points to the multibyte character
n number of bytes to check

Return Value: Returns zero if *s* points to a null character, otherwise, returns 1.

Remarks: The 32-bit C 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 below.)

Include: `<stdlib.h>`

Prototype: `size_t mbstowcs(wchar_t *wcs, const char *s, size_t n);`

Arguments: *wcs* points to the wide character string
s points to the multibyte string
n the number of wide characters to convert.

Return Value: Returns the number of wide characters stored excluding the null character.

Remarks: `mbstowcs` converts *n* number of wide characters unless it encounters a null wide character first. The 32-bit C compiler does not support multibyte characters with length greater than 1 byte.

mbtowc

Description: Converts a multibyte character to a wide character. (See Remarks below.)

Include: `<stdlib.h>`

Prototype: `int mbtowc(wchar_t *pwc, const char *s, size_t n);`

Arguments: *pwc* points to the wide character
s points to the multibyte character
n number of bytes to check

Return Value: Returns zero if *s* points to a null character, otherwise, returns 1

Remarks: The resulting wide character will be stored at *pwc*. The 32-bit C compiler does not support multibyte characters with length greater than 1 byte.

Standard C Libraries with Math Functions

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:

<i>base</i>	pointer to the start of the array
<i>nelem</i>	number of elements
<i>size</i>	size of the elements
<i>cmp</i>	pointer to the comparison function
<i>e1</i>	pointer to the key for the search
<i>e2</i>	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. `qsort` sorts the buffer in ascending order. The comparison function should return negative if the first argument is less than the second, zero if they are equal, and positive if the first argument is greater than the second.

rand

Description: Generates a pseudo-random integer.

Include: <stdlib.h>

Prototype: int rand(void);

Return Value: Returns an integer between 0 and `RAND_MAX`.

Remarks: Calls to this function return pseudo-random integer values in the range [0,RAND_MAX]. To use this function effectively, you must seed the random number generator using the `srand` function. This function will always return the same sequence of integers when no seeds are used or when identical seed values are used.

realloc

Description: Reallocates memory to allow a size change.

Include: <stdlib.h>

Prototype: void *realloc(void *ptr, size_t size);

Arguments:

<i>ptr</i>	points to previously allocated memory
<i>size</i>	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.

In order to allocate memory using `mrealloc`, a heap must be created by specifying a linker command option. See Section 7.7 in the *MPLAB® XC32 C/C++ Compiler User's Guide* for more information

32-Bit Language Tools Libraries

srand

Description: Set the starting seed for the pseudo-random number sequence.

Include: `<stdlib.h>`

Prototype: `void srand(unsigned int seed);`

Argument: *seed* starting value for the pseudo-random number sequence

Return Value: None

Remarks: This function sets the starting seed for the pseudo-random number sequence generated by the `rand` function. The `rand` function will always return the same sequence of integers when identical seed values are used. If `rand` is called with a seed value of 1, the sequence of numbers generated will be the same as if `rand` had been called without `srand` having been called first.

strtod

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

Include: `<stdlib.h>`

Prototype: `long double strtod(const char *s, char **endptr);`

Arguments: *s* string to be converted
endptr pointer to the character at which the conversion stopped

Return Value: Returns the converted number if successful, otherwise, returns 0.

Remarks: The number may consist of the following:
[whitespace] [sign] digits [.digits]
[{ e | E } [sign] digits]
optional whitespace, followed by an optional sign, then a sequence of one or more digits with an optional decimal point, followed by one or more optional digits and an optional e or E followed by an optional signed exponent.
`strtod` converts the string until it reaches a character that cannot be converted to a number. *endptr* will point to the remainder of the string starting with the first unconverted character.
If a range error occurs, `errno` will be set.

strtof

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

Include: `<stdlib.h>`

Prototype: `float strtof(const char *s, char **endptr);`

Arguments: *s* string to be converted
endptr pointer to the character at which the conversion stopped

Return Value: Returns the converted number if successful, otherwise, returns 0.

Remarks: The number may consist of the following:
[whitespace] [sign] digits [.digits]
[{ e | E } [sign] digits]
optional whitespace, followed by an optional sign, then a sequence of one or more digits with an optional decimal point, followed by one or more optional digits and an optional e or E followed by an optional signed exponent.
`strtof` 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.

Standard C Libraries with Math Functions

strtol

Description: Converts a partial string to a long integer.

Include: `<stdlib.h>`

Prototype: `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.

strtoll

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

Include: `<stdlib.h>`

Prototype: `long long strtoll(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, `strtoll` 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 `strtoll` 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.

strtoul

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

Include: `<stdlib.h>`

Prototype: `unsigned long strtoul(const char *s, char **endptr, int base);`

Arguments: *s* string to be converted
endptr pointer to the character at which the conversion stopped
base number base to use in conversion

Return Value: Returns the converted number if successful, otherwise, returns 0.

Remarks: If *base* is zero, `strtoul` attempts to determine the base automatically. It can be octal, determined by a leading zero, hexadecimal, determined by a leading `0x` or `0X`, or decimal in any other case. If *base* is specified `strtoul` converts a sequence of digits and letters a-z (case insensitive), where a-z represents the numbers 10-36. Conversion stops when an out-of-base number is encountered. *endptr* will point to the remainder of the string starting with the first unconverted character. If a range error occurs, `errno` will be set.

32-Bit Language Tools Libraries

strtoull

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

Include: <stdlib.h>

Prototype: `unsigned long long strtoull(const char *s, char **endptr, int base);`

Arguments:

<i>s</i>	string to be converted
<i>endptr</i>	pointer to the character at which the conversion stopped
<i>base</i>	number base to use in conversion

Return Value: Returns the converted number if successful, otherwise, returns 0.

Remarks: If *base* is zero, `strtoull` 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 `strtoull` 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.

system

Description: Execute a command.

Include: <stdlib.h>

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

Argument: *s* command to be executed

Return Value: Returns zero if a null argument is passed, otherwise, returns -1.

Remarks: In a hosted environment, this function can be used to execute commands on the host operating system. By default the 32-bit C compiler does not constitute a hosted environment, and as such this function does nothing.

wcstombs

Description: Converts a wide character string to a multibyte string. (See Remarks below.)

Include: <stdlib.h>

Prototype: `size_t wcstombs(char *s, const wchar_t *wcs, size_t n);`

Arguments:

<i>s</i>	points to the multibyte string
<i>wcs</i>	points to the wide character string
<i>n</i>	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 32-bit C compiler does not support multibyte characters with length greater than 1 character.

Standard C Libraries with Math Functions

wctomb

Description: Converts a wide character to a multibyte character. (See Remarks below.)

Include: `<stdlib.h>`

Prototype: `int wctomb(char *s, wchar_t wchar);`

Arguments: *s* points to the multibyte character
wchar the wide character to be converted

Return Value: Returns zero if *s* points to a null character, otherwise, returns 1.

Remarks: The resulting multibyte character is stored at *s*. The 32-bit C compiler does not support multibyte characters with length greater than 1 character.

32-Bit Language Tools Libraries

2.15 <STRING.H> STRING FUNCTIONS

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

2.15.1 Types

size_t

Description: The type of the result of the `sizeof` operator.

Include: `<string.h>`

2.15.2 Functions and Macros

ffs

Description: Find the first bit set.

Include: `<string.h>`

Prototype: `int ffs (int num);`

Arguments: *num* the value to be tested

Return Value: Returns an integer representing the index of the first bit set in *num*, starting from the Least Significant bit, which is numbered one.

Remarks: If no bits are set (i.e., the argument is zero) zero is returned.

ffsl

Description: Find the first bit set long.

Include: `<string.h>`

Prototype: `int ffsl (long num);`

Arguments: *num* the value to be tested

Return Value: Returns an integer representing the index of the first bit set in *num*, starting from the Least Significant bit, which is numbered one.

Remarks: If no bits are set (i.e., the argument is zero) zero is returned.

ffsll

Description: Find the first bit set long long.

Include: `<string.h>`

Prototype: `int ffsll (long long num);`

Arguments: *num* the value to be tested

Return Value: Returns an integer representing the index of the first bit set in *num*, starting from the Least Significant bit, which is numbered one.

Remarks: If no bits are set (i.e., the argument is zero) zero is returned.

Standard C Libraries with Math Functions

memchr

Description: Locates a character in a buffer.

Include: `<string.h>`

Prototype: `void *memchr(const void *s, int c, size_t n);`

Arguments:

<i>s</i>	pointer to the buffer
<i>c</i>	character to search for
<i>n</i>	number of characters to check

Return Value: Returns a pointer to the location of the match if successful, otherwise, returns null.

Remarks: `memchr` stops when it finds the first occurrence of *c* or after searching *n* number of characters.

memcmp

Description: Compare the contents of two buffers.

Include: `<string.h>`

Prototype: `int memcmp(const void *s1, const void *s2, size_t n);`

Arguments:

<i>s1</i>	first buffer
<i>s2</i>	second buffer
<i>n</i>	number of characters to compare

Return Value: Returns a positive number if *s1* is greater than *s2*, zero if *s1* is equal to *s2*, or a negative number if *s1* is less than *s2*.

Remarks: 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.

memcpy

Description: Copies characters from one buffer to another.

Include: `<string.h>`

Prototype: `void *memcpy(void *dst, const void *src, size_t n);`

Arguments:

<i>dst</i>	buffer to copy characters to
<i>src</i>	buffer to copy characters from
<i>n</i>	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.

32-Bit Language Tools Libraries

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:

<i>s1</i>	buffer to copy characters to (destination)
<i>s2</i>	buffer to copy characters from (source)
<i>n</i>	number of characters to copy from <i>s2</i> to <i>s1</i>

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.

memset

Description: Copies the specified character into the destination buffer.

Include: `<string.h>`

Prototype: `void *memset(void *s, int c, size_t n);`

Arguments:

<i>s</i>	buffer
<i>c</i>	character to put in buffer
<i>n</i>	number of times

Return Value: Returns the buffer with characters written to it.

Remarks: The character *c* is written to the buffer *n* times.

strcasecmp

Description: Compares two strings, ignoring case.

Include: `<string.h>`

Prototype: `int strcasecmp (const char *s1, const char *s2);`

Arguments:

<i>s1</i>	first string
<i>s2</i>	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.

strcat

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

Include: `<string.h>`

Prototype: `char *strcat(char *s1, const char *s2);`

Arguments:

<i>s1</i>	null terminated destination string to copy to
<i>s2</i>	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.

Standard C Libraries with Math Functions

strchr

Description: Locates the first occurrence of a specified character in a string.

Include: `<string.h>`

Prototype: `char *strchr(const char *s, int c);`

Arguments: *s* pointer to the string
c character to search for

Return Value: Returns a pointer to the location of the match if successful, otherwise, returns a Null Pointer.

Remarks: This function searches the string *s* to find the first occurrence of the character *c*.

strcmp

Description: Compares two strings.

Include: `<string.h>`

Prototype: `int strcmp(const char *s1, const char *s2);`

Arguments: *s1* first string
s2 second string

Return Value: Returns a positive number if *s1* is greater than *s2*, zero if *s1* is equal to *s2*, or a negative number if *s1* is less than *s2*.

Remarks: This function compares successive characters from *s1* and *s2* until they are not equal or the null terminator is reached.

strcoll

Description: Compares one string to another. (See Remarks below.)

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 32-bit C 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.

32-Bit Language Tools Libraries

strcspn

Description: Calculate the number of consecutive characters at the beginning of a string that are not contained in a set of characters.

Include: <string.h>

Prototype: `size_t strcspn(const char *s1, const char *s2);`

Arguments: *s1* pointer to the string to be searched
s2 pointer to characters to search for

Return Value: Returns the length of the segment in *s1* not containing characters found in *s2*.

Remarks: This function will determine the number of consecutive characters from the beginning of *s1* that are not contained in *s2*.

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.

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.

strncasecmp

Description: Compares two strings, ignoring case, up to a specified number of characters.

Include: <string.h>

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

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: *strncasecmp* returns a value based on the first character that differs between *s1* and *s2*. Characters that follow a null character are not compared.

Standard C Libraries with Math Functions

strncat

Description: Append a specified number of characters from the source string to the destination string.

Include: `<string.h>`

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

Arguments:

<i>s1</i>	destination string to copy to
<i>s2</i>	source string to copy from
<i>n</i>	number of characters to append

Return Value: Returns a pointer to the destination string.

Remarks: This function appends up to *n* characters (a null character and characters that follow it are not appended) from the source string to the end of the destination string. If a null character is not encountered, then a terminating null character is appended to the result. If the strings overlap, the behavior is undefined.

strncmp

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

Include: `<string.h>`

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

Arguments:

<i>s1</i>	first string
<i>s2</i>	second string
<i>n</i>	number of characters to compare

Return Value: Returns a positive number if *s1* is greater than *s2*, zero if *s1* is equal to *s2*, or a negative number if *s1* is less than *s2*.

Remarks: `strncmp` returns a value based on the first character that differs between *s1* and *s2*. Characters that follow a null character are not compared.

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:

<i>s1</i>	destination string to copy to
<i>s2</i>	source string to copy from
<i>n</i>	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.

32-Bit Language Tools Libraries

strupbrk

Description: Search a string for the first occurrence of a character from a specified set of characters.

Include: <string.h>

Prototype: `char *strupbrk(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*.

strrchr

Description: Search for the last occurrence of a specified character in a string.

Include: <string.h>

Prototype: `char *strrchr(const char *s, int c);`

Arguments: *s* pointer to the string to be searched
c character to search for

Return Value: Returns a pointer to the character if found, otherwise, returns a Null Pointer.

Remarks: The function searches the string *s*, including the terminating null character, to find the last occurrence of character *c*.

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*.

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.

Standard C Libraries with Math Functions

strtok

Description: Break a string into substrings, or tokens, by inserting null characters in place of specified delimiters.

Include: <string.h>

Prototype: char *strtok(char *s1, const char *s2);

Arguments: s1 pointer to the null terminated string to be searched
s2 pointer to characters to be searched for (used as delimiters)

Return Value: Returns a pointer to the first character of a token (the first character in s1 that does not appear in the set of characters of s2). If no token is found, the Null Pointer is returned.

Remarks: A sequence of calls to this function can be used to split up a string into substrings (or tokens) by replacing specified characters with null characters. The first time this function is invoked on a particular string, that string should be passed in s1. After the first time, this function can continue parsing the string from the last delimiter by invoking it with a null value passed in s1.

It skips all leading characters that appear in the string s2 (delimiters), then skips all characters not appearing in s2 (this segment of characters is the token), and then overwrites the next character with a null character, terminating the current token. The function strtok then saves a pointer to the character that follows, from which the next search will start. If strtok finds the end of the string before it finds a delimiter, the current token extends to the end of the string pointed to by s1. If this is the first call to strtok, it does not modify the string (no null characters are written to s1). The set of characters that is passed in s2 need not be the same for each call to strtok.

If strtok is called with a non-null parameter for s1 after the initial call, the string becomes the new string to search. The old string previously searched will be lost.

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 32-bit C 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.

32-Bit Language Tools Libraries

2.16 <TIME.H> DATE AND TIME FUNCTIONS

The header file `time.h` consists of types, macros and functions that manipulate time. For values, refer to the header file.

2.16.1 Types

clock_t

Description: Stores processor time values.

Include: `<time.h>`

Prototype: `unsigned long clock_t`

Remarks: This value is established by convention, and does not reflect the actual execution environment. The actual timing will depend upon the helper function `settimeofday`, which is not provided by default.

size_t

Description: The type of the result of the `sizeof` operator.

Include: `<stddef.h>`

struct timeval

Description: Structure to hold current processor time.

Include: `<sys/time.h>`

Prototype:

```
struct timeval {
    long    tv_sec;           /* seconds */
    long    tv_usec;        /* microseconds */
};
```

Return Value: Returns the calendar time encoded as a value of `time_t`.

Remarks: Used by helper functions `gettimeofday` and `settimeofday`, which are not provided by default.

Standard C Libraries with Math Functions

struct tm

Description: Structure used to hold the time and date (calendar time).

Include: <time.h>

Prototype:

```
struct tm {
    int tm_sec; /*seconds after the minute ( 0 to 61 )*/
                /*allows for up to two leap seconds*/
    int tm_min; /*minutes after the hour ( 0 to 59 )*/
    int tm_hour; /*hours since midnight ( 0 to 23 )*/
    int tm_mday; /*day of month ( 1 to 31 )*/
    int tm_mon; /*month ( 0 to 11 where January = 0 )*/
    int tm_year; /*years since 1900*/
    int tm_wday; /*day of week ( 0 to 6 where Sunday = 0 )*/
    int tm_yday; /*day of year ( 0 to 365 where January 1 = 0
)*/
    int tm_isdst; /*Daylight Savings Time flag*/
}
```

Remarks: If `tm_isdst` is a positive value, Daylight Savings is in effect. If it is zero, Daylight Saving time is not in effect. If it is a negative value, the status of Daylight Saving Time is not known.

time_t

Description: Represents calendar time values.

Include: <time.h>

Prototype: typedef long time_t

Remarks: Calendar time is reported in seconds.

2.16.2 Constants

CLOCKS_PER_SEC

Description: Number of processor clocks per second.

Include: <time.h>

Prototype: #define CLOCKS_PER_SEC

Remarks: This value is established by convention, and may not reflect the actual execution environment. The actual timing will depend upon helper function `settimeofday`, which is not provided by default.

32-Bit Language Tools Libraries

2.16.3 Functions and Macros

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

clock

Description: Calculates the processor time.

Include: `<time.h>`

Prototype: `clock_t clock(void);`

Return Value: Returns the number of clock ticks of elapsed processor time.

Remarks: If the target environment cannot measure elapsed processor time, the function returns -1, cast as a `clock_t`. (i.e., `(clock_t)-1`). This value is established by convention, and may not reflect the actual execution environment. The actual timing will depend upon helper function `settimeofday`, which is not provided by default.

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))`.

difftime

Description: Find the difference between two times.

Include: `<time.h>`

Prototype: `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`.

Standard C Libraries with Math Functions

gettimeofday (User Provided)

Description: Gets the current processor time.

Include: `<time.h>`

Prototype: `int gettimeofday(struct timeval *tv , void *tz);`

Argument: `tv` a structure to contain the current time
`tz` obsolete argument; should be null

Return Value: Returns 0 if successful, -1 on error.

Remarks: This helper function should interact with the target environment and write the current processor time in seconds and microseconds to `tv`. It is not provided by default, but is required by `clock` and `time`.

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: `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`. `gmtime` and `localtime` are equivalent except `gmtime` will return `tm_isdst` (Daylight Savings Time flag) as zero to indicate that Daylight Savings Time is not in effect.

localtime

Description: Converts a value to the local time.

Include: `<time.h>`

Prototype: `struct tm *localtime(const time_t *tod);`

Argument: `tod` pointer to stored time

Return Value: Returns the address of the time structure.

Remarks: `localtime` and `gmtime` are 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.

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`).

32-Bit Language Tools Libraries

settimeofday (User Provided)

Description: Sets the current processor time.

Include: `<time.h>`

Prototype: `int settimeofday(const struct timeval *tv , void *tz);`

Argument: *tv* a structure containing the current time
tz obsolete argument; should be null

Return Value: Returns 0 if successful, -1 on error.

Remarks: This function should interact with the target environment and set the current time using values specified in *tv*. It is not required by other functions.

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)
- %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 %

Standard C Libraries with Math Functions

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`. This function requires the helper function `gettimeofday`, which is not provided by default. Calendar time will be returned in seconds.

32-Bit Language Tools Libraries

2.17 <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 **Section 2.5 “<errno.h> Errors”**).

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.

2.17.1 Constants

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`.

2.17.2 Functions and Macros

acos

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

Include: `<math.h>`

Prototype: `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.

acosf

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

Include: `<math.h>`

Prototype: `float acosf (float x);`

Argument: `x` value between -1 and 1

Return Value: Returns the arc cosine in radians in the range of 0 to pi (inclusive).

Remarks: A domain error occurs if `x` is less than -1 or greater than 1.

Standard C Libraries with Math Functions

asin

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

Include: `<math.h>`

Prototype: `double asin (double x);`

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

Return Value: Returns the arc sine in radians in the range of $-\pi/2$ to $+\pi/2$ (inclusive).

Remarks: A domain error occurs if `x` is less than -1 or greater than 1.

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.

asinh

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

Include: `<math.h>`

Prototype: `double asinh (double x);`

Argument: `x` floating-point value

Return Value: Returns the hyperbolic arc sine of `x`.

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.

atan2

Description: Calculates the trigonometric arc tangent function of y/x .

Include: `<math.h>`

Prototype: `double atan2 (double y, double x);`

Arguments: `y` y value for which to return the arc tangent
`x` x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of $-\pi$ to π (inclusive) with the quadrant determined by the signs of both parameters.

Remarks: A domain error occurs if both `x` and `y` are zero or both `x` and `y` are \pm infinity.

32-Bit Language Tools Libraries

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 π with the quadrant determined by the signs of both parameters.

Remarks: A domain error occurs if both x and y are zero or both x and y are \pm infinity.

atanf

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

Include: `<math.h>`

Prototype: `float atanf (float x);`

Argument: x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of $-\pi/2$ to $+\pi/2$ (inclusive).

Remarks: No domain or range error will occur.

atanh

Description: Calculates the hyperbolic arc tan function of a double precision floating-point value.

Include: `<math.h>`

Prototype: `double atanh (double x);`

Argument: x floating-point value

Return Value: Returns the hyperbolic arc tangent of x .

cbrt

Description: Calculates the cube root of a double precision floating-point value.

Include: `<math.h>`

Prototype: `double cbrt (double x);`

Argument: x a non-negative floating-point value

Return Value: Returns the cube root of x . If x is $+\text{INF}$, $+\text{INF}$ is returned. If x is NaN, NaN is returned.

ceil

Description: Calculates the ceiling of a value.

Include: `<math.h>`

Prototype: `double ceil(double x);`

Argument: x a floating-point value for which to return the ceiling.

Return Value: Returns the smallest integer value greater than or equal to x .

Remarks: No domain or range error will occur. See `floor`.

Standard C Libraries with Math Functions

ceilf

Description: Calculates the ceiling of a value.
Include: `<math.h>`
Prototype: `float ceilf(float x);`
Argument: `x` floating-point value.
Return Value: Returns the smallest integer value greater than or equal to `x`.
Remarks: No domain or range error will occur. See `floorf`.

copysign

Description: Copies the sign of one floating-point number to another.
Include: `<math.h>`
Prototype: `double copysign (double x, double y);`
Argument: `x` floating-point value
`y` floating-point value
Return Value: Returns `x` with its sign changed to match the sign of `y`.

COS

Description: Calculates the trigonometric cosine function of a double precision floating-point value.
Include: `<math.h>`
Prototype: `double cos (double x);`
Argument: `x` value for which to return the cosine
Return Value: Returns the cosine of `x` in radians in the ranges of -1 to 1 inclusive.
Remarks: A domain error will occur if `x` is a NaN or infinity.

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.

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` would cause overflow.

32-Bit Language Tools Libraries

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 would cause overflow.

drem

Description: Calculates the double precision remainder function.

Include: <math.h>

Prototype: double drem(double x, double y)

Argument: x floating-point value

y floating-point value

Return Value: Returns $x - [x/y] * y$, where $[x/y]$ is the value x divided by y, rounded to the nearest integer. If $[x/y]$ is equidistant between two integers, round to the even one.

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 would cause overflow.

expf

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

Include: <math.h>

Prototype: float expf (float x);

Argument: x floating-point value for which to return the exponential

Return Value: Returns the exponential of x. On an overflow, expf returns inf and on an underflow exp returns 0.

Remarks: A range error occurs if the magnitude of x would cause overflow.

Standard C Libraries with Math Functions

expm1

Description: Calculates the exponential function $e^x - 1.0$.

Include: `<math.h>`

Prototype: `double expm1 (double x);`

Argument: `x` floating-point value

Return Value: Returns $e^x - 1.0$, unless that value is too large to represent in a double, in which case `HUGE_VAL` is returned.

Remarks: If a range error occurs, `errno` will be set.

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.

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.

finite

Description: Tests for the value "finite".

Include: `<math.h>`

Prototype: `int finite(double x);`

Argument: `x` floating-point value

Return Value: Returns a non-zero value if `x` is neither infinite or "Not a Number" (NaN), otherwise zero is returned.

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`.

32-Bit Language Tools Libraries

floor

Description: Calculates the floor of a single precision floating-point value.

Include: `<math.h>`

Prototype: `float floorf(float x);`

Argument: `x` floating-point value.

Return Value: Returns the largest integer value less than or equal to `x`.

Remarks: No domain or range error will occur. See `ceilf`.

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`.

fmodf

Description: Calculates the remainder of `x/y` as a single precision value.

Include: `<math.h>`

Prototype: `float fmodf(float x, float y);`

Arguments: `x` a single precision floating-point value

`y` a single precision floating-point value

Return Value: Returns the remainder of `x` divided by `y`.

Remarks: If `y = 0`, a domain error occurs. If `y` is non-zero, the result will have the same sign as `x` and the magnitude of the result will be less than the magnitude of `y`.

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.

Standard C Libraries with Math Functions

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.

hypot

Description: Calculates the Euclidean distance function.

Include: `<math.h>`

Prototype: `double hypot (double x, double y);`

Argument: `x` floating-point value
`y` floating-point value

Return Value: Returns $\sqrt{x^2 + y^2}$, unless that value is too large to represent in a double, in which case `HUGE_VAL` is returned. If `x` or `y` is `+INF` or `-INF`, `INF` is returned. If `x` or `y` is `Nan`, `NaN` is returned.

Remarks: If a range error occurs, `errno` will be set.

isinf

Description: Tests for the value "infinity."

Include: `<math.h>`

Prototype: `int isinf (double x);`

Argument: `x` floating-point value

Return Value: Returns -1 if `x` represents negative infinity, 1 if `x` represents positive infinity, otherwise 0 is returned.

isnan

Description: Tests for the value "Not a Number" (NaN).

Include: `<math.h>`

Prototype: `int isnan (double x);`

Argument: `x` floating-point value

Return Value: Returns a non-zero value if `x` represents "Not a Number" (NaN), otherwise 0 is returned.

32-Bit Language Tools Libraries

ldexp

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

Include: `<math.h>`

Prototype: `double ldexp(double x, int ex);`

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

Return Value: Returns $x * 2^{ex}$. On an overflow, `ldexp` returns `inf` and on an underflow, `ldexp` returns 0.

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

ldexpf

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

Include: `<math.h>`

Prototype: `float ldexpf(float x, int ex);`

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

Return Value: Returns $x * 2^{ex}$. On an overflow, `ldexp` returns `inf` and on an underflow, `ldexp` returns 0.

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

log

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

Include: `<math.h>`

Prototype: `double log(double x);`

Argument: `x` any positive value for which to return the log

Return Value: Returns the natural logarithm of `x`. `-inf` is returned if `x` is 0 and NaN is returned if `x` is a negative number.

Remarks: A domain error occurs if $x \leq 0$.

log10

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

Include: `<math.h>`

Prototype: `double log10(double x);`

Argument: `x` any double precision floating-point positive number

Return Value: Returns the base-10 logarithm of `x`. `-inf` is returned if `x` is 0 and NaN is returned if `x` is a negative number.

Remarks: A domain error occurs if $x \leq 0$.

Standard C Libraries with Math Functions

log10f

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

Include: `<math.h>`

Prototype: `float log10f(float x);`

Argument: `x` any single precision floating-point positive number

Return Value: Returns the base-10 logarithm of `x`. `-inf` is returned if `x` is 0 and NaN is returned if `x` is a negative number.

Remarks: A domain error occurs if $x \leq 0$.

log1p

Description: Calculates the natural logarithm of $(1.0 + x)$.

Include: `<math.h>`

Prototype: `double log1p (double x);`

Argument: `x` floating-point value

Return Value: Returns the natural logarithm of $(1.0 + x)$.

Remarks: If $x = -1$, a domain error occurs and `-INF` is returned. If $x < -1$, a domain error occurs and NaN is returned. If `x` is NaN, NaN is returned. If `x` is INF, `+INF` is returned.

logb

Description: Calculates the unbiased exponent of a floating-point number.

Include: `<math.h>`

Prototype: `double logb(x);`

Argument: `x` floating-point value

Return Value: Returns a signed integral value (in floating-point format) that represents the unbiased exponent of `x`. If `x` is 0., `-INF` is returned. If `x` is INF, `+INF` is returned. If `x` is NaN, NaN is returned.

logf

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

Include: `<math.h>`

Prototype: `float logf(float x);`

Argument: `x` any positive value for which to return the log

Return Value: Returns the natural logarithm of `x`. `-inf` is returned if `x` is 0 and NaN is returned if `x` is a negative number.

Remarks: A domain error occurs if $x \leq 0$.

32-Bit Language Tools Libraries

modf

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

Include: `<math.h>`

Prototype: `double modf(double x, double *pint);`

Arguments: *x* double precision floating-point value
pint pointer to the 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.

modff

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

Include: `<math.h>`

Prototype: `float modff(float x, float *pint);`

Arguments: *x* single precision floating-point value
pint pointer to the 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.

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.

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.

Standard C Libraries with Math Functions

rint

Description: Calculates the integral value nearest to x , in floating-point format.

Include: `<math.h>`

Prototype: `double rint (double x);`

Argument: x floating-point value

Return Value: Returns the integral value nearest to x , represented in floating-point format.

Remarks: If x is +INF or -INF, x is returned. If x is Nan, NaN is returned.

sin

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

Include: `<math.h>`

Prototype: `double sin (double x);`

Argument: x value for which to return the sine

Return Value: Returns the sine of x in radians in the ranges of -1 to 1 inclusive.

Remarks: A domain error will occur if x is a NaN or infinity.

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.

sinh

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

Include: `<math.h>`

Prototype: `double sinh (double x);`

Argument: x value for which to return the hyperbolic sine

Return Value: Returns the hyperbolic sine of x

Remarks: A range error will occur if the magnitude of x is too large.

sinhf

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

Include: `<math.h>`

Prototype: `float sinhf (float x);`

Argument: x value for which to return the hyperbolic sine

Return Value: Returns the hyperbolic sine of x

Remarks: A range error will occur if the magnitude of x is too large.

32-Bit Language Tools Libraries

sqrt

Description: Calculates the square root of a double precision floating-point value.
Include: `<math.h>`
Prototype: `double sqrt(double x);`
Argument: `x` a non-negative floating-point value
Return Value: Returns the non-negative square root of `x`.
Remarks: If `x` is negative, a domain error occurs.

sqrtf

Description: Calculates the square root of a single precision floating-point value.
Include: `<math.h>`
Prototype: `float sqrtf(float x);`
Argument: `x` non-negative floating-point value
Return Value: Returns the non-negative square root of `x`.
Remarks: If `x` is negative, a domain error occurs.

tan

Description: Calculates the trigonometric tangent function of a double precision floating-point value.
Include: `<math.h>`
Prototype: `double tan(double x);`
Argument: `x` value for which to return the tangent
Return Value: Returns the tangent of `x` in radians.
Remarks: A domain error will occur if `x` is a NaN or infinity.

tanf

Description: Calculates the trigonometric tangent function of a single precision floating-point value.
Include: `<math.h>`
Prototype: `float tanf(float x);`
Argument: `x` value for which to return the tangent
Return Value: Returns the tangent of `x`
Remarks: A domain error will occur if `x` is a NaN or infinity.

tanh

Description: Calculates the hyperbolic tangent function of a double precision floating-point value.
Include: `<math.h>`
Prototype: `double tanh(double x);`
Argument: `x` value for which to return the hyperbolic tangent
Return Value: Returns the hyperbolic tangent of `x` in the ranges of -1 to 1 inclusive.
Remarks: No domain or range error will occur.

Standard C Libraries with Math Functions

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.

32-Bit Language Tools Libraries

2.18 <UNISTD.H> MISCELLANEOUS FUNCTIONS

The header file `unistd.h` includes prototypes for helper functions that are not provided by default. These functions must be customized for the target environment.

close

Description: Closes the file associated with `fd`.

Include: `<unistd.h>`

Prototype: `int close(int fd);`

Argument: `fd` file descriptor of previously opened file.

Return Value: This function returns 0 if successful and -1 to indicate an error.

Remarks: This function is not provided by the default libraries and is required to be provided if `fclose()` is used. This function should close a file. A file need not necessarily be associated with a storage device. This function should return -1 to signal an error and a strict implementation will set `errno` to some appropriate value such as `EBADF` or `EIO`.

link

Description: Create a new file.

Include: `<unistd.h>`

Prototype: `int link(const char *from, const char *to);`

Argument: `from` filename from which to link
`to` destination filename of link

Return Value: Zero is returned to indicate success and -1 indicates an error condition.

Remarks: This function is not provided by default. Its purpose, in a file system, is to create a new filename, `to`, which contains the same data as the file named `from`. `errno` should also be set on error. This function is used by `rename`.

lseek

Description: Modify the current read or write position within a file.

Include: `<unistd.h>`

Prototype: `__off_t lseek(int fd, __off_t offset, int whence);`

Argument: `fd` file descriptor (returned by `open`) for file to seek
`offset` amount by which to seek
`whence` describes how to apply `offset` to the current file position

Return Value: `lseek` returns the resulting offset from the start of the file, measured in bytes. The function returns -1 to indicate an error and sets `errno`. Appropriate values might be `EBADF` or `EINVAL`.

Remarks: This function is not provided by default. This function is required to support `fflush`, `fseek`, and `ftell`.

Standard C Libraries with Math Functions

read

Description: Read bytes from an already `opened` file

Include: `<unistd.h>`

Prototype: `int read(int fd, void *buffer, size_t length);`

Argument: *fd* file from which to read
buffer storage buffer for at least *length* bytes
length maximum number of bytes to read

Return Value: Returns the number of bytes read and stores those bytes into memory pointed to by *buffer*. The value -1 is returned to signal an error and `errno` is set to indicate the kind of error. Appropriate values may be `EBADF` or `EINVAL`, among others.

Remarks: This function is not provided by default. It is required to support reading files in full mode, such as via `fgetc`, `fgets`, `fread`, and `gets`.

unlink

Description: Low level command to remove a file link.

Include: `<unistd.h>`

Prototype: `int unlink(const char *name);`

Argument: *name* file to be removed

Return Value: Returns zero if successful and -1 to signify an error.

Remarks: This function is not provided by default and is required for `remove` and `rename`. This function deletes a link between a filename and the file contents. The contents are also deleted when the last link is destroyed. A file may have multiple links to it if the `link` function has been used.

write

Description: Low-level support function for writing data to an already `opened` file.

Include: `<unistd.h>`

Prototype: `int write(int fd, void *buffer, size_t length);`

Arguments: *fd* file descriptor indicating which file should be written
buffer data to be written
length length, in bytes, of data to write

Return Value: Returns number of characters written with -1 indicating an error condition.

Remarks: This function is not provided by default. In the event that an error occurs, `errno` should be set to indicate the type of error. Suitable values may be `EBADF` or `EINVAL`, among others.

32-Bit Language Tools Libraries

NOTES:

Chapter 3. PIC32 DSP Library

3.1 INTRODUCTION**3.1.1 Overview**

The PIC32 DSP library consists of a set of functions that are applicable to many multimedia application areas. Most of the functions, like vector operations, filters, and transforms, are commonly used in many DSP and multimedia applications.

Some functions are designed to be used in specific applications such as video decoding or voice compression. It is beyond the scope of this manual to describe the operation of such applications.

Functions whose performance is considered critical are implemented in assembly and tuned where appropriate for a particular processor pipeline implementation and instruction set features. When a function is typically not considered to be performance critical, or the benefit from an assembly implementation is not significant, it is implemented in C. Often such functions perform initialization of data structures and are used only once during the lifetime of an application.

Table 3-1 lists all the functions currently available in the DSP Library, arranged by category, with the available implementation versions. All general purpose functions work with data in 16-bit fractional format, also known as Q15. Some of the functions also have a version that operates on 32-bit data in Q31 fractional format.

TABLE 3-1: GENERAL PURPOSE DSP LIBRARY FUNCTIONS BY CATEGORY

Category	Function Name	Description
Vector Math Functions	mips_vec_abs16/32	Compute the absolute value of each Q15/Q31 vector element.
	mips_vec_add16/32	Add the corresponding elements of two Q15/Q31 vectors.
	mips_vec_addc16/32	Add a constant to all elements of a vector.
	mips_vec_dotp16/32	Compute dot product of two Q15/Q31 vectors.
	mips_vec_mul16/32	Multiply the corresponding elements of two Q15/Q31 vectors. Can be used for applying windows.
	mips_vec_mulc16/32	Multiply all elements of a vector by a constant.
	mips_vec_sub16/32	Subtract the corresponding elements of two Q15/Q31 vectors.
	mips_vec_sum_squares16/32	Calculate the sum of squares of elements of a vector in Q15/Q31 format.
Filters	mips_fir16	Applies a block FIR filter to a Q15 vector.
	mips_fir16_setup	Prepare the filter coefficients for the mips_fir16 function.
	mips_iir16	Single-sample IIR filter.
	mips_iir16_setup	Prepare the filter coefficients for the mips_iir16 function.
	mips_lms16	Single-sample LMS filter
Transforms	mips_fft16	Compute the complex FFT of a vector containing Q15 complex samples, i.e., 16-bit fractional real and imaginary parts.
	mips_fft16_setup (deprecated)	Create a vector of twiddle factors used by the mips_fft16 function.
	mips_fft32	Compute the complex FFT of a vector containing Q31 complex samples, i.e., 32-bit fractional real and imaginary parts.
	mips_fft32_setup (deprecated)	Create a vector of twiddle factors used by the mips_fft32 function.
Video	mips_h264_iqt	Inverse quantization and transform for H.264 decoding.
	mips_h264_iqt_setup	Create inverse quantization matrix used by the mips_h264_iqt function.
	mips_h264_mc_luma	1/4-pixel motion compensation for luma pixels in H.264 video decoding.

3.1.2 Fixed-Point Types

Input and output data for most functions is represented in 16-bit fractional numbers, in Q15 format. This is the most commonly used data format for signal processing. Some function may use other data formats internally for increased precision of the intermediate results. The Q15 data type used by the DSP functions is specified as *int16* in the C header files supplied with the library. This data type is defined in the common *dsplib_def.h* header file.

Note that within C code care must be taken not to confuse fixed-point values with integers. To the C compiler, objects declared with *int16* type are integers, not fixed-point, and any arithmetic performed on those objects in C will be done as integers.

Fixed-point values have been declared as *int16* only because the standard C language does not include intrinsic support for fixed-point data types.

3.1.3 Saturation, Scaling, and Overflow

In the majority of DSP applications, overflow or underflow during computation is not desirable. It is best to design for appropriate scaling of the data path and avoid the possibility of overflow and underflow. However, such scaling can significantly limit the usable data range. Hence, many algorithm implementations relax the scaling and introduce saturation operations that clip the values that would otherwise overflow to the maximum or minimum limit of the data range.

Some of the general purpose DSP library module functions accumulate a series of values before producing the final result. Examples of these accumulations could include the vector dot product calculation, the FIR filter, the sum of squared values and even the FFT transform. All of these functions, with the exception of the FFT, include a parameter that controls the output scaling, i.e., additional amount of right shift applied when the result is converted to a Q15 value. The FFT results are automatically scaled down by $2^{\log_2(N)}$.

3.1.4 Array Alignment and Length Restrictions

For the sake of efficiency, most functions require that array pointer arguments are aligned on 4-byte boundaries. Arrays of the *int16* data type declared in C will be correctly aligned. Furthermore, there are often restrictions on the number of elements that a function can operate on. Typically the number of elements must be a multiple of a small integer (e.g., four or eight), and must be larger than, or equal to, a specified minimum. Note that to improve performance, the functions do not verify the validity of their input parameters. Supplying incorrect parameters may lead to unpredictable results.

32-Bit Language Tools Libraries

3.2 VECTOR MATH FUNCTIONS

mips_vec_abs16

Description: Computes the absolute value of each element of *indata* and stores it to *outdata*. The number of samples to be processed is given by the parameter *N*.

Mathematically,

$$outdata[n] = abs(indata[N])$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_abs16
(
    int16 *outdata,
    int16 *indata,
    int N
);
```

Argument: *outdata:* Output array of 16-bit fixed-point elements in Q15 format.

indata: Input array with 16-bit fixed-point elements in Q15 format.

N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_abs32

Description: Computes the absolute value of each element of *indata* and stores it to *outdata*. The number of samples to be processed is given by the parameter *N*.

Mathematically,

$$outdata[n] = abs(indata[N])$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_abs32
(
    int32 *outdata,
    int32 *indata,
    int N
);
```

Argument: *outdata:* Output array of 32-bit fixed-point elements in Q31 format.

indata: Input array with 32-bit fixed-point elements in Q31 format.

N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_add16

Description: Adds each element of *indata1* to the corresponding element of *indata2*. The number of samples to be processed is given by the parameter *N*.

Mathematically,

$$outdata[n] = indata1[n] + indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_add16
(
    int16 *outdata,
    int16 *indata1,
    int16 *indata2,
    int N
);
```

Argument:

outdata: Output array of 16-bit fixed-point elements in Q15 format.

indata1: First input array with 16-bit fixed-point elements in Q15 format.

indata2: Second input array with 16-bit fixed-point elements in Q15 format.

N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata*, *indata1*, and *indata2* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_add32

Description: Adds each element of *indata1* to the corresponding element of *indata2*. The number of samples to be processed is given by the parameter *N*.

Mathematically,

$$outdata[n] = indata1[n] + indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_add32
(
    int32 *outdata,
    int32 *indata1,
    int32 *indata2,
    int N
);
```

Argument:

outdata: Output array of 32-bit fixed-point elements in Q31 format.

indata1: First input array with 32-bit fixed-point elements in Q31 format.

indata2: Second input array with 32-bit fixed-point elements in Q31 format.

N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata*, *indata1*, and *indata2* must be aligned on 4-byte boundaries.
- *N* must be larger than, or equal to, 4, and a multiple of 4.

32-Bit Language Tools Libraries

mips_vec_addc16

Description: Adds the Q15 constant *c* to all elements of *indata*. The number of samples to be processed is given by the parameter *N*.

Mathematically,

$$outdata[n] = indata[n] + c$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_addc16
(
    int16 *outdata,
    int16 *indata,
    int16 c,
    int N
);
```

Argument:

outdata: Output array of 16-bit fixed-point elements in Q15 format.
indata: Input array with 16-bit fixed-point elements in Q15 format.
c: Constant added to all elements of the vector.
N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_addc32

Description: Adds the Q31 constant *c* to all elements of *indata*. The number of samples to be processed is given by the parameter *N*.

Mathematically,

$$outdata[n] = indata[n] + c$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_addc32
(
    int32 *outdata,
    int32 *indata,
    int32 c,
    int N
);
```

Argument:

outdata: Output array of 32-bit fixed-point elements in Q31 format.
indata: Input array with 32-bit fixed-point elements in Q31 format.
c: Constant added to all elements of the vector.
N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_dotp16

Description: Computes the dot product of the Q15 vectors *indata1* and *indata2*. The number of samples to be processed is given by the parameter *N*. The *scale* parameter specifies the amount of right shift applied to the final result. Mathematically,

$$result = \frac{1}{2^{scale}} \sum_{n=0}^{N-1} indata1[n] \times indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
int16
mips_vec_dotp16
(
    int16 *indata1,
    int16 *indata2,
    int N,
    int scale
);
```

Argument: *indata1*: First input array with 16-bit fixed point elements in Q15 format.

indata2: Second input array.

N: Number of samples.

scale: Scaling factor: divide the result by 2^{scale} .

Return Value: Scaled result of the calculation in fractional Q15 format.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

32-Bit Language Tools Libraries

mips_vec_dotp32

Description: Computes the dot product of the Q31 vectors *indata1* and *indata2*. The number of samples to be processed is given by the parameter *N*. The *scale* parameter specifies the amount of right shift applied to the final result. Mathematically,

$$result = \frac{1}{2^{scale}} \sum_{n=0}^{N-1} indata1[n] \times indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
int32
mips_vec_dotp32
(
    int32 *indata1,
    int32 *indata2,
    int N,
    int scale
);
```

Argument:

indata1: First input array with 32-bit fixed point elements in Q31 format.
indata2: Second input array.
N: Number of samples.
scale: Scaling factor: divide the result by 2^{scale} .

Return Value: Scaled result of the calculation in fractional Q31 format.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_mul16

Description: Multiplies each Q15 element of *indata1* by the corresponding element of *indata2* and stores the results to *outdata*. The number of samples to be processed is given by the parameter *N*. Mathematically,

$$outdata[n] = indata1[n] \times indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_mull16
(
    int16 *outdata,
    int16 *indata1,
    int16 *indata2,
    int N
);
```

Argument:

outdata: Output array of 16-bit fixed-point elements in Q15 format.
indata1: First input array with 16-bit fixed-point elements in Q15 format.
indata2: Second input array.
N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata*, *indata1*, and *indata2* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_mul32

Description: Multiplies each Q31 element of *indata1* by the corresponding element of *indata2* and stores the results to *outdata*. The number of samples to be processed is given by the parameter *N*.
Mathematically,

$$outdata[n] = indata1[n] \times indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_mul32
(
    int32 *outdata,
    int32 *indata1,
    int32 *indata2,
    int N
);
```

Argument:

outdata: Output array of 32-bit fixed-point elements in Q31 format.
indata1: First input array with 32-bit fixed-point elements in Q31 format.
indata2: Second input array.
N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata*, *indata1*, and *indata2* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_mulc16

Description: Multiplies each Q15 element of *indata* by the Q15 constant *c* and stores the results to *outdata*. The number of samples to be processed is given by the parameter *N*.
Mathematically,

$$outdata[n] = indata1[n] \times c$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_mulc16
(
    int16 *outdata,
    int16 *indata,
    int16 c,
    int N
);
```

Argument:

outdata: Output array of 16-bit fixed-point elements in Q15 format.
indata: Input array with 16-bit fixed-point elements in Q15 format.
c: 16-bit fixed-point constant.
N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

32-Bit Language Tools Libraries

mips_vec_mulc32

Description: Multiplies each Q31 element of *indata* by the Q31 constant *c* and stores the results to *outdata*. The number of samples to be processed is given by the parameter *N*.

Mathematically,

$$outdata[n] = indata1[n] \times c$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_mulc32
(
    int32 *outdata,
    int32 *indata,
    int32 c,
    int N
);
```

Argument:

outdata: Output array of 32-bit fixed-point elements in Q31 format.
indata: Input array with 32-bit fixed-point elements in Q31 format.
c: 32-bit fixed-point constant.
N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata* and *indata* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_sub16

Description: Subtracts each element of *indata2* from the corresponding element of *indata1*. The number of samples to be processed is given by the parameter *N*.
Mathematically,

$$outdata[n] = indata1[n] - indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_sub16
(
    int16 *outdata,
    int16 *indata1,
    int16 *indata2,
    int N
);
```

Argument:

outdata: Output array of 16-bit fixed-point elements in Q15 format.
indata1: First input array with 16-bit fixed-point elements in Q15 format.
indata2: Second input array with 16-bit fixed-point elements in Q15 format.
N: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata*, *indata1*, and *indata2* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_sub32

Description: Subtracts each element of *indata2* from the corresponding element of *indata1*. The number of samples to be processed is given by the parameter *N*. Mathematically,

$$outdata[n] = indata1[n] - indata2[n]$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_vec_sub32
(
    int32 *outdata,
    int32 *indata1,
    int32 *indata2,
    int N
);
```

Argument:

- outdata*: Output array of 32-bit fixed-point elements in Q31 format.
- indata1*: First input array with 32-bit fixed-point elements in Q31 format.
- indata2*: Second input array with 32-bit fixed-point elements in Q31 format.
- N*: Number of samples.

Return Value: None.

Remarks:

- The pointers *outdata*, *indata1*, and *indata2* must be aligned on 4-byte boundaries.
- *N* must be larger than or equal to 4 and a multiple of 4.

mips_vec_sum_squares16

Description: Computes the sum of squared values of all elements of *indata*. The number of samples to be processed is given by the parameter *N*. The *scale* parameter specifies the amount of right shift applied to the final result. Mathematically,

$$result = \frac{1}{2^{scale}} \sum_{n=0}^{N-1} indata[n]^2$$

Include: dsplib_dsp.h

Prototype:

```
int16
mips_vec_sum_squares16
(
    int16 *indata,
    int N,
    int scale
);
```

Argument:

- indata*: Input array with 16-bit fixed-point elements in Q15 format
- N*: Number of samples
- scale*: Scaling factor: divide the result by 2^{scale} .

Return Value: Scaled result of the calculation in fractional Q15 format.

Remarks:

- The pointer *indata* must be aligned on a 4-byte boundary.
- *N* must be larger than or equal to 4 and a multiple of 4.

32-Bit Language Tools Libraries

mips_vec_sum_squares32

Description: Computes the sum of squared values of all elements of *indata*. The number of samples to be processed is given by the parameter *N*. The *scale* parameter specifies the amount of right shift applied to the final result. Mathematically,

$$result = \frac{1}{2^{scale}} \sum_{n=0}^{N-1} indata[n]^2$$

Include: dsplib_dsp.h

Prototype:

```
int32
mips_vec_sum_squares32
(
    int32 *indata,
    int N,
    int scale
);
```

Argument: *indata*: Input array with 32-bit fixed-point elements in Q31 format.

N: Number of samples.

scale: Scaling factor: divide the result by 2^{scale} .

Return Value: Scaled result of the calculation in fractional Q31 format.

Remarks:

- The pointer *indata* must be aligned on a 4-byte boundary.
- *N* must be larger than or equal to 4 and a multiple of 4.

3.3 FILTERING FUNCTIONS

mips_fir16

Description: Computes a finite impulse response (FIR) filter with coefficients specified in *coeffs2x* over the input data samples in *indata*. The function updates the *delayline*, which is used to initialize the filter the next time *mips_fir16()* is called. The number of samples to be processed is given by the parameter *N* and the number of filter coefficients is given by *K*. The *scale* parameter specifies the amount of right shift applied to the final result. Mathematically,

$$output[n] = \frac{1}{2^{scale}} \sum_{k=0}^{K-1} indata[n-k] \times coeffs[k]$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_fir16
(
    int16 *outdata,
    int16 *indata,
    int16 *coeffs2x,
    int16 *delayline,
    int N,
    int K,
    int scale
);
```

Argument:

- outdata*: Output array with 16-bit fixed-point elements in Q15 format.
- indata*: Input array with 16-bit fixed-point elements in Q15 format.
- coeffs2x*: Array of 2*K* 16-bit fixed-point coefficients prepared by *mips_fir16_setup()*.
- delayline*: Delay line array holding the last *K* input samples.
- N*: Number of samples.
- K*: Number of coefficients (filter taps).
- scale*: Scaling factor: divide the result by 2^{scale}.

Return Value: None.

Remarks:

- The pointers *outdata*, *indata*, *coeffs2x*, and *delayline* must be aligned on a 4-byte boundary.
- *K* must be larger than or equal to 4 and a multiple of 4.

Notes: The *coeffs2x* array is twice the size of the original coefficient array, *coeffs*. The function *mips_fir16_setup()* takes the original coefficient array *coeffs* and rearranges the coefficients into the *coeffs2x* array to enable more efficient processing. All elements of the *delayline* array must be initialized to zero before the first call to *mips_fir16()*. Both *delayline* and *coeffs2x* have formats that are implementation-dependent and their contents should not be changed directly.

32-Bit Language Tools Libraries

mips_fir16 (Continued)

Example:

```
int i;
int K = 8;
int N = 32;

int16 coeffs[K];
int16 coeffs2x[2*K];
int16 delayline[K];

int16 indata[N];
int16 outdata[N];

for (i = 0; i < K; i++)
    delayline[i] = 0;

// load coefficients into coeffs here
...

mips_fir16_setup(coeffs2x, coeffs, K);

while (true)
{
    // load input data into indata
    ...

    mips_fir16(outdata, indata, coeffs2x, delayline, N,
K, 3);

    // do something with outdata
    ...
}
```

mips_fir16_setup

Description: Rearranges the coefficients from the input array, *coeffs*, into the output array *coeffs2x*, which is used by the *mips_fir16()* function. The number of coefficients to process is given by the parameter *K*.

Include: dsplib_dsp.h

Prototype:

```
void
mips_fir16_setup
(
    int16 *coeffs2x,
    int16 *coeffs,
    int K
);
```

Argument:

<i>coeffs2x</i> :	Output array holding 2 <i>K</i> coefficients rearranged for <i>mips_fir16()</i> .
<i>coeffs</i> :	Input array holding <i>K</i> 16-bit fixed-point coefficients in Q15 format.
<i>K</i> :	Number of coefficients.

Return Value: None.

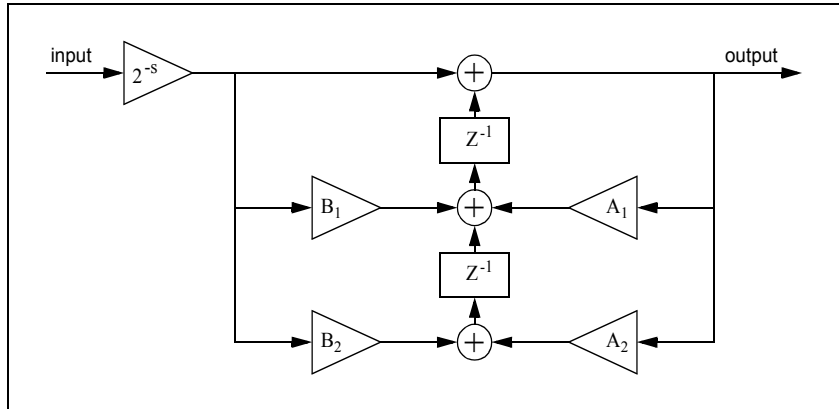
Remarks: None.

Note: This function is implemented in C.

mips_iir16

Description: Computes a single-sample infinite impulse response (IIR) filter with coefficients specified in *coeffs*. The number of biquad sections composing the filter is given by the parameter *B*. The *scale* parameter specifies the amount of right shift applied to the input value of each biquad. Each biquad section is specified by four coefficients – A_1 , A_2 , B_1 , and B_2 – and has two state variables stored inside *delayline*.^{°±} The output of each biquad section becomes input to the next one. The output of the final section is returned as result of the *mips_iir16()* function.

The operations performed for each biquad section are illustrated below:



Include: dsplib_dsp.h

```

Prototype: int16
mips_iir16
(
    int16 in,
    int16 *coeffs,
    int16 *delayline,
    int B,
    int scale
);

```

Argument:

- in*: Input value in Q15 format.
- coeffs*: Array of $4B$ 16-bit fixed-point coefficients prepared by *mips_iir16_setup()*.
- delayline*: Delay line array holding $2B$ state 16-bit state variables.
- B*: Number of biquad sections.
- scale*: Scaling factor: divide the input to each biquad by 2^{scale} .

Return Value: IIR filter output value in fractional Q15 format.

Remarks:

- The pointers *coeffs* and *delayline* must be aligned on a 4-byte boundary.
- *B* must be larger than or equal to 2 and a multiple of 2.

Notes: The *coeffs* array contains four coefficients for each biquad. The coefficients are conveniently specified in an array of *biquad16* structures, which is converted to the appropriate internal representation by the *mips_iir16_setup()* function. All elements of the *delayline* array must be initialized to zero before the first call to *mips_iir16()*. Both *delayline* and *coeffs* have formats that are implementation-dependent and their contents should not be changed directly.

32-Bit Language Tools Libraries

mips_iir16 (Continued)

Example:

```
int i;
int B = 4;

biquad16 bq[B];
int16 coeffs[4*B];
int16 delayline[2*B];

int16 indata, outdata;

for (i = 0; i < 2*B; i++)
    delayline[i] = 0;

// load coefficients into bq here
...

mips_iir16_setup(coeffs, bq, B);

while (true)
{
    // get input data value into indata
    ...

    outdata = mips_iir16(indata, coeffs, delayline, B,
2);

    // do something with outdata
    ...
}
```

mips_iir16_setup

Description: Rearranges the coefficients from the input array, *bq*, into the output array *coeffs*, which is used by the *mips_iir16()* function. The number of biquad sections to process is given by the parameter *B*.

Include: dsplib_dsp.h

Prototype:

```
void
mips_iir16_setup
(
    int16 *coeffs,
    biquad16 *bq,
    int B
);
```

Argument:

<i>coeffs</i> :	Output array holding $4B$ coefficients rearranged for <i>mips_iir16()</i> .
<i>bq</i> :	Input array holding Q15 coefficients for B biquad sections.
<i>B</i> :	Number of biquad sections.

Return Value: None.

Remarks: None.

Notes: This function is implemented in C.

mips_lms16

Description: Computes a Least Mean Squares (LMS) adaptive filter and updates its coefficients. The new coefficients are computed using the *error* between the last filter output and the reference signal *ref*. The function takes one input sample *in* and computes one output sample. The parameter *mu* controls the adaptation rate of the filter.

Include: dsplib_dsp.h

Prototype:

```
int16
mips_lms16
(
    int16 in,
    int16 ref,
    int16 *coeffs,
    int16 *delayline,
    int16 *error,
    int16 K,
    int mu
);
```

Argument:

<i>in</i> :	Input value in Q15 format.
<i>ref</i> :	Desired (reference) value in Q15 format.
<i>coeffs</i> :	Input/output array of 16-bit fixed-point coefficients.
<i>delayline</i> :	Delay line array holding the last <i>K</i> input samples.
<i>error</i> :	Input/output value indicating the difference between the filter output and the reference value.
<i>K</i> :	Number of coefficients (filter taps).
<i>mu</i> :	Adaptation rate in Q15 format.

Return Value: LMS filter output value in Q15 format.

Remarks:

- The pointers *coeffs* and *delayline* must be aligned on a 4-byte boundary.
- *K* must be larger than or equal to 4 and a multiple of 2.

Notes: The order of the elements of the *coeffs* and *delayline* arrays is implementation dependent. The *delayline* array must be initialized to zero before the first call to *mips_lms16()*.

32-Bit Language Tools Libraries

3.4 FREQUENCY DOMAIN TRANSFORM FUNCTIONS

mips_fft16

Description: Computes the complex fast Fourier transform (FFT) of the input sequence *din*. The number of samples to be processed is specified by the parameter *log2N*: $N = 2^{\log 2N}$. The *fft* array holds complex coefficients needed by the FFT algorithm. The *scratch* hold intermediate data; its contents are destroyed on each call to *mips_fft16()*.
Mathematically,

$$output[n] = \frac{1}{2^{\log 2N}} \sum_{k=0}^{N-1} din[k] \times e^{-j\frac{2\pi kn}{N}}$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_fft16
(
    int16c *dout,
    int16c *din,
    int16c *fft,
    int16c *scratch,
    int log2N
);
```

Argument:

dout: Output array with 16-bit complex fixed-point elements in Q15 format.

din: Input array with 16-bit complex fixed-point elements in Q15 format.

fft: Input array with 16-bit complex fixed-point twiddle factors in Q15 format.

scratch: Intermediate results array holding 16-bit complex fixed-point data.

log2N: Logarithm base 2 of the number of samples: $N = 2^{\log 2N}$.

Return Value: None.

Remarks:

- The pointers *dout*, *din*, *fft*, and *scratch* must be aligned on 4-byte boundaries.
- *log2N* must be larger than or equal to 3.

Notes: The *scratch* array must be large enough to hold N 16-bit complex data samples having 16-bit real part and 16-bit imaginary part. Copying *fft* to RAM prior to calling this function can be used to improve performance.

mips_fft16 (Continued)

Example:

```
#include "fftc.h" // pre-computed coefficients
int log2N = 6; // log2(64) = 6
int N = 1 << log2N; // N = 2^6 = 64
int16c din[N];
int16c dout[N];
int16c scratch[N];
#define fftc fft16c64 // from fftc.h, for N = 64
while (true)
{
    // load complex input data into din
    ...
    mips_fft16(dout, din, fftc, scratch, log2N);
    // do something with dout
    ...
}
```

mips_fft16_setup – Function Deprecated

Description: Calculates the twiddle factors need to compute an FFT of size N . The twiddle factors are used by the *mips_fft16()* function. The number of samples to be processed is specified by the parameter *log2N*: $N = 2^{\log_2 N}$.

Include: dsplib_dsp.h

Prototype:

```
void
mips_fft16_setup
(
    int16c *twiddles,
    int log2N
);
```

Argument: *twiddles*: Output array containing N 16-bit complex twiddle factors.

log2N: Logarithm base 2 of the number of samples: $N = 2^{\log_2 N}$.

Return Value: None.

Remarks: This function requires floating-point support.

Notes: This function is implemented in C.

32-Bit Language Tools Libraries

mips_fft32

Description: Computes the complex Fast Fourier Transform (FFT) of the input sequence *din*. The number of samples to be processed is specified by the parameter *log2N*: $N = 2^{\log 2N}$. The *fftc* array holds complex coefficients needed by the FFT algorithm. The *scratch* hold intermediate data; its contents are destroyed on each call to *mips_fft32()*.
Mathematically,

$$output[n] = \frac{1}{2^{\log 2N}} \sum_{k=0}^{N-1} din[k] \times e^{-j\frac{2\pi kn}{N}}$$

Include: dsplib_dsp.h

Prototype:

```
void
mips_fft32
(
    int32c *dout,
    int32c *din,
    int32c *fftc,
    int132 *scratch,
    int log2N
);
```

Argument:

dout: Output array with 32-bit complex fixed-point elements in Q31 format.

din: Input array with 32-bit complex fixed-point elements in Q31 format.

fftc: Input array with 32-bit complex fixed-point twiddle factors in Q31 format.

scratch: Intermediate results array holding 32-bit complex fixed-point data.

log2N: Logarithm base 2 of the number of samples: $N = 2^{\log 2N}$.

Return Value: None.

Remarks:

- The pointers *dout*, *din*, *fftc*, and *scratch* must be aligned on 4-byte boundaries.
- *log2N* must be larger than or equal to 3.

Notes: The *scratch* array must be large enough to hold N 32-bit complex data samples having 32-bit real part and 32-bit imaginary part. Copying *fftc* to RAM prior to calling this function can be used to improve performance.

Example:

```
#include "fftc.h" // pre-computed coefficients
int log2N = 6; // log2(64) = 6
int N = 1 << log2N; // N = 2^6 = 64
int32c din[N];
int32c dout[N];
int32c scratch[N];
#define fftc fft32c64 // from fftc.h, for N = 64
while (true)
{
    // load complex input data into din
    ...
    mips_fft32(dout, din, fftc, scratch, log2N);
    // do something with dout
    ...
}
```

mips_fft32_setup – Function Deprecated

Description: Calculates the twiddle factors need to compute an FFT of size N . The twiddle factors are used by the `mips_fft32()` function. The number of samples to be processed is specified by the parameter $\log_2 N$: $N = 2^{\log_2 N}$.

Include: `dsplib_dsp.h`

Prototype:

```
void  
mips_fft32_setup  
(  
    int32c *twiddles,  
    int log2N  
);
```

Argument: *twiddles*: Output array containing N 32-bit complex twiddle factors.

log2N: Logarithm base 2 of the number of samples: $N = 2^{\log_2 N}$.

Return Value: None.

Remarks: This function requires floating-point support.

Notes: This function is implemented in C.

32-Bit Language Tools Libraries

3.5 VIDEO PROCESSING FUNCTIONS

mips_h264_iqt

Description: Combined inverse quantization and inverse transform function. The input DCT coefficients are inverse quantized by multiplying them with corresponding elements of the inverse quantization matrix. The results are transformed by a 4x4-element integer inverse DCT as specified in the H.264 video compression standard.

Include: dsplib_video.h

Prototype:

```
void
mips_h264_iqt
(
    uint8 b[4][4],
    int16 c[4][4],
    int16 iq[4][4]
);
```

Argument:

b: Output 4x4-pixel array in 8-bit unsigned integer format.

c: Input 4x4-element array of DCT coefficients in signed 16-bit integer format.

iq: Inverse quantization matrix in signed 16-bit integer format.

Return Value: None.

Remarks: The pointers *b*, *c*, and *iq* must be aligned on 4-byte boundaries.

Notes: The *mips_iqt_setup()* function can be used to initialize the *iq* array.

Example:

```
uint8 b[4][4]
int16 dct_data[4][4];
int16 iq_matrix[4][4];

// quantization parameter
int QP = 28;

// initialize the inverse quantization matrix
mips_h264_iqt_setup(iq_matrix, mips_h264_iq_coeffs, QP);

...

// load DCT data into dct_data
...

mips_h264_iqt(b, dct_data, iq_matrix);
```

mips_h264_iqt_setup

Description: Computes the inverse quantization matrix used by the *mips_iqt()* function. The default inverse quantization coefficient array as specified by the H.264 video compression standard is provided as *mips_h264_iq_coeffs* and can be used in place of the *q* parameter.

Include: dsplib_video.h

Prototype:

```
void
mips_h264_iqt_setup
(
    int16 iq[4][4],
    int16 q[6][4][4],
    int16 qp
);
```

Argument:

- iq*: Output 4x4-element inverse quantization matrix in signed 16-bit integer format.
- q*: Input 6x4x4-element inverse quantization coefficient array in signed 16-bit integer format.
- qp*: Quantization parameter.

Return Value: None.

Remarks: None.

Notes: This function is implemented in C.

32-Bit Language Tools Libraries

mips_h264_mc_luma

Description: This function computes 1/4-pixel motion compensation for luma blocks as specified by the H.264 video compression standard. The function performs all necessary interpolations depending on the fractional offset of the desired block as specified by the *dx* and *dy* input parameters. Note, however, that there is no special handling of cases that cross the picture edge. It is expected that the image will be enlarged by four pixels in each direction and the pixels along the edges of the image will be replicated to the expanded borders.

Include: dsplib_video.h

Prototype:

```
void
mips_h264_mc_luma
(
    uint8 b[4][4],
    uint8 *src,
    int ystride,
    int dx,
    int dy
);
```

Argument: *b* Output 4x4-pixel array in 8-bit unsigned integer format.

src Pointer to the top-left pixel of the source image block.

ystride Vertical stride, i.e., distance in bytes between corresponding pixels on adjacent rows.

dx, dy Fractional pixel offsets multiplied by four, e.g., *dx = 1* specifies a 1/4-pixel offset.

Return Value: None.

Remarks: The offsets *dx* and *dy* must have values between 0 and 3 inclusive.

Example:

```
uint8 b[4][4];
uint8 luma[HEIGHT][WIDTH];

int ystride = WIDTH;

...

// obtain 1/4-pixel coordinates of desired block
int x4 = ...;
int y4 = ...;

// compute the integer and fractional parts
int x = x4 >> 2;
int y = y4 >> 2;
int dx4 = x4 & 0x03;
int dy4 = y4 & 0x03;

mips_h264_mc_luma(b, &luma[y][x], ystride, dx4, dy4);
```

3.5.1 MIPS Technologies Inc.'s DSP Library Notices:

Please note that the following notices apply to the MIPS Technologies Inc. DSP Library. Copyright © 2003, 2005, 2006, 2007 MIPS Technologies, Inc. All rights reserved.

Unpublished rights (if any) reserved under the copyright laws of the United States of America and other countries.

This document contains information that is proprietary to MIPS Technologies, Inc. (“MIPS Technologies”). Any copying, reproducing, modifying or use of this information (in whole or in part) that is not expressly permitted in writing by MIPS Technologies or an authorized third party is strictly prohibited. At a minimum, this information is protected under unfair competition and copyright laws. Violations thereof may result in criminal penalties and fines.

Any document provided in source format (i.e., in a modifiable form such as in FrameMaker or Microsoft Word format) is subject to use and distribution restrictions that are independent of and supplemental to any and all confidentiality restrictions. UNDER NO CIRCUMSTANCES MAY A DOCUMENT PROVIDED IN SOURCE FORMAT BE DISTRIBUTED TO A THIRD PARTY IN SOURCE FORMAT WITHOUT THE EXPRESS WRITTEN PERMISSION OF MIPS TECHNOLOGIES, INC.

MIPS Technologies reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information, or of any error or omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Except as expressly provided in any written license agreement from MIPS Technologies or an authorized third party, the furnishing of this document does not give recipient any license to any intellectual property rights, including any patent rights, that cover the information in this document.

The information contained in this document shall not be exported, reexported, transferred, or released, directly or indirectly, in violation of the law of any country or international law, regulation, treaty, Executive Order, statute, amendments or supplements thereto. Should a conflict arise regarding the export, reexport, transfer, or release of the information contained in this document, the laws of the United States of America shall be the governing law.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government (“Government”), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or an authorized third party.

MIPS, MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS-3D, MIPS16, MIPS16e, MIPS32, MIPS64, MIPS-Based, MIPSsim, MIPSpro, MIPS Technologies logo, MIPS RISC CERTIFIED POWER logo, MIPS-VERIFIED, 4K, 4Kc, 4Km, 4Kp, 4KE, 4KEc, 4KEm, 4KEp, 4KS, 4KSc, 4KSd, M4K, 5K, 5Kc, 5Kf, 20K, 20Kc, 24K, 24Kc, 24Kf, 24KE, 24KEc, 24KEf, 25Kf, 34K, 34Kc, 34Kf, R3000, R4000, R5000, ASMACRO, Atlas, “At the core of the user experience.”, BusBridge, CorExtend, CoreFPGA, CoreLV, EC, JALGO, Malta, MDMX, MGB, PDtrace, the Pipeline, Pro Series, QuickMIPS, SEAD, SEAD-2, SmartMIPS, SOC-it, and YAMON are trademarks or registered trademarks of MIPS Technologies, Inc. in the United States and other countries.

All other trademarks referred to herein are the property of their respective owners.

32-Bit Language Tools Libraries

NOTES:

Chapter 4. PIC32 Debug-Support Library

4.1 OVERVIEW

This library supports both the Application Input/Output debugging feature and the PIC32 Starter Kit Debug I/O feature.

4.1.1 Application Input/Output with `printf()` and `scanf()`

Many PIC32 devices support the APPIN/APPOUT debugging feature. This PIC32 feature allows the PIC32 application to write text or data to an MPLAB IDE window, invoked from the Tools menu, without halting the target device. Similarly, you may use the display window to send text or data back to the target PIC32 device. This feature requires an MPLAB REAL ICE emulator or MPLAB ICD 3 debugger.

4.1.2 Starter Kit Debug Print Mechanism with `DBPRINTF()` and `DBSCANF()`

A similar target input/output feature is available for the PIC32 Starter Kit (DM320001) featuring the PIC32MX360F512L MCU and the PIC32 USB Starter Board (DM320003) featuring the PIC32MX460F512L MCU.

The print output functionality is routed to the Output window on the MPLAB PIC32MX tab of the interface window.

For input using the Starter Kit, MPLAB IDE uses a TargetIN window. To send text to the target, type your text into the *Enter Information to be Sent to Target* box, and click **Send**.

4.2 CONFIGURING DEBUG INPUT/OUTPUT FOR THE TARGET AND TOOL

The debug-support library, for both the APPIN/APPOUT mechanism and the Starter Kit mechanism, works by providing alternate I/O helper functions: `_mon_write()`, `_mon_putc()`, and `_mon_getc()`, as described in **Section 2.13.2 “Customizing STDIO”**. These alternate functions use the APPIN/APPOUT or Starter Kit mechanism as requested in the project. These debug-support function implementations override the default helper I/O function implementations.

You can choose which implementation to use by defining a preprocessor symbol. To choose the APPIN/APPOUT implementation, pass the `-mappio-debug` option to `pic32-gcc.exe`. To choose the PIC32 Starter Kit implementation, pass `-DPIC32_STARTER_KIT` to the compiler shell. Also use `#include <p32xxxx.h>` to include the generic header file in your source code.

With one of the above options passed to the compiler and the `sys/appio.h` include file added to your source code, the debugging-support library provides alternate I/O helper functions to the linker. These alternate I/O helper functions redirect `stdin` and `stdout` to the appropriate debugging mechanism. Standard I/O functions now use the selected mechanism.

32-Bit Language Tools Libraries

4.3 <SYS/APPIO.H> PIC32 DEBUGGING SUPPORT

The `sys/appio.h` header file contains conditional-compilation directives that cause the compiler to pull in the correct aliased functions. In addition, it provides macros that simplify enabling and disabling the debugging feature.

DBINIT()

Description: Selects the appropriate mechanism (APPIN/APPOUT or Starter Kit) and initializes buffering as appropriate. When the `-mappio-debug` option is passed to the compiler, the `init` function initializes the debug library for APPIN/APPOUT. When the `-DPIC32_STARTER_KIT` option is passed to the compiler, the `init` function initializes the debug library for the PIC32 Starter Kit.

The APPIN/APPOUT mechanism disables `stdin/stdout` buffering while the PIC32 Starter Kit mechanism uses default line buffering.

Include: `<sys/appio.h>`

Remarks: Behaves as `((void)0)` when APPIO debugging or Starter Kit I/O debugging is not enabled.

DBPRINTF()

Description: Calls `printf()` but is enabled only with the `-mappio-debug` or `-DPIC32_STARTER_KIT` option. When one of these options is not specified on the compiler command line, `DBPRINTF()` behaves as `((void)0)` and `printf` is not called.

Include: `<sys/appio.h>`

Remarks: Behaves as `((void)0)` when APPIO debugging or Starter Kit I/O debugging is not enabled via the `-mappio-debug` or `-DPIC32_STARTER_KIT` option. Use this macro to insert messages that should print only when debugging.

DBSCANF()

Description: Calls `scanf()`. Available for only the APPIN/APPOUT mechanism, not for the PIC32 Starter Kit mechanism.

Include: `<sys/appio.h>`

Remarks: Behaves as `((void)0)` when APPIN/APPOUT debugging is not enabled via the `-mappio-debug` or `-DPIC32_STARTER_KIT` option. Use this macro to read formatted input that should read only when debugging. PIC32 Starter Kit users should consider `DBGGETS` instead.

DBGGETC(canblock)

Description: Get a single `char` from the input mechanism.

Include: `<sys/appio.h>`

Remarks: Behaves as `((void)0)` when APPIN/APPOUT debugging or Starter Kit I/O debugging is not enabled via the `-mappio-debug` or `-DPIC32_STARTER_KIT` option.

DBGETWORD(int canblock)

Description: Read a 32-bit word from the APPIN mechanism. Available only for the APPIN/APPOUT mechanism, not for the PIC32 Starter Kit mechanism.

Include: <sys/appio.h>

Remarks: Behaves as ((void)0) when APPIN/APPOUT debugging is not enabled via the `-mappio-debug` or `-DPIC32_STARTER_KIT` option.

DBPUTC(char c)

Description: Writes a single `char` to the output mechanism

Include: <sys/appio.h>

Remarks: Behaves as ((void)0) when APPIN/APPOUT debugging or Starter Kit I/O debugging is not enabled via the `-mappio-debug` or `-DPIC32_STARTER_KIT` option.

DBPUTWORD(int w)

Description: Writes a 32-bit integer word to the APPOUT mechanism. Available only for the APPIN/APPOUT mechanism, not for the PIC32 Starter Kit mechanism.

Include: <sys/appio.h>

Remarks: Behaves as ((void)0) when APPIN/APPOUT is not enabled via the `-mappio-debug` or `-DPIC32_STARTER_KIT` option.

Example Code:

```
#include <p32xxxx.h>
int main (void)
{
    int num;
    char buf[256] = {0};
    DBINIT();

    while(1)
    {
        DBPRINTF ("Hello there!\n");
        DBPRINTF ("Enter a string\n");
#ifdef __APPIO_DEBUG
        DBSCANF ("%s", &buf[0]);
#elif defined(PIC32_STARTER_KIT)
        DBGETS (&buf[0],128);
#endif
        DBPRINTF ("Entered \"%s\"\n\n", &buf[0]);

        printf ("Prints to UART2 by default or APPOUT when
enabled\n");
    }
    return 0;
}
```

32-Bit Language Tools Libraries

NOTES:

Appendix A. ASCII Character Set

TABLE A-1: ASCII CHARACTER SET

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

Least
Significant
Character

32-Bit Language Tools Libraries

NOTES:

Appendix B. Types, Constants, Functions and Macros

- _IOFBF
- _IOLBF
- _IONBF
- _mon_getc
- _mon_putc
- abort
- abs
- acos
- acosf
- asctime
- asin
- asinf
- asinh
- asprintf
- assert
- atan
- atan2
- atan2f
- atanf
- atanh
- atexit
- atof
- atoi
- atol
- atoll
- bsearch
- BUFSIZ
- calloc
- cbrt
- ceil
- ceilf
- CHAR_BIT
- CHAR_MAX
- CHAR_MIN
- clearerr
- clock
- clock_t
- CLOCKS_PER_SEC
- close
- copysign
- cos
- cosf
- cosh
- coshf
- ctime
- DBL_DIG
- DBL_EPSILON
- DBL_MANT_DIG
- DBL_MAX
- DBL_MAX_10_EXP
- DBL_MAX_EXP
- DBL_MIN
- DBL_MIN_10_EXP
- DBL_MIN_EXP
- difftime
- div
- div_t
- drem
- EBADF
- EDOM
- EINVAL
- ENOMEM
- EOF
- ERANGE
- errno
- exit
- EXIT_FAILURE
- EXIT_SUCCESS
- exp
- expf
- expm1
- fabs
- fabsf
- fclose
- feof
- ferror
- fflush
- ffs
- ffsf
- fgetc
- fgetpos
- fgets
- FILE
- FILENAME_MAX
- finite
- floor
- floorf
- FLT_DIG
- FLT_EPSILON
- FLT_MANT_DIG
- FLT_MAX
- FLT_MAX_10_EXP
- FLT_MAX_EXP
- FLT_MIN
- FLT_MIN_10_EXP
- FLT_MIN_EXP
- FLT_RADIX
- FLT_ROUNDS
- fmod
- fmodf
- fopen
- FOPEN_MAX
- fpos_t
- fprintf
- fputc
- fputs
- fread
- free
- freopen
- frexp
- frexpf
- fscanf
- fseek
- fsetpos
- fflush
- ftell
- fwrite
- getc
- getchar
- getenv

32-Bit Language Tools Libraries

- gets
- gettimeofday (User Provided)
- gmtime
- HUGE_VAL
- hypot
- INT_MAX
- INT_MIN
- isalnum
- isalpha
- isascii
- iscntrl
- isdigit
- isgraph
- isinf
- islower
- isnan
- isprint
- ispunct
- isspace
- isupper
- isxdigit
- jmp_buf
- L_tmpnam
- labs
- LDBL_DIG
- LDBL_EPSILON
- LDBL_MANT_DIG
- LDBL_MAX
- LDBL_MAX_10_EXP
- LDBL_MAX_EXP
- LDBL_MIN
- LDBL_MIN_10_EXP
- LDBL_MIN_EXP
- ldexp
- ldexpf
- ldiv
- ldiv_t
- link
- llabs
- lldiv
- lldiv_t
- LLONG_MAX
- LLONG_MIN
- localtime
- log
- log10
- log10f
- log1p
- logb
- logf
- LONG_MAX
- LONG_MIN
- longjmp
- lseek
- malloc
- MB_CUR_MAX
- MB_LEN_MAX
- mblen
- mbstowcs
- mbtowc
- memchr
- memcmp
- memcpy
- memmove
- memset
- mktime
- modf
- modff
- NULL (stddef.h)
- offsetof
- open
- perror
- pow
- powf
- printf
- ptrdiff_t
- putc
- putchar
- puts
- qsort
- raise
- rand
- RAND_MAX
- read
- realloc
- remove
- rename
- rewind
- rint
- scanf
- SCHAR_MAX
- SCHAR_MIN
- SEEK_CUR
- SEEK_END
- SEEK_SET
- setbuf
- setjmp
- gettimeofday (User Provided)
- setvbuf
- SHRT_MAX
- SHRT_MIN
- sig_atomic_t
- SIG_DFL
- SIG_ERR
- SIG_IGN
- SIGABRT
- SIGFPE
- SIGILL
- SIGINT
- signal
- SIGSEGV
- SIGTERM
- sin
- sinf
- sinh
- sinhf
- size_t (stddef.h)
- size_t (stdio.h)
- size_t (string.h)
- size_t (time.h)
- snprintf
- sprintf
- sqrt
- sqrtf
- srand
- sscanf
- stderr
- stdin
- stdout
- strcasecmp
- strcat
- strchr
- strcmp
- strcoll
- strcpy
- strcspn
- strerror
- strftime
- strlen
- strncasecmp
- strncat

Types, Constants, Functions and Macros

- strcmp
- strncpy
- strpbrk
- strrchr
- strstr
- strtod
- strtouf
- strtok
- strtol
- strtoll
- strtoul
- strtoull
- struct timeval
- struct tm
- strxfrm
- system
- tan
- tanf
- tanh
- tanhf
- time
- time_t
- TMP_MAX
- tmpfile
- tmpnam
- tolower
- toupper
- UCHAR_MAX
- UINT_MAX
- ULLONG_MAX
- ULONG_MAX
- ungetc
- unlink
- USHRT_MAX
- va_arg
- va_end
- va_list
- va_start
- vfprintf
- vfscanf
- vprintf
- vscanf
- vsnprintf
- vsprintf
- vsscanf
- wchar_t
- wcstombs
- wctomb
- write

32-Bit Language Tools Libraries

NOTES:

Appendix C. 16-Bit DSP Wrapper Functions

C.1 INTRODUCTION

The PIC32 DSP wrapper functions are intended to help port existing 16-bit application software using dsPIC[®] DSP library functions to PIC32 with the least modifications in the software. The wrapper functions internally call the DSP library functions provided by MIPS Technologies. The wrapper functions are available for some of the functions supported by dsPIC DSP library.

Note: The DSP libraries from MIPS Technologies support a variety of signal processing functions that have applicability in speech compression, echo cancellation, noise cancellation, channel equalization, audio decoding, and many other DSP and media applications. It is always advisable for the new users to use MIPS Technologies DSP libraries.

C.2 PIC32 DSP WRAPPER FUNCTIONS LIST

These functions are supported by the DSP wrapper functions for PIC32 MCUs:

- VectorAdd16
- VectorAdd32
- VectorDotProduct16
- VectorDotProduct32
- VectorMultiply16
- VectorMultiply32
- VectorScale16
- VectorScale32
- VectorSubtract16
- VectorSubtract32
- VectorPower16
- VectorPower32
- FIR
- FFTComplex16
- TwidFactorInit16
- FFTComplex32
- TwidFactorInit32

32-Bit Language Tools Libraries

C.3 DIFFERENCES BETWEEN WRAPPER FUNCTIONS AND dsPIC® DSP LIBRARY

PIC32 DSP wrapper function names, input parameters and return parameters are maintained the same as that of dsPIC DSP library. However, these are some differences:

TABLE C-1: DIFFERENCES IN WRAPPER FUNCTIONS

PIC32 DSP Wrapper Function Name	FIR (int numSamps, short int* dstSamps, short int* srcSamps, FIRStruct* filter)	TwidFactorInit16 (int log2N, fractcomplex16* twidFactors, int conjFlag) TwidFactorInit32 (int log2N, fractcomplex32* twidFactors, int conjFlag)
Differences with Corresponding Function of dsPIC® DSP Library	Some of the parameters of the structure "FIRStruct" are not necessary for PIC32 library function. Hence, it is not necessary to initialize these parameters before the FIR function is called. These parameters are namely: filter->coeffsEnd, filter -> coeffsPage, filter->delay End, filter->delay	There is a provision in the "TwidFactorInit" function of dsPIC library, either to generate or not generate a complex conjugates of twiddles. It is controlled by flag "conjFlag". There is no such facility in the PIC32 DSP library. "TwidFactorInit16" and "TwidFactorInit32" in PIC32 do not generate a complex conjugate of twiddles. However, the parameter is kept in the function prototype of "TwidFactorInit" of PIC32 to make it compatible with dsPIC.
General Comments Regarding PIC32 DSP Library	Number of coefficients in filter (filter->numCoeffs) must be larger than or equal to 4 and multiple of 4.	-

- Note 1:** PIC32 supports both 16-bit and 32-bit vector math operations.
- 2:** The current version of PIC32 DSP wrapper functions does not support floating-point calculations.
- 3:** For all the vector math operations, the number of samples must be larger than or equal to 4 or multiple of 4.
- 4:** log2N must be larger than or equal to 3 for function "FFTComplex16" and "FFTComplex32".
- 5:** All the source and destination pointers used for math operations must be aligned on 4-byte boundaries.
- 6:** The include file for these DSP wrapper functions is mchp_dsp_wrapper.h.

Index

Symbols

^, Caret..... 50
 __FILE__ 14
 __LINE__ 14
 _IOFBF 35, 50
 _IOLBF 35, 50
 _IONBF 36, 50
 _mon_putc 38
 _NSETJMP 28
 -, Dash 50
 \f, Form Feed 17
 \n, Newline 17, 34, 40, 42, 45, 46, 48
 \r, Carriage Return 17
 \t, Horizontal Tab 17
 \v, Vertical Tab 17
 #if 25
 #include 14
 %, Percent 47, 49, 50, 80

Numerics

0x 17, 46, 65, 66

A

Abnormal Termination Signal..... 30
 abort..... 57
 abs 57
 Absolute Value
 Double Floating Point 87
 Integer..... 57
 Long Integer..... 60
 Single Floating Point 87
 Absolute Value Function
 abs 57
 fabs 87
 fabsf 87
 labs 60
 Access Mode
 Binary..... 41
 Text..... 41
 acos 82
 acosf 82
 Allocate Memory 61
 calloc..... 59
 Free 60
 realloc 63
 Alphabetic Character
 Defined 15
 Test for..... 15
 Alphanumeric Character
 Defined 15
 Test for..... 15
 AM/PM 80

Append..... 70, 73
 arccosine
 Double Floating Point..... 82
 Single Floating Point 82
 arcsine
 Double Floating Point..... 83
 Single Floating Point 83
 arctangent
 Double Floating Point..... 83
 Single Floating Point 84
 arctangent of y/x
 Double Floating Point..... 83
 Single Floating Point 84
 Argument List 32, 52, 53, 54
 Array Alignment and Length Restrictions 101
 ASCII Character Set..... 129
 asctime 78
 asin..... 83
 asinf..... 83
 asinh..... 83
 asprintf 38
 assert 14
 assert.h 14
 assert 14
 Assignment Suppression 49
 Asterisk 46, 49
 atan 83
 atan2 83
 atan2f 84
 atanf 84
 atanh 84
 atexit..... 57, 60
 atof 57
 atoi 58
 atol 58
 atoll..... 58

B

Base 65, 66
 10 21, 22, 23, 24, 90, 91
 2 23
 e 90, 91
 FLT_RADIX..... 21, 22, 23, 24
 Binary
 Base 23
 Mode 41, 51
 Search..... 59
 Streams..... 34
 Bitfields..... 33
 bsearch 59
 Buffer Size..... 36, 50
 Buffering Modes 50

32-Bit Language Tools Libraries

Buffering, See File Buffering	
BUFSIZ	36, 50
C	
C Locale	15, 27
Calendar Time	77, 78, 79, 81
calloc	59, 60
Caret (^)	50
Carriage Return	17
cbrt	84
ceil	84
ceilf	85
ceiling	
Double Floating Point	84
Single Floating Point	85
char	
Maximum Value	25
Minimum Value	25
Number of Bits	25
CHAR_BIT	25
CHAR_MAX	25
CHAR_MIN	25
Character Array	50
Character Case Mapping	
Lower Case Alphabetic Character	18
Upper Case Alphabetic Character	18
Character Case Mapping Functions	
tolower	18
toupper	18
Character Handling, See ctype.h	
Character Input/Output Functions	
fgetc	39
fgets	40
fputc	42
fputs	42
getc	44
getchar	45
gets	45
putc	47
putchar	48
puts	48
ungetc	52
Character Testing	
Alphabetic Character	15
Alphanumeric Character	15
Control Character	15
Decimal Digit	16
Graphical Character	16
Hexadecimal Digit	17
Lower Case Alphabetic Character	16
Printable Character	16
Punctuation Character	17
Upper Case Alphabetic Character	17
White-Space Character	17
Character Testing Functions	
isalnum	15
isalpha	15
iscntrl	15
isdigit	16
isgraph	16
islower	16
isprint	16
ispunct	17
isspace	17
isupper	17
isxdigit	17
Characters	
Alphabetic	15
Alphanumeric	15
Control	15
Convert to Lower Case Alphabetic	18
Convert to Upper Case Alphabetic	18
Decimal Digit	16
Graphical	16
Hexadecimal Digit	17
Lower Case Alphabetic	16
Printable	16
Punctuation	17
Upper Case Alphabetic	17
White-Space	17
Classifying Characters	15
clearerr	38
Clearing Error Indicator	38, 96
clock	78
clock_t	76, 78
CLOCKS_PER_SEC	77
close	96
Common Definitions, See stddef.h	
Compare Strings	71
Comparison Function	59, 63
Comparison Functions	
memcmp	69
strcmp	71
strcoll	71
strncmp	73
strxfrm	75
Compiler Options	
-fno-short-double	34
-msmart-io	34
Concatenation Functions	
strcat	70
strncat	73
Control Character	
Defined	15
Test for	15
Control Transfers	28
Conversion	46, 49, 51
Convert	
Character to Multibyte Character	67
Multibyte Character to Wide Character	62
Multibyte String to Wide Character String	62
String to Double Floating Point	57, 64
String to Integer	58
String to Long Integer	58, 65
String to Unsigned Long Integer	65, 66
To Lower Case Alphabetic Character	18
To Upper Case Alphabetic Character	18
Wide Character String to Multibyte String	66
Copying Functions	
memcpy	69
memmove	70

memset	70	Test for	16
strcpy	71	Decimal Point	46
strncpy	73	Default Handler	29
copy sign	85	Diagnostics, See assert.h	
cos	85	Diagnostics, See unistd.h	
cosf	85	diff time	78
cosh	85	Digit, Decimal, See Decimal Digit	
coshf	86	Digit, Hexadecimal, See Hexadecimal Digit	
cosine		Direct Input/Output Functions	
Double Floating Point	85	fread	42
Single Floating Point	85	fwrite	44
ctime	78	div	56, 59
ctype.h	15	div_t	56
isalnum	15	Divide	
isascii	15	Integer	59
iscntrl	15	Long Integer	61
isdigit	16	Divide by Zero	30, 59
isgraph	16	Documentation	
isalpha	15	Conventions	6
islower	16	Layout	5
isprint	16	Domain Error	19, 82, 83, 84, 85, 88, 90, 91, 93, 94
ispunct	17	dot	46
isspace	17	Double Precision Floating Point	
isupper	17	Machine Epsilon	21
isxdigit	17	Maximum Exponent (base 10)	21
tolower	18	Maximum Exponent (base 2)	21
toupper	18	Maximum Value	21
Current Argument	32	Minimum Exponent (base 10)	21
Customer Notification Service	8	Minimum Exponent (base 2)	22
Customer Support	9	Minimum Value	21
D		Number of Binary Digits	21
Dash (-)	50	Number of Decimal Digits	21
Date and Time	80	double Type	34
Date and Time Functions, See time.h		Dream Function	46
Day of the Month	77, 78, 80	drem	86
Day of the Week	77, 78, 80	DSP Library Functions by Category (General Purpose)	
Day of the Year	77, 80	100	
Daylight Savings Time	77, 79	E	
DBGETC(canblock)	126	EBADF	19
DBGETWORD(int canblock)	127	EDOM	19
DBINIT()	126	edom	82
DBL_DIG	21	EINVAL	19
DBL_EPSILON	21	Ellipses (...)	32, 50
DBL_MANT_DIG	21	Empty Binary File	41
DBL_MAX	21	Empty Text File	41
DBL_MAX_10_EXP	21	End Of File	36
DBL_MAX_EXP	21	Indicator	34
DBL_MIN	21	Seek	43
DBL_MIN_10_EXP	21	Test For	39
DBL_MIN_EXP	22	ENOMEM	19
DBPRINTF()	126	Environment Function	
DBPUTC(char c)	127	getenv	60
DBPUTWORD(int w)	127	EOF	36
DBSCANF()	126	ERANGE	19
Deallocate Memory	60, 63	erange	82
Debugging Logic Errors	14	errno	19, 20, 82
Decimal	47, 50, 65, 66	errno.h	19, 82
Decimal Digit		EBADF	19
Defined	16	EDOM	19
Number Of	21, 22, 23	EINVAL	19

32-Bit Language Tools Libraries

ENOMEM	19	fopen	41
ERANGE	19	freopen	43
errno	20	setbuf	50
Error Codes	19, 72	setvbuf	50
Error Conditions	82	File Access Modes	34, 41
Error Handler	59	File Buffering	
Error Handling Functions		Fully Buffered	34, 35
clearerr	38, 96	Line Buffered	34, 35
feof	39	Unbuffered	34, 36
ferror	39	File Operations	
perror	46	Remove	48
Error Indicator	34	Rename	48
Error Indicators		File Positioning Functions	
Clearing	38, 49, 96	fgetpos	40
End Of File	38, 40	fseek	43
Error	38, 40	fsetpos	44
Test For	39	ftell	44
Error Signal	29	rewind	49
Errors, See errno.h		FILENAME_MAX	36
Errors, Testing For	19	File-Position Indicator	34, 35, 39, 40, 42, 44
Exception Error	59	Files, Maximum Number Open	36
exit	51, 57, 60	finite	87
EXIT_FAILURE	56	Fixed-Point Types	100
EXIT_SUCCESS	56	flags	46
exp	86	float.h	21
expf	86	DBL_DIG	21
expm1	87	DBL_EPSILON	21
Exponential and Logarithmic Functions		DBL_MANT_DIG	21
exp	86	DBL_MAX	21
expf	86	DBL_MAX_10_EXP	21
frexp	88	DBL_MAX_EXP	21
frexpf	89	DBL_MIN	21
ldexp	90	DBL_MIN_10_EXP	21
ldexpf	90	DBL_MIN_EXP	22
log	90	FLT_DIG	22
log10	90	FLT_EPSILON	22
log10f	91	FLT_MANT_DIG	22
logf	91	FLT_MAX	22
modf	92	FLT_MAX_10_EXP	22
modff	92	FLT_MAX_EXP	22
Exponential Function		FLT_MIN	22
Double Floating Point	86	FLT_MIN_10_EXP	22
Single Floating Point	86	FLT_MIN_EXP	23
F		FLT_RADIX	23
fabs	87	FLT_ROUNDS	23
fabsf	87	LDBL_DIG	23
fclose	38, 96	LDBL_EPSILON	23
feof	38, 39	LDBL_MANT_DIG	23
ferror	38, 39	LDBL_MAX	23
fgetc	97	LDBL_MAX_10_EXP	23
fflush	39, 96	LDBL_MAX_EXP	24
ffs	68	LDBL_MIN	24
fgetc	39	LDBL_MIN_10_EXP	24
fgetpos	40	LDBL_MIN_EXP	24
fgets	40, 97	Floating Point	
Field Width	46	Limits	21
FILE	14, 34, 35	Types, Properties Of	21
File Access Functions		Floating Point, See float.h	
fclose	38	Floating-Point Error Signal	30
fflush	39	floor	87

Double Floating Point	87	G	
Single Floating Point	88	getc	44
floor	88	getchar	45
FLT_DIG	22	getenv	60
FLT_EPSILON	22	gets	45, 97
FLT_MANT_DIG	22	gettimeofday	79
FLT_MAX	22	GMT	79
FLT_MAX_10_EXP	22	gmtime	79
FLT_MAX_EXP	22	Graphical Character	
FLT_MIN	22	Defined	16
FLT_MIN_10_EXP	22	Test for	16
FLT_MIN_EXP	23	Greenwich Mean Time	79
FLT_RADIX	23	H	
FLT_RADIX Digit		h modifier	47, 49
Number Of	21, 22, 23	Handler	
FLT_ROUND	23	Default	29
Flush	39, 60	Error	59
fmod	88	Nested	28
fmodf	88	Signal	29
-fno-short-double	34	Signal Type	29
fopen	34, 41, 45, 50	Handling	
FOPEN_MAX	36	Interrupt Signal	31
Form Feed	17	Header Files	
Format Specifiers	46, 49	assert.h	14
Formatted I/O Routines	34	ctype.h	15
Formatted Input/Output Functions		errno.h	19, 82
fprintf	41	float.h	21
fscanf	43	limits.h	25
printf	46	locale.h	27
scanf	49	math.h	82
sprintf	51	setjmp.h	28
sscanf	51	signal.h	29
vfprintf	52	stdarg.h	32
vprintf	53	stddef.h	33
vsprintf	54	stdio.h	34
Formatted Text		stdlib.h	56
Printing	51	string.h	68
Scanning	51	sys/appio.h	126
fpos_t	35	time.h	76
fprintf	34, 41	unistd.h	96
fputc	42	Hexadecimal	47, 50, 65, 66
fputs	42	Hexadecimal Conversion	46
fraction and exponent function		Hexadecimal Digit	
Double Floating Point	88	Defined	17
Single Floating Point	89	Test for	17
Fraction Digits	46	Horizontal Tab	17
fread	42, 97	Hour	77, 78, 80
free	60	HUGE_VAL	82
Free Memory	60	Hyperbolic Cosine	
freopen	34, 43, 45	Double Floating Point	85
frexp	88	Single Floating Point	86
frexpf	89	Hyperbolic Functions	
fscanf	34, 43	cosh	85
fseek	43, 52, 96	coshf	86
fsetpos	44, 52	sinh	93
fsl	68	sinhf	93
ftell	44, 96	tanh	94
Full Buffering	50	tanhf	95
Fully Buffered	34, 35	Hyperbolic Sine	
fwrite	44		

32-Bit Language Tools Libraries

Double Floating Point	93	LC_CTYPE	27
Single Floating Point	93	LC_MONETARY	27
Hyperbolic Tangent		LC_NUMERIC	27
Double Floating Point	94	LC_TIME	27
hyperbolic tangent		Iconv, struct	27
Single Floating Point	95	LDBL_DIG	23
hypot	89	LDBL_EPSILON	23
I		LDBL_MANT_DIG	23
Ignore Signal	29	LDBL_MAX	23
Illegal Instruction Signal	30	LDBL_MAX_10_EXP	23
Implementation-Defined Limits, See limits.h		LDBL_MAX_EXP	24
Indicator		LDBL_MIN	24
End Of File	34, 36	LDBL_MIN_10_EXP	24
Error	34, 39	LDBL_MIN_EXP	24
File Position	34, 39, 40, 42, 44	ldexp	90
Infinity	82	ldexpf	90
Input and Output, See stdio.h		ldiv	56, 61
Input Formats	34	ldiv_t	56
int		Leap Second	77, 80
Maximum Value	25	Left Justify	46
Minimum Value	25	Libraries	
INT_MAX	25	Standard C	13
INT_MIN	25	Standard C Math	82
Integer Limits	25	Limits	
Internal Error Message	72	Floating Point	21
Internet Address, Microchip	8	Integer	25
Interrupt Signal	30	limits.h	25
Interrupt Signal Handling	31	CHAR_BITS	25
Inverse Cosine, See arccosine		CHAR_MAX	25
Inverse Sine, See arcsine		CHAR_MIN	25
Inverse Tangent, See arctangent		INT_MAX	25
IOFBF	35, 50	INT_MIN	25
IOLBF	35, 50	LLONG_MAX	25
IONBF	36, 50	LLONG_MIN	25
isalnum	15	LONG_MAX	25
isascii	15	LONG_MIN	25
iscntrl	15	MB_LEN_MAX	26
isdigit	16	SCHAR_MAX	26
isgraph	16	SCHAR_MIN	26
isinf	89	SHRT_MAX	26
islaoha	15	SHRT_MIN	26
islower	16	UCHAR_MAX	26
isnan	89	UINT_MAX	26
isprint	16	ULLONG_MAX	26
ispunct	17	ULONG_MAX	26
isspace	17	USHRT_MAX	26
isupper	17	LINE	14
isxdigit	17	Line Buffered	34, 35
J		Line Buffering	50
jmp_buf	28	link	48, 96, 97
Justify	46	ll modifier	47, 49
L		llabs	61
L modifier	47, 49	lldiv	56, 61
l modifier	47, 49	lldiv_t	56
L_tmpnam	36, 52	LLONG_MAX	25
labs	60	LLONG_MIN	25
LC_ALL	27	Load Exponent Function	
LC_COLLATE	27	Double Floating Point	90
		Single Floating Point	90
		Local Time	78, 79

Locale, C	15, 27	asin	83
Locale, Other	27	asinf	83
locale.h	27	asinh	83
localeconv	27	atan	83
Localization, See locale.h		atan2	83
localtime	78, 79	atan2f	84
Locate Character	71	atanf	84
log	90	atanh	84
log10	90	cbrt	84
log10f	91	ceil	84
log1p	91	ceilf	85
Logarithm Function		copysign	85
Double Floating Point	90	cos	85
Single Floating Point	91	cosf	85
Logarithm Function, Natural		cosh	85
Double Floating Point	90	coshf	86
Single Floating Point	91	drem	86
logb	91	exp	86
logf	91	expf	86
Logic Errors, Debugging	14	expm1	87
Long Double Precision Floating Point		fabs	87
Machine Epsilon	23	fabsf	87
Maximum Exponent (base 10)	23	finite	87
Maximum Exponent (base 2)	24	floor	87
Maximum Value	23	floorf	88
Minimum Exponent (base 10)	24	fmod	88
Minimum Exponent (base 2)	24	fmodf	88
Minimum Value	24	frexp	88
Number of Binary Digits	23	frexpf	89
Number of Decimal Digits	23	HUGE_VAL	82
long int		hypot	89
Maximum Value	25	isinf	89
Minimum Value	25	isnan	89
long long int		ldexp	90
Maximum Value	25	ldexpf	90
Minimum Value	25	log	90
long long unsigned int		log10	90
Maximum Value	26	log10f	91
long unsigned int		log1p	91
Maximum Value	26	logb	91
LONG_MAX	25	logf	91
LONG_MIN	25	modf	92
longjmp	28	modff	92
Lower Case Alphabetic Character		pow	92
Convert To	18	powf	92
Defined	16	rint	93
Test for	16	sin	93
lseek	39, 43, 44, 45, 96	sinf	93
M		sinh	93
Machine Epsilon		sinhf	93
Double Floating Point	21	sqrt	94
Long Double Floating Point	23	sqrtf	94
Single Floating Point	22	tan	94
Magnitude	86, 88, 93	tanf	94
malloc	60, 61	tanh	94
Mapping Characters	15	tanhf	95
Math Exception Error	59	Mathematical Functions, See math.h	
math.h	82	Maximum	
acos	82	Multibyte Character	56
acosf	82	Maximum Value	

32-Bit Language Tools Libraries

Double Floating-Point Exponent (base 10)	21	mips_fft32_setup	119
Double Floating-Point Exponent (base 2)	21	mips_fir16	111
Long Double Floating-Point Exponent (base 10) ..	23	mips_fir16_setup	112
Long Double Floating-Point Exponent (base 2) ..	24	mips_h264_iqt	120
Multibyte Character	26	mips_h264_iqt_setup	121
rand	57	mips_h264_mc_luma	122
Single Floating-Point Exponent (base 10)	22	mips_iir16	113
Single Floating-Point Exponent (base 2)	22	mips_iir16_setup	114
Type char	25	mips_lms16	115
Type Double	21	mips_vec_abs16	102
Type int	25	mips_vec_abs32	102
Type Long Double	23	mips_vec_add16	103
Type long int	25	mips_vec_add32	103
Type long long int	25	mips_vec_addc16	104
Type long long unsigned int	26	mips_vec_addc32	104
Type long unsigned int	26	mips_vec_dotp16	105
Type short int	26	mips_vec_dotp32	106
Type signed char	26	mips_vec_mul16	106
Type Single	22	mips_vec_mul32	107
Type unsigned char	26	mips_vec_mulc16	107
Type unsigned int	26	mips_vec_mulc32	108
Type unsigned short int	26	mips_vec_sub16	108
MB_CUR_MAX	56	mips_vec_sub32	109
MB_LEN_MAX	26	mips_vec_sum_squares16	109
mblen	62	mips_vec_sum_squares32	110
mbstowcs	62	mktime	79
mbtowc	62	modf	92
memchr	69	modff	92
memcmp	69	modulus function	
memcpy	69	Double Floating Point	92
memmove	70	Single Floating Point	92
Memory		Month	77, 78, 80
Allocate	59, 61	-msmart-io	34
Deallocate	60	Multibyte Character	56, 62, 67
Free	60	Maximum Number of Bytes	26
Reallocate	63	Multibyte String	62, 66
memset	70	N	
Minimum Value		NaN	82
Double Floating-Point Exponent (base 10)	21	Natural Logarithm	
Double Floating-Point Exponent (base 2)	22	Double Floating Point	90
Long Double Floating-Point Exponent (base 10) ..	24	Single Floating Point	91
Long Double Floating-Point Exponent (base 2) ..	24	NDEBUG	14
Single Floating-Point Exponent (base 10)	22	Nearest Integer Functions	
Single Floating-Point Exponent (base 2)	23	ceil	84
Type char	25	ceilf	85
Type Double	21	floor	87
Type int	25	floorf	88
Type Long Double	24	Nested Signal Handler	28
Type long int	25	Newline	17, 34, 40, 42, 45, 46, 48
Type long long int	25	No Buffering	34, 36, 50
Type short int	26	Non-Local Jumps, See setjmp.h	
Type signed char	26	NSETJMP	28
Type Single	22	NULL	33, 36
Minute	77, 78, 80	O	
MIPS Technologies Inc.'s DSP Library Notices	123	Octal	47, 50, 65, 66
mips_fft16	116	Octal Conversion	46
mips_fft16_setup	117	offsetof	33
mips_fft32	118	open	45
		Output Formats	34

Overflow Errors	19, 82, 86, 90, 92	rename	48, 96, 97
Overlap	69, 70, 71, 73	Reset	57
P		Reset File Pointer	49
Pad Characters	46	rewind	49, 52
Percent	47, 49, 50, 80	rint	93
perorr	46	Rounding Mode	23
PIC32 Debugging Support, See sys/appio.h		S	
PIC32 DSP Library	99	Saturation, Scaling, and Overflow	101
Plus Sign	46	Scan Formats	34
Pointer, Temporary	63	scanf	34, 49
pow	92	SCHAR_MAX	26
Power Function		SCHAR_MIN	26
Double Floating Point	92	Search Functions	
Single Floating Point	92	memchr	69
Power Functions		strchr	71
pow	92	strcspn	72
powf	92	strpbrk	74
powf	92	strchr	74
precision	46	strspn	74
Prefix	17, 46	strstr	74
Print Formats	34	strtok	75
Printable Character		Second	77, 78, 80
Defined	16	Seed	63, 64
Test for	16	Seek	
printf	34, 46	From Beginning of File	43
Processor Clocks per Second	77	From Current Position	43
Processor Time	76, 78	From End Of File	43
Pseudo-Random Number	63, 64	SEEK_CUR	36, 43
ptrdiff_t	33	SEEK_END	37, 43
Punctuation Character		SEEK_SET	37, 43
Defined	17	setbuf	34, 36, 50
Test for	17	setjmp	28
Pushed Back	52	setjmp.h	28
putc	47	jmp_buf	28
putchar	48	longjmp	28
puts	48	setjmp	28
Q		setlocale	27
qsort	63	settimeofday	80
Quick Sort	63	setvbuf	34, 35, 50
R		short int	
Radix	23	Maximum Value	26
raise	29, 30, 31	Minimum Value	26
rand	63, 64	SHRT_MAX	26
RAND_MAX	57, 63	SHRT_MIN	26
Range	50	sig_atomic_t	29
Range Error	19, 65, 66, 85, 86, 90, 92, 93	SIG_DFL	29
read	97	SIG_ERR	29
Reading, Recommended	7	SIG_IGN	29
realloc	60, 63	SIGABRT	30
Reallocate Memory	63	SIGFPE	30
Registered Functions	57, 60	SIGILL	30
Remainder		SIGINT	30
Double Floating Point	88	Signal	
Single Floating Point	88	Abnormal Termination	30
Remainder Functions		Error	29
fmod	88	Floating-Point Error	30
fmodf	88	Ignore	29
remove	48, 97	Illegal Instruction	30
		Interrupt	30
		Reporting	31

32-Bit Language Tools Libraries

Termination Request.....	30	sqrtf.....	94
signal.....	30, 31	srand.....	64
Signal Handler.....	29	sscanf.....	34, 51
Signal Handler Type.....	29	Standard C Library.....	13
Signal Handling, See signal.h		Standard C Locale.....	15
signal.h.....	29	Standard Error.....	34, 37
raise.....	31	Standard Input.....	34, 37
sig_atomic_t.....	29	Standard Output.....	34, 37
SIG_DFL.....	29	Start-up.....	34
SIG_ERR.....	29	stdarg.h.....	32
SIG_IGN.....	29	va_arg.....	32
SIGABRT.....	30	va_end.....	32
SIGFPE.....	30	va_list.....	32
SIGILL.....	30	va_start.....	32
SIGINT.....	30	stddef.h.....	33
signal.....	31	NULL.....	33
SIGSEGV.....	30	offsetof.....	33
SIGTERM.....	30	ptrdiff_t.....	33
signed char		size_t.....	33
Maximum Value.....	26	wchar_t.....	33
Minimum Value.....	26	stderr.....	34, 36, 37, 46
SIGSEGV.....	30	stdin.....	34, 36, 37, 45, 49
SIGTERM.....	30	stdio.h.....	34
sin.....	93	_IOFBF.....	35
sine		_IOLBF.....	35
Double Floating Point.....	93	_IONBF.....	36
Single Floating Point.....	93	_mon_putc.....	38
sinf.....	93	asprintf.....	38
Single Precision Floating Point		BUFSIZ.....	36
Machine Epsilon.....	22	clearerr.....	38
Maximum Exponent (base 10).....	22	EOF.....	36
Maximum Exponent (base 2).....	22	fclose.....	38
Maximum Value.....	22	feof.....	38, 39
Minimum Exponent (base 10).....	22	ferror.....	39
Minimum Exponent (base 2).....	23	fflush.....	39
Minimum Value.....	22	fgetc.....	39
Number of Binary Digits.....	22	fgetpos.....	40
Number of Decimal Digits.....	22	fgets.....	40
sinh.....	93	FILE.....	35
sinhf.....	93	FILENAME_MAX.....	36
size.....	47	fopen.....	41
size_t.....	33, 35, 68, 76	FOPEN_MAX.....	36
sizeof.....	33, 35, 68, 76	fpos_t.....	35
snprintf.....	51	fprintf.....	41
Sort, Quick.....	63	fputc.....	42
Source File Name.....	14	fputs.....	42
Source Line Number.....	14	fread.....	42
Space.....	46	freopen.....	43
Space Character		fscanf.....	43
Defined.....	17	fseek.....	43
Test for.....	17	fsetpos.....	44
Specifiers.....	46, 49	ftell.....	44
sprintf.....	34, 51	fwrite.....	44
sqrt.....	94	getc.....	44
sqrtf.....	94	getchar.....	45
Square Root Function		gets.....	45
Double Floating Point.....	94	L_tmpnam.....	36
Single Floating Point.....	94	NULL.....	36
Square Root Functions		open.....	45
sqrt.....	94	perror.....	46

printf	46	rand	63
putc	47	RAND_MAX	57
putchar	48	realloc	63
puts	48	rand	64
remove	48	strtod	64
rename	48	strtof	64
rewind	49	strtol	65
scanf	49	strtoll	65
SEEK_CUR	36	strtoul	65
SEEK_END	37	strtoull	66
SEEK_SET	37	system	66
setbuf	50	wctomb	67
setvbuf	50	wxstombs	66
size_t	35	stdout	34, 36, 37, 46, 48
snprintf	51	strcasecmp	70
sprintf	51	strcat	70
sscanf	51	strchr	71
stderr	37	strcmp	71
stdin	37	strcoll	71
stdout	37	strcpy	71
TMP_MAX	37	strcspn	72
tmpfile	51	Streams	34
tmpnam	52	Binary	34
ungetc	52	Buffering	50
vfprintf	52	Closing	38, 60
vfscanf	53	Opening	41
vprintf	53	Reading From	44
vscanf	53	Text	34
vsprintf	54	Writing To	44, 47
vsprintf	54	sterror	72
vsscanf	55	strftime	80
stdlib.h	56	String	
abort	57	Length	72
abs	57	Search	74
atexit	57	Transform	75
atof	57	String Functions, See string.h	
atoi	58	string.h	68
atol	58	ffs	68
atoll	58	fsl	68
bsearch	59	memchr	69
calloc	59	memcmp	69
div	59	memcpy	69
div_t	56	memmove	70
exit	60	memset	70
EXIT_FAILURE	56	size_t	68
EXIT_SUCCESS	56	strcasecmp	70
free	60	strcat	70
getenv	60	strchr	71
labs	60	strcmp	71
ldiv	61	strcoll	71
ldiv_t	56	strcpy	71
llabs	61	strcspn	72
lldiv	61	sterror	72
lldiv_t	56	strlen	72
malloc	61	strncasecmp	72
MB_CUR_MAX	56	strncat	73
mblen	62	strncmp	73
mbstowcs	62	strncpy	73
mbtowc	62	strpbrk	74
qsort	63	strchr	74

32-Bit Language Tools Libraries

strspn	74	time_t	76, 77, 79, 81
strstr	74	time.h	76
strtok	75	asctime	78
strxfrm	75	clock	78
strlen	72	clock_t	76
strncasecmp	72	CLOCKS_PER_SEC	77
strncat	73	ctime	78
strncmp	73	difftime	78
strncpy	73	gettimeofday	79
strpbrk	74	gmtime	79
strchr	74	localtime	79
strspn	74	mktime	79
strstr	74	settimeofday	80
strtod	57, 64	size_t	76
strtof	64	strftime	80
strtok	75	struct timeval	76
strtol	58, 65	struct tm	77
strtoll	65	time	81
strtoul	65	time_t	77
strtoull	66	TMP_MAX	37
struct lconv	27	tmpfile	51
struct timeval	76	tmpnam	52
struct tm	77	Tokens	75
strxfrm	75	tolower	18
Substrings	75	toupper	18
Subtracting Pointers	33	Transferring Control	28
Successful Termination	56	Transform String	75
sys/appio.h	126	Trigonometric Functions	
DBGETC(canblock)	126	acos	82
DBGETWORD(int canblock)	127	acosf	82
DBINIT()	126	asin	83
DBPRINTF()	126	asinf	83
DBPUTC(char c)	127	atan	83
DBPUTWORD(int w)	127	atan2	83
DBSCANF()	126	atan2f	84
system	66	atanf	84
T		cos	85
Tab	17	cosf	85
tan	94	sin	93
tanf	94	sinf	93
tangent		tan	94
Double Floating Point	94	tanf	94
Single Floating Point	94	type	47, 50
tanh	94	U	
tanhf	95	UCHAR_MAX	26
Temporary		UINT_MAX	26
File	51, 60	ULLONG_MAX	26
Filename	36, 52	ULONG_MAX	26
Pointer	63	Underflow Errors	19, 82, 86, 90, 92
Termination		ungetc	52
Request Signal	30	unistd.h	96
Successful	56	close	96
Unsuccessful	56	link	96
Text Mode	41	lseek	96
Text Streams	34	read	97
Ticks	78	unlink	97
time	81	write	97
Time Difference	78	Universal Time Coordinated	79
Time Structure	77, 80	unlink	48, 97
Time Zone	80	unsigned char	

Maximum Value	26
unsigned int	
Maximum Value	26
unsigned short int	
Maximum Value	26
Unsuccessful Termination.....	56
Upper Case Alphabetic Character	
Convert To.....	18
Defined	17
Test for.....	17
USHRT_MAX.....	26
UTC.....	79
Utility Functions, See stdlib.h	
V	
va_arg	32, 52, 53, 54
va_end	32, 52, 53, 54
va_list.....	32
va_start	32, 52, 53, 54
Variable Argument Lists, See stdarg.h	
Variable Length Argument List.....	32, 52, 53, 54
Vertical Tab.....	17
vfprintf	34, 52
vfscanf.....	53
vprintf	34, 53
vscanf.....	53
vsprintf	54
vsprintf	34, 54
vsscanf.....	55
W	
wchar_t	33
wcstombs	66
wctomb.....	67
Web Site, Microchip	8
Week.....	80
White Space.....	49, 57, 58, 64
White-Space Character	
Defined	17
Test for.....	17
Wide Character	62, 67
Wide Character String.....	62, 66
Wide Character Value	33
Width.....	46
width.....	46, 49
Wrapper Functions.....	135
write	97
Y	
Year	77, 78, 80
Z	
Zero.....	82
Zero, divide by	30, 59



MICROCHIP

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://www.microchip.com/support>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hangzhou
Tel: 86-571-2819-3187
Fax: 86-571-2819-3189

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Osaka
Tel: 81-66-152-7160
Fax: 81-66-152-9310

Japan - Yokohama
Tel: 81-45-471-6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-330-9305

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

11/29/11