



MPLAB XC16 Libraries Reference Guide

Notice to Customers



Important:

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 website (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA," where "XXXXX" is the document number and "A" is the alphabetic 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 Help Content to open a list of available online help files.



Table of Contents

Notice to Customers.....	1
1. Libraries Overview.....	4
1.1. OMF-Specific Libraries and Startup Modules.....	4
1.2. Startup Code.....	4
1.3. DSP Library.....	5
1.4. 16-Bit Peripheral Libraries.....	5
1.5. Standard C Libraries with Math and Support Functions.....	5
1.6. Fixed-Point Math Functions.....	5
1.7. Compiler Built-in Functions.....	5
2. Standard C Libraries.....	6
2.1. Using the Standard C Libraries.....	6
2.2. Multiple Header Types and Macros.....	6
2.3. <assert.h> Diagnostics.....	7
2.4. <ctype.h> Character Handling.....	8
2.5. <errno.h> Errors.....	19
2.6. <float.h> Floating-Point Characteristics.....	20
2.7. <iso646.h> Alternate Spellings.....	25
2.8. <limits.h> Implementation-Defined Limits.....	26
2.9. <locale.h> Localization.....	26
2.10. <setjmp.h> Non-Local Jumps.....	27
2.11. <signal.h> Signal Handling.....	28
2.12. <stdarg.h> Variable Argument Lists.....	34
2.13. <stddef.h> Common Definitions.....	36
2.14. <stdio.h> Input and Output.....	36
2.15. <stdlib.h> Utility Functions.....	81
2.16. <string.h> String Functions.....	97
2.17. <time.h> Date and Time Functions.....	118
3. Math Libraries.....	127
3.1. <math.h> Mathematical Functions.....	127
4. Support Libraries.....	165
4.1. Rebuilding the libpic30 Library.....	165
4.2. Standard C Library Helper Functions.....	165
4.3. Standard C Library Functions That Require Modification.....	171
4.4. Functions/Constants to Support A Simulated UART.....	171
4.5. Functions for Erasing and Writing EEDATA Memory.....	175
4.6. Functions for Erasing and Writing Flash Memory.....	178
4.7. Functions for Specialized Copying and Initialization.....	182
4.8. Functions to Support Secondary Core PRAM.....	189
5. Fixed-Point Math Functions.....	192
5.1. Overview of Fixed-Point Data Formats.....	192
5.2. Using the Fixed-Point Libraries.....	196
5.3. <libq.h> Mathematical Functions.....	197

6. Revision History.....	220
6.1. Revision N (February 2021).....	220
6.2. Revision M (July 2020).....	220
6.3. Revision L (December 2019).....	220
6.4. Revision K (February 2018).....	220
6.5. Revision J (December 2014).....	220
6.6. Revision H (September 2013).....	221
6.7. Revision G (October 2010).....	221
6.8. Revision F (March 2009).....	221
6.9. Revision E (January 2008).....	221
6.10. Revision D (December 2006).....	222
6.11. Revision C (October 2005).....	223
6.12. Revision B (September 2004).....	224
6.13. Revision A (May 2004).....	224
The Microchip Website.....	225
Product Change Notification Service.....	225
Customer Support.....	225
Microchip Devices Code Protection Feature.....	225
Legal Notice.....	225
Trademarks.....	226
Quality Management System.....	226
Worldwide Sales and Service.....	227

1. Libraries Overview

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

Compiler Installation Locations

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

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

where x.xx is the version number.

Assembly Code Applications

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

C Code Applications

The 16-bit language tool libraries are included in the `lib` subdirectory of the MPLAB XC16 C Compiler install directory (see “Compiler Installation Locations”). These libraries can be linked directly into an application with a 16-bit linker.

1.1 OMF-Specific Libraries and Startup Modules

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

- ELF – Executable and Linkable Format (default). The debugging format used for ELF object files is DWARF 2.0.
- COFF – Common Object File Format.

There are two ways to select the OMF:

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

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

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

1.2 Startup Code

In order to initialize variables in data memory, the linker creates a data initialization template. This template must be processed at startup, before the application proper takes control. For C programs, this function is performed by the startup modules `inlibpic30-coff.a` (either `crt0.o` or `crt1.o`) or `inlibpic30-elf.a` (either `crt0.eo` or `crt1.eo`). Assembly language programs can utilize these modules directly by linking with the desired startup module file. The source code for the startup modules is provided in corresponding `.s` files.

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

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

1.3 DSP Library

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

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

1.4 16-Bit Peripheral Libraries

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

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

1.5 Standard C Libraries with Math and Support Functions

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

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

A typical C application will require these libraries.

1.6 Fixed-Point Math Functions

Fixed-point math functions may be found in the library file `libq-omf.a`.

1.7 Compiler Built-in Functions

The MPLAB XC16 C Compiler contains built-in functions that, to the developer, work like library functions. These functions are listed in the “*MPLAB® XC16 C Compiler Users’ Guide*” (DS50002071).

2. Standard C Libraries

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

Assembly Code Applications

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

C Code Applications

The MPLAB XC16 C Compiler install directory (see “Libraries Overview, Compiler Installation Locations”) contains the following subdirectories with library-related files:

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

2.1 Using the Standard C Libraries

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

Header Files

All standard C library entities are declared or defined in one or more standard headers. To make use of a library entity in a program, write an include directive that names the relevant standard header.

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

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

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

A standard header never includes another standard header.

Library Files

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

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

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

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

2.2 Multiple Header Types and Macros

Some functions, macros, variables and types may be found in more than one or two standard headers. These are listed below.

2.2.1 Null Macro

Description

The value of a null pointer constant.

Include

```
<locale.h>
<stddef.h>
<stdio.h>
<stdlib.h>
<string.h>
<time.h>
```

2.2.2 **size_t Type** **Description**

The result type of the `sizeof` operator.

Include

```
<stddef.h>
<stdio.h>
<stdlib.h>
<string.h>
<time.h>
```

2.3 **<assert.h> Diagnostics**

The content of the header file `assert.h` is useful for debugging logic errors in programs. By using these features in critical locations where certain conditions should be true, the logic of the program may be tested.

2.3.1 **assert Macro**

If the argument is false, an assertion failure message is printed and the program is aborted.

Include

```
<assert.h>
```

Prototype

```
void assert(scalar 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 and `abort()` is called, terminating program execution. In the case of MPLAB XC8, the message is printed to `stdout`; for other compilers, this is printed to `stderr`.

When using MPLAB XC8 for 8-bit AVR devices, the message is only printed if `__ASSERT_USE_STDERR` has been defined before the inclusion of `<assert.h>`.

The message includes the source file name (`__FILE__`), the source line number (`__LINE__`), the expression being evaluated.

If the macro `NDEBUG` is defined at the point where `<assert.h>` is included, the `assert()` macro will evaluate to a void expression, `((void)0)`, and not print any assertion message, nor abort program execution. Inclusion of `<assert.h>` can occur multiple times, even in the same source file, and the action of the `assert()` macro for each inclusion will be based on the state of `NDEBUG` at the point at which that inclusion takes place.

Example

See the notes at the beginning of this chapter or section for information on using `printf()` or `scanf()` (and other functions reading and writing the `stdin` or `stdout` streams) in the example code.

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

    a = 4;
#define NDEBUG          /* negate debugging - disable assert() functionality */
#include <assert.h>
    assert(a == 6);      /* no action performed, even though expression is false */

#undef NDEBUG           /* ensure assert() is active */
#include <assert.h>
    a = 7;
    assert(a == 7);      /* true - no action performed */
    assert(a == 8);      /* false - print message and abort */
}
```

Example Output

```
sampassert.c:14 a == 8 -- assertion failed
ABRT
```

2.3.2 `__conditional_software_breakpoint` Macro

If the argument is false, a software breakpoint is triggered.

Include

```
<assert.h>
```

Prototype

```
void __conditional_software_breakpoint(scalar expression);
```

Argument

expression The expression to test.

Remarks

The expression evaluates to zero or non-zero. If zero, a software breakpoint is triggered and execution will be suspended. If the target device does not support software breakpoints, a compiler error is triggered.

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

Example

```
#include <assert.h>

int main(void)
{
    int a;

    a = 2 * 2;
    __conditional_software_breakpoint(a == 4); /* if true-no action */
    __conditional_software_breakpoint(a == 6); /* if false-break on this line */
}
```

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.

2.4.1 isalnum Function

Test for an alphanumeric character.

Include

```
<ctype.h>
```

Prototype

```
int isalnum(int c);
```

Argument

c The character to test.

Return Value

Returns a non-zero integer value if the character, *c*, is alphanumeric; otherwise, returns a zero.

Remarks

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

Example

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

int main(void)
{
    int ch;

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

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

Example Output

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

2.4.2 isalpha Function

Test for an alphabetic character.

Include

```
<ctype.h>
```

Prototype

```
int isalpha(int c);
```

Argument

c The character to test.

Return Value

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

Remarks

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

Example

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

int main(void)
{
    int ch;

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

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

Example Output

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

2.4.3 isblank Function

Test for a space or tab character.

Include

<ctype.h>

Prototype

```
int isblank (int c);
```

Argument

c The character to test.

Return Value

Returns a non-zero integer value if the character is a space or tab character; otherwise, returns zero.

Remarks

A character is considered to be a white-space character if it is one of the following: space (' ') or horizontal tab ('\t').

Example

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

int main(void)
{
    int ch;

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

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

```
    printf("a tab is NOT a white-space character\n");
}
```

Example Output

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

2.4.4 iscntrl Function

Test for a control character.

Include

```
<ctype.h>
```

Prototype

```
int iscntrl(int c);
```

Argument

c The character to test.

Return Value

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

Remarks

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

Example

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

int main(void)
{
    char ch;

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

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

Example Output

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

2.4.5 isdigit Function

Test for a decimal digit.

Include

```
<ctype.h>
```

Prototype

```
int isdigit(int c);
```

Argument

c The character to test.

Return Value

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

Remarks

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

Example

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

int main(void)
{
    int ch;

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

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

Example Output

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

2.4.6 isgraph Function

Test for a graphical character.

Include

<ctype.h>

Prototype

```
int isgraph (int c);
```

Argument

c The character to test.

Return Value

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

Remarks

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

Example

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

int main(void)
{
    int ch;

    ch = '3';
    if (isgraph(ch))
        printf("3 is a graphical character\n");
}
```

```

else
    printf("3 is NOT a graphical character\n");

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

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

```

Example Output

```

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

```

2.4.7 islower Function

Test for a lowercase alphabetic character.

Include

<ctype.h>

Prototype

```
int islower (int c);
```

Argument

c The character to test.

Return Value

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

Remarks

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

Example

```

#include <ctype.h>
#include <stdio.h>

int main(void)
{
    int ch;

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

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

```

Example Output

```

B is NOT lowercase
b is lowercase

```

2.4.8 isprint Function

Test for a printable character (includes a space).

Include

```
<ctype.h>
```

Prototype

```
int isprint (int c);
```

Argument

c The character to test.

Return Value

Returns a non-zero integer value if the character, **c**, is printable; otherwise, returns zero.

Remarks

A character is considered to be a printable character if it is in the range 0x20 to 0x7e inclusive.

Example

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

int main(void)
{
    int ch;

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

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

Example Output

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

2.4.9 ispunct Function

Test for a punctuation character.

Include

```
<ctype.h>
```

Prototype

```
int ispunct (int c);
```

Argument

c The character to test.

Return Value

Returns a non-zero integer value if the character, **c**, is a punctuation character; otherwise, returns zero.

Remarks

A character is considered to be a punctuation character if it is a printable character which is neither a space nor an alphanumeric character. Punctuation characters consist of the following:

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

Example

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

int main(void)
{
    int ch;

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

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

Example Output

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

2.4.10 isspace Function

Test for a white-space character.

Include

<ctype.h>

Prototype

```
int isspace (int c);
```

Argument

c The character to test.

Return Value

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

Remarks

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

Example

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

int main(void)
{
    int ch;

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

    ch = '\t';
```

```

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

```

Example Output

```

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

```

2.4.11 isupper Function

Test for an uppercase letter.

Include

<ctype.h>

Prototype

```
int isupper (int c);
```

Argument

c The character to test.

Return Value

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

Remarks

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

Example

```

#include <ctype.h>
#include <stdio.h>

int main(void)
{
    int ch;

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

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

```

Example Output

```

B is uppercase
b is NOT uppercase

```

2.4.12 isxdigit Function

Test for a hexadecimal digit.

Include

<ctype.h>

Prototype

```
int isxdigit (int c);
```

Argument

c The character to test.

Return Value

Returns a non-zero integer value if the character, *c*, 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.

Example

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

int main(void)
{
    int ch;

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

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

Example Output

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

2.4.13 tolower Function

Convert a character to a lowercase alphabetical character.

Include

<ctype.h>

Prototype

```
int tolower (int c);
```

Argument

c The character to convert to lowercase.

Return Value

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

Remarks

Only uppercase alphabetical characters may be converted to lowercase.

Example

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

int main(void)
{
    int ch;

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

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

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

Example Output

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

2.4.14 toupper Function

Convert a character to an uppercase alphabetical character.

Include

<ctype.h>

Prototype

```
int toupper (int c);
```

Argument

c The character to convert to uppercase.

Return Value

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

Remarks

Only lowercase alphabetical characters may be converted to uppercase.

Example

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

int main(void)
{
    int ch;

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

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

    ch = '@';
    printf("@ has no uppercase, ");
```

```
    printf("so %c is returned\n", toupper(ch));
}
```

Example Output

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

2.5 <errno.h> Errors

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

2.5.1 EDOM Macro

Represents a domain error.

Include

```
<errno.h>
```

Remarks

`EDOM` represents a domain error, which occurs when an input argument is outside the domain in which the function is defined.

2.5.2 EILSEQ Macro

Represents a wide character encoding error.

Include

```
<errno.h>
```

Remarks

`EILSEQ` represents a wide character encoding error, when the character sequence presented to the underlying `mbrtowc` function does not form a valid (generalized) multibyte character, or if the code value passed to the underlying `wcrtomb` does not correspond to a valid (generalized) overflow or underflow error, which occurs when a result is too large or too small to be stored.

2.5.3 ERANGE Macro

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.

2.5.4 errno Variable

Contains the value of an error when an error occurs in a function.

Include

```
<errno.h>
```

Remarks

The variable `errno` is set to a non-zero integer value by a library function when an error occurs. At program start-up, `errno` is set to zero. `errno` should be reset to zero prior to calling a function that sets it.

2.6 <float.h> Floating-Point Characteristics

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

2.6.1 DBL_DIG Macro

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

Include

`<float.h>`

Value

The value 6, by default; 15 if the switch `-fno-short-double` is used.

Remarks

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

2.6.2 DBL_EPSILON Macro

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

Include

`<float.h>`

Value

The value 1.192093e-07 by default, 2.220446e-16 if the switch `-fno-short-double` is used.

Remarks

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

2.6.3 DBL_MANT_DIG Macro

Number of base-FLT_RADIX digits in a double precision floating-point significand.

Include

`<float.h>`

Value

The value 24 by default, 53 if the switch `-fno-short-double` is used.

Remarks

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

2.6.4 DBL_MAX Macro

Maximum finite double precision floating-point value.

Include

`<float.h>`

Value

The value 3.402823e+38 by default; 1.797693e+308 if the switch `-fno-short-double` is used.

Remarks

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

2.6.5 DBL_MAX_10_EXP Macro

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

Include

`<float.h>`

Value

The value 38 by default, 308 if the switch `-fno-short-double` is used.

Remarks

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

2.6.6 DBL_MAX_EXP Macro

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

Include

`<float.h>`

Value

The value 128 by default; 1024 if the switch `-fno-short-double` is used.

Remarks

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

2.6.7 DBL_MIN Macro

Minimum double precision floating-point value.

Include

`<float.h>`

Value

The value 1.175494e-38 by default; 2.225074e-308 if the switch `-fno-short-double` is used.

Remarks

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

2.6.8 DBL_MIN_10_EXP Macro

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

Include

`<float.h>`

Value

The value -37 by default; -307 if the switch `-fno-short-double` is used.

Remarks

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

2.6.9 DBL_MIN_EXP Macro

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

Include

`<float.h>`

Value

The value -125 by default; -1021 if the switch `-fno-short-double` is used.

Remarks

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

2.6.10 FLT_DIG Macro

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

Include

`<float.h>`

Value

The value 6.

2.6.11 FLT_EPSILON Macro

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

Include

`<float.h>`

Value

The value 1.192093e-07.

2.6.12 FLT_MANT_DIG Macro

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

Include

`<float.h>`

Value

The value 24.

2.6.13 FLT_MAX Macro

Maximum finite single precision floating-point value.

Include

`<float.h>`

Value

The value 3.402823e+38.

2.6.14 FLT_MAX_10_EXP Macro

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

Include

`<float.h>`

Value

The value 38.

2.6.15 FLT_MAX_EXP Macro

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

Include

`<float.h>`

Value

The value 128.

2.6.16 FLT_MIN Macro

Minimum single precision floating-point value.

Include

`<float.h>`

Value

The value 1.175494e-38.

2.6.17 FLT_MIN_10_EXP Macro

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

Include

`<float.h>`

Value

The value -37.

2.6.18 FLT_MIN_EXP Macro

Minimum negative integer value for a single precision floating-point exponent in base `FLT_RADIX`.

Include

`<float.h>`

Value

The value -125.

2.6.19 FLT_RADIX Macro

Radix of the exponent representation.

Include

`<float.h>`

Value

2

Remarks

The value 2 (binary).

2.6.20 FLT_ROUNDS Macro

Represents the rounding mode for floating-point operations.

- | | |
|----|--------------------------------|
| -1 | indeterminable |
| 0 | toward zero |
| 1 | to nearest representable value |
| 2 | toward positive infinity |
| 3 | toward negative infinity |

Include

`<float.h>`

Value

1

Remarks

The value 1 (nearest representable value).

2.6.21 LDBL_DIG Macro

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

Include

`<float.h>`

Value

The value 15.

2.6.22 LDBL_EPSILON Macro

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

Include

`<float.h>`

Value

The value 2.220446e-16.

2.6.23 LDBL_MANT_DIG Macro

Number of base-`FLT_RADIX` digits in a long double precision floating-point significand.

Include

`<float.h>`

Value

The value 53.

2.6.24 LDBL_MAX Macro

Maximum finite long double precision floating-point value.

Include

`<float.h>`

Value

The value 1.797693e+308.

2.6.25 LDBL_MAX_10_EXP Macro

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

Include

`<float.h>`

Value

The value 308.

2.6.26 LDBL_MAX_EXP Macro

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

Include

`<float.h>`

Value

The value 1024.

2.6.27 LDBL_MIN Macro

Minimum long double precision floating-point value.

Include

`<float.h>`

Value

The value 2.225074e-308.

2.6.28 LDBL_MIN_10_EXP Macro

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

Include

`<float.h>`

Value

The value -307.

2.6.29 LDBL_MIN_EXP Macro

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

Include

`<float.h>`

Value

The value -1021.

2.7 <iso646.h> Alternate Spellings

The `<iso646.h>` header file consists of macros that can be used to replace the logical and bitwise operators.

2.7.1 iso6464 Alternate Spelling Macros

Macro Name	Definition
<code>and</code>	<code>& &</code>
<code>and_eq</code>	<code>&=</code>
<code>bitand</code>	<code>&</code>
<code>bitor</code>	<code> </code>
<code>compl</code>	<code>~</code>
<code>not</code>	<code>!</code>
<code>not_eq</code>	<code>!=</code>
<code>or</code>	<code> </code>
<code>or_eq</code>	<code> =</code>
<code>xor</code>	<code>^</code>
<code>xor_eq</code>	<code>^=</code>

2.8 <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.

Table 2-1. Declarations Provided by <limits.h>

Macro Name	Description	Value
<code>CHAR_BIT</code>	Number of bits to represent type <code>char</code>	8
<code>CHAR_MAX</code>	Maximum value of a <code>char</code>	127
<code>CHAR_MIN</code>	Minimum value of a <code>char</code>	-128
<code>INT_MAX</code>	Maximum value of a <code>int</code>	32767
<code>INT_MIN</code>	Minimum value of a <code>int</code>	-32768
<code>LLONG_MAX</code>	Maximum value of a long long <code>int</code>	9223372036854775807
<code>LLONG_MIN</code>	Minimum value of a long long <code>int</code>	-9223372036854775808
<code>LONG_MAX</code>	Maximum value of a long <code>int</code>	2147483647
<code>LONG_MIN</code>	Minimum value of a long <code>int</code>	-2147483648
<code>MB_LEN_MAX</code>	Maximum number of bytes in a multibyte character	1
<code>SCHAR_MAX</code>	Maximum value of a signed <code>char</code>	127
<code>SCHAR_MIN</code>	Minimum value of a signed <code>char</code>	-128
<code>SHRT_MAX</code>	Maximum value of a short <code>int</code>	32767
<code>SHRT_MIN</code>	Minimum value of a short <code>int</code>	-32768
<code>UCHAR_MAX</code>	Maximum value of an unsigned <code>char</code>	255
<code>UINT_MAX</code>	Maximum value of an unsigned <code>int</code>	65535
<code>ULLONG_MAX</code>	Maximum value of a long long unsigned <code>int</code>	18446744073709551615
<code>ULONG_MAX</code>	Maximum value of a long unsigned <code>int</code>	4294967295
<code>USHRT_MAX</code>	Maximum value of an unsigned short <code>int</code>	65535

2.9 <locale.h> Localization

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

- `struct lconv`
- `NULL`
- `LC_ALL`

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

Related Links[2.2 Multiple Header Types and Macros](#)

2.10 <setjmp.h> Non-Local Jumps

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

2.10.1 <setjmp.h> Types

jmp_buf Type

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

Include

```
<setjmp.h>
```

Prototype

```
typedef int jmp_buf[_NSETJMP];
```

Remarks

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

2.10.2 longjmp Function

A function that restores the environment saved by `setjmp`.

Include

```
<setjmp.h>
```

Prototype

```
void longjmp(jmp_buf env, int val);
```

Arguments

env variable where environment is stored.

val value to be returned to `setjmp` call.

Remarks

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

Example

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>

jmp_buf jfb;

void inner (void)
{
```

```

    longjmp(jb, 5);
}
int main (void)
{
    int i;
    if(i = setjmp(jb)) {
        printf("setjmp returned %d\n" i);
        exit(0);
    }
    printf("setjmp returned 0 - good\n");
    printf("calling inner...\n");
    inner();
    printf("inner returned - bad!\n");
}

```

Example Output

```

setjmp returned 0 - good
calling inner...

```

2.10.3 setjmp Function

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

Include

`<setjmp.h>`

Prototype

```
#define setjmp(jmp_buf env)
```

Argument

env variable where environment is stored

Return Value

If the return is from a direct call, `setjmp` returns zero. If the return is from a call to `longjmp`, `setjmp` returns a non-zero value.

Note: If the argument `val` from `longjmp` is 0, `setjmp` returns 1.

Example

See `longjmp`.

2.11 <signal.h> Signal Handling

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

A signal may be handled by:

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

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

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

The following type is included in `signal.h`.

sig_atomic_t - A type used by a signal handler. Prototype: `typedef int sig_atomic_t;`

The following argument and return value macros is included in `signal.h`.

SIG_DFL - 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.

SIG_ERR - Used as the return value for `signal` when it cannot complete a request due to an error.

SIG_IGN - Used as the second argument and/or the return value for `signal` to specify that the signal should be ignored.

2.11.1 SIGABRT Macro

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`. The default `raise` behavior (action identified by **SIG_DFL**) is to output to the standard error stream:

```
abort - terminating
```

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

Example

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

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

Example Output

```
ABRT
```

where **ABRT** stands for “abort.”

2.11.2 SIGFPE Macro

Signals floating-point errors 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`. When used, the default behavior is to print an arithmetic error message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user-defined function.

Example

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

int main(void)
{
    raise(SIGFPE);
}
```

```
    printf("Program never reaches here");
}
```

Example Output

```
FPE
```

where `FPE` stands for “floating-point error.”

2.11.3 SIGILL Macro

Signals illegal instruction.

Include

```
<signal.h>
```

Prototype

```
#define SIGILL
```

Remarks

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

Example

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

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

Example Output

```
ILL
```

where `ILL` stands for “illegal instruction.”

2.11.4 SIGINT Macro

Interrupt signal.

Include

```
<signal.h>
```

Prototype

```
#define SIGINT
```

Remarks

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

Example

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

int main(void)
{
    raise(SIGINT);
}
```

```
    printf("Program never reaches here.");  
}
```

Example Output

```
INT
```

where `INT` stands for “interruption.”

2.11.5 SIGSEGV Macro

Signals invalid access to storage.

Include

```
<signal.h>
```

Prototype

```
#define SIGSEGV
```

Remarks

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

Example

```
#include <signal.h> /* for raise, SIGSEGV */  
#include <stdio.h> /* for printf */  
  
int main(void)  
{  
    raise(SIGSEGV);  
    printf("Program never reaches here.");  
}
```

Example Output

```
SEGV
```

where `SEGV` stands for “invalid storage access.”

2.11.6 SIGTERM Macro

Signals a termination request.

Include

```
<signal.h>
```

Prototype

```
#define SIGTERM
```

Remarks

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

Example

```
#include <signal.h> /* for raise, SIGTERM */  
#include <stdio.h> /* for printf */  
  
int main(void)  
{  
    raise(SIGTERM);  
}
```

```
    printf("Program never reaches here.");
}
```

Example Output

```
TERM
```

where `TERM` stands for “termination request.”

2.11.7 raise Function

Reports a synchronous signal.

Include

```
<signal.h>
```

Prototype

```
int raise(int sig);
```

Argument

sig signal name

Return Value

Returns a 0 if successful; otherwise, returns a non-zero value.

Remarks

`raise` sends the signal identified by `sig` to the executing program.

Example

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

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

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

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

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

Example Output

```
Illegal instruction executed
```

Example Explanation

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

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

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

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

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

2.11.8 signal Function

Controls interrupt signal handling.

Include

`<signal.h>`

Prototype

```
void (*signal(int sig, void(*func)(int)))(int);
```

Arguments

sig	signal name
func	function to be executed

Return Value

Returns the previous value of `func`.

Example

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

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

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

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

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

Example Output

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

Example Explanation

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

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

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

2.12 <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.

2.12.1 va_list Type

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

Include

`<stdarg.h>`

Example

See `va_arg`.

2.12.2 va_arg Macro

Gets the current argument.

Include

`<stdarg.h>`

Prototype

```
#define va_arg(va_list ap, Ty)
```

Arguments

ap	pointer to list of arguments
Ty	type of argument to be retrieved

Return Value

Returns the current argument

Remarks

`va_start` must be called before `va_arg`.

Example

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

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

```

va_start(ap, fmt);
while (*fmt)
{
    switch (*fmt)
    {
        case '%':
            fmt++;
            if (*fmt == 'd')
            {
                int d = va_arg(ap, int);
                printf("<%d> is an integer\n", d);
            }
            else if (*fmt == 's')
            {
                char *s = va_arg(ap, char*);
                printf("<%s> is a string\n", s);
            }
            else
            {
                printf("%%%c is an unknown format\n",
                    *fmt);
            }
            fmt++;
            break;
        default:
            printf("%c is unknown\n", *fmt);
            fmt++;
            break;
    }
}
va_end(ap);
}
int main(void)
{
    tprint("%d%s.%c", 83, "This is text.", 'a');
}

```

Example Output

```

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

```

2.12.3 va_end Macro

Ends the use of ap.

Include

<stdarg.h>

Prototype

```
#define va_end(va_list ap)
```

Argument

ap	pointer to list of arguments
-----------	------------------------------

Remarks

After a call to `va_end`, the argument list pointer, `ap`, is considered to be invalid. Further calls to `va_arg` should not be made until the next `va_start`. In the 16-bit compiler, `va_end` does nothing, so this call is not necessary but should be used for readability and portability.

Example

See `va_arg`.

2.12.4 va_start Macro

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)
```

Arguments

ap	pointer to list of arguments
last_arg	last named parameter before the optional arguments

Example

See `va_arg`.

2.13 <stddef.h> Common Definitions

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

Related Links

[2.2 Multiple Header Types and Macros](#)

2.14 <stdio.h> Input and Output

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

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

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

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

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

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

Compiler Options

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

- The `-msmart-io` option, when enabled, will attempt to convert `printf`, `scanf` and other functions that use the input output formats to an integer-only variant. The functionality is the same as that of the C standard forms,

minus the support for floating-point output. `-msmart-io=0` disables this feature and no conversion will take place. `-msmart-io=1` or `-msmart-io` (the default) will convert a function call if it can be proven that an I/O function will never be presented with a floating-point conversion. `-msmart-io=2` is more optimistic than the default and will assume that non-constant format strings or otherwise unknown format strings will not contain a floating-point format. In the event that `-msmart-io=2` is used with a floating-point format, the format letter will appear as literal text and its corresponding argument will not be consumed.

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

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

Customizing STDIO

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

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

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

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

A complete generic solution might be:

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

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

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

In `write()`, you would write:

```
int __attribute__((__section__(".libc.write")))
write(int handle, void *buffer, unsigned int len) {
    int i;
    volatile UxMODEBITS *umode = &U1MODEbits;
    volatile UxSTABITS *ustatus = &U1STABits;
    volatile unsigned int *txreg = &U1TXREG;
    volatile unsigned int *brg = &U1BRG;
```

```

switch (handle)
{
default:
case 0:
case 1:
case 2:
    if ((C30_UART != 1) && (&U2BRG)) {
        umode = &U2MODEbits;
        ustatus = &U2STABits;
        txreg = &U2TXREG;
        brg = &U2BRG;
    }
    if ((umode->UARTEN) == 0)
    {
        *brg = 0;
        umode->UARTEN = 1;
    }
    if ((ustatus->UTXEN) == 0)
    {
        ustatus->UTXEN = 1;
    }
    for (i = len; i; --i)
    {
        while ((ustatus->TRMT) == 0);
        *txreg = *(char*)buffer++;
    }
    break;
case handle_can1: /* code to support can1 */
    break;
case handle_can2: /* code to support can2 */
    break;
case handle_spi1: /* code to support spi1 */
    break;
case handle_spi2: /* code to support spi2 */
    break;
}
return(len);
}

```

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

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

```

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

```

2.14.1 stdio File Macros

The following macros are included in `stdio.h`.

FILE	Stores information for a file stream.
fpos_t	Type of a variable used to store a file position.
EOF	A negative number indicating the end-of-file has been reached or to report an error condition. 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	Maximum number of characters in a filename including the null terminator. Value = 260.
FOPEN_MAX	Defines the maximum number of files that can be simultaneously open. <code>stderr</code> , <code>stdin</code> and <code>stdout</code> are included in the <code>FOPEN_MAX</code> count. Value = 8.

Related Links

[2.2 Multiple Header Types and Macros](#)

2.14.2 Standard Stream File Pointers

These pointers specify I/O in the standard stream.

2.14.2.1 stderr

File pointer to the standard error stream.

Include

<stdio.h>

2.14.2.2 stdin

File pointer to the standard input stream.

Include

<stdio.h>

2.14.2.3 stdout

File pointer to the standard output stream.

Include

<stdio.h>

2.14.3 clearerr Function

Resets the error indicator for the stream.

Include

<stdio.h>

Prototype

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

Argument

stream stream to reset error indicators

Remarks

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

Example

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

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

```

        printf("Error indicator reset\n");
    }
    fclose(myfile);
}

```

Example Output

```

Error
Error indicator reset

```

2.14.4 fclose Function

Close a stream.

Include

<stdio.h>

Prototype

```
int fclose(FILE *stream);
```

Argument

stream pointer to the stream to close

Return Value

Returns 0 if successful; otherwise, returns EOF if any errors were detected.

Remarks

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

Example

```

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

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

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

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

```

Example Output

```

afile1 was opened
afile1 was closed

```

2.14.5 fdopen Function

Associate a stream with a file descriptor.

Include

<stdio.h>

Prototype

```
FILE *fdopen(int filides, const char *mode);
```

Arguments

filides	file descriptor
mode	type of access permitted

Return Value

Returns a pointer to the open stream. If the function fails, a null pointer is returned.

Remarks

The following are types of file access:

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

This function requires a heap.

Example

```
#include <stdio.h> /* for fdopen, fclose, printf, FILE, NULL, EOF */
int main(void)
{
    FILE *myfile1, *myfile2;
    int y;
    if ((myfile1 = fdopen("afile1", "r")) == NULL)
        printf("Cannot open afile1\n");
    else
    {
        printf("afile1 was opened\n");
        y = fclose(myfile1);
        if (y == EOF)
            printf("afile1 was not closed\n");
        else
            printf("afile1 was closed\n");
    }
    if ((myfile1 = fdopen("afile1", "w+")) == NULL)
        printf("Second try, cannot open afile1\n");
    else
    {
        printf("Second try, afile1 was opened\n");
        y = fclose(myfile1);
        if (y == EOF)
            printf("afile1 was not closed\n");
        else
            printf("afile1 was closed\n");
    }
}
```

```

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

```

Example Output

```

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

```

Example Explanation

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

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

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

2.14.6 feof Function

Tests for end-of-file.

Include

<stdio.h>

Prototype

```
int feof(FILE *stream);
```

Argument

stream stream to check for end-of-file

Return Value

Returns non-zero if stream is at the end-of-file; otherwise, returns zero.

Example

```

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

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

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

```

```
}
}
```

Example Input

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

```
This is a sentence.
```

Example Output

```
This is a sentence.
```

2.14.7 ferror Function

Tests if error indicator is set.

Include

```
<stdio.h>
```

Prototype

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

Argument

stream pointer to `FILE` structure

Return Value

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

Example

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

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

int main(void)
{
    FILE *myfile;

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

        fclose(myfile);
    }
}
```

Example Output

```
Error
Error indicator reset
```

2.14.8 fflush Function

Flushes the buffer in the specified stream.

Include

```
<stdio.h>
```

Prototype

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

Argument

stream pointer to the stream to flush

Return Value

Returns `EOF` if a write error occurs; otherwise, returns zero for success.

Remarks

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

2.14.9 fgetc Function

Get a character from a stream

Include

```
<stdio.h>
```

Prototype

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

Argument

stream pointer to the open stream

Return Value

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

Remarks

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

Example

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

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

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

```

        printf("%c\\", y);
        y = fgetc(buf);
    }
    fclose(buf);
}

```

Example Input

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

```

Short
Longer string

```

Example Output

```

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

```

2.14.10 fgetpos Function

Gets the stream's file position.

Include

```
<stdio.h>
```

Prototype

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

Arguments

stream	target stream
pos	position-indicator storage

Return Value

Returns 0 if successful; otherwise, returns a non-zero value.

Remarks

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

Example

```

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

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

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

    if ((myfile = fopen("sampfgetpos.c", "rb")) ==
        NULL)

```

```

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

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

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

```

Example Output

```

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

```

2.14.11 fgets Function

Get a string from a stream.

Include

```
<stdio.h>
```

Prototype

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

Arguments

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

Return Value

Returns a pointer to the string s if successful; otherwise, returns a null pointer.

Remarks

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

Example

```

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

#define MAX 50

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

```

```

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

```

Example Input

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

```

Short
Longer string

```

Example Output

```

Short
|Longer string
|

```

2.14.12 fopen Function

Opens a file.

Include

`<stdio.h>`

Prototype

```
FILE *fopen(const char *filename, const char *mode);
```

Arguments

filename	name of the file
mode	type of access permitted

Return Value

Returns a pointer to the open stream. If the function fails, a null pointer is returned.

Remarks

The following are types of file access:

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

w+b or wb+	Opens an empty binary file for reading and writing (an existing file is overwritten.)
a+b or ab+	Opens a binary file for reading and appending (a file is created if it doesn't exist).

This function requires a heap.

Example

```
#include <stdio.h> /* for fopen, fclose, printf, FILE, NULL, EOF */
int main(void)
{
    FILE *myfile1, *myfile2;
    int y;
    if ((myfile1 = fopen("afile1", "r")) == NULL)
        printf("Cannot open afile1\n");
    else
    {
        printf("afile1 was opened\n");
        y = fclose(myfile1);
        if (y == EOF)
            printf("afile1 was not closed\n");
        else
            printf("afile1 was closed\n");
    }
    if ((myfile1 = fopen("afile1", "w+")) == NULL)
        printf("Second try, cannot open afile1\n");
    else
    {
        printf("Second try, afile1 was opened\n");
        y = fclose(myfile1);
        if (y == EOF)
            printf("afile1 was not closed\n");
        else
            printf("afile1 was closed\n");
    }
    if ((myfile2 = fopen("afile2", "a+")) == NULL)
        printf("Cannot open afile2\n");
    else
    {
        printf("afile2 was opened\n");
        y = fclose(myfile2);
        if (y == EOF)
            printf("afile2 was not closed\n");
        else
            printf("afile2 was closed\n");
    }
}
```

Example Output

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

Example Explanation

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

If the fopen function opens a file for writing (w+) it does not have to exist, it will be created and then opened. If the file does exist, it cannot be overwritten and will be appended.

2.14.13 fprintf Function

Prints formatted text 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; see "Remarks"

Return Value

Returns number of characters generated or a negative number if an error occurs.

Remarks

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

Example

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

int main(void)
{
    FILE *myfile;
    int y;
    char s[]="Print this string";
    int x = 1;
    char a = '\n';
    if ((myfile = fopen("afile", "w")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        y = fprintf(myfile, "%s %d time%c", s, x, a);

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

        fclose(myfile);
    }
}
```

Example Output

Number of characters printed to file = 25

Contents of afile:

Print this string 1 time

Related Links

[2.14.28 printf Function](#)

2.14.14 fputc Function

Puts a character to the stream.

Include

<stdio.h>

Prototype

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

Arguments

c	character to be written
stream	pointer to the open stream

Return Value

Returns the character written or `EOF` if a write error occurs.

Remarks

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

Example

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

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

    x = 0;

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

Example Output

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

2.14.15 fputs Function

Puts a string to the stream.

Include

`<stdio.h>`

Prototype

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

Arguments

s	string to be written
stream	pointer to the open stream

Return Value

Returns a non-negative value if successful; otherwise, returns `EOF`.

Remarks

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

Example

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

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

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

Example Output

```
This is text
|
```

2.14.16 fread Function

Reads data from the stream.

Include

```
<stdio.h>
```

Prototype

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

Arguments

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

Return Value

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

Remarks

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

Example

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

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

    if ((buf = fopen("afile.out", "w+")) != NULL) {
        for (x = 0; x < 10; x++) {
            nums[x] = 10.0/(x + 1);
            printf("10.0/%d = %f\n", x+1, nums[x]);
        }
        numwrote = fwrite(nums, sizeof(double), 10, buf);
        printf("Wrote %d numbers\n\n", numwrote);
        fclose(buf);
    }
    else printf("Cannot open afile.out\n");

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

Example Output

```

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

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

```

Example Explanation

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

2.14.17 freopen Function

Reassigns an existing stream to a new file.

Include

```
<stdio.h>
```

Prototype

```
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

Arguments

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

Return Value

Returns a pointer to the new open file. If the function fails a null pointer is returned.

Remarks

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

This function requires a heap.

Example

```

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

```

```

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

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

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

```

Example Output

```

afile1 was opened
afile2 was opened

```

Example Explanation

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

2.14.18 fscanf Function

Scans formatted text from a stream.

Include

<stdio.h>

Prototype

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

Arguments

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

Return Value

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

Remarks

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

Example

```

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

```

```

int main(void)
{
    FILE *myfile;
    char s[30];
    int x;
    char a;
    if ((myfile = fopen("afile", "w+")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        fprintf(myfile, "%s %d times%c",
                "Print this string", 100, '\n');

        fseek(myfile, 0L, SEEK_SET);

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

        fclose(myfile);
    }
}

```

Example Input

Contents of afile:

```
Print this string 100 times
```

Example Output

```
Print
this
string
100
times
```

2.14.19 fseek Function

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 set `errno`.

Remarks

`mode` can be one of the following:

SEEK_SET – seeks from the beginning of the file

SEEK_CUR – seeks from the current position of the file pointer

SEEK_END – seeks from the end of the file

Example

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

int main(void)
{
    FILE *myfile;
    char s[70];
    int y;
    myfile = fopen("afile.out", "w+");
    if (myfile == NULL)
        printf("Cannot open afile.out\n");
    else
    {
        fprintf(myfile, "This is the beginning, "
                    "this is the middle and "
                    "this is the end.");

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

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

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

Example Output

```
"This is the beginning"

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

"this is the end."
```

Example Explanation

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

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

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

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

2.14.20 `fseek` Macros

These macros are used by the function `fseek`.

2.14.20.1 `SEEK_CUR`

Indicates that `fseek` should seek from the current position of the file pointer.

Include

```
<stdio.h>
```

Example

See example for `fseek`.

2.14.20.2 `SEEK_END`

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

Include

```
<stdio.h>
```

Example

See example for `fseek`.

2.14.20.3 `SEEK_SET`

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

Include

```
<stdio.h>
```

Example

See example for `fseek`.

2.14.21 `fsetpos` Function

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 <code>fgetpos</code>

Return Value

Returns 0 if successful; otherwise, returns a non-zero value.

Remarks

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

Example

```

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

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

int main(void)
{
    FILE *myfile;
    fpos_t pos;
    char buf[25];
    if ((myfile = fopen("sampfgetpos.c", "rb")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fread(buf, sizeof(char), 8, myfile);
        if (fgetpos(myfile, &pos) != 0)
            perror("fgetpos error");
        else
        {
            fread(buf, sizeof(char), 21, myfile);
            printf("Bytes read: %.21s\n", buf);
            fread(buf, sizeof(char), 18, myfile);
            printf("Bytes read: %.18s\n", buf);
        }
        if (fsetpos(myfile, &pos) != 0)
            perror("fsetpos error");

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

```

Example Output

```

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

```

2.14.22 ftell Function

Gets the current position of a file pointer.

Include

```
<stdio.h>
```

Prototype

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

Argument

stream stream in which to get the current file position

Return Value

Returns the position of the file pointer if successful; otherwise, returns -1.

Example

```

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

int main(void)
{
    FILE *myfile;
    char s[75];
    long y;
    myfile = fopen("afile.out", "w+");
    if (myfile == NULL)
        printf("Cannot open afile.out\n");
    else
    {
        fprintf(myfile, "This is a very long sentence "
                    "for input into the file named "
                    "afile.out for testing.");

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

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

```

Example Output

```

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

```

2.14.23 fwrite Function

Writes data to the stream.

Include

<stdio.h>

Prototype

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

Arguments

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

Return Value

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

Remarks

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

Example

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

int main(void)
{
    FILE *buf;
    int x, numwrote, numread;
    double nums[10], readnums[10];
    if ((buf = fopen("afile.out", "w+")) != NULL)
    {
        for (x = 0; x < 10; x++)
        {
            nums[x] = 10.0/(x + 1);
            printf("10.0/%d = %f\n", x+1, nums[x]);
        }

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

Example Output

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

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

Example Explanation

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

2.14.24 getc Function

Get a character from the stream.

Include

```
<stdio.h>
```

Prototype

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

Argument

stream pointer to the open stream

Return Value

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

Remarks

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

Example

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

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

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

Example Input

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

```
Short
Longer string
```

Example Output

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

2.14.25 getchar Function

Get a character from `stdin`.

Include

`<stdio.h>`

Prototype

```
int getchar(void);
```

Return Value

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

Remarks

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

Example

```
#include <stdio.h>

int main(void)
{
    char y;

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

Example Input

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

```
Short
Longer string
```

Example Output

```
S|h|o|r|t|
```

2.14.26 gets Function

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.

Example

```
#include <stdio.h>

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

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

Example Input

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

```
Short
Longer string
```

Example Output

```
Text: Short
```

2.14.27 perror Function

Prints an error message to `stderr`.

Include

```
<stdio.h>
```

Prototype

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

Argument

s string to print

Return Value

None.

Remarks

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

Example

```
#include <stdio.h>

int main(void)
{
    FILE *myfile;

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

    fclose(myfile);
}
```

Example Output

```
Cannot open samp.fil:  file open error
```

2.14.28 printf Function

Prints formatted text to `stdout`.

Include

```
<stdio.h>
```

Prototype

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

Arguments

format	format control string
...	optional arguments; see "Remarks"

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.

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.
ll modifier	Used with s; specifies that the string pointer is an __eds__ pointer. This is an MPLAB XC16 extension.
L modifier	Used with e, E, f, g, G; converts the value to a long double.

type

d, i	signed int.
o	unsigned int in octal.
u	unsigned int in decimal.
x	unsigned int in lowercase hexadecimal.
X	unsigned int in uppercase hexadecimal.
e, E	double in scientific notation.
f	double decimal notation.
g, G	double (takes the form of e, E or f as appropriate).
c	char - a single character.
s	string.
p	value of a pointer.
n	The associated argument shall be an integer pointer into which is placed the number of characters written so far. No characters are printed.
%	A % character is printed.

This function requires a heap.

Example

```
#include <stdio.h>

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

    /* print an integer, left justified (as      */
    /* specified by the minus sign in the format */
    /* string) in a 4 character space. Print a   */
    /* second integer that is right justified in */
}
```



```

/* a 4 character space using the pipe (|) as */
/* a separator between the integers.      */
printf("%-4d|%4d\n", -4, 4);

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

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

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

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

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

```

Example Output

```

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

```

2.14.29 putc Function

Puts a character to the stream.

Include

```
<stdio.h>
```

Prototype

```
int putc(int c, FILE *stream);
```

Arguments

c	character to be written
stream	pointer to FILE structure

Return Value

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

Remarks

putc is the same as the function fputc.

Example

```

#include <stdio.h>

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

    x = 0;

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

```

```
    putc('|', stdout);
}
}
```

Example Output

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

2.14.30 putchar Function

Put a character to `stdout`.

Include

`<stdio.h>`

Prototype

```
int putchar(int c);
```

Argument

c character to be written

Return Value

Returns the character or `EOF` if an error occurs or end-of-file is reached.

Remarks

Same effect as `fputc` with `stdout` as an argument.

Example

```
#include <stdio.h>

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

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

Example Output

```
This is text
```

2.14.31 puts Function

Put a string to `stdout`.

Include

`<stdio.h>`

Prototype

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

Argument

s string to be written

Return Value

Returns a non-negative value if successful; otherwise, returns `EOF`.

Remarks

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

Example

```
#include <stdio.h>

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

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

Example Output

```
This is text
|
```

2.14.32 remove Function

Deletes the specified file.

Include

`<stdio.h>`

Prototype

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

Argument

filename name of file to be deleted

Return Value

Returns 0 if successful; otherwise, returns -1.

Remarks

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

Example

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

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

Example Output

```
File removed
```

2.14.33 rename Function

Renames the specified file.

Include

`<stdio.h>`

Prototype

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

Arguments

old	pointer to the old name
new	pointer to the new name

Return Value

Return 0 if successful; otherwise, returns a non-zero value.

Remarks

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

Example

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

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

Example Output

```
File renamed
```

2.14.34 rewind Function

Resets the file pointer to the beginning of the file.

Include

```
<stdio.h>
```

Prototype

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

Argument

stream	stream to reset the file pointer
---------------	----------------------------------

Remarks

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

Example

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

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

    if ((myfile = fopen("afile", "w+")) == NULL)
        printf("Cannot open afile\n");
    else
    {
```

```

    fprintf(myfile, "%d %s", x, s);
    printf("I have %d %s.\n", x, s);

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

    fclose(myfile);
}

```

Example Output

```

I have 10 cookies.
I ate 10 cookies.

```

2.14.35 scanf Function

Scans formatted text from `stdin`.

Include

`<stdio.h>`

Prototype

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

Arguments

format	format control string
...	optional arguments

Return Value

Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if an input failure is encountered before the first.

Remarks

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

`%[*][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.

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.

Example

For MPLAB X Simulator:

```
#include <stdio.h>
#include <libpic30.h>

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

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

    printf("Number of items scanned = %d\n", items);
    printf("Favorite number = %d, ", number);
    printf("Favorite letter = %c\n", letter);
    printf("Favorite color = %s, ", color);
    printf("Desired salary = $%.2f\n", salary);
    printf("Social Security Number = %s, ", string);
}
// If not using the simulator, remove these lines:
// #include <libpic30.h>
// __attach_input_file("uart_in.txt");
```

Example Input

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

```
5 T Green 300000 123-45-6789
```

Example Output

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

2.14.36 setbuf Function

Defines how a stream is buffered.

Include

```
<stdio.h>
```

Prototype

```
void setbuf(FILE *stream, char *buf);
```

Arguments

stream	pointer to the open stream
buf	user allocated buffer

Remarks

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

This function requires a heap.

Example

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

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

    if ((myfile1 = fopen("afile1", "w+")) != NULL)
    {
        setbuf(myfile1, NULL);
        printf("myfile1 has no buffering\n");
        fclose(myfile1);
    }

    if ((myfile2 = fopen("afile2", "w+")) != NULL)
    {
        setbuf(myfile2, buf);
        printf("myfile2 has full buffering");
        fclose(myfile2);
    }
}
```

Example Output

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

This macro is used by the function `setbuf`.

2.14.36.1 BUFSIZ

Defines the size of the buffer used.

Include

`<stdio.h>`

Value

512

2.14.37 setvbuf Function

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

This function requires a heap.

Example

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

int main(void)
{
    FILE *myfile1, *myfile2;
    char buf[256];
    if ((myfile1 = fopen("afile1", "w+")) != NULL)
    {
        if (setvbuf(myfile1, NULL, _IONBF, 0) == 0)
            printf("myfile1 has no buffering\n");
        else
            printf("Unable to define buffer stream "
                  "and/or size\n");
    }
    fclose(myfile1);
    if ((myfile2 = fopen("afile2", "w+")) != NULL)
    {
        if (setvbuf(myfile2, buf, _IOFBF, sizeof(buf)) ==
```



```

    0)
    printf("myfile2 has a buffer of %d "
           "characters\n", sizeof(buf));
    else
    printf("Unable to define buffer stream "
           "and/or size\n");
}
fclose(myfile2);
}

```

Example Output

```

myfile1 has no buffering
myfile2 has a buffer of 256 characters

```

2.14.38 setvbuf Macros

These macros are used by the function `setvbuf`.

2.14.38.1 _IOFBF

Indicates full buffering.

Include

`<stdio.h>`

2.14.38.2 _IOLBF

Indicates line buffering.

Include

`<stdio.h>`

2.14.38.3 _IONBF

Indicates no buffering.

Include

`<stdio.h>`

2.14.39 sprintf Function

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`.

Example

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

```

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

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

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

```

Example Output

```

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

```

Related Links

[2.14.28 printf Function](#)

2.14.40 sscanf Function

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`.

Example

```

#include <stdio.h>

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

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

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

```

Example Output

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

2.14.41 tmpfileprintf Function

Creates a temporary file.

Include

<stdio.h>

Prototype

```
FILE *tmpfile(void)
```

Return Value

Returns a stream pointer if successful; otherwise, returns a `NULL` pointer.

Remarks

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

Example

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

int main(void)
{
    FILE *mytmpfile;

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

Example Output

```
Temporary file was created
```

2.14.42 tmpnam Function

Creates a unique temporary filename.

Include

<stdio.h>

Prototype

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

Argument

s pointer to the temporary name

Return Value

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

Remarks

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

Example

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

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

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

Example Output

```
Temporary file ctm00001.tmp was created
```

2.14.43 tmpnam Macros

These macros are used by the function `tmpnam`.

2.14.43.1 L_tmpnam

Defines the number of characters for the longest temporary file name created by the function `tmpnam`.

Include

<stdio.h>

Value

16

Remarks

`L_tmpnam` is used to define the size of the array used by `tmpnam`.

2.14.43.2 TMP_MAX

The maximum number of unique file names the function `tmpnam` can generate.

Include

<stdio.h>

Value

32

2.14.44 ungetc Function

Pushes character back onto stream.

Include

<stdio.h>

Prototype

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

Arguments

c character to be pushed back

stream pointer to the open stream

Return Value

Returns the pushed character if successful; otherwise, returns EOF.

Remarks

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

Example

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

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

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

Example Input

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

```
Short
Longer string
```

Example Output

```
Sho2rt
Longe2r st2ring
```

2.14.45 vasprintf Function

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

Include

`<stdio.h>`

`<stdarg.h>`

Prototype

```
int vasprintf(char **s, const char *format, va_list ap);
```

Arguments

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

Return Value

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

Remarks

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

To access the variable length argument list, the *ap* variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vasprintf` function is called. Invoke `va_end` after the function returns. For more details, see `stdarg.h`.

This function requires a heap.

Example

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

void errmsg(const char *fmt, ...)
{
    va_list ap;
    char buf[100];
    char **bufptr;
    bufptr = &buf;

    va_start(ap, fmt);
    vasprintf(bufptr, fmt, ap);
    va_end(ap);
    printf("Error: %s", buf);
}

int main(void)
{
    int num = 3;

    errmsg("The letter '%c' is not %s\n", 'a',
           "an integer value.");
    errmsg("Requires %d%s\n", num,
           " or more characters.\n");
}
```

Example Output

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

Related Links

[2.14.28 printf Function](#)

2.14.46 vfprintf Function

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 `stdarg.h`.

This function requires a heap.

Example

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

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

FILE *myfile;

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

    va_start(ap, fmt);
    vfprintf(myfile, fmt, ap);
    va_end(ap);
}

int main(void)
{
    int num = 3;

    if ((myfile = fopen("afile.txt", "w")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        errmsg("Error: The letter '%c' is not %s\n", 'a',
              "an integer value.");
        errmsg("Error: Requires %d%s%c", num,
              " or more characters.", '\n');
    }
    fclose(myfile);
}
```

Example Output

Contents of `afile.txt`:

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

Related Links

[2.14.28 printf Function](#)

2.14.47 vprintf Function

Prints formatted text to `stdout` using a variable length argument list.

Include

```
<stdio.h>
<stdarg.h>
```

Prototype

```
int vprintf(const char *format, va_list ap);
```

Arguments

format	format control string
ap	pointer to a list of arguments

Return Value

Returns number of characters generated or a negative number if an error occurs.

Remarks

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

To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vprintf` function is called. Invoke `va_end` after the function returns. For more details, see `stdarg.h`

Example

```
#include <stdio.h> /* for vprintf, printf */
#include <stdarg.h> /* for va_start,      */
/* va_list, va_end */
void errmsg(const char *fmt, ...)
{
    va_list ap;

    va_start(ap, fmt);
    printf("Error: ");
    vprintf(fmt, ap);
    va_end(ap);
}
int main(void)
{
    int num = 3;

    errmsg("The letter '%c' is not %s\n", 'a',
           "an integer value.");
    errmsg("Requires %d%s\n", num,
           " or more characters.\n");
}
```

Example Output

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

Related Links

[2.14.28 printf Function](#)

2.14.48 vsprintf Function

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

Include

```
<stdio.h>
```

```
<stdarg.h>
```

Prototype

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

Arguments

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

Return Value

Returns number of characters stored in s excluding the terminating null character.

Remarks

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

To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vsprintf` function is called. Invoke `va_end` after the function returns. For more details, see `stdarg.h`.

This function requires a heap.

Example

```
#include <stdio.h> /* for vsprintf, printf */
#include <stdarg.h> /* for va_start, */
/* va_list, va_end */
void errmsg(const char *fmt, ...)
{
    va_list ap;
    char buf[100];

    va_start(ap, fmt);
    vsprintf(buf, fmt, ap);
    va_end(ap);
    printf("Error: %s", buf);
}
int main(void)
{
    int num = 3;

    errmsg("The letter '%c' is not %s\n", 'a',
           "an integer value.");
    errmsg("Requires %d%s\n", num,
           " or more characters.\n");
}
```

Example Output

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

Related Links

[2.14.28 printf Function](#)

2.15 <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.

2.15.1 **stdlib.h** Types and Macros

The following types are included in `stdlib.h`:

- `div_t`
- `ldiv_t`
- `wchar_t`

The following macros are included in `stdlib.h`:

- `EXIT_FAILURE`
- `EXIT_SUCCESS`
- `MB_CUR_MAX`
- `RAND_MAX`

Related Links

[2.2 Multiple Header Types and Macros](#)

2.15.1.1 **div_t**

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

Prototype

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

Remarks

This is the structure type returned by the function, `div`.

2.15.1.2 **ldiv_t**

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

Prototype

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

Remarks

This is the structure type returned by the function, `ldiv`.

2.15.1.3 **wchar_t**

A type that holds a wide character value. In `stdlib.h` and `stddef.h`.

2.15.1.4 **EXIT_FAILURE**

Reports unsuccessful termination.

Remarks

`EXIT_FAILURE` is a value for the `exit` function to return an unsuccessful termination status.

Example

See `exit` for example of use.

2.15.1.5 **EXIT_SUCCESS**

Reports successful termination.

Remarks

`EXIT_SUCCESS` is a value for the `exit` function to return a successful termination status.

Example

See `exit` for example of use.

2.15.1.6 **MB_CUR_MAX**

Maximum number of characters in a multibyte character.

Value

1

2.15.1.7 RAND_MAX

Maximum value capable of being returned by the rand function.

Value

32767

2.15.2 abort Function

Aborts the current process.

Include

<stdlib.h>

Prototype

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

Remarks

abort will cause the processor to reset.

Example

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

int main(void)
{
    FILE *myfile;

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

    fclose(myfile);
}
```

Example Output

```
Cannot open samp.fil
ABRT
```

2.15.3 abs Function

Calculates the absolute value.

Include

<stdlib.h>

Prototype

```
int abs(int i);
```

Argument

i integer value

Return Value

Returns the absolute value of *i*.

Remarks

A negative number is returned as positive; a positive number is unchanged.

Example

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

int main(void)
{
    int i;

    i = 12;
    printf("The absolute value of  %d is  %d\n", i, abs(i));

    i = -2;
    printf("The absolute value of  %d is  %d\n", i, abs(i));

    i = 0;
    printf("The absolute value of  %d is  %d\n", i, abs(i));
}
```

Example Output

```
The absolute value of  12 is  12
The absolute value of  -2 is   2
The absolute value of   0 is   0
```

2.15.4 atexit Function

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

Include

```
<stdlib.h>
```

Prototype

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

Argument

func function to be called

Return Value

Returns a zero if successful; otherwise, returns a non-zero value.

Remarks

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

Example

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

void good_msg(void);
void bad_msg(void);
void end_msg(void);

int main(void)
{
    int number;

    atexit(end_msg);
    printf("Enter your favorite number:");
    scanf("%d", &number);
    printf(" %d\n", number);
    if (number == 5)
    {
        printf("Good Choice\n");
        atexit(good_msg);
        exit(0);
    }
    else
```

```

    {
        printf("%d!?\n", number);
        atexit(bad_msg);
        exit(0);
    }
}

void good_msg(void)
{
    printf("That's an excellent number\n");
}

void bad_msg(void)
{
    printf("That's an awful number\n");
}

void end_msg(void)
{
    printf("Now go count something\n");
}

```

Example Input 1

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

```
5
```

Example Output 1

```

Enter your favorite number: 5
Good Choice
That's an excellent number
Now go count something

```

Example Input 2

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

```
42
```

Example Output 2

```

Enter your favorite number: 42
42!?
That's an awful number
Now go count something

```

2.15.5 atof Function

Converts a string to a double precision floating-point value.

Include

```
<stdlib.h>
```

Prototype

```
double atof(const char *s);
```

Argument

s pointer to the string to be converted

Return Value

Returns the converted value if successful; otherwise, returns 0.

Remarks

The number may consist of the following:

[whitespace] [sign] digits [.digits] [{ e | E } [sign] digits]

Optional whitespace followed by an optional sign, then a sequence of one or more digits with an optional decimal point, followed by one or more optional digits and an optional `e` or `E` followed by an optional signed exponent. The conversion stops when the first unrecognized character is reached. The conversion is the same as `strtod(s, 0)` except it does no error checking so `errno` will not be set.

Example

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

int main(void)
{
    char a[] = " 1.28";
    char b[] = "27.835e2";
    char c[] = "Number1";
    double x;

    x = atof(a);
    printf("String = \"%s\" float = %f\n", a, x);

    x = atof(b);
    printf("String = \"%s\" float = %f\n", b, x);

    x = atof(c);
    printf("String = \"%s\" float = %f\n", c, x);
}
```

Example Output

```
String = "1.28"      float = 1.280000
String = "27.835:e2" float = 2783.500000
String = "Number1"  float = 0.000000
```

2.15.6 atoi Function

Converts a string to an integer.

Include

`<stdlib.h>`

Prototype

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

Argument

s string to be converted

Return Value

Returns the converted integer if successful; otherwise, returns 0.

Remarks

The number may consist of the following:

[whitespace] [sign] digits

Optional whitespace followed by an optional sign, then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to `(int) strtol(s, 0, 10)`, except it does no error checking so `errno` will not be set.

Example

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

int main(void)
{
```

```

char a[] = " -127";
char b[] = "Number1";
int x;

x = atoi(a);
printf("String = \"%s\" \tint = %d\n", a, x);

x = atoi(b);
printf("String = \"%s\" \tint = %d\n", b, x);
}

```

Example Output

```

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

```

2.15.7 atol Function

Converts a string to a long integer.

Include

<stdlib.h>

Prototype

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

Argument

s string to be converted

Return Value

Returns the converted long integer if successful; otherwise, returns 0.

Remarks

The number may consist of the following:

[whitespace] [sign] digits

Optional whitespace followed by an optional sign, then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to `(int) strtol(s, 0, 10)`, except it does no error checking so `errno` will not be set.

Example

```

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char a[] = " -123456";
    char b[] = "2Number";
    long x;

    x = atol(a);
    printf("String = \"%s\"   int = %ld\n", a, x);

    x = atol(b);
    printf("String = \"%s\"   int = %ld\n", b, x);
}

```

Example Output

```

String = " -123456"    int = -123456
String = "2Number"     int = 2

```

2.15.8 bsearch Function

Performs a binary search.

Include

```
<stdlib.h>
```

Prototype

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

Arguments

key	object to search for
base	pointer to the start of the search data
nelem	number of elements
size	size of elements
cmp	pointer to the comparison function

Arguments to the comparison function are as follows.

ck	pointer to the key for the search
ce	pointer to the element being compared with the key

Return Value

Returns a pointer to the object being searched for if found; otherwise, returns `NULL`.

Remarks

The value returned by the compare function is <0 if `ck` is less than `ce`, 0 if `ck` is equal to `ce` or >0 if `ck` is greater than `ce`.

In the following example, `qsort` is used to sort the list before `bsearch` is called. `bsearch` requires the list to be sorted according to the comparison function. This `comp` uses ascending order.

Example

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

#define NUM 7

int comp(const void *e1, const void *e2);

int main(void)
{
    int list[NUM] = {35, 47, 63, 25, 93, 16, 52};
    int x, y;
    int *r;

    qsort(list, NUM, sizeof(int), comp);

    printf("Sorted List:  ");
    for (x = 0; x < NUM; x++)
        printf("%d ", list[x]);

    y = 25;
    r = bsearch(&y, list, NUM, sizeof(int), comp);
    if (r)
        printf("\nThe value %d was found\n", y);
    else
        printf("\nThe value %d was not found\n", y);

    y = 75;
    r = bsearch(&y, list, NUM, sizeof(int), comp);
    if (r)
```



```

    printf("\nThe value %d was found\n", y);
else
    printf("\nThe value %d was not found\n", y);
}

int comp(const void *e1, const void *e2)
{
    const int * a1 = e1;
    const int * a2 = e2;

    if (*a1 < *a2)
        return -1;
    else if (*a1 == *a2)
        return 0;
    else
        return 1;
}

```

Example Output

```

Sorted List:  16  25  35  47  52  63  93
The value 25 was found

The value 75 was not found

```

2.15.9 calloc Function

Allocates an array in memory and initializes the elements to 0.

Include

```
<stdlib.h>
```

Prototype

```
void *calloc(size_t nelem, size_t size);
```

Arguments

nelem	number of elements
size	length of each element

Return Value

Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

Remarks

Memory returned by `calloc` is aligned correctly for any size data element and is initialized to zero. This function requires a heap.

Example

```

/* This program allocates memory for the      */
/* array 'i' of long integers and initializes */
/* them to zero.                             */
/*
#include <stdio.h> /* for printf, NULL */
#include <stdlib.h> /* for calloc, free */

int main(void)
{
    int x;
    long *i;

    i = (long *)calloc(5, sizeof(long));
    if (i != NULL)
    {
        for (x = 0; x < 5; x++)
            printf("i[%d] = %ld\n", x, i[x]);
        free(i);
    }
}

```

```

    }
    else
        printf("Cannot allocate memory\n");
}

```

Example Output

```

i[0] = 0
i[1] = 0
i[2] = 0
i[3] = 0
i[4] = 0

```

2.15.10 div Function

Calculates the quotient and remainder of two numbers.

Include

<stdlib.h>

Prototype

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

Arguments

numer	numerator
denom	denominator

Return Value

Returns the quotient and the remainder.

Remarks

The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator ($\text{quot} * \text{denom} + \text{rem} = \text{numer}$). Division by zero will invoke the math exception error, which, by default, will cause a Reset. Write a math error handler to do something else.

Example

```

#include <stdlib.h>
#include <stdio.h>

void __attribute__((__interrupt__))
_MathError(void)
{
    printf("Illegal instruction executed\n");
    abort();
}

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

    x = 7;
    y = 3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\n", z.quot, z.rem);

    x = 7;
    y = -3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\n", z.quot, z.rem);
}

```

```

x = -5;
y = 3;
printf("For div(%d, %d)\n", x, y);
z = div(x, y);
printf("The quotient is %d and the "
      "remainder is %d\n\n", z.quot, z.rem);

x = 7;
y = 7;
printf("For div(%d, %d)\n", x, y);
z = div(x, y);
printf("The quotient is %d and the "
      "remainder is %d\n\n", z.quot, z.rem);

x = 7;
y = 0;
printf("For div(%d, %d)\n", x, y);
z = div(x, y);
printf("The quotient is %d and the "
      "remainder is %d\n\n", z.quot, z.rem);
}

```

Example Output

```

For div(7, 3)
The quotient is 2 and the remainder is 1

For div(7, -3)
The quotient is -2 and the remainder is 1

For div(-5, 3)
The quotient is -1 and the remainder is -2

For div(7, 7)
The quotient is 1 and the remainder is 0

For div(7, 0)
Illegal instruction executed
ABRT

```

2.15.11 exit Function

Terminates program after clean up.

Include

```
<stdlib.h>
```

Prototype

```
void exit(int status);
```

Argument

status	exit status
---------------	-------------

Remarks

`exit` calls any functions registered by `atexit` in reverse order of registration, flushes buffers, closes stream, closes any temporary files created with `tmpfile` and resets the processor. This function is customizable. See `pic30-libs`.

Example

```

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samp.fil", "r" )) == NULL)
    {

```

```

    printf("Cannot open samp.fil\n");
    exit(EXIT_FAILURE);
}
else
{
    printf("Success opening samp.fil\n");
    exit(EXIT_SUCCESS);
}
printf("This will not be printed");
}

```

Example Output

```
Cannot open samp.fil
```

2.15.12 free Function

Frees memory.

Include

<stdlib.h>

Prototype

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

Argument

ptr points to memory to be freed

Remarks

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

Example

```

#include <stdio.h> /* for printf, sizeof, */
                /* NULL */
#include <stdlib.h> /* for malloc, free */

int main(void)
{
    long *i;

    if ((i = (long *)malloc(50 * sizeof(long))) ==
        NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        free(i);
        printf("Memory freed\n");
    }
}

```

Example Output

```
Memory allocated
Memory freed
```

2.15.13 getenv Function

Get a value for an environment variable.

Include

<stdlib.h>

Prototype

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

Argument

name name of environment variable

Return Value

Returns a pointer to the value of the environment variable if successful; otherwise, returns a null pointer.

Remarks

This function must be customized to be used as described (see `pic30-libs`). By default, there are no entries in the environment list for `getenv` to find.

Example

```
#include <stdio.h> /* for printf, NULL */
#include <stdlib.h> /* for getenv */

int main(void)
{
    char *incvar;

    incvar = getenv("INCLUDE");
    if (incvar != NULL)
        printf("INCLUDE environment variable = %s\n",
              incvar);
    else
        printf("Cannot find environment variable "
              "INCLUDE ");
}
```

Example Output

```
Cannot find environment variable INCLUDE
```

2.15.14 labs Function

Calculates the absolute value of a long integer.

Include

```
<stdlib.h>
```

Prototype

```
long labs(long i);
```

Argument

i long integer value

Return Value

Returns the absolute value of *i*.

Remarks

A negative number is returned as positive; a positive number is unchanged.

Example

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

int main(void)
{
    long i;
```

```

    i = 123456;
    printf("The absolute value of %7ld is %6ld\n",
           i, labs(i));

    i = -246834;
    printf("The absolute value of %7ld is %6ld\n",
           i, labs(i));

    i = 0;
    printf("The absolute value of %7ld is %6ld\n",
           i, labs(i));
}

```

Example Output

```

The absolute value of  123456 is 123456
The absolute value of -246834 is 246834
The absolute value of      0 is      0

```

2.15.15 ldiv Function

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 ($\text{quot} * \text{denom} + \text{rem} = \text{numer}$). If the denominator is zero, the behavior is undefined.

Example

```

#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    long x,y;
    ldiv_t z;

    x = 7;
    y = 3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = -3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = -5;
    y = 3;

```

```

printf("For ldiv(%ld, %ld)\n", x, y);
z = ldiv(x, y);
printf("The quotient is %ld and the "
      "remainder is %ld\n\n", z.quot, z.rem);

x = 7;
y = 7;
printf("For ldiv(%ld, %ld)\n", x, y);
z = ldiv(x, y);
printf("The quotient is %ld and the "
      "remainder is %ld\n\n", z.quot, z.rem);

x = 7;
y = 0;
printf("For ldiv(%ld, %ld)\n", x, y);
z = ldiv(x, y);
printf("The quotient is %ld and the "
      "remainder is %ld\n\n",
      z.quot, z.rem);
}

```

Example Output

```

For ldiv(7, 3)
The quotient is 2 and the remainder is 1

For ldiv(7, -3)
The quotient is -2 and the remainder is 1

For ldiv(-5, 3)
The quotient is -1 and the remainder is -2

For ldiv(7, 7)
The quotient is 1 and the remainder is 0

For ldiv(7, 0)
The quotient is -1 and the remainder is 7

```

Example Explanation

In the last example (`ldiv(7, 0)`) the denominator is zero, the behavior is undefined.

2.15.16 malloc Function

Allocates memory.

The default implementation of `malloc` will require an additional 4 bytes of heap memory per allocation.

The legacy library's `malloc` will use an additional 2 bytes of heap memory per allocation.

Include

```
<stdlib.h>
```

Prototype

```
void *malloc(size_t size);
```

Argument

size number of characters to allocate

Return Value

Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

Remarks

`malloc` does not initialize memory it returns. This function requires a heap.

Example

```
#include <stdio.h> /* for printf, sizeof, */
                  /* NULL */
#include <stdlib.h> /* for malloc, free */

int main(void)
{
    long *i;

    if ((i = (long *)malloc(50 * sizeof(long))) ==
        NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        free(i);
        printf("Memory freed\n");
    }
}
```

Example Output

```
Memory allocated
Memory freed
```

2.15.17 system Function

Execute a command.

Include

```
<stdlib.h>
```

Prototype

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

Argument

s command to be executed

Default Behavior

As distributed, this function acts as a stub or placeholder for your function. If *s* is not NULL, an error message is written to *stdout* and the program will reset; otherwise, a value of -1 is returned.

2.15.18 wcstombs Function

Converts a wide character string to a multibyte string (see Remarks).

Include

```
<stdlib.h>
```

Prototype

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

Arguments

s points to the multibyte string
wcs points to the wide character string
n the number of characters to convert

Return Value

Returns the number of characters stored excluding the null character.

Remarks

`wcstombs` converts `n` number of multibyte characters unless it encounters a null character first. The 16-bit compiler does not support multibyte characters with length greater than 1 character.

2.15.19 `wctomb` Function

Converts a wide character to a multibyte character (see Remarks).

Include

`<stdlib.h>`

Prototype

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

Arguments

<code>s</code>	points to the multibyte character
<code>wchar</code>	the wide character to be converted

Return Value

Returns zero if `s` points to a null character; otherwise, returns 1.

Remarks

The resulting multibyte character is stored at `s`. The 16-bit compiler does not support multibyte characters with length greater than 1 character.

2.15.20 `strtol` family functions

Any initial whitespace characters in the string are skipped. The following characters representing the integer are assumed to be in a radix specified by the `base` argument. Conversion stops once an unrecognized character is encountered in the string. If the correct converted value is out of range, the value of the macro `ERANGE` is stored in `errno`.

If the value of `base` is zero, the characters representing the integer can be in any valid C constant form (i.e., in decimal, octal, or hexadecimal), but any integer suffix is ignored. If the value of `base` is between 2 and 36 (inclusive), the expected form of the integer characters is a sequence of letters and digits representing an integer with the radix specified by `base`, optionally preceded by a plus or minus sign, but again, the integer suffix is ignored. The letters from `a` (or `A`) through `z` (or `Z`) are ascribed the values 10 through 35; only letters and digits whose ascribed values are less than that of `base` are permitted. If the value of `base` is 16, the characters `0x` or `0X` may optionally precede the sequence of letters and digits, following the sign if present.

2.16 `<string.h>` String Functions

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

Related Links

[2.2 Multiple Header Types and Macros](#)

2.16.1 `memchr` Function

Locates a character in a buffer.

Include

`<string.h>`

Prototype

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

Arguments

s pointer to the buffer
c character to search for
n number of characters to check

Return Value

Returns a pointer to the location of the match if successful; otherwise, returns `NULL`.

Remarks

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

Example

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

int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'i', ch2 = 'y';
    char *ptr;
    int res;
    printf("buf1 : %s\n\n", buf1);

    ptr = memchr(buf1, ch1, 50);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
        printf("%c not found\n", ch1);
    printf("\n");

    ptr = memchr(buf1, ch2, 50);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch2, res);
    }
    else
        printf("%c not found\n", ch2);
}
```

Example Output

```
buf1 : What time is it?
i found at position 7
y not found
```

2.16.2 memcmp Function

Compare the contents of two buffers.

Include

`<string.h>`

Prototype

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

Arguments

s1 first buffer
s2 second buffer

n number of characters to compare

Return Value

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

Remarks

This function compares the first *n* characters in *s1* to the first *n* characters in *s2* and returns a value indicating whether the buffers are less than, equal to or greater than each other.

Example

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

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

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

    res = memcmp(buf1, buf2, 6);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("6 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");
    printf("\n");

    res = memcmp(buf1, buf2, 20);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("20 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");
    printf("\n");

    res = memcmp(buf1, buf3, 20);
    if (res < 0)
        printf("buf1 comes before buf3\n");
    else if (res == 0)
        printf("20 characters of buf1 and buf3 "
              "are equal\n");
    else
        printf("buf3 comes before buf1\n");
}
```

Example Output

```
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?

6 characters of buf1 and buf2 are equal

buf2 comes before buf1

buf1 comes before buf3
```

2.16.3 memcpy Function

Copies characters from one buffer to another.

Include

```
<string.h>
```

Prototype

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

Arguments

dst	buffer to copy characters to
src	buffer to copy characters from
n	number of characters to copy

Return Value

Returns dst.

Remarks

`memcpy` copies `n` characters from the source buffer `src` to the destination buffer `dst`. If the buffers overlap, the behavior is undefined.

For `memcpy_eds`, `memcpy_packed`, `memcpy_p2d16` or `memcpy_p2d1624`, see “Functions for Specialized Copying and Initialization.”

Example

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

int main(void)
{
    char buf1[50] = "";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";

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

    memcpy(buf1, buf2, 6);
    printf("buf1 after memcpy of 6 chars of "
           "buf2: \n\t%s\n", buf1);

    printf("\n");

    memcpy(buf1, buf3, 5);
    printf("buf1 after memcpy of 5 chars of "
           "buf3: \n\t%s\n", buf1);
}
```

Example Output

```
buf1 :
buf2 : Where is the time?
buf3 : Why?

buf1 after memcpy of 6 chars of buf2:
    Where

buf1 after memcpy of 5 chars of buf3:
    Why?
```

2.16.4 memmove Function

Copies `n` characters of the source buffer into the destination buffer, even if the regions overlap.

Include

```
<string.h>
```

Prototype

```
void *memmove(void *s1, const void *s2, size_t n);
```

Arguments

s1 buffer to copy characters to (destination)
s2 buffer to copy characters from (source)
n number of characters to copy from s2 to s1

Return Value

Returns a pointer to the destination buffer.

Remarks

If the buffers overlap, the effect is as if the characters are read first from s2, then written to s1, so the buffer is not corrupted.

Example

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

int main(void)
{
    char buf1[50] = "When time marches on";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";

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

    memmove(buf1, buf2, 6);
    printf("buf1 after memmove of 6 chars of "
           "buf2: \n\t%s\n", buf1);

    printf("\n");

    memmove(buf1, buf3, 5);
    printf("buf1 after memmove of 5 chars of "
           "buf3: \n\t%s\n", buf1);
}
```

Example Output

```
buf1 : When time marches on
buf2 : Where is the time?
buf3 : Why?

buf1 after memmove of 6 chars of buf2:
    Where ime marches on

buf1 after memmove of 5 chars of buf3:
    Why?
```

2.16.5 memset Function

Copies the specified character into the destination buffer.

Include

```
<string.h>
```

Prototype

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

Arguments

s	buffer
c	character to put in buffer
n	number of times

Return Value

Returns the buffer with characters written to it.

Remarks

The character `c` is written to the buffer `n` times.

Example

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

int main(void)
{
    char buf1[20] = "What time is it?";
    char buf2[20] = "";
    char ch1 = '?', ch2 = 'y';
    char *ptr;
    int res;

    printf("memset(\"%s\", \"%c\",4);\n", buf1, ch1);
    memset(buf1, ch1, 4);
    printf("buf1 after memset: %s\n", buf1);

    printf("\n");
    printf("memset(\"%s\", \"%c\",10);\n", buf2, ch2);
    memset(buf2, ch2, 10);
    printf("buf2 after memset: %s\n", buf2);
}
```

Example Output

```
memset("What time is it?", '?',4);
buf1 after memset: ??? time is it?

memset("", 'y',10);
buf2 after memset: yyyyyyyyyy
```

2.16.6 strcat Function

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

Include

```
<string.h>
```

Prototype

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

Arguments

s1	null terminated destination string to copy to
s2	null terminated source string to be copied

Return Value

Returns a pointer to the destination string.

Remarks

This function appends the source string (including the terminating null character) to the end of the destination string. The initial character of the source string overwrites the null character at the end of the destination string. If the buffers overlap, the behavior is undefined.

Example

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

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";

    printf("buf1 : %s\n", buf1);
    printf("\t\t(%d characters)\n\n", strlen(buf1));
    printf("buf2 : %s\n", buf2);
    printf("\t\t(%d characters)\n\n", strlen(buf2));

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

    strcat(buf1, "Why?");
    printf("buf1 after strcat of \"Why?\": \n\t%s\n",
        buf1);
    printf("\t\t(%d characters)\n", strlen(buf1));
}
```

Example Output

```
buf1 : We're here
      (10 characters)

buf2 : Where is the time?
      (18 characters)

buf1 after strcat of buf2:
      We're hereWhere is the time?
      (28 characters)

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

2.16.7 strchr Function

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*.

Example

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

int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'm', ch2 = 'y';
    char *ptr;
    int res;

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

    ptr = strchr(buf1, ch1);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
        printf("%c not found\n", ch1);
    printf("\n");

    ptr = strchr(buf1, ch2);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch2, res);
    }
    else
        printf("%c not found\n", ch2);
}
```

Example Output

```
buf1 : What time is it?

m found at position 8

y not found
```

2.16.8 strcmp Function

Compares two strings.

Include

<string.h>

Prototype

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

Arguments

s1	first string
s2	second string

Return Value

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

Remarks

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

Example

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

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

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

    res = strcmp(buf1, buf2);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("buf1 and buf2 are equal\n");
    else
        printf("buf2 comes before buf1\n");
    printf("\n");

    res = strcmp(buf1, buf3);
    if (res < 0)
        printf("buf1 comes before buf3\n");
    else if (res == 0)
        printf("buf1 and buf3 are equal\n");
    else
        printf("buf3 comes before buf1\n");
    printf("\n");

    res = strcmp("Why?", buf3);
    if (res < 0)
        printf("\"Why?\" comes before buf3\n");
    else if (res == 0)
        printf("\"Why?\" and buf3 are equal\n");
    else
        printf("buf3 comes before \"Why?\" \n");
}
```

Example Output

```
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?

buf2 comes before buf1
buf1 comes before buf3

"Why?" and buf3 are equal
```

2.16.9 strcoll Function

Compares one string to another (see Remarks).

Include

```
<string.h>
```

Prototype

```
int strcoll(const char *s1, const char *s2);
```

Arguments

s1	first string
s2	second string

Return Value

Using the locale-dependent rules, it returns a positive number if `s1` is greater than `s2`, zero if `s1` is equal to `s2` or a negative number if `s1` is less than `s2`.

Remarks

Since alternate locales are not supported, this function is equivalent to `strcmp`.

2.16.10 strcpy Function

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.

For `strcpy_eds` or `strcpy_packed`, see “Functions for Specialized Copying and Initialization.”

Example

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

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";

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

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

    strcpy(buf1, buf3);
    printf("buf1 after strcpy of buf3: \n\t%s\n",
        buf1);
}
```

Example Output

```
buf1 : We're here
buf2 : Where is the time?
buf3 : Why?

buf1 after strcpy of buf2:
    Where is the time?

buf1 after strcpy of buf3:
    Why?
```

2.16.11 strcspn Function

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

Include

```
<string.h>
```

Prototype

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

Arguments

s1 pointer to the string to be searched

s2 pointer to characters to search for

Return Value

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

Remarks

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

Example

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

int main(void)
{
    char str1[20] = "hello";
    char str2[20] = "aeiou";
    char str3[20] = "animal";
    char str4[20] = "xyz";
    int res;

    res = strcspn(str1, str2);
    printf("strcspn(\"%s\", \"%s\") = %d\n",
           str1, str2, res);

    res = strcspn(str3, str2);
    printf("strcspn(\"%s\", \"%s\") = %d\n",
           str3, str2, res);

    res = strcspn(str3, str4);
    printf("strcspn(\"%s\", \"%s\") = %d\n",
           str3, str4, res);
}
```

Example Output

```
strcspn("hello", "aeiou") = 1
strcspn("animal", "aeiou") = 0
strcspn("animal", "xyz") = 6
```

Example Explanation

In the first result, e is in *s2* so it stops counting after h.

In the second result, a is in *s2*.

In the third result, none of the characters of *s1* are in *s2* so all characters are counted.

2.16.12 strerror Function

Gets an internal error message.

Include

```
<string.h>
```

Prototype

```
char *strerror(int errcode);
```

Argument

errcode number of the error code

Return Value

Returns a pointer to an internal error message string corresponding to the specified error code `errcode`.

Remarks

The array pointed to by `strerror` may be overwritten by a subsequent call to this function.

Example

```
#include <stdio.h>
#include <string.h>
#include <errno.h>

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samp.fil", "r+")) == NULL)
        printf("Cannot open samp.fil: %s\n",
               strerror(errno));
    else
        printf("Success opening samp.fil\n");
    fclose(myfile);
}
```

Example Output

```
Cannot open samp.fil: file open error
```

2.16.13 strlen Function

Finds the length of a string.

Include

```
<string.h>
```

Prototype

```
size_t strlen(const char *s);
```

Argument

s the string

Return Value

Returns the length of a string.

Remarks

This function determines the length of the string, not including the terminating null character.

Example

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

int main(void)
{
    char str1[20] = "We are here";
    char str2[20] = "";
```

```

char str3[20] = "Why me?";

printf("str1 : %s\n", str1);
printf("\t(string length = %d characters)\n\n",
      strlen(str1));
printf("str2 : %s\n", str2);
printf("\t(string length = %d characters)\n\n",
      strlen(str2));
printf("str3 : %s\n", str3);
printf("\t(string length = %d characters)\n\n\n",
      strlen(str3));
}

```

Example Output

```

str1 : We are here
      (string length = 11 characters)

str2 :
      (string length = 0 characters)

str3 : Why me?
      (string length = 7 characters)

```

2.16.14 strncat Function

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

s1	destination string to copy to
s2	source string to copy from
n	number of characters to append

Return Value

Returns a pointer to the destination string.

Remarks

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

Example

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";

    printf("buf1 : %s\n", buf1);
    printf("\t(%d characters)\n\n", strlen(buf1));
    printf("buf2 : %s\n", buf2);
    printf("\t(%d characters)\n\n", strlen(buf2));
    printf("buf3 : %s\n", buf3);
    printf("\t(%d characters)\n\n\n", strlen(buf3));
}

```

```

    strncat(buf1, buf2, 6);
    printf("buf1 after strncat of 6 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));
    printf("\n");

    strncat(buf1, buf2, 25);
    printf("buf1 after strncat of 25 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));
    printf("\n");

    strncat(buf1, buf3, 4);
    printf("buf1 after strncat of 4 characters "
           "of buf3: \n\t%s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));
}

```

Example Output

```

buf1 : We're here
      (10 characters)

buf2 : Where is the time?
      (18 characters)

buf3 : Why?
      (4 characters)

buf1 after strncat of 6 characters of buf2:
      We're hereWhere
      (16 characters)

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

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

```

2.16.15 strncmp Function

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

s1	first string
s2	second string
n	number of characters to compare

Return Value

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

Remarks

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

Example

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

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

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

    res = strncmp(buf1, buf2, 6);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("6 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");
    printf("\n");

    res = strncmp(buf1, buf2, 20);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("20 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");
    printf("\n");

    res = strncmp(buf1, buf3, 20);
    if (res < 0)
        printf("buf1 comes before buf3\n");
    else if (res == 0)
        printf("20 characters of buf1 and buf3 "
              "are equal\n");
    else
        printf("buf3 comes before buf1\n");
}
```

Example Output

```
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?

6 characters of buf1 and buf2 are equal

buf2 comes before buf1

buf1 comes before buf3
```

2.16.16 strncpy Function

Copy the source string into the destination string, up to the specified number of characters.

Include

```
<string.h>
```

Prototype

```
char *strncpy(char *s1, const char *s2, size_t n);
```

Arguments

s1 destination string to copy to

s2 source string to copy from
n number of characters to copy

Return Value

Returns a pointer to the destination string.

Remarks

Copies *n* characters from the source string to the destination string. If the source string is less than *n* characters, the destination is filled with null characters to total *n* characters. If *n* characters were copied and no null character was found, then the destination string will not be null-terminated. If the strings overlap, the behavior is undefined.

For `strncpy_eds`, `strncpy_packed`, `strncpy_p2d16` or `strncpy_p2d24`, see “Functions for Specialized Copying and Initialization.”

Example

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

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";
    char buf4[7]  = "Where?";

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);
    printf("buf4 : %s\n", buf4);

    strncpy(buf1, buf2, 6);
    printf("buf1 after strncpy of 6 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));
    printf("\n");

    strncpy(buf1, buf2, 18);
    printf("buf1 after strncpy of 18 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));
    printf("\n");

    strncpy(buf1, buf3, 5);
    printf("buf1 after strncpy of 5 characters "
           "of buf3: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));
    printf("\n");

    strncpy(buf1, buf4, 9);
    printf("buf1 after strncpy of 9 characters "
           "of buf4: \n\t%s\n", buf1);
    printf("\t( %d characters)\n", strlen(buf1));
}
```

Example Output

```
buf1 : We're here
buf2 : Where is the time?
buf3 : Why?
buf4 : Where?
buf1 after strncpy of 6 characters of buf2:
    Where here
    ( 10 characters)

buf1 after strncpy of 18 characters of buf2:
    Where is the time?
    ( 18 characters)

buf1 after strncpy of 5 characters of buf3:
```



```

    Why?
    ( 4 characters)

buf1 after strncpy of 9 characters of buf4:
    Where?
    ( 6 characters)

```

Example Explanation

Each buffer contains the string shown, followed by null characters for a length of 50. Using `strlen` will find the length of the string up to, but not including, the first null character.

In the first example, 6 characters of `buf2` ("Where ") replace the first 6 characters of `buf1` ("We're ") and the rest of `buf1` remains the same ("here" plus null characters).

In the second example, 18 characters replace the first 18 characters of `buf1` and the rest remain null characters.

In the third example, 5 characters of `buf3` ("Why?" plus a null terminating character) replace the first 5 characters of `buf1`. `buf1` now actually contains ("Why?", 1 null character, " is the time?", 32 null characters). `strlen` shows 4 characters because it stops when it reaches the first null character.

In the fourth example, since `buf4` is only 7 characters, `strncpy` uses 2 additional null characters to replace the first 9 characters of `buf1`. The result of `buf1` is 6 characters ("Where?") followed by 3 null characters, followed by 9 characters ("the time?"), followed by 32 null characters.

2.16.17 strpbrk Function

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

Include

```
<string.h>
```

Prototype

```
char *strpbrk(const char *s1, const char *s2);
```

Arguments

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

Return Value

Returns a pointer to the matched character in `s1` if found; otherwise, returns a null pointer.

Remarks

This function will search `s1` for the first occurrence of a character contained in `s2`.

Example

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char str1[20] = "What time is it?";
    char str2[20] = "xyz";
    char str3[20] = "eou?";
    char *ptr;
    int res;

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

```

```

printf("\n");

printf("strpbrk(\"%s\", \"%s\")\n", str1, str3);
ptr = strpbrk(str1, str3);
if (ptr != NULL)
{
    res = ptr - str1 + 1;
    printf("match found at position %d\n", res);
}
else
    printf("match not found\n");
}

```

Example Output

```

strpbrk("What time is it?", "xyz")
match not found

strpbrk("What time is it?", "eou?")
match found at position 9

```

2.16.18 strrchr Function

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
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 last occurrence of the character, *c*.

Example

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'm', ch2 = 'y';
    char *ptr;
    int res;

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

    ptr = strrchr(buf1, ch1);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
        printf("%c not found\n", ch1);
    printf("\n");

    ptr = strrchr(buf1, ch2);
    if (ptr != NULL)
    {

```

```

    res = ptr - buf1 + 1;
    printf("%c found at position %d\n", ch2, res);
}
else
    printf("%c not found\n", ch2);
}

```

Example Output

```

buf1 : What time is it?

m found at position 8

y not found

```

2.16.19 strspn Function

Calculate the number of consecutive characters at the beginning of a string that are contained in a set of characters.

Include

```
<string.h>
```

Prototype

```
size_t strspn(const char *s1, const char *s2);
```

Arguments

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

Return Value

Returns the number of consecutive characters from the beginning of s1 that are contained in s2.

Remarks

This function stops searching when a character from s1 is not in s2.

Example

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char str1[20] = "animal";
    char str2[20] = "aeiounm";
    char str3[20] = "aimnl";
    char str4[20] = "xyz";
    int res;

    res = strspn(str1, str2);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str2, res);

    res = strspn(str1, str3);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str3, res);

    res = strspn(str1, str4);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str4, res);
}

```

Example Output

```

strspn("animal", "aeiounm") = 5
strspn("animal", "aimnl") = 6
strspn("animal", "xyz") = 0

```

Example Explanation

In the first result, `l` is not in `s2`.

In the second result, the terminating null is not in `s2`.

In the third result, `a` is not in `s2`, so the comparison stops.

2.16.20 strstr Function

Search for the first occurrence of a string inside another string.

Include

```
<string.h>
```

Prototype

```
char *strstr(const char *s1, const char *s2);
```

Arguments

- s1** pointer to the string to be searched
- s2** pointer to substring to be searched for

Return Value

Returns the address of the first element that matches the substring if found; otherwise, returns a null pointer.

Remarks

This function will find the first occurrence of the string `s2` (excluding the null terminator) within the string `s1`. If `s2` points to a zero length string, `s1` is returned.

Example

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

int main(void)
{
    char str1[20] = "What time is it?";
    char str2[20] = "is";
    char str3[20] = "xyz";
    char *ptr;
    int res;

    printf("str1 : %s\n", str1);
    printf("str2 : %s\n", str2);
    printf("str3 : %s\n\n", str3);

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

    ptr = strstr(str1, str3);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("\"%s\" found at position %d\n",
            str3, res);
    }
    else
        printf("\"%s\" not found\n", str3);
}
```

Example Output

```
str1 : What time is it?
str2 : is
str3 : xyz

"is" found at position 11

"xyz" not found
```

2.16.21 strtok Function

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.

Example

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

int main(void)
{
    char str1[30] = "Here, on top of the world!";
    char delim[5] = ", .";
    char *word;
    int x;

    printf("str1 : %s\n", str1);
    x = 1;
    word = strtok(str1, delim);
    while (word != NULL)
    {
        printf("word %d: %s\n", x++, word);
        word = strtok(NULL, delim);
    }
}
```

```
}
}
```

Example Output

```
str1 : Here, on top of the world!
word 1: Here
word 2: on
word 3: top
word 4: of
word 5: the
word 6: world!
```

2.16.22 strxfrm Function

Transforms a string using the locale-dependent rules (see Remarks).

Include

```
<string.h>
```

Prototype

```
size_t strxfrm(char *s1, const char *s2, size_t n);
```

Arguments

s1 destination string

s2 source string to be transformed

n number of characters to transform

Return Value

Returns the length of the transformed string not including the terminating null character. If *n* is zero, the string is not transformed (*s1* may be a point null in this case) and the length of *s2* is returned.

Remarks

If the return value is greater than or equal to *n*, the content of *s1* is indeterminate. Since the 16-bit compiler does not support alternate locales, the transformation is equivalent to `strcpy`, except that the length of the destination string is bounded by *n*-1.

2.17 <time.h> Date and Time Functions

The header file `time.h` consists of types, macros and functions that manipulate date and time.

2.17.1 time.h Types and Macros**clock_t**

Stores processor time values.

Prototype

```
typedef long clock_t
```

struct tm

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

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

Represents calendar time values.

Prototype

```
typedef long time_t
```

The following macro is included in `time.h`

CLOCKS_PER_SEC

Number of processor clocks per second.

Prototype

```
#define CLOCKS_PER_SEC
```

Value

1

Remarks

The compiler returns clock ticks (instruction cycles) not actual time.

Related Links

[2.2 Multiple Header Types and Macros](#)

2.17.2 asctime Function

Converts the time structure to a character string.

Include

```
<time.h>
```

Prototype

```
char *asctime(const struct tm *tptr);
```

Argument

tptr	time/date structure
-------------	---------------------

Return Value

Returns a pointer to a character string of the following format:

```
DDD MMM dd hh:mm:ss YYYY
```

DDD is day of the week

MMM is month of the year

dd is day of the month

hh is hour

mm is minute

ss is second

YYYY is year

Example

```
#include <time.h> /* for asctime, tm */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    struct tm when;
    time_t whattime;

    when.tm_sec = 30;
    when.tm_min = 30;
    when.tm_hour = 2;
    when.tm_mday = 1;
    when.tm_mon = 1;
    when.tm_year = 103;

    whattime = mktime(&when);
    printf("Day and time is %s\n", asctime(&when));
}
```

Example Output

```
Day and time is Sat Feb  1 02:30:30 2003
```

2.17.3 clock Function

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

This function uses the device Timer2 and Timer3 to compute clock ticks. By default, the compiler returns the time as instruction cycles.

If the target environment cannot measure elapsed processor time, the function returns -1 cast as a `clock_t` (i.e. `(clock_t) -1`).

Example

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

volatile int i;

int main(void)
{
    clock_t start, stop;
    int cT;

    start = clock();
    for (i = 0; i < 10; i++)
        stop = clock();
    printf("start = %ld\n", start);
}
```



```
    printf("stop = %ld\n", stop);
}
```

Example Output

```
start = 0
stop = 317
```

2.17.4 ctime Function

Converts calendar time to a string representation of local time.

Include

```
<time.h>
```

Prototype

```
char *ctime(const time_t *tod);
```

Argument

tod pointer to stored time

Return Value

Returns the address of a string that represents the local time of the parameter passed.

Remarks

This function is equivalent to `asctime(localtime(tod))`.

Example

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

int main(void)
{
    time_t whattime;
    struct tm nowtime;

    nowtime.tm_sec = 30;
    nowtime.tm_min = 30;
    nowtime.tm_hour = 2;
    nowtime.tm_mday = 1;
    nowtime.tm_mon = 1;
    nowtime.tm_year = 103;

    whattime = mktime(&nowtime);
    printf("Day and time %s\n", ctime(&whattime));
}
```

Example Output

```
Day and time Sat Feb  1 02:30:30 2003
```

2.17.5 difftime Function

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`.

Remarks

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

Example

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

volatile int i;

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

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

Example Output

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

2.17.6 gmtime Function

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`. By default, the compiler returns the time as instruction cycles. With this default, `gmtime` and `localtime` will be equivalent, except `gmtime` will return `tm_isdst` (Daylight Savings Time flag) as zero to indicate that Daylight Savings Time is not in effect.

Example

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

```
int main(void)
{
    time_t timer;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */

    newtime = gmtime(&timer);
    printf("UTC time = %s\n", asctime(newtime));
}
```

Example Output

```
UTC time = Mon Oct 20 16:43:02 2003
```

2.17.7 localtime Function

Converts a value to the local time.

Include

<time.h>

Prototype

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

Argument

tod pointer to stored time

Return Value

Returns the address of the time structure.

Remarks

By default, the 16-bit compiler returns the time as instruction cycles. With this default, `localtime` and `gmtime` will be equivalent, except `localtime` will return `tm_isdst` (Daylight Savings Time flag) as -1 to indicate that the status of Daylight Savings Time is not known.

Example

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

int main(void)
{
    time_t timer;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */

    newtime = localtime(&timer);
    printf("Local time = %s\n", asctime(newtime));
}
```

Example Output

```
Local time = Mon Oct 20 16:43:02 2003
```

2.17.8 mktime Function

Converts local time to a calendar value.

Include

<time.h>

Prototype

```
time_t mktime(struct tm *tptr);
```

Argument

tptr a pointer to the time structure

Return Value

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

Remarks

If the calendar time cannot be represented, the function returns -1 cast as a `time_t` (i.e. `(time_t) -1`).

Example

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

int main(void)
{
    time_t timer, whattime;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */
    /* localtime allocates space for struct tm */
    newtime = localtime(&timer);
    printf("Local time = %s", asctime(newtime));

    whattime = mktime(newtime);
    printf("Calendar time as time_t = %ld\n",
           whattime);
}
```

Example Output

```
Local time = Mon Oct 20 16:43:02 2003
Calendar time as time_t = 1066668182
```

2.17.9 strftime Function

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 %

Example

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

int main(void)
{
    time_t timer, whattime;
    struct tm *newtime;
    char buf[128];
    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */
    /* localtime allocates space for structure */
    newtime = localtime(&timer);
    strftime(buf, 128, "It was a %A, %d days into the "
                "month of %B in the year %Y.\n", newtime);
    printf(buf);
    strftime(buf, 128, "It was %W weeks into the year "
                "or %j days into the year.\n", newtime);
    printf(buf);
}
```

Example Output

```
It was a Monday, 20 days into the month of October in the year 2003.
It was 42 weeks into the year or 293 days into the year.
```

2.17.10 time Function

Calculates the current calendar time.

Include

`<time.h>`

Prototype

`time_t time(time_t *tod);`

Argument

tod pointer to storage location for time

Return Value

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

Remarks

If the target environment cannot determine the time, the function returns -1 cast as a `time_t`. By default, the compiler returns the time as instruction cycles.

This function is customizable (see `pic30-libs`).

Example

See the notes at the beginning of this chapter or section for information on using `printf()` or `scanf()` (and other functions reading and writing the `stdin` or `stdout` streams) in the example code.

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

volatile int i;

int main(void)
{
    time_t ticks;

    time(0); /* start time */
    for (i = 0; i < 10; i++) /* waste time */
        continue;
    time(&ticks); /* get time */
    printf("Time = %ld\n", ticks);
}
```

Example Output

```
Time = 256
```

3. Math Libraries

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

For details on using the standard C libraries, see “Standard C Libraries.”

3.1 <math.h> Mathematical Functions

The header file `math.h` consists of a macro and various functions that calculate common mathematical operations. Error conditions may be handled with a domain error or range error (see `errno.h`).

A domain error occurs when the input argument is outside the domain over which the function is defined. The error is reported by storing the value of `EDOM` in `errno` and returning a particular value defined for each function.

A range error occurs when the result is too large or too small to be represented in the target precision. The error is reported by storing the value of `ERANGE` in `errno` and returning `HUGE_VAL` if the result overflowed (return value was too large) or a zero if the result underflowed (return value is too small).

Responses to special values, such as NaNs, zeros and infinities, may vary depending upon the function. Each function description includes a definition of the function's response to such values.

3.1.1 HUGE_VAL

`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`.

3.1.2 acos Function

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.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>
```

```

int main(void)
{
    double x,y;

    errno = 0;
    x = -2.0;
    y = acos (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n", x, y);

    errno = 0;
    x = 0.10;
    y = acos (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n", x, y);
}

```

Example Output

```

Error: domain error
The arccosine of -2.000000 is nan
The arccosine of 0.100000 is 1.470629

```

3.1.3 acosf Function

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

Include

```
<math.h>
```

Prototype

```
float acosf (float x);
```

Argument

x value between -1 and 1

Return Value

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

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = acosf (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = acosf (x);
    if (errno)
        perror("Error");
}

```



```
printf("The arccosine of %f is %f\n", x, y);
}
```

Example Output

```
Error: domain error
The arccosine of 2.000000 is nan
The arccosine of 0.000000 is 1.570796
```

3.1.4 asin Function

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

Include

```
<math.h>
```

Prototype

```
double asin (double x);
```

Argument

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

Return Value

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

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x, y;

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

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

Example Output

```
Error: domain error
The arcsine of 2.000000 is nan
The arcsine of 0.000000 is 0.000000
```

3.1.5 asinf Function

Calculates the trigonometric arc sine function of a single precision floating-point value.

Include

```
<math.h>
```

Prototype

```
float asinf (float x);
```

Argument

x value between -1 and 1

Return Value

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

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = asinf(x);
    if (errno)
        perror("Error");
    printf("The arcsine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = asinf(x);
    if (errno)
        perror("Error");
    printf("The arcsine of %f is %f\n", x, y);
}
```

Example Output

```
Error: domain error
The arcsine of 2.000000 is nan
The arcsine of 0.000000 is 0.000000
```

3.1.6 atan Function

Calculates the trigonometric arc tangent function of a double precision floating-point value.

Include

```
<math.h>
```

Prototype

```
double atan (double x);
```

Argument

x value for which to return the arc tangent

Return Value

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

Remarks

No domain or range error will occur.

Example

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

int main(void)
{
    double x, y;

    x = 2.0;
    y = atan (x);
    printf("The arctangent of %f is %f\n", x, y);

    x = -1.0;
    y = atan (x);
    printf("The arctangent of %f is %f\n", x, y);
}
```

Example Output

```
The arctangent of 2.000000 is 1.107149
The arctangent of -1.000000 is -0.785398
```

3.1.7 atanf Function

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

Include

```
<math.h>
```

Prototype

```
float atanf (float x);
```

Argument

x value for which to return the arc tangent

Return Value

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

Remarks

No domain or range error will occur.

Example

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

int main(void)
{
    float x, y;

    x = 2.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n", x, y);

    x = -1.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n", x, y);
}
```

Example Output

```
The arctangent of 2.000000 is 1.107149
The arctangent of -1.000000 is -0.785398
```

3.1.8 atan2 Function

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.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

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

    errno = 0;
    x = 0.0;
    y = 2.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n", y, x, z);

    errno = 0;
    x = -1.0;
    y = 0.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n", y, x, z);

    errno = 0;
    x = 0.0;
    y = 0.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n", y, x, z);
}
```

Example Output

```
The arctangent of 2.000000/0.000000 is 1.570796
The arctangent of 0.000000/-1.000000 is 3.141593
Error: domain error
The arctangent of 0.000000/0.000000 is nan
```

3.1.9 atan2f Function

Calculates the trigonometric arc tangent function of y/x .

Include

<math.h>

Prototype

```
float atan2f (float y, float x);
```

Arguments

y y value for which to return the arc tangent

x x value for which to return the arc tangent

Return Value

Returns the arc tangent in radians in the range of -pi to pi with the quadrant determined by the signs of both parameters.

Remarks

A domain error occurs if both *x* and *y* are zero or both *x* and *y* are +/- infinity.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

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

    errno = 0;
    x = 2.0F;
    y = 0.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n", y, x, z);

    errno = 0;
    x = 0.0F;
    y = -1.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n", y, x, z);

    errno = 0;
    x = 0.0F;
    y = 0.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n", y, x, z);
}
```

Example Output

```
The arctangent of 2.000000/0.000000 is 1.570796
The arctangent of 0.000000/-1.000000 is 3.141593
Error: domain error
The arctangent of 0.000000/0.000000 is nan
```

3.1.10 ceil Function

Calculates the ceiling of a value.

Include

<math.h>

Prototype

```
double ceil(double x);
```

Argument

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

Return Value

Returns the smallest integer value greater than or equal to *x*.

Remarks

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

Example

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

int main(void)
{
    double x[8] = {2.0, 1.75, 1.5, 1.25, -2.0, -1.75, -1.5, -1.25};
    double y;
    int i;

    for (i=0; i<8; i++)
    {
        y = ceil (x[i]);
        printf("The ceiling for  %f is  %f\n", x[i], y);
    }
}
```

Example Output

```
The ceiling for  2.000000 is  2.000000
The ceiling for  1.750000 is  2.000000
The ceiling for  1.500000 is  2.000000
The ceiling for  1.250000 is  2.000000
The ceiling for -2.000000 is -2.000000
The ceiling for -1.750000 is -1.000000
The ceiling for -1.500000 is -1.000000
The ceiling for -1.250000 is -1.000000
```

3.1.11 ceilf Function

Calculates the ceiling of a value.

Include

```
<math.h>
```

Prototype

```
float ceilf(float 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 `floorf`.

Example

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

int main(void)
```

```

{
    float x[8] = {2.0F, 1.75F, 1.5F, 1.25F, -2.0F, -1.75F, -1.5F, -1.25F};
    float y;
    int i;

    for (i=0; i<8; i++)
    {
        y = ceilf (x[i]);
        printf("The ceiling for  %f is  %f\n", x[i], y);
    }
}

```

Example Output

```

The ceiling for  2.000000 is  2.000000
The ceiling for  1.750000 is  2.000000
The ceiling for  1.500000 is  2.000000
The ceiling for  1.250000 is  2.000000
The ceiling for -2.000000 is -2.000000
The ceiling for -1.750000 is -1.000000
The ceiling for -1.500000 is -1.000000
The ceiling for -1.250000 is -1.000000

```

3.1.12 cos Function

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

Include

```
<math.h>
```

Prototype

```
double cos (double x);
```

Argument

x value for which to return the cosine

Return Value

Returns the cosine of *x* in radians in the ranges of -1 to 1 inclusive.

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x,y;

    errno = 0;
    x = -1.0;
    y = cos (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0;
    y = cos (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n", x, y);
}

```

Example Output

```
The cosine of -1.000000 is 0.540302
The cosine of 0.000000 is 1.000000
```

3.1.13 cosf Function

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

Include

```
<math.h>
```

Prototype

```
float cosf (float x);
```

Argument

x value for which to return the cosine

Return Value

Returns the cosine of *x* in radians in the ranges of -1 to 1 inclusive.

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

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

    errno = 0;
    x = 0.0F;
    y = cosf (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n", x, y);
}
```

Example Output

```
The cosine of -1.000000 is 0.540302
The cosine of 0.000000 is 1.000000
```

3.1.14 cosh Function

Calculates the hyperbolic cosine function of a double precision floating-point value.

Include

```
<math.h>
```

Prototype

```
double cosh (double x);
```

Argument

x value for which to return the hyperbolic cosine

Return Value

Returns the hyperbolic cosine of *x*.

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.5;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);

    errno = 0;
    x = 720.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);
}
```

Example Output

```
The hyperbolic cosine of -1.500000 is 2.352410
The hyperbolic cosine of 0.000000 is 1.000000
Error: range error
The hyperbolic cosine of 720.000000 is inf
```

3.1.15 coshf Function

Calculates the hyperbolic cosine function of a single precision floating-point value.

Include

<math.h>

Prototype

```
float coshf (float x);
```

Argument

x value for which to return the hyperbolic cosine

Return Value

Returns the hyperbolic cosine of *x*.

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);

    errno = 0;
    x = 720.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);
}
```

Example Output

```
The hyperbolic cosine of -1.000000 is 1.543081
The hyperbolic cosine of 0.000000 is 1.000000
Error: range error
The hyperbolic cosine of 720.000000 is inf
```

3.1.16 exp Function

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

Include

<math.h>

Prototype

double exp(double x);

Argument

x value for which to return the exponential

Return Value

Returns the exponential of x . On an overflow, exp returns *inf* and on an underflow exp returns 0.

Remarks

A range error occurs if the magnitude of x is too large.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

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

```

    errno = 0;
    x = 1.0;
    y = exp(x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n", x, y);

    errno = 0;
    x = 1E3;
    y = exp(x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n", x, y);

    errno = 0;
    x = -1E3;
    y = exp(x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n", x, y);
}

```

Example Output

```

The exponential of 1.000000 is 2.718282
Error: range error
The exponential of 1000.000000 is inf
Error: range error
The exponential of -1000.000000 is 0.000000

```

3.1.17 expf Function

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 expf returns 0.

Remarks

A range error occurs if the magnitude of x is too large.

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = 1.0F;
    y = expf(x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n", x, y);

    errno = 0;
    x = 1.0E3F;
    y = expf(x);
}

```

```

if (errno)
    perror("Error");
printf("The exponential of %f is %f\n", x, y);

errno = 0;
x = -1.0E3F;
y = expf(x);
if (errno)
    perror("Error");
printf("The exponential of %f is %f\n", x, y);
}

```

Example Output

```

The exponential of 1.000000 is 2.718282
Error: range error
The exponential of 1000.000000 is inf
Error: range error
The exponential of -1000.000000 is 0.000000

```

3.1.18 fabs Function

Calculates the absolute value of a double precision floating-point value.

Include

```
<math.h>
```

Prototype

```
double fabs(double x);
```

Argument

x floating-point value for which to return the absolute value

Return Value

Returns the absolute value of **x**. A negative number is returned as positive; a positive number is unchanged.

Remarks

No domain or range error will occur.

Example

```

#include <math.h>
#include <stdio.h>

int main(void)
{
    double x, y;

    x = 1.75;
    y = fabs(x);
    printf("The absolute value of %f is %f\n", x, y);

    x = -1.5;
    y = fabs(x);
    printf("The absolute value of %f is %f\n", x, y);
}

```

Example Output

```

The absolute value of 1.750000 is 1.750000
The absolute value of -1.500000 is 1.500000

```

3.1.19 fabsf Function

Calculates the absolute value of a single precision floating-point value.

Include

```
<math.h>
```

Prototype

```
float fabsf(float x);
```

Argument

x floating-point value for which to return the absolute value

Return Value

Returns the absolute value of **x**. A negative number is returned as positive; a positive number is unchanged.

Remarks

No domain or range error will occur.

Example

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

int main(void)
{
    float x,y;

    x = 1.75F;
    y = fabsf (x);
    printf("The absolute value of  %f is  %f\n", x, y);

    x = -1.5F;
    y = fabsf (x);
    printf("The absolute value of %f is  %f\n", x, y);
}
```

Example Output

```
The absolute value of  1.750000 is  1.750000
The absolute value of -1.500000 is  1.500000
```

3.1.20 floor Function

Calculates the floor of a double precision floating-point value.

Include

```
<math.h>
```

Prototype

```
double floor (double x);
```

Argument

x floating-point value for which to return the floor

Return Value

Returns the largest integer value less than or equal to **x**.

Remarks

No domain or range error will occur. See `ceil`.

Example

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

int main(void)
{
    double x[8] = {2.0, 1.75, 1.5, 1.25, -2.0,
```

```

        -1.75, -1.5, -1.25};
double y;
int i;

for (i=0; i<8; i++)
{
    y = floor (x[i]);
    printf("The ceiling for %f is %f\n", x[i], y);
}
}

```

Example Output

```

The floor for 2.000000 is 2.000000
The floor for 1.750000 is 1.000000
The floor for 1.500000 is 1.000000
The floor for 1.250000 is 1.000000
The floor for -2.000000 is -2.000000
The floor for -1.750000 is -2.000000
The floor for -1.500000 is -2.000000
The floor for -1.250000 is -2.000000

```

3.1.21 floorf Function

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

Include

```
<math.h>
```

Prototype

```
float floorf(float 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 `ceilf`.

Example

```

#include <math.h>
#include <stdio.h>

int main(void)
{
    float x[8] = {2.0F, 1.75F, 1.5F, 1.25F,
                  -2.0F, -1.75F, -1.5F, -1.25F};
    float y;
    int i;

    for (i=0; i<8; i++)
    {
        y = floorf (x[i]);
        printf("The floor for  %f is  %f\n", x[i], y);
    }
}

```

Example Output

```

The floor for 2.000000 is 2.000000
The floor for 1.750000 is 1.000000
The floor for 1.500000 is 1.000000
The floor for 1.250000 is 1.000000
The floor for -2.000000 is -2.000000
The floor for -1.750000 is -2.000000

```

```
The floor for -1.500000 is -2.000000
The floor for -1.250000 is -2.000000
```

3.1.22 fmod Function

Calculates the remainder of x/y as a double precision value.

Include

```
<math.h>
```

Prototype

```
double fmod(double x, double y);
```

Arguments

x a double precision floating-point value

y a double precision floating-point value

Return Value

Returns the remainder of x divided by y .

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

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

    errno = 0;
    x = 7.0;
    y = 3.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n", x, y, z);

    errno = 0;
    x = 7.0;
    y = 7.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n", x, y, z);

    errno = 0;
    x = -5.0;
    y = 3.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n", x, y, z);

    errno = 0;
    x = 5.0;
    y = -3.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n", x, y, z);

    errno = 0;
    x = -5.0;
```

```

y = -5.0;
z = fmod(x, y);
if (errno)
    perror("Error");
printf("For fmod(%f, %f) the remainder is %f\n", x, y, z);

errno = 0;
x = 7.0;
y = 0.0;
z = fmod(x, y);
if (errno)
    perror("Error");
printf("For fmod(%f, %f) the remainder is %f\n", x, y, z);
}

```

Example Output

```

For fmod(7.000000, 3.000000) the remainder is 1.000000
For fmod(7.000000, 7.000000) the remainder is 0.000000
For fmod(-5.000000, 3.000000) the remainder is -2.000000
For fmod(5.000000, -3.000000) the remainder is 2.000000
For fmod(-5.000000, -5.000000) the remainder is -0.000000
Error: domain error
For fmod(7.000000, 0.000000) the remainder is nan

```

3.1.23 fmodf Function

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

Include

<math.h>

Prototype

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

Arguments

- x** a double precision floating-point value
- y** a double precision floating-point value

Return Value

Returns the remainder of x divided by y .

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

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

    errno = 0;
    x = 7.0F;
    y = 3.0F;
    z = fmodf(x, y);
    if(errno)
        perror("Error");
    printf("For fmodf(%f, %f) the remainder is"
           " %f\n\n", x, y, z);

    errno = 0;
    x = -5.0F;

```



```

y = 3.0F;
z = fmodf(x, y);
if(errno)
    perror("Error");
    printf("For fmodf(%f, %f) the remainder is %f\n", x, y, z);

errno = 0;
x = 5.0F;
y = -3.0F;
z = fmodf(x, y);
if(errno)
    perror("Error");
    printf("For fmodf(%f, %f) the remainder is %f\n", x, y, z);

errno = 0;
x = 5.0F;
y = -5.0F;
z = fmodf(x, y);
if(errno)
    perror("Error");
    printf("For fmodf (%f, %f) the remainder is %f\n", x, y, z);

errno = 0;
x = 7.0F;
y = 0.0F;
z = fmodf(x, y);
if(errno)
    perror("Error");
    printf("For fmodf(%f, %f) the remainder is %f\n", x, y, z);

errno = 0;
x = 7.0F;
y = 7.0F;
z = fmodf(x, y);
if(errno)
    perror("Error");
    printf("For fmodf(%f, %f) the remainder is %f\n", x, y, z);
}

```

Example Output

```

For fmodf (7.000000, 3.000000) the remainder is 1.000000
For fmodf (-5.000000, 3.000000) the remainder is -2.000000
For fmodf (5.000000, -3.000000) the remainder is 2.000000
For fmodf (5.000000, -5.000000) the remainder is 0.000000
Error: domain error
For fmodf (7.000000, 0.000000) the remainder is nan
For fmodf (7.000000, 7.000000) the remainder is 0.000000

```

3.1.24 frexp Function

Gets the fraction and the exponent of a double precision floating-point number.

Include

<math.h>

Prototype

```
double frexp (double x, int *exp);
```

Arguments

- x** floating-point value for which to return the fraction and exponent
- exp** pointer to a stored integer exponent

Return Value

Returns the fraction, `exp` points to the exponent. If `x` is 0, the function returns 0 for both the fraction and exponent.

Remarks

The absolute value of the fraction is in the range of 1/2 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example

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

int main(void)
{
    double x,y;
    int n;

    x = 50.0;
    y = frexp (x, &n);
    printf("For frexp of %f\n the fraction is %f\n ", x, y);
    printf(" and the exponent is %d\n\n", n);

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

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

Example Output

```
For frexp of 50.000000
the fraction is 0.781250
and the exponent is 6

For frexp of -2.500000
the fraction is -0.625000
and the exponent is 2

For frexp of 0.000000
the fraction is 0.000000
and the exponent is 0
```

3.1.25 frexpf Function

Gets the fraction and the exponent of a single precision floating-point number.

Include

<math.h>

Prototype

```
float frexpf (float x, int *exp);
```

Arguments

- x** floating-point value for which to return the fraction and exponent
- exp** pointer to a stored integer exponent

Return Value

Returns the fraction, `exp` points to the exponent. If `x` is 0, the function returns 0 for both the fraction and exponent.

Remarks

The absolute value of the fraction is in the range of 1/2 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example

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

int main(void)
{
    float x,y;
    int n;

    x = 0.15F;
    y = frexpf (x, &n);
    printf("For frexpf of %f\n the fraction is %f\n ", x, y);
    printf(" and the exponent is %d\n\n", n);

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

    x = 0.0F;
    y = frexpf (x, &n);
    printf("For frexpf of %f\n the fraction is %f\n ", x, y);
    printf(" and the exponent is %d\n\n", n);
}
```

Example Output

```
For frexpf of 0.150000
the fraction is 0.600000
and the exponent is -2

For frexpf of -2.500000
the fraction is -0.625000
and the exponent is 2

For frexpf of 0.000000
the fraction is 0.000000
and the exponent is 0
```

3.1.26 ldexp Function

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

Include

```
<math.h>
```

Prototype

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

Arguments

x	floating-point value
ex	integer exponent

Return Value

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

Remarks

A range error will occur on overflow or underflow.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
```

```

{
    double x,y;
    int n;

    errno = 0;
    x = -0.625;
    n = 2;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexp(%f, %d) = %f\n\n",
           x, n, y);

    errno = 0;
    x = 2.5;
    n = 3;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexp(%f, %d) = %f\n\n",
           x, n, y);

    errno = 0;
    x = 15.0;
    n = 10000;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexp(%f, %d) = %f\n\n",
           x, n, y);
}

```

Example Output

```

For a number = -0.625000 and an exponent = 2
ldexp(-0.625000, 2) = -2.500000

For a number = 2.500000 and an exponent = 3
ldexp(2.500000, 3) = 20.000000

Error: range error
For a number = 15.000000 and an exponent = 10000
ldexp(15.000000, 10000) = inf

```

3.1.27 ldexpf Function

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, ldexpf returns 0.

Remarks

A range error will occur on overflow or underflow.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x,y;
    int n;

    errno = 0;
    x = -0.625F;
    n = 2;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n", x, n);
    printf("  ldexpf(%f, %d) = %f\n\n", x, n, y);

    errno = 0;
    x = 2.5F;
    n = 3;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n", x, n);
    printf("  ldexpf(%f, %d) = %f\n\n", x, n, y);

    errno = 0;
    x = 15.0F;
    n = 10000;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n", x, n);
    printf("  ldexpf(%f, %d) = %f\n\n", x, n, y);
}
```

Example Output

```
For a number = -0.625000 and an exponent = 2
  ldexpf(-0.625000, 2) = -2.500000

For a number = 2.500000 and an exponent = 3
  ldexpf(2.500000, 3) = 20.000000

Error: range error
For a number = 15.000000 and an exponent = 10000
  ldexpf(15.000000, 10000) = inf
```

3.1.28 log Function

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

Include

```
<math.h>
```

Prototype

```
double log(double x);
```

Argument

x any positive value for which to return the log

Return Value

Returns the natural logarithm of *x*. *-inf* is returned if *x* is 0 and NaN is returned if *x* is a negative number.

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = log(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n",
           x, y);

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

    errno = 0;
    x = -2.0;
    y = log(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n",
           x, y);
}
```

Example Output

```
The natural logarithm of 2.000000 is 0.693147
The natural logarithm of 0.000000 is -inf
Error: domain error
The natural logarithm of -2.000000 is nan
```

3.1.29 logf Function

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

Include

<math.h>

Prototype

```
float log(float x);
```

Argument

x any positive value for which to return the log

Return Value

Returns the natural logarithm of x . $-\text{inf}$ is returned if x is 0 and NaN is returned if x is a negative number.

Remarks

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

Example Output

```
#include <math.h>
#include <stdio.h>
#include <errno.h>
```

```

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = logf(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n",
           x, y);

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

    errno = 0;
    x = -2.0F;
    y = logf(x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n",
           x, y);
}

```

Example Output

```

The natural logarithm of 2.000000 is 0.693147
The natural logarithm of 0.000000 is -inf
Error: domain error
The natural logarithm of -2.000000 is nan

```

3.1.30 log10 Function

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

Include

<math.h>

Prototype

double log10(double x);

Argument

x any double precision floating-point positive number

Return Value

Returns the base-10 logarithm of *x*. *-inf* is returned if *x* is 0 and NaN is returned if *x* is a negative number.

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = log10 (x);
}

```

```

if (errno)
    perror("Error");
printf("The base-10 logarithm of %f is %f\n", x, y);

errno = 0;
x = 0.0;
y = log10 (x);
if (errno)
    perror("Error");
printf("The base-10 logarithm of %f is %f\n", x, y);

errno = 0;
x = -2.0;
y = log10 (x);
if (errno)
    perror("Error");
printf("The base-10 logarithm of %f is %f\n", x, y);
}

```

Example Output

```

The base-10 logarithm of 2.000000 is 0.301030
The base-10 logarithm of 0.000000 is -inf
Error: domain error
The base-10 logarithm of -2.000000 is nan

```

3.1.31 log10f Function

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

Include

```
<math.h>
```

Prototype

```
float log10(float x);
```

Argument

x any double precision floating-point positive number

Return Value

Returns the base-10 logarithm of *x*. *-inf* is returned if *x* is 0 and NaN is returned if *x* is a negative number.

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n", x, y);
}

```



```

    errno = 0;
    x = -2.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n", x, y);
}

```

Example Output

```

The base-10 logarithm of 2.000000 is 0.301030
Error: domain error
The base-10 logarithm of 0.000000 is -inf
Error: domain error
The base-10 logarithm of -2.000000 is nan

```

3.1.32 modf Function

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

Include

```
<math.h>
```

Prototype

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

Arguments

x double precision floating-point value

pint pointer to a stored the integer part

Return Value

Returns the signed fractional part and `pint` points to the integer part.

Remarks

The absolute value of the fractional part is in the range of 0 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example

```

#include <math.h>
#include <stdio.h>

int main(void)
{
    double x, y, n;

    x = 0.707;
    y = modf(x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf(" and the integer is %0.f\n\n", n);

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

```

Example Output

```

For 0.707000 the fraction is 0.707000
 and the integer is 0

For -15.212100 the fraction is -0.212100
 and the integer is -15

```

3.1.33 modff Function

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 a stored the integer part

Return Value

Returns the signed fractional part and `pint` points to the integer part.

Remarks

The absolute value of the fractional part is in the range of 0 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example

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

int main(void)
{
    float x, y, n;

    x = 0.707F;
    y = modff(x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf(" and the integer is %0.f\n\n", n);

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

Example Output

```
For 0.707000 the fraction is 0.707000
and the integer is 0

For -15.212100 the fraction is -0.212100
and the integer is -15
```

3.1.34 pow Function

Calculates x raised to the power y .

Include

```
<math.h>
```

Prototype

```
double pow(double x, double y);
```

Arguments

x	the base
y	the exponent

Return Value

Returns x raised to the power y (x^y).

Remarks

If y is 0, `pow` returns 1. If x is 0.0 and y is less than 0, `pow` returns `inf` and a domain error occurs. If the result overflows or underflows, a range error occurs.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

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

    errno = 0;
    x = -2.0;
    y = 3.0;
    z = pow(x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n ", x, y, z);

    errno = 0;
    x = 3.0;
    y = -0.5;
    z = pow(x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n ", x, y, z);

    errno = 0;
    x = 4.0;
    y = 0.0;
    z = pow(x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n ", x, y, z);

    errno = 0;
    x = 0.0;
    y = -3.0;
    z = pow(x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n ", x, y, z);
}
```

Example Output

```
-2.000000 raised to 3.000000 is -8.000000
3.000000 raised to -0.500000 is 0.577350
4.000000 raised to 0.000000 is 1.000000
Error: domain error
0.000000 raised to -3.000000 is inf
```

3.1.35 powf Function

Calculates x raised to the power y .

Include

<math.h>

Prototype

```
float powf(float x, float y);
```

Arguments

x the base
y the exponent

Return Value

Returns x raised to the power y (x^y).

Remarks

If y is 0, `pow` returns 1. If x is 0.0 and y is less than 0, `pow` returns `inf` and a domain error occurs. If the result overflows or underflows, a range error occurs.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

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

    errno = 0;
    x = -2.0F;
    y = 3.0F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n ", x, y, z);

    errno = 0;
    x = 3.0F;
    y = -0.5F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n ", x, y, z);

    errno = 0;
    x = 0.0F;
    y = -3.0F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n ", x, y, z);
}
```

Example Output

```
-2.000000 raised to 3.000000 is -8.000000
3.000000 raised to -0.500000 is 0.577350
Error: domain error
0.000000 raised to -3.000000 is inf
```

3.1.36 sin Function

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

Include

<math.h>

Prototype

```
double sin (double x);
```

Argument

x value for which to return the sine

Return Value

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

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.0;
    y = sin (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n", x, y);

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

Example Output

```
The sine of -1.000000 is -0.841471
The sine of 0.000000 is 0.000000
```

3.1.37 sinf Function

Calculates the trigonometric sine function of a single precision floating-point value.

Include

<math.h>

Prototype

```
float sin (float x);
```

Argument

x value for which to return the sine

Return Value

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

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = sinf (x);
}
```

```

if (errno)
    perror("Error");
printf("The sine of %f is %f\n", x, y);

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

```

Example Output

```

The sine of -1.000000 is -0.841471
The sine of 0.000000 is 0.000000

```

3.1.38 sinh Function

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

Include

<math.h>

Prototype

```
double sinh (double x);
```

Argument

x value for which to return the hyperbolic sine

Return Value

Returns the hyperbolic sine of *x*.

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.5;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n",
        x, y);

    errno = 0;
    x = 0.0;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n",
        x, y);

    errno = 0;
    x = 720.0;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n",

```

```

    x, y);
}

```

Example Output

```

The hyperbolic sine of -1.500000 is -2.129279
The hyperbolic sine of 0.000000 is 0.000000
Error: range error
The hyperbolic sine of 720.000000 is inf

```

3.1.39 sinhF Function

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

Include

```
<math.h>
```

Prototype

```
float sinhF (float x);
```

Argument

x value for which to return the hyperbolic sine

Return Value

Returns the hyperbolic sine of *x*.

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = sinhF (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = sinhF (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n", x, y);
}

```

Example Output

```

The hyperbolic sine of -1.000000 is -1.175201
The hyperbolic sine of 0.000000 is 0.000000

```

3.1.40 sqrt Function

Calculates the square root of a double precision floating-point value.

Include

```
<math.h>
```

Prototype

```
double sqrt(double x);
```

Argument

x a non-negative floating-point value

Return Value

Returns the non-negative square root of *x*.

Remarks

If *x* is negative, a domain error occurs.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x, y;

    errno = 0;
    x = 0.0;
    y = sqrt(x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n", x, y);

    errno = 0;
    x = 9.5;
    y = sqrt(x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n", x, y);

    errno = 0;
    x = -25.0;
    y = sqrt(x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n", x, y);
}
```

Example Output

```
The square root of 0.000000 is 0.000000
The square root of 9.500000 is 3.082207
Error: domain error
The square root of -25.000000 is nan
```

3.1.41 sqrtf Function

Calculates the square root of a single precision floating-point value.

Include

```
<math.h>
```

Prototype

```
float sqrtf(float x);
```

Argument

x a non-negative floating-point value

Return Value

Returns the non-negative square root of x .

Remarks

If x is negative, a domain error occurs.

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    double x;

    errno = 0;
    x = sqrtf (0.0F);
    if (errno)
        perror("Error");
    printf("The square root of 0.0F is %f\n", x);

    errno = 0;
    x = sqrtf (9.5F);
    if (errno)
        perror("Error");
    printf("The square root of 9.5F is %f\n", x);

    errno = 0;
    x = sqrtf (-25.0F);
    if (errno)
        perror("Error");
    printf("The square root of -25F is %f\n", x);
}
```

Example Output

```
The square root of 0.0F is 0.000000
The square root of 9.5F is 3.082207
Error: domain error
The square root of -25F is nan
```

3.1.42 tan Function

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

Include

```
<math.h>
```

Prototype

```
double tan (double x);
```

Argument

x value for which to return the tangent

Return Value

Returns the tangent of x in radians.

Remarks

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

Example

```
#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
```

```

{
    double x, y;

    errno = 0;
    x = -1.0;
    y = tan (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n", x, y);

    errno = 0;
    x = 0.0;
    y = tan (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n", x, y);
}

```

Example Output

```

The tangent of -1.000000 is -1.557408
The tangent of 0.000000 is 0.000000

```

3.1.43 tanf Function

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* in radians.

Remarks

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

Example

```

#include <math.h>
#include <stdio.h>
#include <errno.h>

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = tanf (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = tanf (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n", x, y);
}

```

Example Output

```
The tangent of -1.000000 is -1.557408
The tangent of 0.000000 is 0.000000
```

3.1.44 tanh Function

Calculates the hyperbolic tangent function of a double precision floating-point value.

Include

```
<math.h>
```

Prototype

```
double tanh(double x);
```

Argument

x value for which to return the hyperbolic tangent

Return Value

Returns the hyperbolic tangent of *x* in the ranges of -1 to 1 inclusive.

Remarks

No domain or range error will occur.

Example

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

int main(void)
{
    double x, y;

    x = -1.0;
    y = tanh(x);
    printf("The hyperbolic tangent of %f is %f\n", x, y);

    x = 2.0;
    y = tanh(x);
    printf("The hyperbolic tangent of %f is %f\n", x, y);
}
```

Example Output

```
The hyperbolic tangent of -1.000000 is -0.761594
The hyperbolic tangent of 2.000000 is 0.964028
```

3.1.45 tanhf Function

Calculates the hyperbolic tangent function of a single precision floating-point value.

Include

```
<math.h>
```

Prototype

```
float tanhf(float x);
```

Argument

x value for which to return the hyperbolic tangent

Return Value

Returns the hyperbolic tangent of *x* in the ranges of -1 to 1 inclusive.

Remarks

No domain or range error will occur.

Example

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

int main(void)
{
    float x, y;

    x = -1.0F;
    y = tanhf(x);
    printf("The hyperbolic tangent of %f is %f\n", x, y);

    x = 0.0F;
    y = tanhf(x);
    printf("The hyperbolic tangent of %f is %f\n", x, y);
}
```

Example Output

```
The hyperbolic tangent of -1.000000 is -0.761594
The hyperbolic tangent of 0.000000 is 0.000000
```

4. Support Libraries

Support libraries contain support functions that either must be customized for correct operation of the Standard C Library in your target environment or are already customized for a Microchip target environment. The default behavior section describes what the function does, as it is distributed. The description and remarks describe what it typically should do. The corresponding object modules are distributed in the `libpic30-omf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder.

For details on using the standard C libraries, see “Standard C Libraries.”

4.1 Rebuilding the libpic30 Library

By default, the Support Libraries helper functions were written to work with the `sim30` simulator. The header file `simio.h` defines the interface between the library and the simulator. It is provided so you can rebuild the libraries and continue to use the simulator. However, your application should not use this interface since the simulator will not be available to an embedded application.

The helper functions must be modified and rebuilt for your target application. The `libpic30-omf.a` library can be rebuilt with the batch file named `makelib.bat` which has been provided with the sources in `src\pic30`. Execute the batch file from a command window. Be sure you are in the `src\pic30` directory, then copy the newly compiled file (`libpic30-omf.a`) into the `lib` directory.

4.2 Standard C Library Helper Functions

These functions are called by other functions in the standard C library and must be modified for the target application. The corresponding object modules are distributed in the `libpic30-omf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder.

4.2.1 `_dump_heap_info` Function

Displays the current use/free state of the heap.

Include

None

Prototype

```
void _dump_heap_info();
```

Argument

None

Remarks

This helper function can be used to profile memory allocation in the heap. It is useful for memory allocation functions.

Default Behavior

This function prints information about the heap to `stderr`. Information includes the memory region where heap is located, the different sections of the heap, whether a section is currently busy or free, and a summary of how much heap is free.

For example, below, a heap of 1000 has been allocated. The heap begins and ends at `d0c` and `10f4`, respectively. The `malloc` function has been executed 3 different times, resulting in 3 different sections of `BUSY` memory and 1 `FREE` memory section of heap that has not been used. If the `free` function were to be used on any of these 3 allocated sections, the status would turn from `BUSY` to `FREE` as well.

```
*** Unused Heap status:
***   start: 0x00000d0c   end: 0x000010f4
```

```

*** 0d0c      8 BUSY
*** 0d14     16 BUSY
*** 0d24     20 BUSY
*** 0d38    956 FREE
*** 44 used, 956 free, 010f0 end

```

File

_dump_heap_info.c

4.2.2 _exit Function

Terminate program execution.

Include

None

Prototype

```
void _exit (int status);
```

Argument

status	exit status
---------------	-------------

Remarks

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

Default Behavior

As distributed, this function flushes `stdout` and terminates. The parameter `status` is the same as that passed to the `exit()` standard C library function.

File

_exit.c

4.2.3 brk Function

Set the end of the process's data space.

Include

None

Prototype

```
int brk(void *endds);
```

Argument

endds	pointer to the end of the data segment
--------------	--

Return Value

Returns '0' if successful; otherwise, returns '-1'.

Remarks

`brk()` is used to dynamically change the amount of space allocated for the calling process's data segment. The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.

Newly allocated space is uninitialized.

This helper function is used by the Standard C Library function `malloc()`.

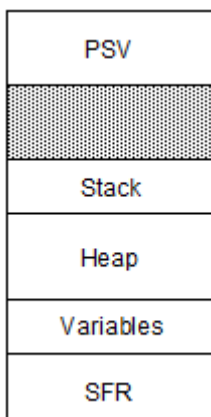
Default Behavior

If the argument `endds` is zero, the function sets the global variable `__curbrk` to the address of the start of the heap and returns zero.

If the argument `endds` is non-zero and has a value less than the address of the end of the heap, the function sets the global variable `__curbrk` to the value of `endds` and returns zero.

Otherwise, the global variable `__curbrk` is unchanged and the function returns -1.

The argument `endds` must be within the heap range (see data space memory map below).



Since the stack is located immediately above the heap, using `brk()` or `sbrk()` has little effect on the size of the dynamic memory pool. The `brk()` and `sbrk()` functions are primarily intended for use in run-time environments where the stack grows downward and the heap grows upward.

The linker allocates a block of memory for the heap if the `-Wl, --heap=n` option is specified, where `n` is the desired heap size in characters. The starting and ending addresses of the heap are reported in variables: `_heap` and `_ehheap`, respectively.

For the 16-bit compiler, using the linker's heap size option is the standard way of controlling heap size, rather than relying on `brk()` and `sbrk()`.

File

`brk.c`

4.2.4 close Function

Close a file.

Include

None

Prototype

```
int close(int handle);
```

Argument

handle	handle referring to an opened file
---------------	------------------------------------

Return Value

Returns '0' if the file is successfully closed. A return value of '-1' indicates an error.

Remarks

This helper function is called by the `fclose()` Standard C Library function.

Default Behavior

As distributed, this function passes the file handle to the simulator, which issues a close in the host file system.

File

close.c

4.2.5 lseek Function

Move a file pointer to a specified location.

Include

None

Prototype

```
long lseek(int handle, long offset, int origin);
```

Arguments

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

Return Value

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

Remarks

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

Default Behavior

As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.

File

lseek.c

4.2.6 open Function

Open a file.

Include

None

Prototype

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

Arguments

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

Return Value

If successful, the function returns a file handle: a small positive integer. This handle is then used on subsequent low-level file I/O operations. A return value of '-1' indicates an error.

Remarks

The access flag is a union of one of the following access methods and zero or more access qualifiers:

- 0 – Open a file for reading.
- 1 – Open a file for writing.
- 2 – Open a file for both reading and writing.

The following access qualifiers must be supported:

- 0x0008 – Move file pointer to end-of-file before every write operation.
- 0x0100 – Create and open a new file for writing.
- 0x0200 – Open the file and truncate it to zero length.
- 0x4000 – Open the file in text (translated) mode.
- 0x8000 – Open the file in binary (untranslated) mode.

The mode parameter may be one of the following:

- 0x0100 – Reading only permitted.
- 0x0080 – Writing permitted (implies reading permitted).

This helper function is called by the Standard C Library functions `fopen()` and `freopen()`.

Default Behavior

As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system. If the host system returns a value of '-1', the global variable `errno` is set to the value of the symbolic constant, `EFOPEN`, defined in `<errno.h>`.

File

`open.c`

4.2.7 read Function

Read data from a file.

Include

None

Prototype

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

Arguments

handle	handle referring to an opened file
buffer	points to the storage location for read data
len	the maximum number of characters to read

Return Value

Returns the number of characters read, which may be less than `len` if there are fewer than `len` characters left in the file or if the file was opened in text mode, in which case, each carriage return-linefeed (CR-LF) pair is replaced with a single linefeed character. Only the single linefeed character is counted in the return value. The replacement does not affect the file pointer. If the function tries to read at end-of-file, it returns '0'. If the handle is invalid, or the file is not open for reading or the file is locked, the function returns '-1'.

Remarks

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

Default Behavior

As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.

File

read.c

4.2.8 sbrk Function

Extend the process' data space by a given increment.

Include

None

Prototype

```
void * sbrk(int incr);
```

Argument

incr	number of characters to increment/decrement
-------------	---

Return Value

Return the start of the new space allocated or '-1' for errors.

Remarks

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

`sbrk()` is used to dynamically change the amount of space allocated for the calling process's data segment. The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.

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

Default Behavior

If the global variable `__curbrk` is zero, the function calls `brk()` to initialize the break value. If `brk()` returns -1, so does this function.

If the *incr* is zero, the current value of the global variable `__curbrk` is returned.

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

Otherwise, the function returns -1.

See the of `brk()`.

File

sbrk.c

4.2.9 write Function

Write data to a file.

Include

None

Prototype

```
int __attribute__((__section__(".libc.write"))) write(int handle, void *buffer,  
unsigned int count);
```

Arguments

handle	refers to an opened file
buffer	points to the storage location of data to be written

count the number of characters to write.

Return Value

If successful, write returns the number of characters actually written. A return value of '-1' indicates an error.

Remarks

If the actual space remaining on the disk is less than the size of the buffer, the function trying to write to the disk write fails and does not flush any of the buffer's contents to the disk. If the file is opened in text mode, each linefeed character is replaced with a carriage return – linefeed pair in the output. The replacement does not affect the return value.

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

Default Behavior

As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.

File

write.c

4.3 Standard C Library Functions That Require Modification

Although these functions are part of the Standard C Library, the object modules are distributed in the `libpic30-omf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder. These modules are not distributed as part of `libc-omf.a`

- `getenv` - in `stdlib.h`
- `remove` - in `stdio.h`
- `rename` - in `stdio.h`
- `system` in `stdlib.h`
- `time` - in `time.h`

Related Links

[2.15.13 getenv Function](#)

[2.14.32 remove Function](#)

[2.14.33 rename Function](#)

[2.15.17 system Function](#)

[2.17.10 time Function](#)

4.4 Functions/Constants to Support A Simulated UART

These functions and constants support UART functionality in the MPLAB SIM simulator.

Examples of Use

Example 1: UART1 I/O

```
#include <libpic30.h> /* a new header file for these definitions */
#include <stdio.h>
void main() {
    if (__attach_input_file("foo.txt")) {
        while (!feof(stdin)) {
            putchar(getchar());
        }
        __close_input_file();
    }
}
```

Example 2: Using UART2

```

/* This program flashes a light and transmits a lot of messages at
   9600 8n1 through uart 2 using the default stdio provided by the
   16-bit compiler. This is for a dsPIC33F DSC on an Explorer 16(tm) board
   (and isn't very pretty) */
#include <libpic30.h> /* a new header file for these definitions */
#include <stdio.h>

#ifndef __dsPIC33F
#error this is a 33F demo for the explorer 16(tm) board
#endif
#include <p33Fxxxx.h>

_FOSCSEL(FNOSC_PRI );
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_XT);
_FWDT(FWDTEN_OFF);

main() {
    ODCA = 0;
    TRISAbits.TRISA6 = 0;
    __C30_UART=2;
    U2BRG = 38;
    U2MODEbits.UARTEN = 1;
    while (1) {
        __builtin_btg(&LATA,6);
        printf("Hello world %d\n",U2BRG);
    }
}

```

Example 3: Millisecond Delay

```

#define FCY 1000000UL
#include <libpic30.h>
int main() {
    /* at 1MHz, these are equivalent */
    __delay_ms(1);
    __delay32(1000);
}

```

Example 4: Microsecond Delay

```

#define FCY 1000000UL
#include <libpic30.h>
int main() {
    /* at 1MHz, these are equivalent */
    __delay_us(1000);
    __delay32(1000);
}

```

4.4.1 __C30_UART Constant

Constant that defines the default UART.

Include

N/A

Prototype

```
int __C30_UART;
```

Remarks

Defines the default UART that `read()` and `write()` will use for `stdin` (unless a file has been attached) and `stdout`.

Default Behavior

By default, or with a value of 1, UART 1 will be used. Otherwise, UART 2 will be used. `read()` and `write()` are the eventual destinations of the C standard I/O functions.

4.4.2 __attach_input_file Function

Attach a hosted file to the standard input stream.

Include

```
<libpic30.h>
```

Prototype

```
int __attach_input_file(const char *p);
```

Argument

p	pointer to file
----------	-----------------

Remarks

This function differs from the MPLAB X IDE mechanism of providing an input file because it provides “on-demand” access to the file. That is, data will only be read from the file upon request and the asynchronous nature of the UART is not simulated. This function may be called more than once; any opened file will be closed. It is only appropriate to call this function in a simulated environment.

Default Behavior

Allows the programmer to attach a hosted file to the standard input stream, `stdin`.

The function will return 0 to indicate failure. If the file cannot be opened for whatever reason, standard in will remain connected (or be re-connected) to the simulated UART.

File

`attach.c`

4.4.3 __close_input_file Function

Close a previously attached file.

Include

```
<libpic30.h>
```

Prototype

```
void __close_input_file(void);
```

Argument

None

Remarks

None.

Default Behavior

This function will close a previously attached file and re-attach `stdin` to the simulated UART. This should occur before a Reset to ensure that the file can be re-opened.

File

`close.c`

4.4.4 __delay32 Function

Produce a delay of a specified number of clock cycles.

Include

```
<libpic30.h>
```

Prototype

```
void __delay32(unsigned long cycles);
```

Argument

cycles	number of cycles to delay
---------------	---------------------------

Remarks

None.

Default Behavior

This function will effect a delay of the requested number of cycles. The minimum supported delay is 12 cycles (an argument of less than or equal to 12 will result in 12 cycles). The delay includes the call and return statements, but not any cycles required to set up the argument (typically this would be two for a literal value).

File

delay32.s

4.4.5 __delay_ms Function

Produce a delay of a specified number of milliseconds (ms).

Include

```
<libpic30.h>
```

Prototype

```
void __delay_ms(unsigned int time);
```

Argument

time	number of ms to delay
-------------	-----------------------

Remarks

This function is implemented as a macro.

Default Behavior

This function relies on a user-supplied definition of FCY to represent the instruction clock frequency. FCY must be defined before header file `libpic30.h` is included. The specified delay is converted to the equivalent number of instruction cycles and passed to `__delay32()`. If FCY is not defined, then `__delay_ms()` is declared external, causing the link to fail unless the user provides a function with that name.

File

delay32.s

4.4.6 __delay_us Function

Produce a delay of a specified number of microseconds (us).

Include

```
<libpic30.h>
```

Prototype

```
void __delay_us(unsigned int time);
```

Argument

time	number of us to delay
-------------	-----------------------

Remarks

This function is implemented as a macro. The minimum delay is equivalent to 12 instruction cycles.

Default Behavior

This function relies on a user-supplied definition of FCY to represent the instruction clock frequency. FCY must be defined before header file `libpic30.h` is included. The specified delay is converted to the equivalent number of instruction cycles and passed to `__delay32()`. If FCY is not defined, then `__delay_ms()` is declared external, causing the link to fail unless the user provides a function with that name.

File

`delay32.s`

4.5 Functions for Erasing and Writing EEDATA Memory

These functions support the erasing and writing of EEDATA memory for devices that have this type of memory.

Examples of Use

Example 1: dsPIC30F DSCs

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

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

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

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

    _init_prog_address(p, dat); /* get address in program space */
    _erase_eedata(p, _EE_ROW); /* erase a row */
    _wait_eedata(); /* wait for operation to complete */
    _write_eedata_row(p, source); /* write a row */
}
```

Example 2: PIC24FXXKA MCUs

```
#include "libpic30.h" /* should use <> here */
#include "p24Fxxxx.h"

int __attribute__((space(eedata), aligned(_EE_4WORDS))) dat[_EE_4WORDS/2];

int main() {
    _prog_addressT p;
    _init_prog_address(p, dat); /* get address in program space */
    _erase_eedata(p, _EE_4WORDS); /* erase the dat[] array */
    _wait_eedata(); /* wait to complete */
    _write_eedata_word(p, 0x1234); /* write a word to dat[0] */
    _wait_eedata();

    p += 2;
    _write_eedata_word(p, 0x5678); /* write a word to dat[1] */
    _wait_eedata();
}
```

4.5.1 _erase_eedata Function

Erase EEDATA memory on dsPIC30F and PIC24FXXKA devices.

Include

`<libpic30.h>`

Prototype

```
void _erase_eedata(_prog_addressT dst, int len);
```

Argument

dst	destination memory address
len	dsPIC30F: length may be <code>_EE_WORD</code> or <code>_EE_ROW</code> (bytes) PIC24FxxKA: length may be <code>_EE_WORD</code> , <code>_EE_4WORDS</code> or <code>_EE_8WORDS</code> (bytes)

Return Value

None.

Remarks

None.

Default Behavior

Erase EEDATA memory as specified by parameters.

File

eedata_helper.c

4.5.2 `_erase_eedata_all` Function

Erase the entire range of EEDATA memory on dsPIC30F and PIC24FXXKA devices.

Include

<libpic30.h>

Prototypevoid `_erase_eedata_all`(void);**Argument**

None

Return Value

None.

Remarks

None.

Default Behavior

Erase all EEDATA memory for the selected device.

File

eedata_helper.c

4.5.3 `_wait_eedata` Function

Wait for an erase or write operation to complete on dsPIC30F and PIC24FXXKA devices.

Include

<libpic30.h>

Prototypevoid `_wait_eedata`(void);**Argument**

None

Return Value

None.

Remarks

None.

Default Behavior

Wait for an erase or write operation to complete.

File

eedata_helper.c

4.5.4 _write_eedata_row Function

Write _EE_ROW bytes of EEDATA memory on dsPIC30F devices.

Include

<libpic30.h>

Prototype

```
void _write_eedata_row(_prog_addressT dst, int *src);
```

Arguments

dst	destination memory address
src	points to the storage location of data to be written

Return Value

None.

Remarks

None.

Default Behavior

Write specified bytes of EEDATA memory.

File

eedata_helper.c

4.5.5 _write_eedata_word Function

Write 16 bits of EEDATA memory on dsPIC30F and PIC24FXXKA devices.

Include

<libpic30.h>

Prototype

```
void _write_eedata_word(_prog_addressT dst, int dat);
```

Arguments

dst	destination memory address
dat	integer data to be written

Return Value

None.

Remarks

None.

Default Behavior

Write one word of EEDATA memory for dsPIC30F devices.

File

eedata_helper.c

4.6 Functions for Erasing and Writing Flash Memory

These functions support the erasing and writing of Flash memory for devices that have this type of memory.

Examples of Use

Example 1: 16-Bit MCUs

```
#include "libpic30.h"
#include "p24Fxxxx.h"

int __attribute__((space(prog),aligned(_FLASH_PAGE*2))) dat[_FLASH_PAGE];

int main() {
    int i;
    int source1[_FLASH_ROW];
    long source2[_FLASH_ROW];
    _prog_addressT p;

    for (i = 0; i < _FLASH_ROW; ) {
        source1[i] = i;
        source2[i] = i++;
    } /* initialize some data */

    _init_prog_address(p, dat); /* get address in program space */
    _erase_flash(p); /* erase a page */
    _write_flash16(p, source1); /* write first row with 16-bit data */

    #if defined (__dsPIC30F__)
        _erase_flash(p); /* on dsPIC30F, only 1 row per page */
    #else
        p += (_FLASH_ROW * 2); /* advance to next row */
    #endif

    _write_flash24(p, source2); /* write second row with 24-bit data */
}
```

Example 2: PIC24FXXKA MCUs

```
#include "libpic30.h" /* should use <> here */
#include "p24Fxxxx.h"

int __attribute__((space(prog),aligned(_FLASH_2ROWS*2))) dat[_FLASH_2ROWS];

int main() {
    int i;
    int source1[_FLASH_ROW];
    long source2[_FLASH_ROW];
    _prog_addressT p;

    for (i = 0; i < _FLASH_ROW; ) {
        source1[i] = i;
        source2[i] = i++;
    } /* initialize some data */

    _init_prog_address(p, dat); /* get address in program space */
    _erase_flash(p, _FLASH_2ROWS); /* erase two rows */
    _write_flash16(p, source1); /* write first row with 16-bit data */

    p += (_FLASH_ROW * 2); /* advance to next row */
    _write_flash24(p, source2); /* write second row with 24-bit data */
}
```

4.6.1 _erase_flash Function

Erase a page of Flash memory. The length of a page is _FLASH_PAGE words (1 word = 3 bytes = 2 PC address units).

Include

```
<libpic30.h>
```

Prototype

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

Argument

dst	destination memory address
------------	----------------------------

Return Value

None.

Remarks

None.

Default Behavior

Erase a page of Flash memory.

File

flash_helper.s

4.6.2 _erase_flash (PIC24FXXKA Only) Function

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

Include

```
<libpic30.h>
```

Prototype

```
void _erase_flash(_prog_addressT dst, int len);
```

Arguments

dst	destination memory address
len	length may be _FLASH_ROW, _FLASH_2ROWS or _FLASH_4ROWS (bytes)

Return Value

None.

Remarks

None.

Default Behavior

Erase rows of Flash memory.

File

flash_helper2.s

4.6.3 _write_flash16 Function

Write a row of Flash memory with 16-bit data. The length of a row is _FLASH_ROW words. The upper byte of each destination word is filled with 0xFF. Note that the row must be erased before any write can be successful.

Include

```
<libpic30.h>
```

Prototype

```
void _write_flash16(_prog_addressT dst, int *src);
```

Arguments

dst	destination memory address
src	points to the storage location of data to be written

Return Value

None.

Remarks

None.

Default Behavior

Write a row of Flash memory with 16-bit data.

File

flash_helper.c

4.6.4 _write_flash24 Function

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

Include

<libpic30.h>

Prototype

```
void _write_flash24(_prog_addressT dst, int *src);
```

Arguments

dst	destination memory address
src	points to the storage location of data to be written

Return Value

None.

Remarks

None.

Default Behavior

Write a row of Flash memory with 24-bit data.

File

flash_helper.c

4.6.5 _write_flash_word16 Function

Write a word of Flash memory with 16-bit data. The upper byte of the destination word is filled with 0xFF. Note that the word must be erased before any write can be successful. This function is currently available only for PIC24F devices (excluding PIC24FXXKA MCUs).

Include

<libpic30.h>

Prototype

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

Arguments

dst	destination memory address
------------	----------------------------

dat integer data to be written

Return Value

None.

Remarks

None.

Default Behavior

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

File

flash_helper.c

4.6.6 _write_flash_word24 Function

Write a word of Flash memory with 24-bit data. Note that the word must be erased before any write can be successful. This function is currently available only for PIC24F devices (excluding PIC24FXXKA MCUs).

Include

<libpic30.h>

Prototype

```
void _write_flash_word24(_prog_addressT dst, long dat);
```

Arguments

dst destination memory address
dat integer data to be written

Return Value

None.

Remarks

None.

Default Behavior

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

File

flash_helper.c

4.6.7 _write_flash_word32 Function

Write two words of FLASH memory with 16 bits of data per word. The 16 bits are written to the low 16 bits of the word. Word writes are supported dsPIC33E and PIC24E devices. The row address is specified with type `_prog_addressT`. Note that the location must be erased before any write can be successful.

This function is currently disabled for devices subject to the Device ID errata as described in DS-80444, DS-80446, or DS-80447 (#32).

Include

<libpic30.h>

Prototype

```
void _write_flash_word32(_prog_addressT dst, int dat1, int dat2);
```

Arguments

dst	destination memory address
dat1, dat2	integer data to be written

Return Value

None.

Remarks

None.

Default Behavior

Write two words of Flash memory with 16-bit data for most dsPIC33E/PIC24E devices.

File

flash_helper.c

4.6.8 _write_flash_word48 Function

Write two words of FLASH memory with 24 bits of data per word. Word writes are supported 33E and 24E devices. The row address is specified with type `_prog_addressT`. Note that the location must be erased before any write can be successful.

This function is currently disabled for devices subject to the Device ID errata as described in DS-80444, DS-80446, or DS-80447 (#32).

Include

<libpic30.h>

Prototype

```
void _write_flash_word48(_prog_addressT dst, int dat1, int dat2);
```

Arguments

dst	destination memory address
dat1, dat2	integer data to be written

Return Value

None.

Remarks

None.

Default Behavior

Write two words of Flash memory with 48-bit data for most dsPIC33E/PIC24E devices.

File

flash_helper.c

4.7 Functions for Specialized Copying and Initialization

These functions support specialized data copying and initialization.

Example of Use

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

void display_mem(char *p, unsigned int len) {
    int i;
    for (i = 0; i < len; i++) {
```

```

    printf(" %d", *p++);
}
printf("\n");
}

char __attribute__((space(prog))) dat[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
char buf[10];

int main() {
    int i;
    _prog_addressT p;

    /* method 1 */
    _init_prog_address(p, dat);
    (void) _memcpy_p2d16(buf, p, 10);
    display_mem(buf, 10);

    /* method 2 */
    _init_prog_address(p, dat);
    p = _memcpy_p2d16(buf, p, 4);
    p = _memcpy_p2d16(&buf[4], p, 6);
    display_mem(buf, 10);
}

```

4.7.1 **_init_prog_address Macro**

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

Include

```
<libpic30.h>
```

Prototype

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

Arguments

- | | |
|----------|--|
| a | variable of type <code>_prog_addressT</code> |
| b | initialization value for variable <i>a</i> |

Return Value

N/A

Remarks

None.

Default Behavior

Initialize variable to specified value.

File

libpic30.c

4.7.2 **_memcpy_eds Function**

Copies bytes from one `__eds__` buffer to another `__eds__` buffer.

Include

```
<libpic30.h>
```

Prototype

```
__eds__ void* _memcpy_eds(__eds__ void *dest, __eds__ void *src, unsigned int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The memory block pointed to by `dest`.

Remarks

`__eds__` pointers are superset of unqualified data pointers; therefore these functions can be used to copy between `__eds__` and unqualified data memory.

Default Behavior

Copy `len` bytes of data from each address pointed to by the `src` pointer to the destination pointed to by the `dest` pointer.

File

`memcpy_helper.s`

4.7.3 `_memcpy_p2d16` Function

Copy 16 bits of data from each address in program memory to data memory. The next unused source address is returned.

Include

`<libpic30.h>`

Prototype

```
_prog_addressT _memcpy_p2d16(char *dest, _prog_addressT src, unsigned int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The next unused source address.

Remarks

None.

Default Behavior

Copy 16 bits of data from each of the `len` bytes of addresses of `src` to the destination pointed to by the `dest` pointer.

File

`memcpy_helper.s`

4.7.4 `_memcpy_p2d24` Function

Copy 24 bits of data from each address in program memory to data memory. The next unused source address is returned.

Include

`<libpic30.h>`

Prototype

```
_prog_addressT _memcpy_p2d24(char *dest, _prog_addressT src, unsigned int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The next unused source address.

Remarks

None.

Default Behavior

Copy 24 bits of data from each of the `len` bytes of addresses of `src` to the destination pointed to by the `dest` pointer.

File

`memcpy_helper.s`

4.7.5 _memcpy_packed Function

Copies bytes from one `__pack_upper_byte` Flash buffer to another buffer in RAM.

Include

```
<libpic30.h>
```

Prototype

```
void* _memcpy_packed(void *dest, __pack_upper_byte void *src, unsigned int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The memory block pointed to by `dest`.

Remarks

None.

Default Behavior

Copy `len` bytes of data from each address pointed to by the `src` pointer to the destination pointed to by the `dest` pointer.

File

`memcpy_helper.s`

4.7.6 _strcpy_eds Function

Copies a string from one `__eds__` buffer to another `__eds__` buffer.

Include

```
<libpic30.h>
```

Prototype

```
__eds__ char* _strcpy_eds(__eds__ char *dest, __eds__ char *src);
```

Arguments

dest	destination memory address
src	address of data to be written

Return Value

The string pointed to by `dest`.

Remarks

`__eds__` pointers are superset of unqualified data pointers; therefore these functions can be used to copy between `__eds__` and unqualified data memory.

Default Behavior

Copy the string `src` pointer to the destination pointed to by the `dest` pointer.

File

`memcpy_helper.s`

4.7.7 _strcpy_packed Function

Copies a string from one `__pack_upper_byte` Flash buffer to another buffer in RAM.

Include

```
<libpic30.h>
```

Prototype

```
void* _strcpy_packed( void *dest, __pack_upper_byte void *src);
```

Arguments

dest	destination memory address
src	address of data to be written

Return Value

The string pointed to by `dest`.

Remarks

None.

Default Behavior

Copy the string pointed to by `src` to the RAM buffer pointed to by `dest`. Unlike the standard `strcpy` function, this function does not zero fill the remaining space in the destination string.

File

`memcpy_helper.s`

4.7.8 _strncpy_eds Function

Copies at most `len` bytes string from one `__eds__` buffer to another `__eds__` buffer.

Include

```
<libpic30.h>
```

Prototype

```
__eds__ char* _strncpy_eds(__eds__ char *dest, __eds__ char *src, unsigned int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The string pointed to by *dest*.

Remarks

`__eds__` pointers are superset of unqualified data pointers; therefore these functions can be used to copy between `__eds__` and unqualified data memory.

Default Behavior

Copy *len* bytes of the string pointed to by the *src* pointer to the destination pointed to by the *dest* pointer. Unlike the standard `strcpy` function, this function does not zero fill the remaining space in the destination string.

File

`memcpy_helper.s`

4.7.9 `_strncpy_p2d16` Function

Copy 16 bits of data from each address in program memory to data memory. The operation terminates early if a NULL char is copied. The next unused source address is returned.

Include

`<libpic30.h>`

Prototype

```
_prog_addressT _strncpy_p2d16(char *dest, _prog_addressT src, unsigned int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The next unused source address.

Remarks

None.

Default Behavior

Copy 16 bits of data from each of the *len* bytes of addresses of *src* to the destination pointed to by the *dest* pointer.

File

`memcpy_helper.s`

4.7.10 `_strncpy_p2d24` Function

Copy 24 bits of data from each address in program memory to data memory. The operation terminates early if a NULL char is copied. The next unused source address is returned.

Include

`<libpic30.h>`

Prototype

```
_prog_addressT _strncpy_p2d24(char *dest, _prog_addressT src, unsigned int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The next unused source address.

Remarks

None.

Default Behavior

Copy 24 bits of data from each of the `len` bytes of addresses of `src` to the destination pointed to by the `dest` pointer.

File

`memcpy_helper.s`

4.7.11 _strncpy_packed Function

Copies at most `len` bytes of a string from one `__pack_upper_byte` Flash buffer to another buffer in RAM.

Include

```
<libpic30.h>
```

Prototype

```
void* _strncpy_packed( void *dest, __pack_upper_byte void *src, int len);
```

Arguments

dest	destination memory address
src	address of data to be written
len	length of program memory

Return Value

The string pointed to by `dest`.

Remarks

None.

Default Behavior

Copy at most `len` bytes of the string pointed to by `src` to the RAM buffer pointed to by `dest`. Unlike the standard `strncpy` function, this function does not zero fill the remaining space in the destination string.

File

`memcpy_helper.s`

4.8 Functions to Support Secondary Core PRAM

These utility functions program the Secondary core with the specified image which has been generated by an MPLAB X IDE project. These functions will not work unless using the XC16 model of programming. Outside of this paradigm, refer to the device programming specification.

Examples of Use

Example 1: Secondary PRAM Load and Verify Routine

```
#include <libpic30.h>
//wipe memory before programming
_wipe_secondary(&secondary_image)

//_program_secondary(core#, verify, &secondary_image)
if (_program_secondary(1, 0, &secondary_image) == 0)
{
    /* now verify */
    if (_program_secondary(1, 1, &secondary_image) == ESEC_VERIFY_FAIL)
    {
        asm("reset") ; // try again
    }
}
```

Example 2: Secondary Core Start and Stop

```
#include <libpic30.h>
int main()
{
    // Main initialization code
    _start_secondary(); // Start Secondary core

    // Main application code
    _stop_secondary(); // Stop Secondary core

    while(1);
}
```

Reference Documents:

ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33CH128MP508-Family-Data-Sheet-DS70005319D.pdf (See Section 4.3)

ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33CH512MP508-Family-Data-Sheet-DS70005371D.pdf (See Section 4.3)

4.8.1 _wipe_secondary Function

Erases the Secondary core to the erase state.

Include

```
<libpic30.h>
```

Prototype

```
int _wipe_secondary(int secondary_number)
```

Arguments

secondary_number	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is 1.
-------------------------	--

Return Value

- ESLV_INVALID - returned if an invalid secondary number is given

Remarks

None.

Default Behaviour

The `_wipe_secondary` routine can be used to completely wipe the PRAM memory of the Secondary core. This routine will always wipe the entire memory range, even if dual partition mode is selected.

File

`wipe_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

4.8.2 _program_secondary Function

Programs the Secondary core with the specified image.

Include

`<libpic30.h>`

Prototype

```
int _program_secondary(int secondary_number, int verify, __eds__ unsigned char *image)
```

Arguments

secondary_number	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is 1.
verify	A 0 will load the entire Secondary image to the PRAM. A 1 will verify the entire image in the PRAM.
*image	Pointer to the image to be programmed into PRAM. Secondary core PRAM images not following the Microchip language tool format will require a custom routine.

Return Value

- `ESLV_INVALID` - returned if an invalid Secondary core number is given
- `ESLV_BAD_IMAGE` - returned if there is an error decoding the Secondary core image
- `ESLV_VERIFY_FAIL` - returned if 'verify' fails

Remarks

None.

Default Behavior

The `_program_secondary` routine uses the `verify` parameter as a switch to either load or verify the Secondary core PRAM image using the `LDSEC` or `VFSEC` instructions.

File

`program_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

4.8.3 _program_inactive_secondary Function

Programs the inactive partition PRAM with the specified image.

Include

`<libpic30.h>`

Prototype

```
int _program_inactive_secondary(int secondary_number, int verify, __eds__ unsigned char *image)
```

Arguments

secondary_number	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is 1.
-------------------------	--

verify	A 0 will load the entire Secondary image to the PRAM. A 1 will verify the entire image in the PRAM.
*image	Pointer to the image to be programmed into PRAM. Secondary PRAM images not following the Microchip language tool format will require a custom routine.

Return Value

- ESLV_INVALID - returned if an invalid Secondary core number is given
- ESLV_BAD_IMAGE - returned if there is an error decoding the Secondary core image
- ESLV_VERIFY_FAIL - returned if 'verify' fails

Remarks

The Main core loads the PRAM inactive partition while the Secondary core is running and then the Secondary core executes the `BOOTSMP` instruction to swap partitions.

Default Behavior

The `_program_inactive_secondary` routine uses the **verify** parameter as a switch to either load or verify the inactive partition image using the `LDSEC` or `VFSEC` instructions.

File

`program_inactive_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

4.8.4 _start_secondary and _stop_secondary Functions

Start or stop the Secondary core after the image has been loaded by the Main core.

Include

`<libpic30.h>`

Prototypes

`void _start_secondary(void)`

`void _stop_secondary(void)`

Arguments

None.

Return Value

None.

Remarks

None.

Default Behavior

The `_start_secondary` and `_stop_secondary` routines perform the MNI1KEY unlock sequence and set or clear the SECEN bit (MNI1CON[15]) respectively.

Files

`start_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

`stop_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

5. Fixed-Point Math Functions

Fixed-point library math functions are contained in the files `libq-omf.a` (standard) and `libq-dsp-omf.a` (DSP), where `omf` will be `elf` or `coff` depending upon the selected object module format. The header file is named `libq.h` and is the same for standard or DSP versions of the library. Linker options `-lq` (standard and DSP) and `-lq-dsp` (DSP only) must be used when linking the respective libraries.

5.1 Overview of Fixed-Point Data Formats

The integer data is encoded as its two's complement to accommodate both positive and negative numbers in binary format. The two's complement can be represented using integer format or the fractional format.

Integer Format

The integer format data is represented as a signed two's complement value, where the Most Significant bit is defined as a sign bit. The range of an N-bit two's complement integer is -2^{N-1} to $2^{N-1}-1$ with a resolution of 1. For a 16-bit integer, the data range is -32768 (0x8000) to +32767 (0x7FFF) with a resolution of 1. For a 32-bit integer, the data range is -2,147,483,648 (0x8000 0000) to +2,147,483,647 (0x7FFF FFFF) with a resolution of 1.

Fractional Format

The fractional data format (Qn.m) has integral part (n) and fractional part (m) and the Most Significant bit represents the sign, thus consisting of (m+n+1) bits. It represents a signed two's complement value. Qn.m format data has a range of $[-2^n, (2^n-2^{-m})]$ with 2^{-m} resolution.

The binary representation of an N-bit (m+n+1 bits) number in Qn.m is shown in Figure 1. The value is given by the equation shown in Figure 2.

Figure 1: Binary Representation

$$\underbrace{b_{m+n} \ b_{m+n-1} \ \dots \ b_m}_{N-1} . b_{m-1} \ \dots \ b_1 \ b_0$$

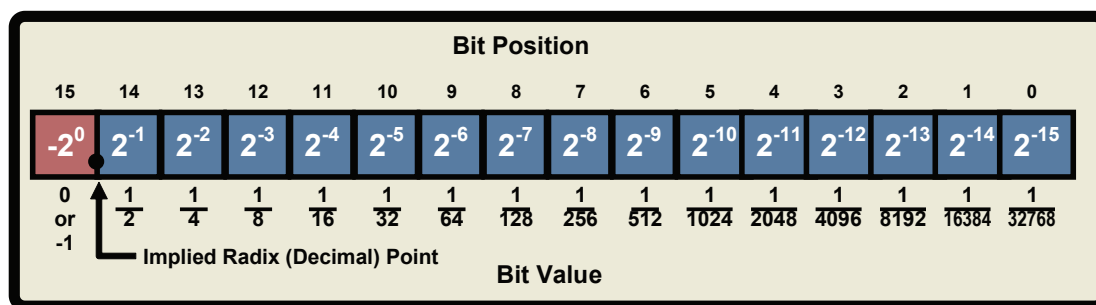
Figure 2: Equation Value

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

Q15 (1.15) Format



In Q15 format, the Most Significant bit is defined as a sign bit and the radix point is implied to lie just after the sign bit followed by the fractional value. This format is commonly referred to as 1.15 or Q15 format, where 1 is the number of bits used to represent the integer portion of the number, and 15 is the number of bits used to represent the fractional portion. The range of an N-bit two's complement fraction with this implied radix point is -1.0 to $(1 - 2^{1-N})$. For a 16-bit fraction, the 1.15 data range is -1.0 (0x8000) to +0.999969482 (0x7FFF) with a precision of 3.05176×10^{-5} .


Figure 3: Fractional Format (16 bits)



The following table shows the conversion of a two's complement 16-bit integer +24576 to Q15 value +0.75.

Table 5-1. Table1: Conversion of a Two's Complement 16-Bit Integer to Q15

Binary		Dec		Q15
0	$0 \times (-2^{15})$	0	$0 \times (-2^0)$	0
				
1	1×2^{14}	16384	1×2^{-1}	0.5
1	1×2^{13}	8192	1×2^{-2}	0.25
0	0×2^{12}	0	0×2^{-3}	0
0	0×2^{11}	0	0×2^{-4}	0
0	0×2^{10}	0	0×2^{-5}	0
0	0×2^9	0	0×2^{-6}	0
0	0×2^8	0	0×2^{-7}	0
0	0×2^7	0	0×2^{-8}	0
0	0×2^6	0	0×2^{-9}	0
0	0×2^5	0	0×2^{-10}	0
0	0×2^4	0	0×2^{-11}	0
0	0×2^3	0	0×2^{-12}	0
0	0×2^2	0	0×2^{-13}	0
0	0×2^1	0	0×2^{-14}	0
0	0×2^0	0	0×2^{-15}	0
	SUM	+24576	SUM	+0.75

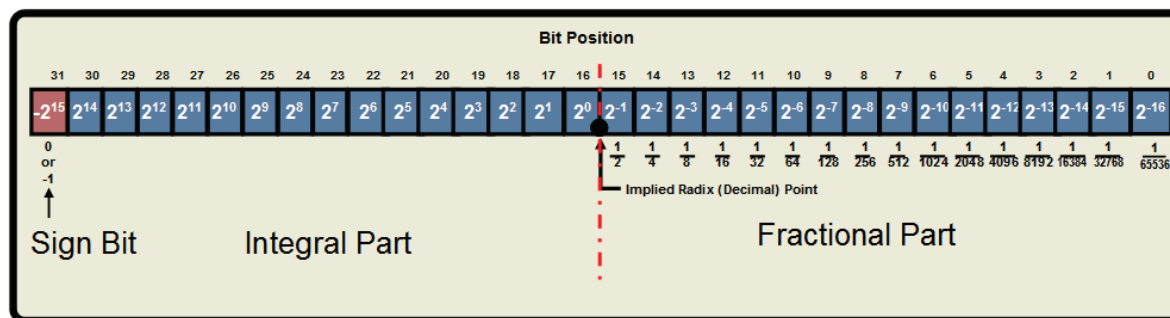
 = Radix Point

Q15.16 Format

In the Q15.16 format, the Most Significant bit is defined as a sign bit followed by 16 bits of the integral part. The radix point is implied to lie just after the integral part, followed by 16 bits of the fractional value. This format

is commonly referred to as Q15.16 format. The range of Q15.16 numbers is from -32768.0 (0x8000 0000) to +32767.9999847412109375 (0x7FFF FFFF) and has a precision of 2⁻¹⁶.

Figure 4: Fractional Format (32 bits)



The following table shows the conversion of a two's complement 32-bit integer, -715827882 to Q15.16 value -10922.6666564941.

Table 5-2. Table 2: Conversion of a Two's Complement 32-Bit Integer to Q15.16

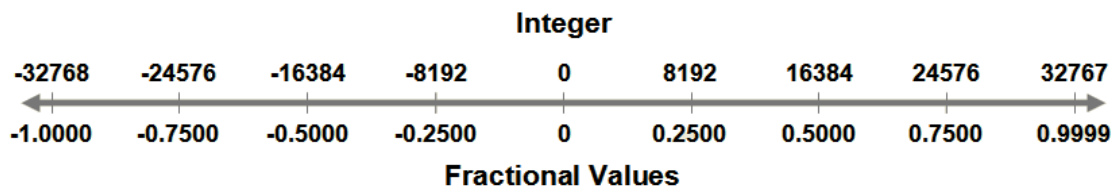
Binary		Dec		Q15.16
1	$1 \times (-2^{31})$	-2147483648	$1 \times (-2^{15})$	-32768
1	1×2^{30}	1073741824	1×2^{14}	16384
0	0×2^{29}	0	0×2^{13}	0
1	1×2^{28}	268435456	1×2^{12}	4096
0	0×2^{27}	0	0×2^{11}	0
1	1×2^{26}	67108864	1×2^{10}	1024
0	0×2^{25}	0	0×2^9	0
1	1×2^{24}	16777216	1×2^8	256
0	0×2^{23}	0	0×2^7	0
1	1×2^{22}	4194304	1×2^6	64
0	0×2^{21}	0	0×2^5	0
1	1×2^{20}	1048576	1×2^4	16
0	0×2^{19}	0	0×2^3	0
1	1×2^{18}	262144	1×2^2	4
0	0×2^{17}	0	0×2^1	0
1	1×2^{16}	65536	1×2^0	1
0	0×2^{15}	0	0×2^{-1}	0
1	1×2^{14}	16384	1×2^{-2}	0.25
0	0×2^{13}	0	0×2^{-3}	0
1	0×2^{12}	4096	1×2^{-4}	0.0625
0	0×2^{11}	0	0×2^{-5}	0
1	1×2^{10}	1024	1×2^{-6}	0.015625

.....continued				
Binary		Dec		Q15.16
0	0×2^9	0	0×2^{-7}	0
1	1×2^8	256	1×2^{-8}	0.00390625
0	0×2^7	0	0×2^{-9}	0
1	1×2^6	64	1×2^{-10}	0.000976563
0	0×2^5	0	0×2^{-11}	0
1	1×2^4	16	1×2^{-12}	0.000244141
0	0×2^3	0	0×2^{-13}	0
1	1×2^2	4	1×2^{-14}	6.10352E-05
1	1×2^1	2	1×2^{-15}	3.01576E-05
0	0×2^0	0	0×2^{-16}	0
	SUM	-715827882	SUM	-10922.6666564941
● = Radix Point				

Integer - Fractional Format Mapping

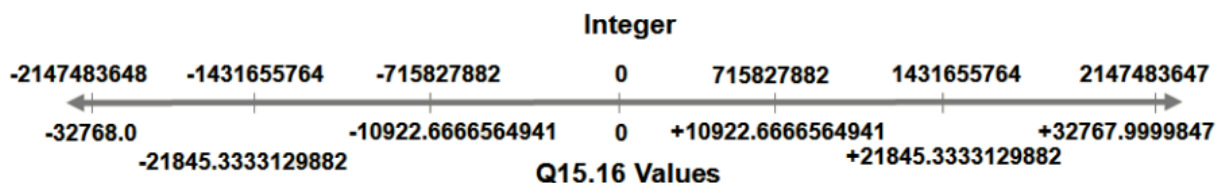
The same two's complement N-bit word may represent an integer format value or a fractional format value. For example., the 16-bit integer range [-32768, +32767] format maps to Q15 range of [-1.0, +0.999969482]. Figure 5 shows the mapping between these formats.

Figure 5: Mapping between 16-bit integer format and Q15 fractional format



A similar relationship exists between the 32-bit integer format and the Q15.16 format, where the integer range [-2147483648, +2147483647] is mapped to the Q15.16 range [-32768.0, +32767.9999847412109375].

Figure 6: Mapping between 32-bit integer format and Q15.16 format



libq Library and Fixed-Point Data Format

The functions in the `libq` library use the fixed-point data format. The parameters passed and the results generated by the functions are fractional in nature. There are two similar sets of math functions which perform the same math operations. One set supports Q15 operands and the other supports Q15.16 operands. Q15.16 operand functions, naturally, have better precision and range compared to Q15 operand functions.

5.2 Using the Fixed-Point Libraries

Building an application which utilizes the fixed-point libraries requires two types of files: header files and library files. Understanding fixed-point function naming conventions is needed for use.

Header Files

All standard C library entities are declared or defined in one or more standard headers. To make use of a library entity in a program, write an include directive that names the relevant standard header. The contents of a standard header are included by naming them in an include directive, as in:

```
#include <libq.h> /* include fixed-point library */
```

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

A standard header never includes another standard header.

Library Files

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

When linking an application, the library file (`libq-omf.a` or `libq-dsp-omf.a`) must be provided as an input to the linker (using the `--library` or `-l` linker option), such that the functions used by the application may be linked into the application. Also, linker options `-lq` and `-lq-dsp` must be used when linking the respective libraries.

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

Function Naming Conventions

Signed fixed-point types are defined as follows:

`Qn_m`

where:

- `n` is the number of data bits to the left of the radix point
- `m` is the number of data bits to the right of the radix point

Note: A sign bit is implied.

For convenience, short names are also defined:

Exact Name	# Bits Required	Short Name
<code>_Q0_15</code>	16	<code>_Q15</code>
<code>_Q15_16</code>	32	<code>_Q16</code>

In this document, the terms `Q15.16` and `Q16` are used interchangeably; however, both imply `Q15.16` format. Functions in the library are prefixed with the type of the return value. For example, `_Q15acos` returns a `Q15` value equal to the arc cosine of its argument.

Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments. In cases where the return value is not a fixed-point type, the argument type is appended to the function name. For example, function `_itoaQ15` accepts a type `Q15` argument.

In cases where two versions of a function are provided with the same return type but different argument types, the argument type is appended to the function name. For example:

Function Name	Return Type	Argument Type
<code>_Q16reciprocalQ15</code>	<code>_Q16</code>	<code>_Q15</code>
<code>_Q16reciprocalQ16</code>	<code>_Q16</code>	<code>_Q16</code>

5.3 <libq.h> Mathematical Functions

The header file libq.h consists of macro definitions and various functions that calculate fixed-point mathematical operations.

Q15 Functions: Many of these functions use fixed-point Q15 (1.15) format. For each function, the entire range may not be used.

Q16 Functions: Many of these functions use fixed-point Q16 (15.16) format. For each function, the entire range may not be used.

5.3.1 _Q15abs Function

The function finds the absolute value of a Q15 value.

Include

<libq.h>

Prototype

```
_Q15 _Q15abs(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the absolute value of x in Q15 format. The value ranges from 0 to 32767.

Note: `_Q15abs(-32768) = 32767`.

Also `abs` (smallest negative number) = largest positive number.

5.3.2 _Q15acos Function

This function finds the arc cosine of a Q15 value.

Include

<libq.h>

Prototype

```
_Q15 _Q15acos(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. Therefore the value of this argument ranges from 17705 to 32767.
----------	--

Return Value

This function returns the arc cosine of x in Q15 format. The value ranges from 256 to 32767.

5.3.3 _Q15acosByPI Function

This function finds the arc cosine of a Q15 value and then divides by PI (π).

Include

<libq.h>

Prototype

```
_Q15 _Q15acosByPI(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the arc cosine of *x*, divided by PI, in Q15 format. The value ranges from 82 to 32767.

5.3.4 _Q15add Function

This function finds the sum value of two Q15 values. The function takes care of saturation during overflow and underflow occurrences.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15add(_Q15 x, _Q15 y);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
y	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.

Return Value

This function returns the sum of *x* and *y* in Q15 format. The value ranges from -32768 to 32767.

5.3.5 _Q15asin Function

This function finds the arc sine of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15asin(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the arc sine of *x* in Q15 format. The value ranges from -32768 to 32767.

5.3.6 _Q15asinByPI Function

This function finds the arc sine of a Q15 value and then divides by PI (π).

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15asinByPI(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the arc sine of x , divided by PI, in Q15 format. The value ranges from -16384 to 16303.

5.3.7 _Q15atan Function

This function finds the arc tangent of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15atan(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the arc tangent of x in Q15 format. The value ranges from -25736 to 25735.

5.3.8 _Q15atanByPI Function

This function finds the arc tangent of a Q15 value and then divides by PI (π).

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15atanByPI(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the arc tangent of x , divided by PI, in Q15 format. The value ranges from -8192 to 8192.

5.3.9 _Q15atanYByX Function

This function finds the arc tangent of a Q15 value divided by a second Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15atanYByX(_Q15 x, _Q15 y);
```

Arguments

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
y	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.

Return Value

This function returns the arc tangent of y divided by x in Q15 format. The value ranges from -25736 to 25735.

5.3.10 _Q15atanYByXByPI Function

This function finds the arc tangent of a Q15 value divided by a second Q15 value and then divides the result by PI (π).

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15atanYByXByPI(_Q15 x, _Q15 y);
```

Arguments

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
y	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.

Return Value

This function returns the arc tangent of y divided by x, divided by PI, in Q15 format. The value ranges from -8192 to 8192.

5.3.11 _Q15atoi Function

This function takes a string which holds the ASCII representation of decimal digits and converts it into a single Q15 number.

Note: The decimal digit should not be beyond the range: -32768 to 32767.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15atoi(const char *s);
```

Argument

s	a buffer holding the ASCII values of each decimal digit.
----------	--

Return Value

This function returns the integer equivalent of s in Q15 format, which range is from -32768 to 32767.

5.3.12 _Q15cos Function

This function finds the cosine of a Q15 value.

Include

```
<libq.h>
```

Prototype

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

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the cosine of x in Q15 format. The value ranges from 17705 to 32767.

5.3.13 _Q15cosPI Function

This function finds the cosine of PI (π) times a Q15 value.

Include

```
<libq.h>
```

Prototype

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

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the cosine of PI times x in Q15 format. The value ranges from -32768 to 32767.

5.3.14 _Q15exp Function

This function finds the exponential value of a Q15 value.

Include

```
<libq.h>
```

Prototype

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

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 0.
----------	---

Return Value

This function returns the exponent value of x in Q15 format. The value ranges from 12055 to 32767.

5.3.15 _Q15ftoi Function

This function converts a single precision floating-point value into its corresponding Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15ftoi(float x);
```

Argument

x	a floating-point equivalent number. The corresponding floating-point range is -1 to 0.99996.
----------	--

Return Value

This function returns a fixed-point number in Q15 format. The value ranges from -32768 to 32767.

5.3.16 _itoaQ15 Function

This function converts each decimal digit of a Q15 value to its representation in ASCII. For example, 1 is converted to 0x31, which is the ASCII representation of 1.

Include

```
<libq.h>
```

Prototype

```
void _itoaQ15(_Q15 x, char *s);
```

Arguments

- x** a fixed-point number in Q15 format, which ranges from -2¹⁵ to 2¹⁵-1. The value of this argument ranges from -32768 to 32767.
- s** a buffer holding values in ASCII, at least 8 characters long.

Return Value

None.

5.3.17 _itofQ15 Function

This function converts a Q15 value into its corresponding floating-point value.

Include

```
<libq.h>
```

Prototype

```
float _itofQ15(_Q15 x);
```

Argument

- x** a fixed-point number in Q15 format, which ranges from -2¹⁵ to 2¹⁵-1. The value of this argument ranges from -32768 to 32767.

Return Value

This function returns a floating-point equivalent number. The corresponding floating-point range is -1 to 0.99996.

5.3.18 _Q15log Function

This function finds the natural log of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15log(_Q15 x);
```

Argument

- x** a fixed-point number in Q15 format, which ranges from -2¹⁵ to 2¹⁵-1. The value of this argument ranges from 12055 to 32767.

Return Value

This function returns the natural log of x in Q15 format. The value ranges from -32768 to -1.

5.3.19 _Q15log10 Function

This function finds the log (base 10) of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15log10(_Q15 x);
```

Argument

- x** a fixed-point number in Q15 format, which ranges from -2¹⁵ to 2¹⁵-1. The value of this argument ranges from 3277 to 32767.

Return Value

This function returns the log of x in Q15 format. The value ranges from -32768 to 0.

5.3.20 _Q15neg Function

This function negates a Q15 value with saturation. The value is saturated in the case where the input is -32768.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15neg(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns $-x$ in Q15 format. The value ranges from -32768 to 32767.

5.3.21 _Q15norm Function

This function finds the normalized value of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15norm(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the square root of x in Q15 format. The value ranges from 16384 to -32767 for a positive number and -32768 to -16384 for a negative number.

5.3.22 _Q15power Function

This function finds the power result given the base value and the power value in Q15 format.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15power(_Q15 x, _Q15 y);
```

Argument

x	a fixed-point number in Q15 format, which ranges from 1 to $2^{15}-1$. The value of this argument ranges from 1 to 32767.
y	a fixed-point number in Q15 format, which ranges from 1 to $2^{15}-1$. The value of this argument ranges from 1 to 32767.

Return Value

This function returns x to the power of y in Q15 format. The value ranges from 1 to 32767.

5.3.23 _Q15random Function

This function generates a random number in the range from -32768 to 32767. The random number generation is periodic with period 65536. The function uses the `_Q15randomSeed` variable as a random seed value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15random(void);
```

Argument

None.

Return Value

This function returns a random number in Q15 format. The value ranges from -32768 to 32767.

5.3.24 _Q15shl Function

This function shifts a Q15 value by `num` bits, to the left if `num` is positive or to the right if `num` is negative. The function takes care of saturating the result, in case of underflow or overflow.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15shl(_Q15 x, short num);
```

Arguments

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
num	an integer number, which ranges from -15 to 15.

Return Value

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

5.3.25 _Q15shlNoSat Function

This function shifts a Q15 value by `num` bits, to the left if `num` is positive or to the right if `num` is negative. The function sets the `_Q15shlSatFlag` variable in case of underflow or overflow but does not take care of saturation.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15shlNoSat(_Q15 x, short num);
```

Arguments

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
num	an integer number, which ranges from -15 to 15.

Return Value

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

5.3.26 _Q15shr Function

This function shifts a Q15 value by `num` bits, to the right if `num` is positive or to the left if `num` is negative. The function takes care of saturating the result, in case of underflow or overflow.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15shr(_Q15 x, short num);
```

Arguments

- | | |
|------------|---|
| x | a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767. |
| num | an integer number, which ranges from -15 to 15. |

Return Value

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

5.3.27 _Q15shrNoSat Function

This function shifts a Q15 value by `num` bits, to the right if `num` is positive or to the left if `num` is negative. The function sets the `_Q15shrSatFlag` variable in case of underflow or overflow but does not take care of saturation.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15shrNoSat(_Q15 x, short num);
```

Arguments

- | | |
|------------|---|
| x | a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767. |
| num | an integer number, which ranges from -15 to 15. |

Return Value

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

5.3.28 _Q15sin Function

This function finds the sine of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15sin(_Q15 x);
```

Argument

- | | |
|----------|---|
| x | a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767. |
|----------|---|

Return Value

This function returns the sine of `x` in Q15 format. The value ranges from -27573 to 27573.

5.3.29 _Q15sinPI Function

This function finds the sine of PI (π) times a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15sinPI(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the sine of PI times *x* in Q15 format. The value ranges from -32768 to 32767.

5.3.30 _Q15sinSeries Function

Generates the sine series with the given normalizing frequency, *f*, and the given number of samples, *num*, starting from *start*. Stores the result in buffer, *buf*.

Include

```
<libq.h>
```

Prototype

```
short _Q15sinSeries(_Q15 f, short start, short num, _Q15 *buf);
```

Arguments

f	a fixed-point number in Q15 format, which ranges from 0 to $(2^{31}-1)$. The valid range of values for this argument is from -16384 to 16384. The argument represents the Normalizing frequency.
start	a fixed-point number in Q16 format, which ranges from 0 to $(2^{31}-1)$. The valid range of values for this argument is from 1 to 32767. This argument represents the Starting Sample number in the Sine Series.
num	a fixed-point number in Q16 format, which ranges from 0 to $(2^{31}-1)$. The valid range of values for this argument is from 1 to 32767. This argument represents the Number of Sine Samples the function is called to generate. Note: <i>num</i> should not be more than 16383 for dsPIC and 32767 for PIC devices.
buf	a pointer to the buffer where the generated sine samples would get copied into.

Return Value

This function returns *num*, the number of generated sine samples.

5.3.31 _Q15sqrt Function

This function finds the square root of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15sqrt(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from 1 to 32767.
----------	--

Return Value

This function returns the square root of x in Q15 format. The value ranges from 1 to 32767.

5.3.32 **_Q15sub Function**

This function finds the difference of two Q15 values. The function takes care of saturation during overflow and underflow occurrences.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15sub(_Q15 x, _Q15 y);
```

Arguments

- | | |
|----------|---|
| x | a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767. |
| y | a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767. |

Return Value

This function returns x minus y in Q15 format. The value ranges from -32768 to 32767.

5.3.33 **_Q15tan Function**

This function finds the tangent of a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15tan(_Q15 x);
```

Argument

- | | |
|----------|---|
| x | a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767. |
|----------|---|

Return Value

This function returns the tangent of x in Q15 format. The value ranges from -32768 to 32767.

5.3.34 **_Q15tanPI Function**

This function finds the tangent of PI (π) times a Q15 value.

Include

```
<libq.h>
```

Prototype

```
_Q15 _Q15tanPI(_Q15 x);
```

Argument

- | | |
|----------|---|
| x | a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767. |
|----------|---|

Return Value

This function returns the tangent of PI times x in Q15 format. The value ranges from -32768 to 32767.

5.3.35 _Q16acos Function

This function finds the arc cosine of a Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16acos(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.
----------	---

Return Value

This function returns the arc cosine of *x* in Q16 format. The value ranges from 205887 to 363 respectively.

5.3.36 _Q16acosByPI Function

This function finds the arc cosine of a Q16 value and then divides by PI (π).

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16acosByPI(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.
----------	---

Return Value

This function returns the arc cosine of *x*, divided by PI, in Q16 format. The value ranges from 65536 to 115 respectively.

5.3.37 _Q16asin Function

This function finds the arc sine of a Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16asin(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.
----------	---

Return Value

This function returns the arc sine of *x* in Q16 format. The value ranges from -102944 to 102579 respectively.

5.3.38 _Q16asinByPI Function

This function finds the arc sine of a Q16 value and then divides by PI (π).

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16asinByPI(_Q16 x);
```


Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.
----------	---

Return Value

This function returns the arc sine of x , divided by π , in Q16 format. The value ranges from -32768 to 32653 respectively.

5.3.39 _Q16atan Function

This function finds the arc tangent of a Q16 value.

Include

<libq.h>

Prototype

```
_Q16 _Q16atan(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

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

5.3.40 _Q16atanByPI Function

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

Include

<libq.h>

Prototype

```
_Q16 _Q16atanByPI(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

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

5.3.41 _Q16atanYByX Function

This function finds the arc tangent of a Q16 value divided by a second Q16 value.

Include

<libq.h>

Prototype

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

Arguments

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

y a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

Return Value

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

5.3.42 _Q16atanYByXByPI Function

This function finds the arc tangent of a Q16 value divided by a second Q16 value and then divides the result by PI (π).

Include

<libq.h>

Prototype

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

Arguments

x a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

y a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

Return Value

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

5.3.43 _Q16cos Function

This function finds the cosine of a Q16 value.

Include

<libq.h>

Prototype

```
_Q16 _Q16cos(_Q16 x);
```

Argument

x a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

Return Value

This function returns the cosine of *x* in Q16 format. The value ranges from -65536 to 65535.

5.3.44 _Q16cosPI Function

This function finds the cosine of PI (π) times a Q16 value.

Include

<libq.h>

Prototype

```
_Q16 _Q16cosPI(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns the cosine of π times x in Q16 format. The value ranges from -65536 to 65535.

5.3.45 _Q16div Function

This function returns the quotient of its arguments.

Include

<libq.h>

Prototype

```
_Q16 _Q16div(_Q16 dividend, _Q16 divisor);
```

Arguments

dividend	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
divisor	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

Return Value

This function returns the quotient of its arguments. The value ranges from 0 to 2147483647.

5.3.46 _Q16divmod Function

This function returns the quotient and remainder of its arguments.

Include

<libq.h>

Prototype

```
_Q16 _Q16divmod(_Q16 dividend, _Q16 divisor, _Q16 *remainder);
```

Arguments

dividend	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
divisor	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
remainder	a pointer to an object large enough to hold the remainder. This must be provided by the caller.

Return Value

This function returns the quotient and remainder of its arguments. The values range from 0 to 2147483647.

5.3.47 _Q16exp Function

This function finds the exponential value of a Q16 value.

Include

<libq.h>

Prototype

```
_Q16 _Q16exp(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -772244 to 681391.
----------	---

Return Value

This function returns the exponent value of x in Q16 format. The value ranges from 0 to 2147483647.

5.3.48 _Q16ftoi Function

This function converts a single precision floating-point value into its corresponding Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16ftoi(float x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -32768 to 32768.
----------	---

Return Value

This function returns a fixed-point number in Q16 format. The value ranges from -2147483648 to 2147483647.

5.3.49 _itofQ16 Function

This function converts a Q16 value into its corresponding floating-point value.

Include

```
<libq.h>
```

Prototype

```
float _itofQ16(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns a floating-point equivalent number. The corresponding floating-point range is -32768 to 32768.

5.3.50 _Q16log Function

This function finds the natural log of a Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16log(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from 1 to 2147483647.
----------	---

Return Value

This function returns the natural log of x in Q16 format. The value ranges from -726817 to 681391.

5.3.51 _Q16log10 Function

This function finds the log (base 10) of a Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16log10(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from 1 to 2147483647.
----------	---

Return Value

This function returns the log of *x* in Q16 format. The value ranges from -315653 to 295925.

5.3.52 _Q16mac Function

This function multiplies the two 32-bit inputs, *x* and *y*, and accumulates the product with *prod*. The function takes care of saturating the result in case of underflow or overflow.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16mac(_Q16 x, _Q16 y, _Q16 prod);
```

Arguments

x	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
y	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
prod	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

Return Value

This function returns the multiplied and accumulated value *prod* in Q16 format. The value ranges from 0 to 2147483647.

5.3.53 _Q16macNoSat Function

This function multiplies the two 32 bit inputs, *x* and *y* and accumulates the product with *prod*. This function only sets the *_Q16macSatFlag* variable in case of an overflow or underflow and does not take care of saturation.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16macNoSat(_Q16 x, _Q16 y, _Q16 prod);
```

Arguments

x	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
y	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
prod	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

Return Value

This function returns the multiplied and accumulated value *prod* in Q16 format. The value ranges from 0 to 2147483647.

5.3.54 _Q16mpy Function

This function returns the product of its arguments.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16mpy(_Q16 a, _Q16 b);
```

Arguments

a	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
b	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

Return Value

This function returns the product of its arguments. The value ranges from 0 to 2147483647.

5.3.55 _Q16neg Function

This function negates a Q16 value with saturation. The value is saturated in the case where the input is -2147483648.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16neg(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns -x in Q16 format. The value ranges from -2147483648 to 2147483647.

5.3.56 _Q16norm Function

This function finds the normalized value of a Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16norm(_Q16 x);
```

Argument

x - a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns the square root of x in Q16 format. The value ranges from 16384 to -32767 for a positive number and -2147483648 to -1073741824 for a negative number.

5.3.57 _Q16power Function

This function finds the power result given the base value and the power value in Q16 format.

Include

```
<libq.h>
```

Prototype

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

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
y	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

Return Value

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

5.3.58 _Q16random Function

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

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16random(void);
```

Argument

None.

Return Value

This function returns a random number in Q16 format. The value ranges from -2147483648 to 2147483647.

Remarks

$\text{RndNum}(n) = (\text{RndNum}(n-1) * \text{RAN_MULT}) + \text{RAN_INC}$

SEED VALUE = 21845

RAN_MULT = 1664525

RAN_INC = 1013904223

5.3.59 _Q16reciprocalQ15 Function

This function returns the reciprocal of a Q15 value. Since the input range lies in the -1 to +1 region, the output is always greater than the -1 or +1 region. So, Q16 format is used to represent the output.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16reciprocalQ15(_Q15 x);
```

Argument

x	a fixed-point number in Q15 format, which ranges from -2^{15} to $2^{15}-1$. The value of this argument ranges from -32768 to 32767.
----------	---

Return Value

This function returns the reciprocal of x in Q16 format. The value ranges from -2147483648 to 2147418112.

5.3.60 _Q16reciprocalQ16 Function

This function returns the reciprocal value of the input.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16reciprocalQ16(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns the reciprocal of *x* in Q16 format. The value of this output ranges from -2147483648 to 2147483647.

5.3.61 _Q16shl Function

This function shifts a Q16 value by *y* bits, to the left if *y* is positive or to the right if *y* is negative. The function takes care of saturating the result, in case of underflow or overflow.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16shl(_Q16 x, short y);
```

Arguments

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
y	an integer number, which ranges from -32 to 32.

Return Value

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

5.3.62 _Q16shlNoSat Function

This function shifts a Q16 value by *y* bits, to the left if *y* is positive or to the right if *y* is negative. The function sets the `_Q16shlSatFlag` variable in case of underflow or overflow but does not take care of saturation.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16shlNoSat(_Q16 x, short y);
```

Arguments

x	a fixed-point number in Q16 format, which ranges from -216 to 216-1. The value of this argument ranges from -2147483648 to 2147483647.
y	an integer number, which ranges from -32 to 32.

Return Value

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

5.3.63 _Q16shr Function

This function shifts a Q16 value by *y* bits, to the right if *y* is positive or to the left if *y* is negative. The function takes care of saturating the result, in case of underflow or overflow.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16shr(_Q16 x, short y);
```

Arguments

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
y	an integer number, which ranges from -32 to 32.

Return Value

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

5.3.64 _Q16shrNoSat Function

This function shifts a Q16 value by *y* bits, to the right if *y* is positive or to the left if *y* is negative. The function sets the `_Q16shrSatFlag` variable in case of underflow or overflow but does not take care of saturation.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16shrNoSat(_Q16 x, short num);
```

Arguments

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
y	an integer number, which ranges from -32 to 32.

Return Value

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

5.3.65 _Q16sin Function

This function finds the sine of a Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16sin(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns the sine of *x* in Q16 format. The value ranges from -65566 to 65565.

5.3.66 _Q16sinPI Function

This function finds the sine of PI (π) times a Q16 value.

Include

```
<libq.h>
```

Prototype

```
_Q16 _Q16sinPI(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns the sine of π times x in Q16 format. The value ranges from -65536 to 65535.

5.3.67 _Q16sinSeries Function

Generates the sine series with the given normalizing frequency, f , and the given number of samples, num , starting from $start$. Stores the result in buffer, buf .

Include

<libq.h>

Prototype

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

Arguments

- f** a fixed-point number in Q16 format, which ranges from 0 to $(2^{31}-1)$. The valid range of values for this argument is from -32768 to 32768. The argument represents the Normalizing frequency.
- start** a fixed-point number in Q16 format, which ranges from 0 to $(2^{31}-1)$. The valid range of values for this argument is from 1 to 32767. This argument represents the Starting Sample number in the Sine Series.
- num** a fixed-point number in Q16 format, which ranges from 0 to $(2^{31}-1)$. The valid range of values for this argument is from 1 to 32767. This argument represents the Number of Sine Samples the function is called to generate. **Note:** num should not be more than 16383 for dsPIC and 32767 for PIC devices.
- buf** a pointer to the buffer where the generated sine samples would get copied into.

Return Value

This function returns num , the number of generated sine samples.

5.3.68 _Q16tan Function

This function finds the tangent of a Q16 value.

Include

<libq.h>

Prototype

```
_Q16 _Q16tan(_Q16 x);
```

Argument

- x** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

Return Value

This function returns the tangent of x in Q16 format. The value ranges from -2147483648 to 2147483647.

5.3.69 _Q16tanPI Function

This function finds the tangent of π times a Q16 value.

Include

<libq.h>

Prototype

```
_Q16 _Q16tanPI(_Q16 x);
```

Argument

x	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

Return Value

This function returns the tangent of PI times x in Q16 format. The value ranges from -2147483648 to 2147483647.

6. Revision History

The following is a list of changes by version to this document.

Note: Some revision letters are not used - the letters I and O - as they can be confused for numbers in some fonts.

6.1 Revision N (February 2021)

Section 2.17.3 “clock Function”: Stated that `clock()` function uses device timer2 and timer3.

Section 4.8 “Functions to Support Secondary PRAM Program Memory”: Terminology updated and added `_wipe_secondary` function.

6.2 Revision M (July 2020)

Section 2.14 “<stdio.h> Input and Output”: Update to `printf` function, additional `ll` modifier.

Section 4.8 “Functions to Support Slave PRAM Program Memory”: Section added from content in dsPIC33CH128MP508 (DS70005319) and dsPIC33CH512MP508 (DS70005371) family data sheets, Section 4.3 “Slave PRAM Program Memory”.

6.3 Revision L (December 2019)

- **Section 2.3 “<assert.h> Diagnostics”:** Added `__conditional_software_breakpoint` macro.
- **Section 2.4 “<ctype.h> Character Handling”:** Added `isblank` function.
- **Section 2.5 “<errono.h> Errors”:** Added `EILSEQ` macro.
- **Section 2.7 “<iso646.h> Alternate Spellings”:** Added a table of alternate spelling macros.
- **Section 2.15 “<stdlib.h> Utility Functions”:** Added “`strtol` family functions” section.
- **Chapter 5. “Fixed-Point Math Functions”:** Updated prototypes for `_Q16div` and `_Q16divmod`. Removed `_Q16reciprocal` as it is not in the library. This function is similar to `_Q16reciprocalQ16`.
- Other minor typographical, wording, and format changes.

6.4 Revision K (February 2018)

- **Section 4.8 “Functions for Specialized Copying and Initialization”** - added `_memcpy_eds`, `_strcpy_eds`, `_strncpy_eds`, `_memcpy_packed`, `_strcpy_packed`, `_strncpy_packed`. Also referenced in **Section 2.14 “<string.h> String Functions”** under `_memcpy`, `_strcpy`, `_strncpy`.
- **Section 5.3.1 “Q15 Functions”** - updated `_Q15abs` return value.
- **Section 5.3.2 “Q16 Functions”** - updated `_Q16norm` return value.

6.5 Revision J (December 2014)

- **Section 2.13 “<stdlib.h> Utility Functions”** - updated `malloc` function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for `_erase_flash` (PIC24FXXKA Only) and `_write_flash_word24()`.
- **Section 4.8 “Functions for Specialized Copying and Initialization”** - the following function descriptions have been updated:
 - `_memcpy_p2d24`
 - `_memcpy_p2d16`
 - `_strncpy_p2d24`
 - `_strncpy_p2d16`

6.6 Revision H (September 2013)

- **Section 1.1 “Introduction”** - added “Compiler Installation Locations”
- **Section 2.9 “<signal.h> Signal Handling”** - updated signal function.
- **Section 2.12 “<stdio.h> Input and Output”** - updated Customizing STDIO handles and descriptions for fdopen and vasprintf.
- **Section 4.3 “Standard C Library Helper Functions”** - updated _dump_heap_info function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for _write_flash_word32 and _write_flash_word48.
- **Section 5.1 “Overview of Fixed-Point Data Formats”** - inserted Table 5-1 and Table 5-2.
- **Section 5.3 “<libq.h> Mathematical Functions”** - the following function descriptions have been updated:
 - _Q16div
 - _Q16divmod
 - _Q16mpy
- Minor typographical changes.

6.7 Revision G (October 2010)

- **Section 2.12 “<stdio.h> Input and Output”** - updated scanf function
- **Section 2.14 “<string.h> String Functions”** - updated memchr function
- Inserted **Section 5.1 “Overview of Fixed-Point Data Formats”**
- Minor typographical changes.

6.8 Revision F (March 2009)

- **Section 4.5 “Functions/Constants to Support A Simulated UART”** - the following function descriptions have been updated:
 - _delay_ms
 - _delay_us
 - _C30UART
- **Section 4.6 “Functions for Erasing and Writing EEDATA Memory”** - updated _write_eedata_row function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for _erase_flash and _write_flash_word24.
- **Section 5.2 “Using the Fixed-Point Libraries”** - updated descriptions for Q16ftoi and itofQ16.
- Minor typographical changes.

6.9 Revision E (January 2008)

Section 2.3 “<ctype.h> Character Handling” - the following function descriptions have been updated:

- islower
- isxdigit
- tolower
- toupper

Section 2.5 “<float.h> Floating-Point Characteristics” - the following function descriptions have been updated:

- FLT_MAX_EXP
- FLT_MIN
- FLT_MIN_EXP

-
-
- LDBL_MIN
 - LDBL_MIN_EXP
 - UCHAR_MAX
 - UINT_MAX
 - ULLONG_MAX
 - ULONG_MAX
 - USHRT_MAX

Section 2.8 “<setjmp.h> Non-Local Jumps” - the following function descriptions have been updated:

- jmp_buf
- longjmp
- SIGILL
- SIGSEGV
- raise
- signal

Divided **Chapter 3. “Standard C Libraries - Math Functions”** into **Chapter 4. “Standard C Libraries - Support Functions”** and **Chapter 5. “Fixed-Point Math Functions”**

Minor typographical changes.

6.10 Revision D (December 2006)

- **Section 3.7 “<limits.h> implementation-defined limits”** - the following function descriptions have been updated:
 - LLONG_MIN
 - LONG_MAX
 - LONG_MIN
 - MB_LEN_MAX
 - SCHAR_MAX
 - SCHAR_MIN
 - SHRT_MAX
 - SHRT_MIN
 - UCHAR_MAX
- **Section 3.9 “<setjmp.h> non-local jumps”** - the following function descriptions have been updated:
 - jmp_buf
 - setjmp
 - longjmp
- **Section 3.10 “<signal.h> signal handling”** - the following function descriptions have been updated:
 - SIGFPE
 - SIGILL
 - SIGINT
 - raise
- **Section 3.13 “<stdio.h> input and output”** - the following function descriptions have been updated:
 - FILE
 - fpos_t
 - _IOFBF
 - _IOLBF
 - _IONBF
 - BUFSIZ

-
-
- FOPEN_MAX
 - L_tmpnam
 - SEEK_CUR
 - stderr
 - stdout
 - TMP_MAX
 - clearerr
 - feof
 - fgetpos
 - fopen
 - fprintf
 - fread
 - Minor typographical changes

6.11 Revision C (October 2005)

- Updated **Section 2.7 “<locale.h> Localization”** and the following function descriptions:
 - PIDInit
 - PIDCoeffCalc
 - PID
- Updated **Section 2.8 “<setjmp.h> Non-Local Jumps”** and the following function descriptions:
 - Fract2Float
 - Float2Fract
- Updated **Section 3.2 “Using the Standard C Libraries”**
- **Section 3.1 “<math.h> Mathematical Functions”** - the following function descriptions have been updated:
 - OpenXLCD
 - putsXLCD
 - SetDDRamAddr
 - WriteDataXLCD
 - WriteCmdXLCD
- Divided **Section 3.1 “<math.h> Mathematical Functions”** into **Section 3.4 “CAN Functions”**. The following function descriptions have been updated:
 - CAN1AbortAll
 - CAN1GetRXErrorCount
 - CAN1GetTXErrorCount
 - CAN1IsBusOff
 - CAN1IsRXReady
 - CAN1IsRXPassive
 - CAN1IsTXPassive
- **Section 5.1 “Overview of Fixed-Point Data Formats”** - the following function descriptions have been updated:
 - _builtin_addab
 - _builtin_add
 - _builtin_btg
 - _builtin_clr
 - _builtin_clr_prefetch
 - _builtin_dmaoffset
 - _builtin_ed
 - _builtin_edac

- `_builtin_fbcl`
- `_builtin_mac`
- `_builtin_movsac`
- `_builtin_mpy`
- `_builtin_mpy_n`
- `_builtin_msc`
- `_builtin_mulss`
- `_builtin_mulsu`
- `_builtin_mulus`
- `_builtin_muluu`
- `_builtin_nop`
- `_builtin_psvpage`
- Minor typographical changes

6.12 Revision B (September 2004)

- Added "NOTICE TO CUSTOMERS"
- Divided **Chapter 4. "Standard C Libraries - Support Functions"** into **Chapter 5. "Fixed-Point Math Functions"**
- Added **Section 1.1 "OMF-Specific Libraries/Start-up Modules"**
- Minor typographical changes

6.13 Revision A (May 2004)

- Initial release of this document.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 design partner 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

Product Change Notification Service

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

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.

Legal Notice

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 unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQL, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark 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.

© 2021, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-7681-8

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com	Australia - Sydney Tel: 61-2-9868-6733 China - Beijing Tel: 86-10-8569-7000 China - Chengdu Tel: 86-28-8665-5511 China - Chongqing Tel: 86-23-8980-9588 China - Dongguan Tel: 86-769-8702-9880 China - Guangzhou Tel: 86-20-8755-8029 China - Hangzhou Tel: 86-571-8792-8115 China - Hong Kong SAR Tel: 852-2943-5100 China - Nanjing Tel: 86-25-8473-2460 China - Qingdao Tel: 86-532-8502-7355 China - Shanghai Tel: 86-21-3326-8000 China - Shenyang Tel: 86-24-2334-2829 China - Shenzhen Tel: 86-755-8864-2200 China - Suzhou Tel: 86-186-6233-1526 China - Wuhan Tel: 86-27-5980-5300 China - Xian Tel: 86-29-8833-7252 China - Xiamen Tel: 86-592-2388138 China - Zhuhai Tel: 86-756-3210040	India - Bangalore Tel: 91-80-3090-4444 India - New Delhi Tel: 91-11-4160-8631 India - Pune Tel: 91-20-4121-0141 Japan - Osaka Tel: 81-6-6152-7160 Japan - Tokyo Tel: 81-3-6880-3770 Korea - Daegu Tel: 82-53-744-4301 Korea - Seoul Tel: 82-2-554-7200 Malaysia - Kuala Lumpur Tel: 60-3-7651-7906 Malaysia - Penang Tel: 60-4-227-8870 Philippines - Manila Tel: 63-2-634-9065 Singapore Tel: 65-6334-8870 Taiwan - Hsin Chu Tel: 886-3-577-8366 Taiwan - Kaohsiung Tel: 886-7-213-7830 Taiwan - Taipei Tel: 886-2-2508-8600 Thailand - Bangkok Tel: 66-2-694-1351 Vietnam - Ho Chi Minh Tel: 84-28-5448-2100	Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 Denmark - Copenhagen Tel: 45-4450-2828 Fax: 45-4485-2829 Finland - Espoo Tel: 358-9-4520-820 France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 Germany - Garching Tel: 49-8931-9700 Germany - Haan Tel: 49-2129-3766400 Germany - Heilbronn Tel: 49-7131-72400 Germany - Karlsruhe Tel: 49-721-625370 Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 Germany - Rosenheim Tel: 49-8031-354-560 Israel - Ra'anana Tel: 972-9-744-7705 Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781 Italy - Padova Tel: 39-049-7625286 Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340 Norway - Trondheim Tel: 47-72884388 Poland - Warsaw Tel: 48-22-3325737 Romania - Bucharest Tel: 40-21-407-87-50 Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 Sweden - Gothenberg Tel: 46-31-704-60-40 Sweden - Stockholm Tel: 46-8-5090-4654 UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820