POSIX / ANSI / ISO C

ANSI / ISO C

ANSI C / ISO C are standards designed to support an incredibly broad array of different systems and allow compiling legacy code from ages past. We can call this “standard C”. Because C is standardized, there are many implementations.

Half of the C standard refers to the language itself, which includes specific guarantees about what types are available, what syntax you can use, et cetera. These guarantees are sometimes too broad to work with comfortably. For example, in standard C, it is guaranteed that at least the following types exist:

- short int, which represents at least -32767..+32767
- int, which represents at least -32767..+32767
- long int, which represents at least -2147483647..+2147483647

Each type in the list must be wider than the last, but there is a lot of leeway for systems to choose different sizes. For example, on a DSP or an old supercomputer all of the types might be exactly the same size. There is also no guarantee that a byte has 8 bits (maybe it has more than 8 bits).

The other half of the C standard specifies the standard library, such as which functions are provided by each header. For example, the <stdlib.h> header must define the malloc() function.

A program written in standard C can be run almost anywhere, as long as you are careful not to rely on non-portable constructs. However, the C standard does not provide very much functionality… so these portable programs cannot do much more than open files or read input from the user on the console.

There are several versions of standard C. The most common ones are C89/C90, C99, and C11. It is not uncommon to find systems which only support C90 (for example, MSVC).

http://en.wikipedia.org/wiki/ANSI_C

POSIX

POSIX is a much larger and more comprehensive standard which includes standard C as a part of it. POSIX also specifies parts of the operating system. Because POSIX is standardized, there are many implementations.

On a POSIX system, there are some restrictions on the C implementation. For example, on POSIX:

- A byte is always 8 bits
- Integers are always two's-complement
- The / character is always used as a path separator
- Certain errors are signaled by setting errno

POSIX also specifies some additions the the standard library, such as

- Network sockets
- Creating new processes
- Multithreaded programming
- Memory-mapped IO

A program written to run on POSIX can be run on Linux, Unix, OS X, or other POSIX-compliant systems. These programs will usually require extra work before they can be run on Windows. The POSIX standard includes interfaces for things like networking, process creation, shells, terminals, and filesystems. It is not too difficult to write a sophisticated POSIX program like a web server or a command-line shell.

There are several versions of POSIX.

http://en.wikipedia.org/wiki/POSIX

GLibc

GLibc is the GNU C library. It implements the standard C library, POSIX extensions to the C library, and some extra functionality. GLibc is not standardized and there is only one implementation.

For example, GLibc provides asprintf(), which is like sprintf() but it allocates the buffer automatically.

Programs that use GLibc extensions are not generally portable, although certain extensions are also available on BSD systems.

http://en.wikipedia.org/wiki/GNU_C_Library

Win32

Win32 is an API specific to Windows. The API provides functions not available in standard C, such as functions for creating a graphical user interface. Win32 is not standardized and there are only two implementations (Windows and WINE). Win32 provides a large set of interfaces, such as:

- Network sockets
- Creating new processes
- Multithreaded programming
- Memory-mapped IO

These interfaces overlap with POSIX, but the function calls are mostly different. For example, on Windows you can create a mutex with CreateMutexEx(), and on POSIX you create a mutex with pthread*mutex*init(). The exception to this is network sockets, which are mostly the same between Windows and POSIX.

Programs written for Win32 will generally only run on Windows and possibly WINE.

http://en.wikipedia.org/wiki/Windows_API

MFC

MFC is a library provided by Microsoft which makes it easier to write Win32 applications. It is effectively obsolete and should not be used for new projects. MFC is not standardized and there is only one implementation.

http://en.wikipedia.org/wiki/Microsoft_Foundation_Class_Library

Source:
https://stackoverflow.com/questions/19697152/what-is-posix-any-other-interface-standards-which-can-replace-it