C/C++ tip: How to detect the compiler name and version using compiler predefined macros

October 7, 2012

Topics: C/C++

Compiler name and version macros are predefined by all C/C++ compilers to enable **`#if/#endif`** sets around compiler-specific code, such as inline assembly, compiler-specific intrinsics, or special language features. This can be necessary in high-performance code that aims at using the best performance tricks available for each compiler. **This article surveys common compilers and shows how to use predefined macros to detect the compiler name and version at compile time.**

Table of Contents

How to list predefined macros

See How to list compiler predefined macros for instructions on getting a list of macros for the compilers referenced here.

How to detect the compiler name

Compiler name macros indicate a specific compiler, such as Intel ICC or Microsoft Visual Studio. There are exceptions. See the notes after the table.

```
#if defined(__clang__)
           /* Clang/LLVM. ---------------------------------------------- */
#elif defined(__ICC) || defined(__INTEL_COMPILER)
           /* Intel ICC/ICPC. ------------------------------------------ */
#elif defined(__GNUC__) || defined(__GNUG__)
           /* GNU GCC/G++. --------------------------------------------- */
#elif defined(__HP_cc) || defined(__HP_aCC)
           /* Hewlett-Packard C/aC++. ---------------------------------- */
#elif defined(__IBMC__) || defined(__IBMCPP__)
           /* IBM XL C/C++. -------------------------------------------- */
#elif defined(_MSC_VER)
           /* Microsoft Visual Studio. --------------------------------- */
#elif defined(__PGI)
           /* Portland Group PGCC/PGCPP. ------------------------------- */
#elif defined(__SUNPRO_C) || defined(__SUNPRO_CC)
           /* Oracle Solaris Studio. ----------------------------------- */


#endif
```

Compiler name macros

Macro	Clang/LLVM		GNU GCC/G++		HP C/aC++		IBM XL C/C++		Intel ICC/ICPC		Microsoft Visual Studio		Oracle Solaris Studio		Portland PGCC/PGCPP	
	C	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*
__clang__	yes	yes														
__GNUC__	yes	yes	yes	yes					yes	yes						
__GNUG__		yes		yes						yes						
__HP_aCC						yes										
__HP_cc					yes											
__IBMC__							yes									
__IBMCPP__								yes								
__ICC									yes	yes						
__INTEL_COMPILER									yes	yes						
_MSC_VER											yes	yes				
__PGI															yes	yes
__SUNPRO_C													yes			
__SUNPRO_CC														yes		

Notes:

- **__GNUC__** and **__GNUG__** were intended to indicate the GNU compilers. However, they're also defined by Clang/LLVM and Intel compilers to indicate compatibility. To detect GCC and G++ today, use an **#if/#endif** that checks that **__GNUC__** or **__GNUG__** are defined, but **__clang__** and **__INTEL_COMPILER** are not:

```
#if (defined(__GNUC__) || defined(__GNUG__)) && !(defined(__clang__) || defined(__INTEL_COMPILER))
  /* GNU GCC/G++. ------------------------------------------- */


#endif
```

- Microsoft Visual Studio and Portland Group PGCC/PGCPP don't define a different compiler name macro for C++ vs. C. To detect C++ compilation using these compilers, check that **__cplusplus** is defined along with **_MSC_VER** or **__PGI**.
- Clang/LLVM and GNU GCC/C++ on Apple's OSX also define **__APPLE_CC__** to support legacy code.

How to detect the compiler version

Compiler version macros indicate major and minor version numbers. In some cases these are the same macros used to indicate the compiler name (see the previous section).

Compiler version macros

Macro	Clang/LLVM		GNU GCC/G++		HP C/aC++		IBM XL C/C++		Intel ICC/ICPC		Microsoft Visual Studio		Oracle Solaris Studio		Portland PGCC/PGCPP	
	C	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*	*C*	*C++*
`__clang_major__`	yes	yes														
`__clang_minor__`	yes	yes														
`__clang_patchlevel__`	yes	yes														
`__clang_version__`	yes	yes														
`__GNUC__`	yes	yes	yes	yes					yes	yes						
`__GNUC_MINOR__`	yes	yes	yes	yes					yes	yes						
`__GNUC_PATCHLEVEL__`	yes	yes	yes	yes					yes	yes						
`__GNUG__`		yes		yes						yes						
`__HP_aCC`						yes										
`__HP_cc`					yes											
`__IBMC__`							yes									
`__IBMCPP__`								yes								
`__ICC`									yes	yes						
`__INTEL_COMPILER`									yes	yes						
`__INTEL_COMPILER_BUILD_DATE`									yes	yes						
`_MSC_BUILD`											yes	yes				
`_MSC_FULL_VER`											yes	yes				
`_MSC_VER`											yes	yes				
`__PGIC__`															yes	yes
`__PGIC_MINOR__`															yes	yes
`__PGIC_PATCHLEVEL__`															yes	yes
`__SUNPRO_C`													yes			
`__SUNPRO_CC`														yes		
`__VERSION__`	yes		yes						yes	yes						yes
`__xlc__`							yes									
`__xlC__`							yes	yes								
`__xlC_ver__`							yes	yes								

Notes:

- Clang/LLVM:
  - **`__clang_major__`**, **`__clang_minor__`**, and **`__clang_patchlevel__`** contain the major version, minor version, and patch level numbers (e.g. "3", "0", and "1" for version 3.0.1).
  - **`__clang_version__`** contains a version string (e.g. "3.0 (tags/RELEASE_30/final)" for version 3.0.0). The format of the string is undefined and varies with different distributions.
  - **`__GNUC__`**, **`__GNUG__`**, **`GNUC_MINOR__`**, and **`__GNUC_PATCHLEVEL__`** are per GNU GCC/G++ below and indicate Clang compatibility.
  - **`__VERSION__`** contains a long version string (e.g. "4.2.1 Compatible Clang 3.0 (tags/RELEASE_30/final)" for version 3.0.0 with GCC 4.2.1 compatibility). The format of the string is undefined and varies with different distributions.

- GNU GCC/G++:
  - **__GNUC__** and **__GNUG__** contain the major version number, and **__GNUC_MINOR__** and **__GNUC_PATCHLEVEL__** the minor version and patch level numbers (e.g. "4", "6", and "1" for version 4.6.1). The **__GNUC_PATCHLEVEL__** macro was introduced in version 3.0.
  - **__VERSION__** contains a version string (e.g. "4.6.1" for version 4.6.1). The format of the string is undefined and varies with different distributions.
- HP C/aCC:
  - **__HP_aCC** and **__HP_cc** contain the major version, minor version, and extension numbers as an integer (e.g. "061700" for version A.06.17.00). However, the macro's value was just "1" for versions before A.01.21.00.
- IBM XL C/C++
  - **__IMBC__** and **__IBMCPP__** contain the version, release, and modification numbers as a hex integer (e.g. "0x0500" for version 5.0.0).
  - **__xlc__** contains the version, release, modification, and fix level numbers as a string (e.g. "5.0.0.3" for version 5.0.0.3).
  - **__xlC__** contains the version and release numbers as a hex integer (e.g. "0x0500" for version 5.0).
  - **__xlC_ver__** contains the modification and fix level numbers as a hex integer with four leading zeroes (e.g. "0x00000003" for 0.3).
- Intel ICC/ICPC:
  - **__ICC** and **__INTEL_COMPILER** contain the major and minor version number as an integer (e.g. "1210" for version 12.10). **__INTEL_COMPILER_BUILD_DATE** contains the build date in yyyymmdd format (e.g. "20111011" for October 11, 2011).
  - **__ICC** is deprecated in favor of **__INTEL_COMPILER**.
  - **__GNUC__**, **__GNUG__**, **__GNUC_MINOR__**, and **__GNUC_PATCHLEVEL__** are per GNU GCC/G++ above and indicate ICC compatibility.
  - **__VERSION__** contains a version string (e.g. "Intel(R) C++ gcc 4.1 mode"). The format of the string is undefined.
- Microsoft Visual Studio:
  - **_MSC_VER** contains the major and minor version numbers as an integer (e.g. "1500" is version 15.00).
  - **_MSC_FULL_VER** contains the major version, minor version, and build numbers as an integer (e.g. "150020706" is version 15.00.20706). The macro was introduced in Visual Studio 2008.
  - **_MSC_BUILD** contains the revision number after the major version, minor version, and build numbers (e.g. "1" is revision 1, such as for 15.00.20706.01). The macro is was introduced in Visual Studio 2008.
- Oracle Solaris Studio:
  - **__SUNPRO_C** and **__SUNPRO_CC** contain the major version, minor version, and patch level numbers as a hex number (e.g. "0x5120" is version 5.12.0).
- Portland Group PGCC/PGCPP:
  - **__PGIC__**, **__PGIC_MINOR**, and **__PGIC_PATCHLEVEL__** contain the major version, minor version, and patch level numbers (e.g. "11", "9", and "0" for version 11.9.0).

Further reading

Related articles at NadeauSoftware.com

C/C++ tip: How to list compiler predefined macros explains how to get a compiler's macros by using command-line options and other methods.

C/C++ tip: How to detect the operating system type using compiler predefined macros provides **#if/#endif** sets for detecting desktop operating systems using compiler macros.

C/C++ tip: How to detect the processor type using compiler predefined macros provides **#if/#endif** sets for detecting desktop and server processors using compiler macros.

Web articles

Compilers at Sourceforge.net provides a long list of compilers and their compiler name and version macros. However the list is very cluttered with obscure and obsolete compilers (Compaq? Convex? SCO? Palm?).

Pre-defined C/C++ Compiler Macros at beefchunk.com has a list of compilers and predefined macros. Like the Sourceforge list, the information is a bit cluttered with obsolete compilers (Codewarrior? Tiny C? SCO?) and it's a little out of date.

Compiler Resources at Apache.org has a good list of compilers, links to vendor documentation, and some tables of compiler predefined macros.