How to determine the version of the C++ standard used by the compiler?

Asked 10 years, 6 months ago Active 4 months ago Viewed 104k times

116

46

How do you determine what version of the C++ standard is implemented by your compiler? As far as I know, below are the standards I've known:

- C++03
- C++98

c++ standards

edited Feb 24 '10 at 9:21

asked Feb 24 '10 at 8:46
jasonline
6,908 18 52 76

3 You tagged this *c++*, but two of the three standards you listed are not C++ standards. Which language(s) are you interested in? – Rob Kennedy Feb 24 '10 at 8:56

1 And the question has been asked just a couple minutes ago. (stackoverflow.com/questions/7132440/...) – Mat Aug 20 '11 at 14:57

1 @Mat: Posed and closed because the question was rubbish and had some other arbitrary nonsense piggybacking on it. I have re-posted it in a decent form. I'd be happy to close this one if it looks like the original will be fixed and revived, but I'm not holding my breath. – Lightness Races in Orbit Aug 20 '11 at 14:58

1 @Mat: Well, the best answer is not a static list of compilers, but *a means of determining for yourself* what is in use. So there you go. – Lightness Races in Orbit Aug 20 '11 at 15:00

1 @Als: It will be soon. I promise. Besides, the `c++-faq` tag doesn't have any *actual* pre-requisite "number of times asked" that you have to pass; it's more about the format and generality of the thing. – Lightness Races in Orbit Aug 20 '11 at 15:03

8 Answers

Active Oldest Votes

13

By my knowledge there is no overall way to do this. If you look at the headers of cross platform/multiple compiler supporting libraries you'll always find **a lot** of defines that use compiler specific constructs to determine such things:

```
/*Define Microsoft Visual C++ .NET (32-bit) compiler */
#if (defined(_M_IX86) && defined(_MSC_VER) && (_MSC_VER >= 1300)
     ...
#endif

/*Define Borland 5.0 C++ (16-bit) compiler */
#if defined(__BORLANDC__) && !defined(__WIN32__)
     ...
#endif
```

You probably will have to do such defines yourself for all compilers you use.

answered Feb 24 '10 at 9:23
RED SOFT ADAIR
11k 9 46 81

1 Not my expected answer though but I guess there's just no universal way of finding it out. – jasonline Feb 28 '10 at 11:26

250

From the Bjarne Stroustrup C++0x FAQ:

> **`__cplusplus`**
>
> In C++0x the macro `__cplusplus` will be set to a value that differs from (is greater than) the current `199711L`.

Although this isn't as helpful as one would like. `gcc` (apparently for nearly 10 years) had this value set to `1`, ruling out one major compiler, until it was fixed when gcc 4.7.0 came out.

These are the C++ standards and what value you should be able to expect in `__cplusplus`:

- C++ pre-C++98: `__cplusplus` is `1`.
- C++98: `__cplusplus` is `199711L`.
- C++98 + TR1: This reads as C++98 and there is no way to check that I know of.
- C++11: `__cplusplus` is `201103L`.
- C++14: `__cplusplus` is `201402L`.
- C++17: `__cplusplus` is `201703L`.

If the compiler might be an older `gcc`, we need to resort to compiler specific hackery (look at a version macro, compare it to a table with implemented features) or use Boost.Config (which provides relevant macros). The advantage of this is that we actually can pick specific features of the new standard, and write a workaround if the feature is missing. This is often preferred over a wholesale solution, as some compilers will claim to implement C++11, but only offer a subset of the features.

The Stdcxx Wiki hosts a comprehensive matrix for compiler support of C++0x features (if you dare to check for the features yourself).

Unfortunately, more finely-grained checking for features (e.g. individual library functions like `std::copy_if`) can only be done in the build system of your application (run code with the feature, check if it compiled and produced correct results - `autoconf` is the tool of choice if taking this route).

Doesn't look like compiler vendors are updating this - maybe they're waiting until they fully conform to the standard?(stackoverflow.com/q/14131454/11698) – Richard Corden Jan 22 '13 at 16:44

2 @prnr: That may be true, but it's up to the user who asked the question to decide which answer to accept. At the time that the answer which is currently marked as accepted was posted, it was correct, so the original poster accepted it. That user could decide to change the accepted answer, but they may no longer be active on the site. See: meta.stackexchange.com/questions/120568/... – Dan Korn Aug 18 '16 at 19:26

3 vs2017 gives __cplusplus's value 199711 – Al Mamun Dec 13 '17 at 11:18

5 @AlMamun Microsoft partly fixed `__cplusplus` only in VS 15.7. See their Visual C++ Team Blog – Ivan_Bereziuk Jul 4 '18 at 9:51

1 The link to the FAQ is broken. – brainplot Apr 29 '19 at 2:13

40

Please, run the following code to check the version.

```
#include<iostream>

int main() {
    if (__cplusplus == 201703L) std::cout << "C++17\n";
    else if (__cplusplus == 201402L) std::cout << "C++14\n";
    else if (__cplusplus == 201103L) std::cout << "C++11\n";
    else if (__cplusplus == 199711L) std::cout << "C++98\n";
    else std::cout << "pre-standard C++\n";
}
```

edited Dec 6 '18 at 20:29
Subangkar KrS
93 2 3

answered Jul 26 '18 at 10:16
Deepanshu
417 4 2

9 Its funny, because on visual studios the value of __cplusplus is 199711L and the code you posted returned c++98 however, I've used features from c++14 including variable templates and decltype(auto). Is it possible the wrong version of the macro was implemented? – Colin Hicks May 11 '19 at 19:37

2 See: devblogs.microsoft.com/cppblog/... (TLDR: specify the flag `/Zc:__cplusplus`) – Daan Timmer Aug 12 '19 at 22:08

@DaanTimmer I'm confused by that article, it seems to assume knowledge of how to use the `/Zc:__cplusplus` flag. I can't simply `std::cout << /Zc:__cplusplus;` because colons and slashes can't be part of variable names of course. Are you able to explain how to do this? Thanks. – A__ Oct 31 '19 at 15:12

Found it: docs.microsoft.com/en-us/cpp/build/reference/... – A__ Oct 31 '19 at 18:41

7

Depending on what you want to achieve, Boost.Config might help you. It does not provide detection of the standard-version, but it provides macros that let you check for support of specific language/compiler-features.

answered Feb 24 '10 at 9:35
Björn Pollex
67.6k 24 169 257

3 Checking for features is probably a better idea than checking standard versions, anyway. Few compilers support *everything* from a standard, but if they all support the limited number of features you need, then it doesn't really matter whether the rest of the features from a given standard are implemented and working correctly. – Rob Kennedy Feb 24 '10 at 10:36

4

> __cplusplus
>
> In C++0x the macro __cplusplus will be set to a value that differs from (is greater than) the current 199711L.

C++0x FAQ by BS

answered Feb 24 '10 at 9:48
Vinzenz
2,644 14 23

2

Use `__cplusplus` as suggested. Only one note for Microsoft compiler, use `Zc:__cplusplus` compiler switch to enable `__cplusplus`

Source https://devblogs.microsoft.com/cppblog/msvc-now-correctly-reports-__cplusplus/

answered Jan 15 at 12:26
dimon4eg
840 10 16

0

After a quick google:

`__STDC__` and `__STDC_VERSION__` , see here

answered Feb 24 '10 at 8:50
Tor Valamo
29.4k 10 67 79

Whether `__STDC__` is defined, and what its value is, are implementation-defined in C++. – Rob Kennedy Feb 24 '10 at 8:57

@Rob: Yes, it is. @Tor: I tried in VC++ 2005 but it says **STDC** is an undeclared identifier. It is listed as one of those pre-defined macros though. However, **STDC_VERSION** does not exist. – jasonline Feb 24 '10 at 9:04

This tells you the version of the C programming language supported by the compiler. It tells you nothing about the version of the C++ language that is supported. – Dan Moulding May 8 '12 at 14:29

0

Normally you should use `__cplusplus` define to detect c++17, but by default microsoft compiler does not define that macro properly, see https://devblogs.microsoft.com/cppblog/msvc-now-correctly-reports-__cplusplus/ - you need to either modify project settings to include `/Zc:__cplusplus` switch, or you could use syntax like this:

```
#if ((defined(_MSVC_LANG) && _MSVC_LANG >= 201703L) || __cplusplus >=
201703L)
     //C++17 specific stuff here
#endif
```

answered May 1 at 22:56
TarmoPikaro
2,954 1 27 40