0.12 — Configuring your compiler: Choosing a language standard

ALEX DECEMBER 8, 2022

With many different versions of C++ available (C++98, C++03, C++11, C++14, C++17, C++20, etc…) how does your compiler know which one to use? Generally, a compiler will pick a standard to default to (typically not the most recent language standard).

If you wish to use a different language standard (and you probably will), you'll have to configure your IDE/compiler to do so.

Code names for in-progress language standards

Finalized language standards are named after the years in which they are finalized (e.g. C++17 was finalized in 2017).

However, when a new language standard is being agreed upon, it's not clear in what year the finalization will take place. Consequently, in-progress language standards are given code names, which are then replaced by the actual names upon finalization of the standard. For example, C++11 was called c++1x while it was being worked on. You may still see the code names used in places (especially for upcoming version of the language standard, which won't have a final name yet).

Here's a mapping of code names to the final names:

- c++1x = C++11
- c++1y = C++14
- c++1z = C++17
- c++2a = C++20
- c++2b = C++23

For example, if you see `c++1z`, this is synonymous with the C++17 language standard.

Which language standard should you choose?

In professional environments, it's common to choose a language standard that is one or two versions back from the latest standard (e.g. if C++20 is the latest version, that means C++14 or C++17). This is typically done to ensure the compiler makers have had a chance to resolve defects, and so that best practices for new features are well understood. Where relevant, this also helps ensure better cross-platform compatibility, as compilers on some platforms may not provide full support for newer language standards immediately.

For personal projects and while learning, there is little downside to choosing the latest finalized standard (as of the time of writing, currently C++20).

Author's note

This website currently targets the C++17 standard, meaning our lessons and examples assume your compiler is C++17 capable. Some C++20 content is available for those with C++20 compatible compilers.

To take full advantage of all the lesson content, we recommend using the C++20 language standard if your compiler supports it. Using the C++17 language standard will also provide a good experience.

If your compiler doesn't support C++17, we recommend upgrading to one that does. If this is not possible for some reason, you will need to skip some content, and alter some examples so that they will compile. This should not impact your overall experience too heavily (especially in the early lessons).

C++14 is the minimum language standard for a decent experience on this site.

There's an example at the end of this lesson which you can use to test if you set up your compiler to use C++17 correctly.

Setting a language standard in Visual Studio

As of the time of writing, Visual Studio 2022 defaults to C++14 capabilities, which does not allow for the use of newer features introduced in C++17 and C++20.

To use these newer features, you'll need to enable a newer language standard. Unfortunately, there is currently no way to do this globally -- you must do so on a project-by-project basis.

Warning

With Visual Studio, you will need to reselect your language standard every time you create a new project.

To select a language standard, open your project, then go to Project menu > (Your application's Name) Properties, then open Configuration Properties > C/C++ > Language.

First, make sure the Configuration is set to "All Configurations".

From there, you can set the C++ Language Standard to the version of C++ you wish to use.

Tip

We recommend choosing the latest standard "ISO C++ Latest (/std:c++latest)", which will ensure you can use as many features as your compiler supports.

Make sure you're selecting the language standard from the dropdown menu (don't type it out).

Related content

For more information on Visual Studio language standard settings, Microsoft has a Visual Studio language standard reference document (https://docs.microsoft.com/en-us/cpp/build/reference/std-specify-language-standard-version?view=msvc-160).

Setting a language standard in Code::Blocks

Code::Blocks may default to a pre-C++11 language standard. You'll definitely want to check and ensure a more modern language standard is enabled.

The good news is that Code::Blocks allows setting your language standard globally, so you can set it once (rather than per-project). To do so, go to Settings menu > Compiler...:

Then find the checkboxes labeled Have g++ follow the C++XX ISO C++ language standard [-std=c++XX], where XX is some number (e.g. 20, 17, etc...) representing a language standard:

Tip

If C++20 or C++17 appears in this list, select the one that represents the latest ISO standard (e.g. select Have g++ follow the C++20 ISO language standard). If you see GNU standards in this list as well, ignore them.

If you do not see C++20 or C++17 in this list, upgrade to the latest version of Code::Blocks.

If upgrading to the latest version is not possible for some reason, your version of Code::Blocks may have support for upcoming (or just released) versions of C++. If so, these will be labeled Have g++ follow the coming C++XX (aka C++YY) ISO C++ language standard [-std=c++XX] (see the blue box above). Select the latest version from this list.

Setting a language standard in g++

For GCC/G++, you can pass compiler flags `-std=c++11`, `-std=c++14`, `-std=c++17`, or `-std=c++20` to enable C++11/14/17/20 support respectively. If you have GCC 8 or 9, you'll need to use `-std=c++2a` for C++20 support instead.

Setting a language standard for VS Code

For VS Code, you can use compiler flags `-std=c++11`, `-std=c++14`, `-std=c++17`, or `-std=c++20` to enable C++11/14/17/20 support respectively. If you have GCC 8 or 9, you'll need to use `-std=c++2a` for C++20 support instead.

Place the appropriate language standard flag in the `tasks.json` configuration file, on its own line before "${file}".

()Testing your compiler for C++17 compatibility (#example)

After enabling the C++17 language standard (or higher), you should be able to compile the following code without any warnings or errors.

```cpp
#include <array>
#include <iostream>
#include <string_view>
#include <tuple>
#include <type_traits>

namespace a::b::c
{
    inline constexpr std::string_view str{ "hello" };
}

template <class... T>
std::tuple<std::size_t, std::common_type_t<T...>> sum(T... args)
{
    return { sizeof...(T), (args + ...) };
}

int main()
{
    auto [iNumbers, iSum]{ sum(1, 2, 3) };
    std::cout << a::b::c::str << ' ' << iNumbers << ' ' << iSum << '\n';

    std::array arr{ 1, 2, 3 };

    std::cout << std::size(arr) << '\n';

    return 0;
}
```

If you can't compile this code, you either haven't enabled C++17, or your compiler doesn't fully support C++17. In the latter case, please install the latest version of your IDE/compiler, as described in lesson 0.6 -- Installing an Integrated Development Environment (IDE) (https://www.learncpp.com/cpp-tutorial/installing-an-integrated-development-environment-ide/).

Exporting your configuration

Having to reselect all of your settings options every time you create a new project is burdensome. Fortunately, most IDEs provide a way to export your settings. This is typically done by creating a new project template with the settings you want, and then selecting that project template when you create a new project.

For Visual Studio users

In Visual Studio, this option is available via Project -> Export Template. Select "Project template", add a name and optional description (e.g. C++20 console application), and then click "Finish".

Next time you create a new project, you'll see this template show up in your list of project templates.

Once you create a new project with this template, it may not open any files. You can open up your .cpp file in the Solution Explorer window by going to Solution -> <Project Name> -> Source Files -> <template name>.cpp.

For Code::Blocks users

In Code::Blocks, choose File -> Save project as template. Give your template a title, and save.

When you create a new project, you will find this template under the "User templates" option.

Where can I view the C++ standards document?

Each C++ language standard is described by a **standards document**, which is a formal technical document that is the authoritative source for the rules and requirements of a given language standard. The standards document is not designed for learning -- rather, it's designed for compiler writers to be able to implement new language standards accurately. You will occasionally see people quoting the standards document when explaining how something works.

The approved C++ standards document for a given language standard is not available for free. There is a link to purchase the latest standard here (https://isocpp.org/std/the-standard).

When a new language standard is being developed, draft standards documents are published for review. These drafts are available online for free. The last draft standard before the approved standard is generally close enough to the official standard to use for most purposes. You can find the draft standards here (https://www.open-std.org/jtc1/sc22/wg21/docs/standards).

Next lesson
1.1 Statements and the structure of a program

Back to table of contents

Previous lesson
0.11 Configuring your compiler: Warning and error levels