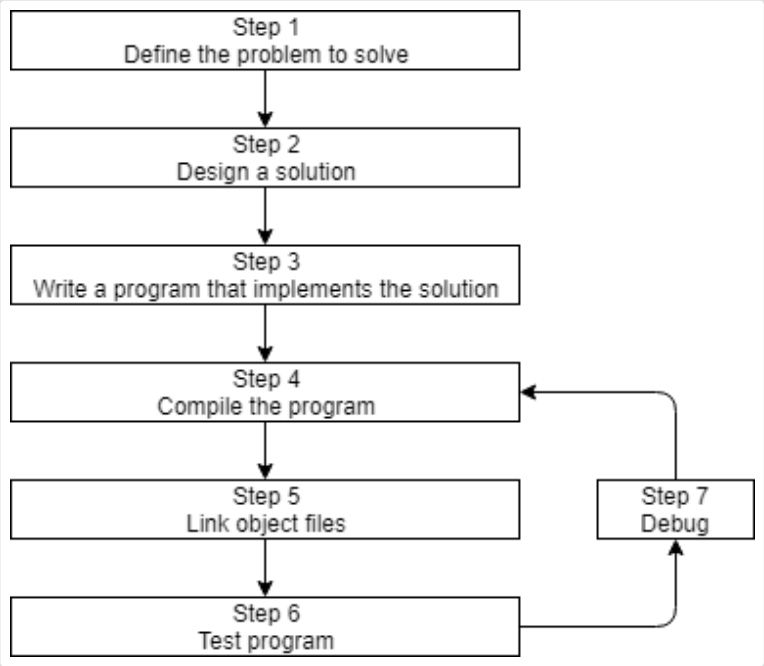
0.5 — Introduction to the compiler, linker, and libraries

BY ALEX ON SEPTEMBER 18TH, 2018 | LAST MODIFIED BY ALEX ON NOVEMBER 13TH, 2019

Continuing our discussion of this diagram from the previous lesson (**0.4 -- Introduction to C++ development**):



Let's discuss steps 4-7.

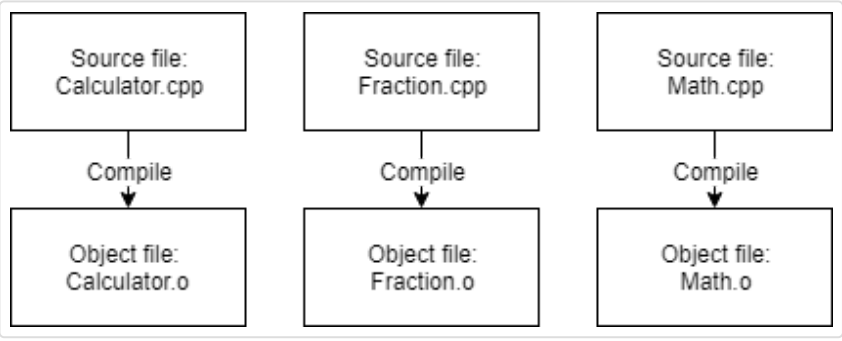
Step 4: Compiling your source code

In order to compile a C++ program, we use a C++ compiler. The C++ compiler sequentially goes through each source code (.cpp) file in your program and does two important tasks:

First, it checks your code to make sure it follows the rules of the C++ language. If it does not, the compiler will give you an error (and the corresponding line number) to help pinpoint what needs fixing. The compilation process will also be aborted until the error is fixed.

Second, it translates your C++ source code into a machine language file called an **object file**. Object files are typically named *name.o* or *name.obj*, where *name* is the same name as the .cpp file it was produced from.

If your program had 3 .cpp files, the compiler would generate 3 object files:

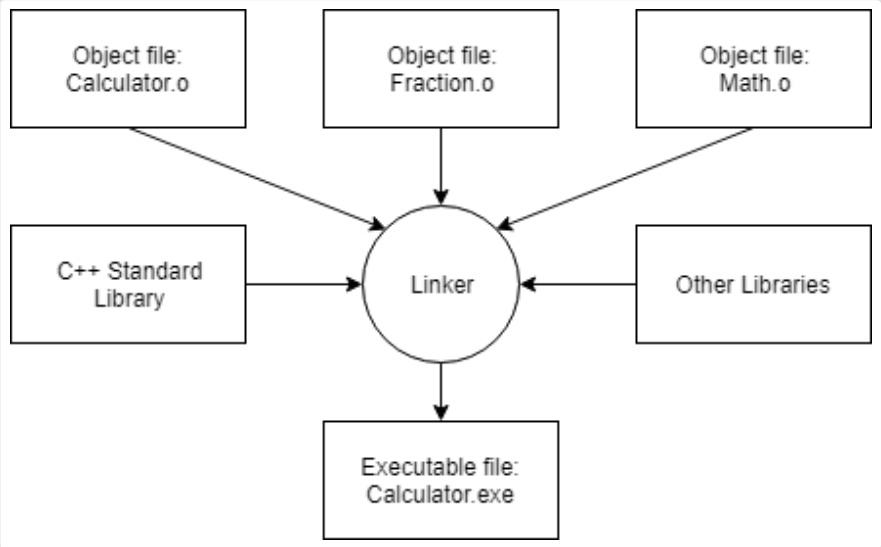


C++ compilers are available for many different operating systems. We will discuss installing a compiler shortly, so there is no need to do so now.

Step 5: Linking object files and libraries

After the compiler creates one or more object files, then another program called the **linker** kicks in. The job of the linker is three fold:

First, to take all the object files generated by the compiler and combine them into a single executable program.



Second, in addition to being able to link object files, the linker also is capable of linking library files. A **library file** is a collection of precompiled code that has been "packaged up" for reuse in other programs.

The C++ core language is actually fairly small and concise (and you'll learn much of it in these tutorials). However, C++ also comes with an extensive library called the **C++ Standard Library** (usually shortened to *standard library*) that provides additional functionality that you can use in your programs. One of the most commonly used parts of the C++ standard library is the *iostream library*, which contains functionality for printing text on a monitor and getting keyboard input from a user. Almost every C++ program written utilizes the standard library in some form, so it's very common for the standard library to get linked into your programs. Most linkers will automatically link in the standard library as soon as you use any part of it, so this generally isn't something you need to worry about.

You can also optionally link in other libraries. For example, if you were going to write a program that played sounds, you probably would not want to write your own code to read in the sound files from disk, check to ensure they were valid, or figure out how to route the sound data to the operating system or hardware to play through the speaker -- that would be a lot of work! Instead, you'd probably download a library that already knew how to do those things, and use that. We'll talk about how to link in libraries (and create your own!) in the appendix.

Third, the linker makes sure all cross-file dependencies are resolved properly. For example, if you define something in one .cpp file, and then use it in another .cpp file, the linker connects the two together. If the linker is unable to connect a reference to something with its definition, you'll get a linker error, and the linking process will abort.

Once the linker is finished linking all the object files and libraries (assuming all goes well), you will have an executable file that you can then run!

For advanced readers

For complex projects, some development environments use a **makefile**, which is a file that describes how to build a program (e.g. which files to compile and link, or otherwise process in various ways). Entire books have been written about how to write and maintain makefiles, and they can be an incredibly powerful tool. However, because makefiles are not part of the C++ core language, nor do you need to use them to proceed, we'll not discuss them as part of this tutorial series.

Steps 6 & 7: Testing and Debugging

This is the fun part (hopefully)! You are able to run your executable and see whether it produces the output you were expecting!

If your program runs but doesn't work correctly, then it's time for some debugging to figure out what's wrong. We will discuss how to test your programs and how to debug them in more detail soon.

Integrated development environments (IDEs)

Note that steps 3, 4, 5, and 7 all involve software (editor, compiler, linker, debugger). While you can use separate programs for each of these activities, a software package known as an integrated development environment (IDE) bundles and integrates all of these features together. We'll discuss IDEs, and install one, in the next section.

 **0.6 -- Installing an Integrated Development Environment (IDE)**

 **Index**

 **0.4 -- Introduction to C++ development**

C++ TUTORIAL | PRINT THIS POST

28 comments to 0.5 — Introduction to the compiler, linker, and libraries

Mariusz Jędrzejowski
December 20, 2019 at 8:24 am · Reply

Explanation is poor. Why to link other people's libraries this way since we can combine them via dll files?

Vitaliy Sh.
December 27, 2019 at 2:13 am · Reply

Hi sir!

I think because: ``In these tutorials, we will avoid any platform specific code.``
https://www.learncpp.com/cpp-tutorial/introduction-to-programming-languages/

And: ``Merging multiple shared libraries into one is indeed practically impossible on all UNIXen, except AIX: the linker considers the .so a "final" product.``
https://stackoverflow.com/questions/915128/merge-multiple-so-shared-libraries#924181

Neo
December 9, 2019 at 4:26 am · Reply

@Murat I think exe. means it is already executed

Helgrin
February 7, 2020 at 6:33 am · Reply

exe means executable i think

murat
August 10, 2019 at 12:14 pm · Reply

Can you please tell me if this is correct?:

1: Translation happens (Preprocessor manipulates text in each code file based on preprocessor directives, and some other translation phases).
2: Code gets compiled (Compiler checks if code is written based on the c++ rules and object files are created for each individual cpp files)
3: Linking happens
- All cpp files are linked together based on declaration and definitions.

I really don't know how what the exe file contains. Does it contain all code in bye form? If so, why should linking take place because all code are placed in one file..

Hope you can help me out with this one..

nascardriver
August 11, 2019 at 12:03 am · Reply

> I really don't know how what the exe file contains.
There are several binary formats. Simply said, they're separate into sections. There's a section for data, one for your code, and several others. The code is compiled and linked, your original source code cannot be restored from an executable. You might be able to decompile functions into code that behaves like yours, but it won't be the same.

> why should linking take place because all code are placed in one file
Each source file is compiled into a separate output file. Those have to be linked together.

murat
August 11, 2019 at 10:34 am · Reply

Hi, thanks for your reply!

So does the exe file contain all the cpp files (after they are linked together), or are all those code in cpp files placed in one place/file?

nascardriver
August 12, 2019 at 11:36 pm · Reply

The code is no longer there, it's all assembly (bytes that your computer understands) now. After they are linked, they're all in 1 file, the exe. There's no indication about files anymore, all assembly looks the same.

Arend
May 6, 2019 at 4:30 am · Reply

where is appendix ;-;

nascardriver
May 6, 2019 at 4:32 am · Reply

After chapter 18

Logan
April 25, 2019 at 5:16 am · Reply

Hi Alex, I am a returning student; it has been sometime since I had a crack at C++. So I am getting back to basics and following your tutorial; which is great and very appreciated!

I am curious ask about the library "using namespace std;" I saw many other videos showing how to program in C++ using that library. I understand it must add to the memory if you include it, and some people mentioned problems using it; if you have big program. I believe they were referring to the possibility of functions being updated and conflicts with the older updates of the library. Some programmers said they got used to typing the extra code, and were better off without the library. I wanted to ask you, what is the main problem using it? And what does the industry do with regards to this library? Do they avoid using it in companies?

nascardriver
April 26, 2019 at 2:57 am · Reply

```
1 using namespace std;
```

doesn't mean that you want to use the @std library. You can always use the @std library, if you include a header from it. The "using" statement allows you to access the members of @std without using the "std::" prefix. eg.

```
1 cout << "Hello\n";
2 // instead of
3 std::cout << "Hello\n";
```

This can lead to name conflicts, if you have an object with the same name as an object in the @std library.

Logan
April 26, 2019 at 4:49 am · Reply

Is that a realistic problem just name conflicts? I thought there were other bigger issues at work here. Again you or anyone what does the industry use? How is this deemed in the actual development world as opposed to what universities teach?

nascardriver
April 26, 2019 at 4:52 am · Reply

> I thought there were other bigger issues at work here.
No, just name conflicts.

"using namespace" shouldn't be and mostly isn't used in practice. Of course, there will be that one guy or company who uses it.
A lot of teaching institutes use "using namespace", because it makes the code more readable and thus friendly for beginners. This comes with the downside of teaching bad practice.

Puya
March 12, 2019 at 10:33 pm · Reply

Grateful reader here. I've heard this phrase that we cannot debug some issue because "we don't have the symbols" for some library (e.g. when stepping through code). What does that mean? Is that related to having the source code?

nascardriver
March 13, 2019 at 8:29 am · Reply

Hi!

When compiling C++, pretty much all information about the source code is lost, because your computer doesn't need to know how you named things or what your file structure was.
In debug mode, symbols can be added to the binary. Those are the names of functions, variables, etc., which of course help immensely when debugging.
You can still debug programs without symbols, but you'll have to figure out what is what on your own.

Robert Young
March 6, 2019 at 2:44 pm · Reply

Comlete newbie here, feeling ambitious about maybe trying to make my own music recording software. Is this possible with C & C++ orwill I need other languages

nascardriver
March 7, 2019 at 9:07 am · Reply

C++ can do everything.

Bruno
January 30, 2019 at 9:44 am · Reply

Hey, Alex!

I'm loving this tutorial, ty so much for this. I have some question tho. Is it ok if i print it? I'm making a "summary", don't know the right word for it, as i study these. At the end i'm gonna have what i wrote down from the tutorial and i think it's wise to also have the source stored with it. I know its silly to ask but i just feel like i should. Also, i bought this book "C++ Primer Plus - 5th Ed - PHE - Pearson Higher Education" to help me with the learning. Is it a good idea to use it as i go through the tutorial. Ty again so much for this :D.

Alex
January 31, 2019 at 5:26 pm · Reply

You are free to print the tutorials for your own personal use, so long as you do not distribute them. :) Have fun!

It's always best to learn from multiple sources, as they will give you different perspectives.

Bruno
February 4, 2019 at 10:56 am · Reply

I would never do that. Again, thanks a lot for this :D. I don't have a credit card to donate :(. Any other way i can donate/help?

Alex
February 4, 2019 at 9:15 pm · Reply

If you see any typos or omissions, point them out. That helps me improve the tutorials for everyone!

Phil
January 29, 2019 at 11:06 pm · Reply

"If the linker is unable to connect a reference to a thing with the definition of that thing, you'll get a linker error, and the linking process will abort."

would sound better as:

"If the linker is unable to connect a referenced thing with its definition, you'll get a linker error, and the linking process will abort.

Hassan Muhammad
November 27, 2018 at 11:52 am · Reply

Hi Alex,
I am new to c++;
assuming I'm n't interested in other libraries and i want to build my own independent OS, can i possibly write my own code to implement it with my own libraries/code without those libraries?

Alex
December 2, 2018 at 2:54 pm · Reply

Sure. You can avoid use of any of the code in the standard library (or any other library) and only use the code you've written yourself.

This generally isn't recommended, because the code in the standard library is useful and tested, and using it will dramatically shorten the time it takes to write functional programs.

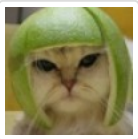
Rob
October 6, 2018 at 10:11 pm · Reply

Ah, the English language is alive and well as long as we have people like Ryan!

And with such great attention to detail, I expect that you' No slouch as a programmer.

Ryan
September 22, 2018 at 7:45 pm · Reply

Hello! I just wanted to point out a couple small typos I noticed. In the Step 5 section, you used 'a' instead of 'an' in "However, C++ also comes with a extensive library". In the box at the bottom, I think you meant to say something like 'describing', or 'which is a file that describes' in "which is a file describes how to build a program".



Alex
September 24, 2018 at 7:45 am · Reply

Typos fixed. Thanks for pointing these out!