1.8 — Programs with multiple files

BY ALEX ON JUNE 2ND, 2007 | LAST MODIFIED BY ALEX ON AUGUST 8TH, 2017

As programs get larger, it is not uncommon to split them into multiple files for organizational or reusability purposes. One advantage of working with an IDE is they make working with multiple files much easier. You already know how to create and compile single-file projects. Adding new files to existing projects is very easy.

Adding files to your project in Visual Studio

In Visual Studio, right click on “Source Files” in the Solution Explorer window on the left, and choose Add -> New Item. Make sure you have “C++ File (.cpp)” selected. Give the new file a name, and it will be added to your project.

Note: If you create a new file from the File menu instead of from your project in the Solution Explorer, the new file won’t be added to your project automatically. You’ll have to add it to the project manually. To do so, right click on “Source Files” in the Solution Explorer, choose Add -> Existing Item, and then select your file.

When you compile your program, the new file will be automatically included, since it’s part of your project.

Adding files to your project in Code::Blocks

In Code::Blocks, go to the file menu and choose “new file”. Give the new file a name (don’t forget the .cpp extension), and Code::Blocks will ask you if you want to add it to the active project. Click “Yes”. Note that you will also have to click the “Release” and “Debug” checkboxes, to make sure it gets added to both versions.

If Code::Blocks doesn’t ask you whether you want to add the file to the active project, or if the file doesn’t show up in the project pane, you’ll need to add it to the project manually. To do so, right click on the project name in the project pane, choose “Add File”, locate the file you just created, and add it to the project.

Now when you compile your program, the new file will be automatically included, since it’s part of your project.

If the new file isn’t compiling or linking with Code::Blocks, make sure the file is set to compile and link. To do so, right click on the file in the project pane, and choose “Properties”. Under the “General” tab, you should see checkboxes that control whether the file is compiled and linked. Make sure they are checked.

Adding files to your project from the command line

From the command line, you can create the additional file yourself, using your favorite editor, and give it a name. When you compile your program, you’ll need to include all of the relevant code files on the compile line. For example: “g++ main.cpp add.cpp -o main”, where main.cpp and add.cpp are the names of your code files, and main is the name of the output file.

A multi-file example

Now, consider the following multiple-file program:

add.cpp:

```
//#include "stdafx.h" // uncomment if using Visual Studio

int add(int x, int y)
{
    return x + y;
}
```

main.cpp:

```
//#include "stdafx.h" // uncomment if using Visual Studio
#include <iostream>

int main()
{
    std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl;
    return 0;
```

```
}
```

Try compiling this program for yourself. You will note that it doesn't compile, and it gives the same compiler error as the program in the previous lesson where the functions were declared in the wrong order:

```
main.cpp(6) : error C3861: 'add': identifier not found
```

When the compiler is compiling a code file, it does not know about anything in other code files, or remember anything it has seen from previously compiled code files. In this case, when compiling main.cpp, it doesn't remember that it previously compiled function add() in add.cpp, so it complains that doesn't know what identifier add is.

This limited visibility and lack of memory is intentional, so that files may have functions or variables that have the same names as those in other files without causing a naming conflict. We'll see an example of this in the next lesson.

However, in this case, we want main.cpp to know about (and use) the add() function that lives in add.cpp. To give main.cpp access to the add function, we can use a forward declaration:

main.cpp with forward declaration:

```
//#include "stdafx.h" // uncomment if using Visual Studio
#include <iostream>

int add(int x, int y); // needed so main.cpp knows that add() is a function declared elsewhere

int main()
{
    std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl;
    return 0;
}
```

Now, when the compiler is compiling main.cpp, it will know what add() is. Using this method, we can give files access to functions that live in another file.

Try compiling add.cpp and the main.cpp with the forward declaration for yourself. If you get a linker error, make sure you've added add.cpp to your project or compilation line properly.

Something went wrong!

There are plenty of things that can go wrong the first time you try to work with multiple files. If you tried the above example and ran into an error, check the following:

1. If you get a compiler error about add() not being defined in main(), you probably forgot the forward declaration in main.

2. If you get a linker error about add not being defined, e.g.

```
unresolved external symbol "int __cdecl add(int,int)" (?add@@YAHHH@Z) referenced in function _main
```

2a. …the most likely reason is that add.cpp is not added to your project correctly. If you're using Visual Studio or Code::Blocks, you should see add.cpp listed in the Solution Explorer/project pane on the left side of the IDE. If you don't, right click on your project, and add the file, then try compiling again. If you're compiling on the command line, don't forget to include both main.cpp and add.cpp in your compile command.

2b. …it's possible that you added add.cpp to the wrong project.

2c. …it's possible that the file is set to not compile or link. Check the file properties and ensure the file is configured to be compiled/linked. In Code::Blocks, compile and link are separate checkboxes that should be checked. In Visual Studio, there's an "exclude from build" option that should be set to "no" or left blank.

3. If you're using Visual Studio with precompiled headers, *every* code file needs to #include "stdafx.h". This includes both main.cpp and add.cpp.

4. Do *not* #include "add.cpp" from main.cpp. This will cause the compiler to insert the contents of add.cpp directly into main.cpp instead of treating them as separate files. While it may compile and run for this simple example, you will encounter problems down the

road using this method.

Conclusion

When the compiler compiles a multi-file program, it may compile the files in any order. Additionally, it compiles each file individually, with no knowledge of what is in other files.

We will begin working with multiple files a lot once we get into object-oriented programming, so now's as good a time as any to make sure you understand how to add and compile multiple file projects.

Reminder: Whenever you create a new code (.cpp) file, you will need to add it to your project so that it gets compiled.


1.8a -- Naming conflicts and the std namespace


Index


1.7 -- Forward declarations and definitions

Share this:

C++ TUTORIAL | PRINT THIS POST

323 comments to 1.8 — Programs with multiple files

« Older Comments 1 … 3 4 5

sri
July 1, 2018 at 3:01 pm · Reply

when calling add function in add.cpp in main.cpp, do we not need to #include "add.cpp" in main.cpp?

nascardriver
July 2, 2018 at 4:19 am · Reply

Hi sri!
No, don't include source files!
The compiler compiles all source files automatically and, because of the declaration of @add in @main.cpp, the compiler knows that the function is defined elsewhere.

Hebi
June 30, 2018 at 11:04 pm · Reply

I don't know what happened to my visual studio but I set up the files right like in the tutorial. Included all the head files that I needed to, did the forward declaration in my main file. I have even made one add.cpp file in each directory that I can find using the solution manager. But the error kept showing up saying that add() is undefined. And I checked out multiple websites they kept saying there is supposedly a 'Source Files' but there isn't. There are only debug and release files. With the Release files containing all the .cpp files. The program worked when I do #include add.cpp but did not when I just compile everything somehow. That means the add.cpp is in the needed directory that my program can find but it