# Files, Functions, and Namespaces

A C++ (C,Java,C#) program is a collection of functions. These functions live in text files you can create/edit with notepad or vi or your favorite editor.

You compile all of these functions together to make an executable program you can launch by double clicking or running from a command prompt.

There are two steps in making an executable: 1) compile all of the text files into binary code … one binary file for each text file. 2) link functions in the binary files together into an executable.

When you use other people’s code you might get their text files to compile into your code. Or you might get just the result of the text-to-binary step.

An IDE (integrated development environment) is an editor/compiler/debugger rolled into one. You can edit code and build it and run it all without leaving the GUI program.

Build up from the ground. One file:

void doWork(int a, int b) {

}

int main() {

doWork(1,2);

return 0;

}

Basic Syntax:

* Brackets and semicolons
* Comments
* Functions can return one value or nothing (void for nothing)
* Names (use verbs). Must be unique. Must start with letter (helps the parser distinguish them from numeric literals).
* Passing values to functions. Type is given.
* Instructions flow from top of function to bottom.
* Calling other functions
* Returning a value
* Spacing doesn’t matter … tabs/spaces … brackets … religious wars … use whatever your company likes. If you have a choice use what you like.
* The “main” is always the first thing executed. The order in the code doesn’t matter.

The order in a single file DOES matter. The compiler works on one file at a time from top to bottom and has no idea what comes later in the file or what’s in other files or even what other files are available.

At runtime the CPU flows from function to function. It keeps up with where it was and how to return to previous functions.

void doWork(int a, int b) {

doMoreWork();

}

int main() {

doWork(1,2);

doMoreWork();

return 0;

}

void doMoreWork() {

}

This does not compile. The “doMoreWork” function isn’t defined until after it is used. The compiler doesn’t know it is coming and gives an error.

You could move the function up, but it gets challenging to keep the order. And maybe you want a specific order for readability.

void doMoreWork();

void doWork(int a, int b) {

doMoreWork();

}

Use a forward-declare by giving the function definition without the body. The compiler makes a note of this and waits to see.

You can pull other files into your file with the “#include” statement. This is a direct text substitution. This happens BEFORE the compiler sees the code.

void doMoreWork();

void doWork(int a, int b) {

#include "myJunk.txt"

}

int main() {

#include "myJunk.txt"

#include "myJunk.txt"

return 0;

}

void doMoreWork() {

}

Important: this is a direct text substitution that happens before the compiler gets the code. The includes can appear anywhere. The compiler never sees them. Note there is no “;” on the end.

Included files can include other files (nesting).

Code is organized into functions. Functions are organized into separate files as you choose. For instance, I am making a set of functions to use in lots of programs:

// mathHelp.cpp

int getZero() {

return 0;

}

int getOne() {

return 1;

}

int getTwo() {

return 2;

}

I add this to my “list of text files” for compiling into my project.

int getZero();

int getOne();

int getTwo();

int main() {

getOne();

getZero();

getTwo();

return 0;

}

Everyone who uses my library must type out all of the forward-declares. I can make a single file for including that does these.

// mathLibFunctions.txt

int getZero();

int getOne();

int getTwo();

#include "mathLibFunctions.txt"

int main() {

getOne();

getZero();

getTwo();

return 0;

}

These includes will appear at the “head” of source files that use them. These code-less files are called “header” files, and the convention is to use “.h”. Some newer headers (system) don’t use any extension. I am changing this to “mathLib.h” to go with my “mathLib.cpp”.

Lots and lots of system libraries out there. The “stdio.h” describes functions that interact with files and the console. You can right-click in the IDE and look at it … just a header file. No code. That’s in binary form in a system library.

#include <stdio.h>

void doMoreWork();

void doWork(int a, int b);

int main() {

printf("main starts\n");

doWork(1,2);

doMoreWork();

printf("main ends\n");

return 0;

}

void doMoreWork() {

printf("doMoreWork starts\n");

printf("doMoreWork ends\n");

}

void doWork(int a, int b) {

printf("doWork starts\n");

doMoreWork();

printf("doWork ends\n");

}

The “<” means to look for the file in the system path. A quote means to look in the user’s path. Separates the concept of “installed” packages and user packages.

main starts

doWork starts

doMoreWork starts

doMoreWork ends

doWork ends

doMoreWork starts

doMoreWork ends

main ends

# Variables and Expressions

A

# Flow (if/while)

A