The Mini Pocket Guide To Learning C++

The easy to use, in-depth way of learning C++

By Jarren Long

Table of Contents

Preface: Introducing C++ and Quick Tips

Welcome to the world of C++! This guide is intended for the complete beginner to get a quick jump on programming without having to read dry, overly technical, and expensive books that aren't really helpful the complete beginner. In this guide, you will learn the basics of C++, as well as general coding techniques that can be used to learn other coding languages.

All programming languages are created equal. I say that, not because they are all exactly the same (none are), but because they all can do the same thing as any other language of the same complexity. All languages allow you to create variables, compute mathematical functions, and step through loops, among other things. Basically, if you are fluent in a language such as C++, learning Java won't so hard, because the only major difference is the syntax.

All that was just some filler crap that you really didn't need to read. Now, you're going to get yourself a "work bench". To use this guide effectively, you'll need a few things. First off, you need a nice and quiet room where you can your computer can "be alone". You really don't want to have a bunch of junk lying around, because it's kind of distracting. Music is ok, so long as it's not too loud.

This entire book is based around C++ for Windows without using MFC (Microsoft Foundation Classes). That means that you don't need to run out and buy the newest version of Visual C++ 2005 or any fancy software. Everything you need (minus the compiler) is already a part of Windows. Windows 2000 should be ok, but XP Home or better is preferred.

You will need a compiler. A compiler is a program that you use to put all of your code together and create a (hopefully) working PE file (Portable Executable A.K.A. Application [.exe]). There are many compilers out there: Borland, Visual C++, and many others, all for different coding languages. You don't have to actually buy one, because you can find a few free ones on the Internet, though you do need to make sure that you find one that is for C++.

All of the code samples included here were written and tested using Bloodshed's Dev C++ 5 compiler. I recommend using it. All of your code can be written inside the program, and then the click of a button will compile and run your application. Bloodshed Dev C++ 5 is available **for free** at www.bloodshed.net. After you install it, open it up and follow the directions to set it up. Make sure that when it asks you if it should parse the header files, say yes. It will speed up your compilation time later.

Now that you have your compiler set up, it's time to grab a microwave burrito and a Coka Cola, and get started learning C++ using the Windows 32-bit API (again, it comes standard with Windows).

Lesson 1: My First Program

Now that you have read the boring introduction and gotten yourself a compiler, it's time to create your first program! And here it is:

```
1: #include <iostream>
2:
3: int main()
4: {
5:       std::cout<<"Hello World!"<<std::endl;
6:       system("PAUSE");
7:       return 1;
8: }
9:
```

Now, compile this file (which can be named whatever you want it to be, so long as it has a .cpp extension) and execute the executable file created. You should see a black console widow with the text "Hello World!" in the top left corner. Congratulations! You wrote your first program! Now, it's time for the lengthy code explanation.

Line 1: #include <iostream>

This line is used to add the file "iostream.h", which contains the information required to use the std::cout and std::endl commands.

Line 2: Blank. The occasional blank line is used to keep your code tidy.

Line 3: int main()

This is the main function. EVERY program that you will ever write is REQUIRED to have this function. This is where the program looks for all of the program information. Without it, your program won't run.

Line 4: {

This is part of the int main() function. The int main() is used to declare the function, but the { is used to signal the beginning of the information inside the function.

Line 5: std::cout<,"Hello World!"<<std::endl;

Ok, this is the main part of your program. When you compiled and ran your program, you did see the black window with the text "Hello World!” right? That's what this line is for. The command std::cout is a part of the file "iostream.h" used to output text to the console (black) window. The symbols << are used to tell the std::cout what to output. The "Hello World!" is what's being outputted. Lastly, the std::endl is used to end the line that is outputted to the console. At the very end of the line is a semicolon (;). The semicolon is used to terminate the line of code. At the end of most any line of code will be a semicolon, with some exceptions (I won't go into them).

Line 6: system("PAUSE");

This part of the program is a Windows specific command that tells your program to wait until a key us pressed on the keyboard before moving on to the next line of code. Once again, a semicolon is used to terminate the line of code.

Line 7: return 1;

When a key on the keyboard is finally pressed, this line of code tells the int main() function that the contents of the function all executed smoothly, and the program can safely terminate itself. Because the int main() function is of the integer type, the function must return an integer (whole number). Using return 1 makes the function return 1, which is an integer. End with a semicolon and everything is dandy.

Line 8: }

Just as Line 3 is used to begin the function, the } symbol is used to end the function.

Line 9: Blank

This line is VERY IMPORTANT! At the end of all of your source code files, you need to have at least one (or more) blank lines. Don't ask why, because I don't really know. All I know is that, without this line, your program won't run correctly.

Now I know what you're thinking: "Holy crap! That’s a lot of info for a 9-line code sample!" Well yeah, it is. But don't worry! After a little practice, this will become second nature to you. To be frank, this is the simplest program you will ever write. But, like I said, it'll get easier with time, even if it looks harder.

Lesson 2: Using Variables

How did the last program run? Hopefully that was fairly easy for you. Now, you’re going to learn to use variables in your program. I bet you remember variables from Algebra, right? X? Y? Z? Here, we’re using the same concept, but adding onto it. In C++, a variable could be a number, character, string, a Boolean value (true or false), or void (nothing at all). There are many kinds of variable types, which I will show you in this next program. And here it is:

```
1: #include <iostream>
2:
3: int main()
4: {
5:      int a;
6:      double b;
7:      float c;
8:      a=10;
9:      b=1.8e30;
10:     c=10.00;
11:
12://This is how you add a
13://comment to your code
14://The two slashes tell the
15://program that this is not
16://a piece of the code!
17:
18:     std::cout<<”Integer: “<<a<<std::endl;
19:     std::cout<<”Double integer:  “<<b<<std::endl;
20:     std::cout<<”Floating Point integer: “<<c<<std::endl;
21:     system(“PAUSE”);
22:     return 1;
23: }
24:
```

Compile the program and execute the .exe file. A console window will open with some text in the top left corner. Once again, it’s time to explain everything:

Lines 1-4: See Lesson 1.
Line 5: int a;

This is the first variable. It is of type int, which can only be a whole (+ or -) number. The letter “a” is the variable’s name. Finish everything off with a semicolon, and you’ve declared your first variable!

Line 6-7: double b; float c;

These are both variables, too. They are both different types, double and float, and they are names “b” and “c”. Once again, don’t forget the semicolon!

Lines 8-10: a=10; b=1.8e30; c=10.10;

Now that you have declared the variables in lines 5-7, it’s time to make them equal something. Variable “a” is being set to equal the integer 10. “b” is being set to equal 1.8x10^30, and “c” is being set to equal 10.10;
Line 11: Blank.
Lines 12-16: Read the lines!
Line 17: Blank.
Lines 18-20: These lines are telling your program to output some text and to output the values of the three variables.
Lines 21-24: See Lesson 1.

You just learned how to use a few types of variables. There are more, however. Here is a table of some variable types:

Type	Size	Values
unsigned short int	2 bytes	0 to 65,535
short int	2 bytes	-32,768 to 32,767
unsigned long int	4 bytes	0 to 4,294,967,295
long int	4 bytes	-2,147,483,648 to 2,147,483,647
char	1 byte	256 char values
bool	1 byte	true or false
float	4 bytes	1.2e-38 to 3.4e38
double	8 bytes	2.2e-308 to 1.8e308 (e=x10^)

Lesson 3:Inputting Data to Your Program

In Lessons one and two, you learned how to create a console program, output text, and use variables. Now that you know how to output information, it's time, to learn how to input data. Inputting data requires that you know how to use variables properly. You must always make sure that you use the right variable type for your input. If you use a "char" type and the user inputs "1", your program will crash. There are ways to counter this problem called function overloading, which I will discuss later on. You are going to write a simple program that prompts the user to input a single character, an integer, and their name, which can be up to 32 characters long. And now for the code:

```
1: #include <iostream>
2:
3: int main()
4: {
5: int a;
6: char b;
7; char name[32];
8: std::cout << "Enter an integer:" << std::endl;
9: std::cin>>a;
10: std::cout << "Enter a character:" << std::endl;
11: std::cin>>b;
12: std::cout << "Enter your name:" << std::endl;
13; std::cin>>name;
14: std::cout << name << " entered " << a << " and " << b << "." << std::endl;
15: system("PAUSE");
16: return 1;
17: }
18:
```

The explanation of this 18-line code:

Lines 1-4: See Lesson One
Line 5: Declare "a" as an integer.
Line 6: Declare "b" as a character.
Line 7: Declare "name" as a string of characters up to 32 characters long.
Line 8: Output text to the console.
Line 9: Here is the new command. "std::cin>>a;" is used to tell the program to prompt the user to enter something, and to store the inputted data in variable "a". In this case, the program needs an integer for the input.
Line 10: Some more text output.
Line 11: std::cin>>b;

This time, your program is asking the user for a character and storing it in variable "b".

Line 12: std::cout << "Enter your name:" << std::endl;

More text output.

Line 13: std::cin>>name;

Lastly, your program wants too enter a string of characters (up to 32) and storing the string in variable “name”.

Line 14: std::cout << name << “ entered “ << a << “ and “ << b << “.” << std::endl;

Here, the program is just outputting some text and the values of the user’s input.

Now that you know how to use variables, the input is pretty easy. Input is vital to your program’s success.

Lesson 4: Basic Math

Let’s recap: You’ve learned how to output text, declare variables, and input text into a console application. Now, I’m going to teach you some math! I bet you’re thinking, “Man, I hated math!” Don’t worry! This lesson is just going to review some basic math skills like addition and division. Here, you’ll see two separate codes, one that is completely automatic, and one that asks for user input. Here is the first code:

```
1: #include <iostream>
2:
3: int main()
4: {
5: int a, b, c;
6: a=5;
7: b=5;
8: c=a*b;
9: std::cout << a << “x” <<b<<”=”<<c<<std::endl;
10: system(“PAUSE”);
11: return 1;
12: }
13:
```

Explanation:

Lines 1-4: As of now, I’m just going to skip these lines completely. If you don’t know these lines by now, see Lesson 1!
Line 5: Variables “a”, “b”, and “c” are all being declared as integers.
Lines 6-7: “a” and “b” are being set to equal 5.
Line 8: “c” is being set to equal “a” times “b”.
Line 9: Output some text and the value of “c”.
Lines 10-13: See Lines 1-4.

That sample is a demonstration of how to use multiplication with variables. Now, I’m going to share the almighty knowledge of how to do the exact same thing using user input. Here it is:

```
1: #include <iostream>
2:
3: int main()
4: {
5: int a, b, c;
6: c=a*b;
7: std::cout<<”What is A:”<<std::endl;
8: std::cin>>a;
9: std::cout<<”What is B:”<<std::endl;
10: std::cin>>b;
11: std::cout<<”a times b=”<<c<<std::endl;
12: system(“PAUSE”);
13: return 1;
14: }
15:
```

Explanation:

Line 5: “a”, “b”, and “c” are all of type integer.
Line 6: “c” is being set to equal “a” times “b”.
Lines 7 and 9: Output some text.
Line 8 and 10: Input a value for “a” and “b”.
Line 11: Output the values of “a”, ”b”, and “c”.

Once again, this is a basic demonstration of multiplying two values inputted by the user. In the next lesson, I’ll show you how to write some functions that will help you with your work.

Lesson 5: Writing a Function

Remember in the last lesson that you learned how to input values, multiply them, and return the answer? We're going to do the same thing here, but we're going to write some functions to do the work for us. Recall that from Lesson 1 that int main() is a function. There are three basic components to a function: the type of the function, the name, and the arguments that the function can accept.

$\boxed{\text{Type}}$ name (arguments)

We are going to write four functions and implement them in our console program. There's the source code that you will write:

```
#include <iostream>

int add(int a, int b)
{
	int c;
	c=a+b;
	return c;
}

int subtract(int a, int b)
{
	int c;
	c=a-b;
	return c;
}

int multiply(int a, int b)
{
	int c;
	c=a*b;
	return c;
}

int divide(int a, int b)
{
	int c;
	c=a/b;
	return c;
}

int main()
{
int a, b;
34: std::cout<<”a:”<<std::endl;
35: std::cin>>a;
36: std::cout<<”b:”<<std::endl;
37: std::cin>>b;
38: std::cout<<”Add function: ”<<add(a,b)<<std::endl;
39: std::cout<<”Subtract function: ”<<subtract(a,b)<<std::endl;
40: std::cout<<”Multiply function: ”<<multiply(a,b)<<std::endl;
41: std::cout<<”Divide function: ”<<divide(a,b)<<std::endl;
42: system(“PAUSE”);
43: return 1;
44: }
45:
```

This is going to be a long explanation!

Lines 3-29: A function called “add()” is created. It accepts two integers, “a” and “b”, and adds them together. The other three functions do the same thing, but with subtraction, multiplication, and division.
Line 33: “a” and “b” are being set as integers.
Lines 34 and 36: Output some text.
Lines 35 and 37: Input two integers for “a” and “b”.
Lines 38-41: Output the values of “a” and “b” after being sent through the four functions.

This is a pretty long code, but it should give you a fairly good feel for writing functions. There are millions of different kinds of functions with different uses. This lesson is just one example of a function.

Lesson 6: Namespaces: A True Timesaver

Have you ever noticed in the last five lessons that you've types std::cout and std::endl a couple dozen times? There is a way to shorten the code, just a little, saving you time in your programming process. Using namespaces to declare a scope will save you loads of time. "I use Scope every day, three times a day after brushing and flossing!" Ha, ha. Wrong scope. When you see the code snippet std::cout, the scope of signaled with the double-colons (::). In this case, the scope is names "std". "Enough talk! How do I shorten my code up?" To make this easy on you, all you have to do is declare the name of the scope you're using as a namespace. To do this, you need to type this at the top of your file:

```
using namespace std;
```

Snippet Breakdown:

Word 1: This tells the program that you are using something.
Word 2: Now, you're telling the program that you're using a namespace.
Word 3: Finally, this is the name of the scope you're using. You're using the namespace std.

Now try it out:

```
1: #include <iostream>
2:
3: using namespace std;
4:
5: int main()
6: {
7: cout<<"Haha! No stupid std::!"<<endl;
8: int a;
9: cout<<"Enter an integer:"<<endl;
10: cin>>a;
11: cout<<"You entered "<<a<<endl;
12: system("PAUSE");
13: return 1;
14: }
15:
```

Now, let's explain the code:

Line 7: Here, you are outputting text. Instead of using std::cout and std::endl, you just need to say "cout" and "endl".
Line 8: "a" is being declared as an integer.
Line 9: See Line 7.

Line 10: Here, you are inputting text. Since you declared the "std" namespace, you just need to say "cin".
Line 11: More text.

Now, when you use "using namespace std;", you are initializing all of the std:: commands. This can be a hassle in a large program that only uses one or two of the std:: commands. To work around this, you should use just "using std::cout", for example. You don't need the word "namespace" if you initialize just one command at a time. This allows you to use "cout", but doesn't initialize all of the std:: commands, just std::cout. Make sense?

A lot less typing, isn't it? Namespaces are very handy in saving space in your codes. From now on, you will be declaring the "std" namespace to save space in your code.

Lesson 7: Creating Your Own Namespaces

In the last chapter, I taught you how to shorten up your code by declaring the std namespace. Now, I’m going to teach you how to create your own namespace. “Why would I want to create my own namespace?” Like I said last chapter, namespaces are a true timesaver, especially when it comes to having several commands with the same name, but different uses. Here is an example of what you could use a namespace for:

```
1: #include <iostream>
2:
3: using namespace std;
4:
5:  namespace Xmas
6: {
7:         int month = 12;
8:         int day = 25;
9:         int year = 2005;
10: }
11:
12:  namespace Easter
13: {
14:       int month = 3;
15:       int day = 27;
16:       int year = 2005;
17: }
18:
19: int main()
20: {
21:       cout<<”Christmas is going to be on
“<<Xmas::month<<”/”<<Xmas::day<<”/”<<Xmas::year<<”.”<<endl;
22:       cout<<”Easter was on
“<<Easter::month<<”/”<<Easter::day<<”/”<<Easter::year<<”.”<<endl;
23:       system(“PAUSE”);
24:       return 1;
25: }
26:
```

By now, you should be pretty familiar with the layout I use in organizing this book. Guess what may be next:

Lines 5-10: Right here, you are creating the namespace called Xmas. Inside of the namespace, you are declaring three variables that can be used throughout the code (because they have a global scope).
Lines 12-17: Just as you created the namespace Xmas in lines 5-10, here, you are creating a namespace called Easter. Inside of the namespace, you are declaring the same variables

as you were in the Xmas namespace. Without the namespaces, this would create a problem in your code.
Lines 21 and 22: Here you are outputting some text, as well as the values of the variables inside of the two namespaces. To access the different variables in the namespaces, you must declare the scope of the namespace (which namespace to search for the variable(s)), and then the name of the variable. For example, to get the value of int month inside of namespace Easter, you would use Easter::month. When the text is outputted to the console, it should look like this:

```
Christmas is going to be on 12/25/2005.
Easter was on 3/27/2005.
```

If you have a namespace or two created in your code, you can use the "using namespace _______" so long as the two namespaces don't have conflicting variables or code snippets in them. For example, in the code above, I could use "using namespace Xmas" or "using namespace Easter", but I couldn't use them both together, because they have the same variables inside of them. That would basically "confuse" your computer and cause a crash.

Lesson 8: #include and #define

Up to this point in this manual, you have written all of your source codes in a single .cpp file. Now, we're going to add in another kind of file, called a "Header File". A header file is a file that contains #defines, #includes, function declarations, and all kinds of things. To create a header file, save a file with a .h (for C) or .hpp (for C++) extension. It doesn't really seem to matter which extension that you use; I use the .h with my .cpp extensions. It probably does in some way, but I don't know about it. It doesn't effect the compilation of the code.

Anyway, when you have a header file that you wish to compile with your main .cpp file, you must us the #include command inside of your .cpp file. Here is a quick example.

```
//begin code for main.cpp
1: #include <iostream>
2: #include "main.h"
//the rest of your code goes below, just as normal.
```

This code snippet searches the root directory of your compiler for the file <iostream> file, and then the specified directory for the "main.h" file. In this case, the compiler is told to search in the directory of the project for the main.h file.

"What's the point of having a second file?" I'm about to explain that. Inside of your header file, you usually have the #define commands that are used in the .cpp file. The #define commands are used to, well, define things. You usually define variables, but you can also define names, conditions, and numbers. Here is an example of using the #define command.

```
//begin code for main.h
1: #define ANUMBER 50
2: #define ASTRING "String"
3: #define REDEFINE TRUE
//etc, etc…
```

Basically, the first line creates a global variable called ANUMBER with the value 50. Line 2 defines ASTRING as being "String". Finally, the last line makes the variable REDEFINE equal to the value TRUE boolean value, so using if(!REDEFINE){} would be the same as if(!TRUE){} (! is a negation symbol, making both of those read if not true do the stuff in the brackets. That'll be covered later.). Now, here is an actual code sample for you to test out.

```
//8.cpp
1: #include <iostream>
2: #include "8.h"
3:
4: using namespace std;
5:
```

```
int main()
{
cout<<STRING<<endl;
cout<<NUMBER<<endl;
system(“PAUSE”);
return 0;
}

```

```
//8.h
#define STRING “This is a string”
#define NUMBER 256

```

In this case, using #define NUMBER could also be rewritten in the main.cpp file as int NUMBER = 256. STRING could also be rewritten as string STRING = “This is a string” or as char *STRING = “This is a string” or as char[16] STRING = “This is a string”. The only reason I didn’t do it the easy way is because I’m not teaching the easy way, I’m teaching how to use the #define and #include commands. Anyway, when this code is compiled, it should look like this:

```
This is a string
256
Press Any Key To Continue…
```

Headers are a simple, easy way to help organize and reuse your code. From now on, you will use them quite often, so get used to them!