

C++ Foundations





Within this section, you can begin to understand the structure of C++ code and how to compile and execute that code. The fundamentals of C++ will teach you the necessary basics, including using comments, variables, data types, strings, and how to use C++ mathematics.

These are the building blocks of a C++ program. With them, you can form your own code, produce an output to the screen and store and retrieve data. Building a good foundation is key to mastering C++, and learning how to structure your C++ code will help you further down the line.

-
- 116** Your First C++ Program
- 118** Structure of a C++ Program
- 120** Compile and Execute
- 122** Using Comments
- 124** Variables
- 126** Data Types
- 128** Strings
- 130** C++ Maths



Your First C++ Program

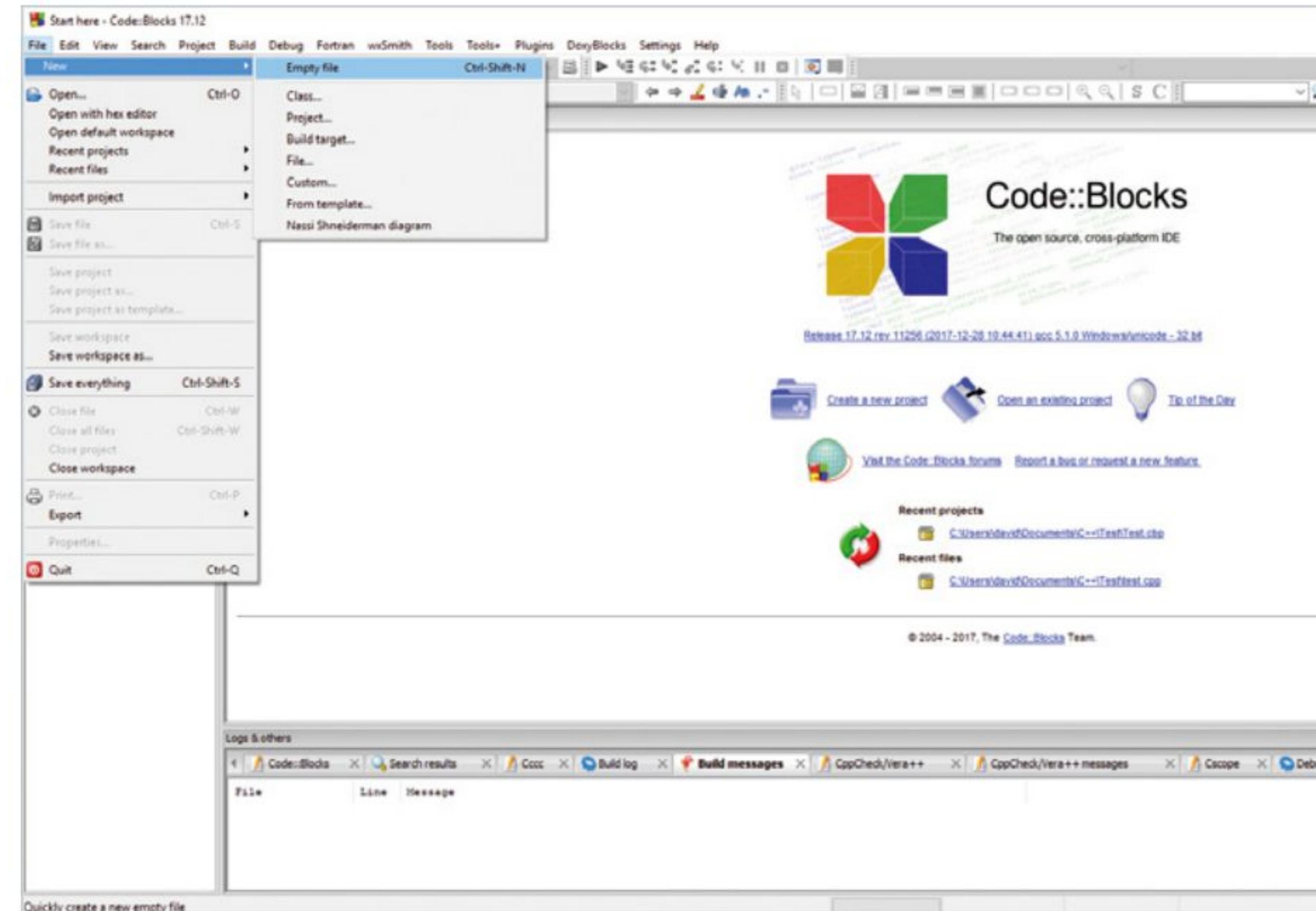
We're going to be working exclusively in Windows with Code::Blocks for this C++ section. Let's begin by writing your first C++ program and taking the first small steps into a larger coding world.

HELLO, WORLD!

It's traditional in programming for the first code to be entered to output the words 'Hello, World!' to the screen. Interestingly, this dates back to 1968 using a language called BCPL.

STEP 1

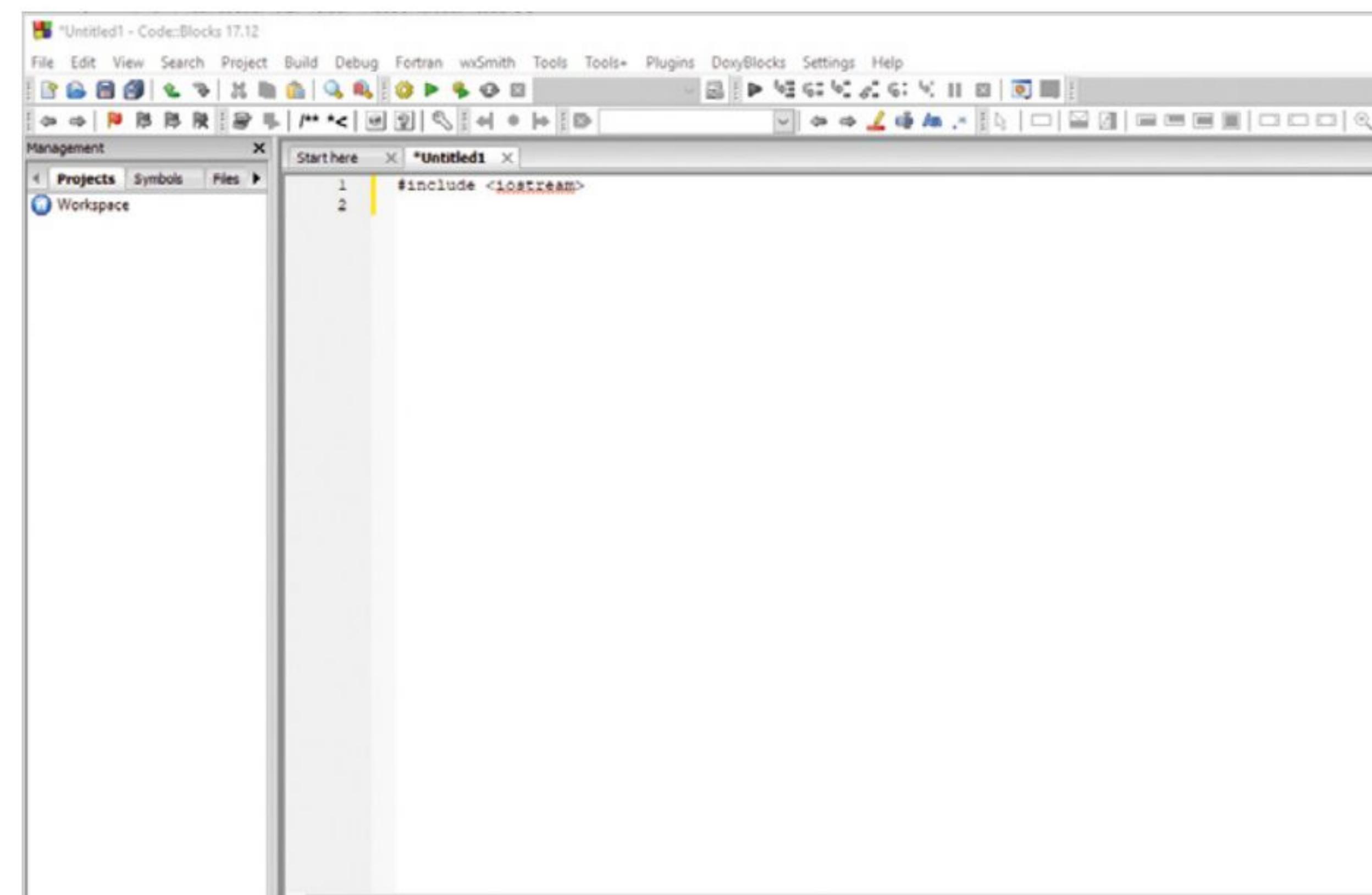
As mentioned, we're using Windows 10 and the latest version of Code::Blocks for the rest of the C++ code in this book. Begin by launching Code::Blocks. When open, click on File > New > Empty File or press Ctrl+Shift+N on the keyboard.



STEP 2

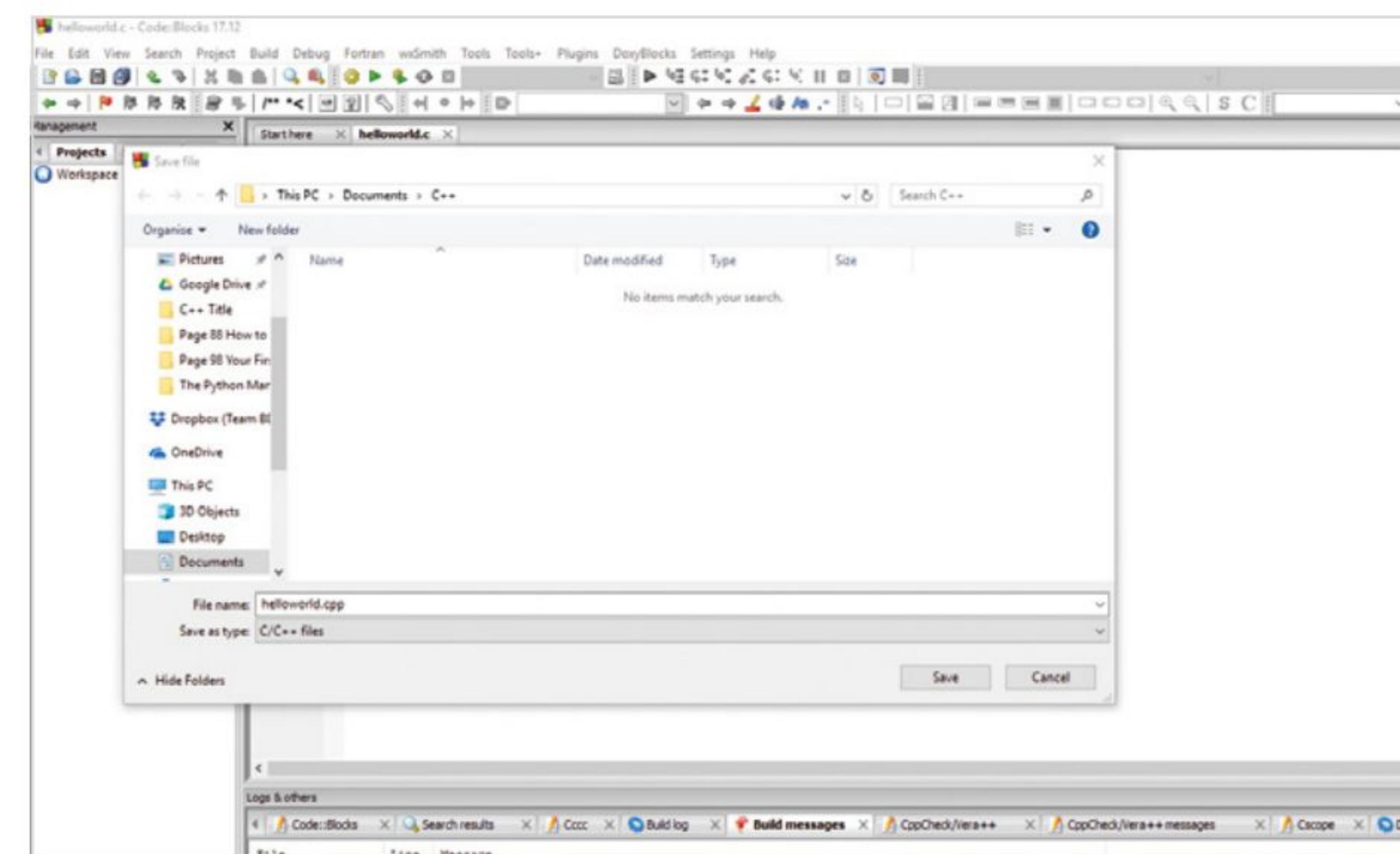
Now you can see a blank screen, with the tab labelled *Untitled1, and the number one in the top left of the main Code::Blocks window. Begin by clicking in the main window, so the cursor is next to the number one, and entering:

```
#include <iostream>
```



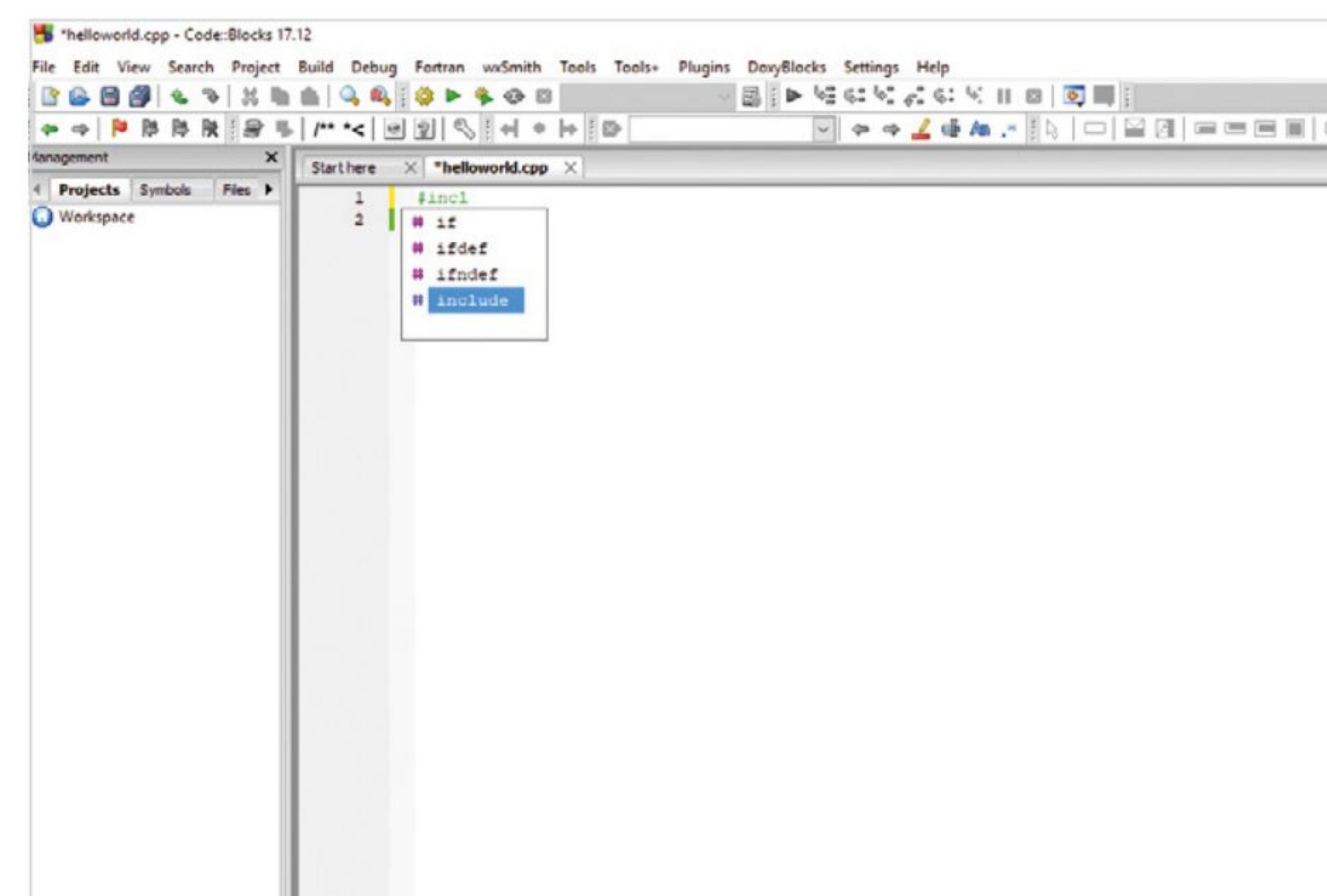
STEP 3

At the moment it doesn't look like much, and it makes even less sense, but we'll get to that in due course. Now click on File > Save File As. Create or find a suitable location on your hard drive and in the File Name box, call it helloworld.cpp. Click the Save as type box and select C/C++ files. Click the Save button.



STEP 4

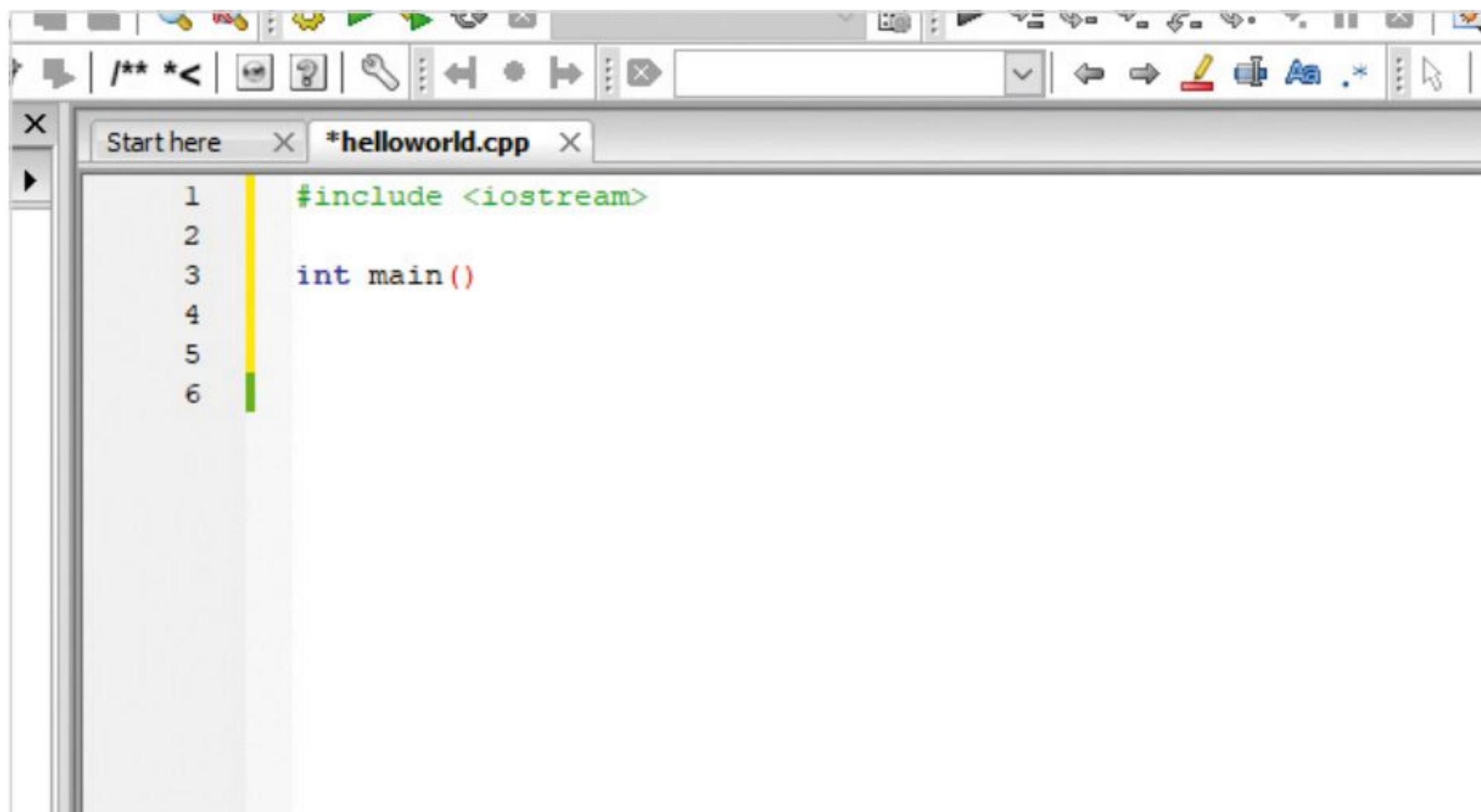
You can see that Code::Blocks has now changed the colour coding, recognising that the file is now C++ code. This means that code can be auto-selected from the Code::Blocks repository. Delete the #include <iostream> line and re-enter it. You can see the auto-select boxes appearing.



STEP 5 Auto-selection of commands is extremely handy and cuts out potential mistyping. Press Return to get to line 3, then enter:

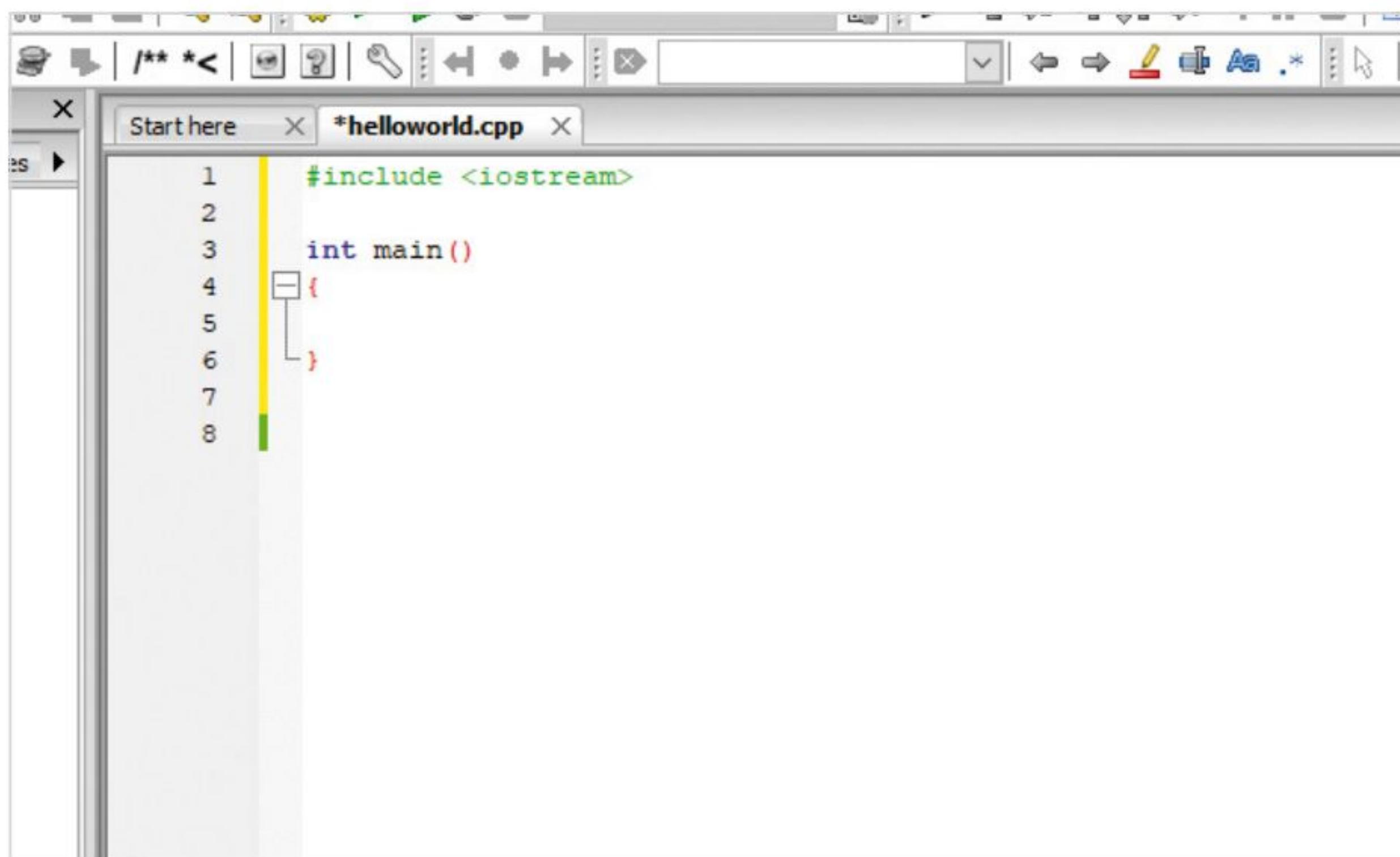
```
int main()
```

Note: there's no space between the brackets.



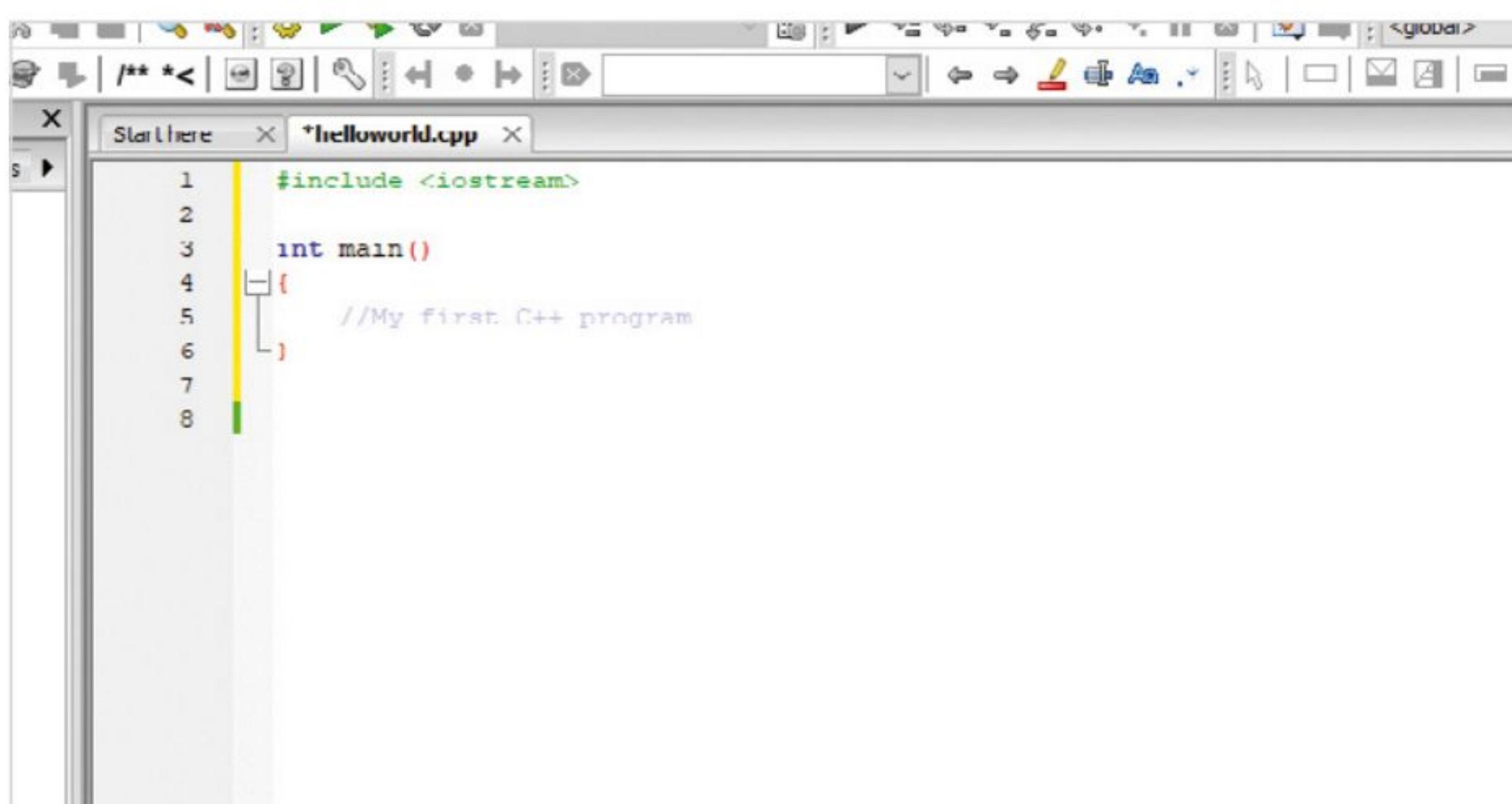
STEP 6 On the next line below int main(), enter a curly bracket:
{

This can be done by pressing Shift and the key to the right of P on an English UK keyboard layout.



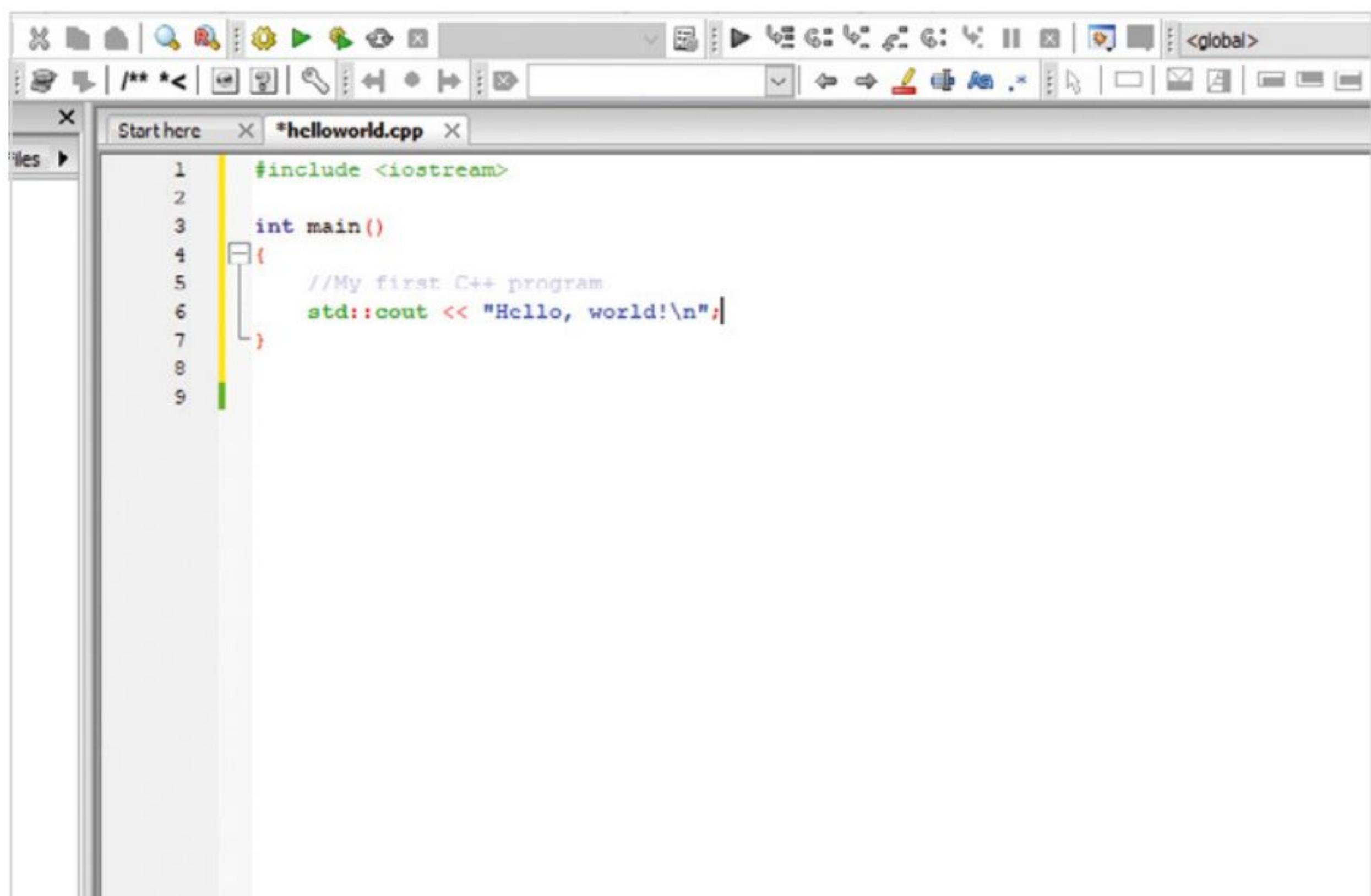
STEP 7 Notice that Code::Blocks has automatically created a corresponding closing curly bracket a couple of lines below, linking the pair, as well as a slight indent. This is due to the structure of C++ and it's where the meat of the code is entered. Now enter:

```
//My first C++ program
```

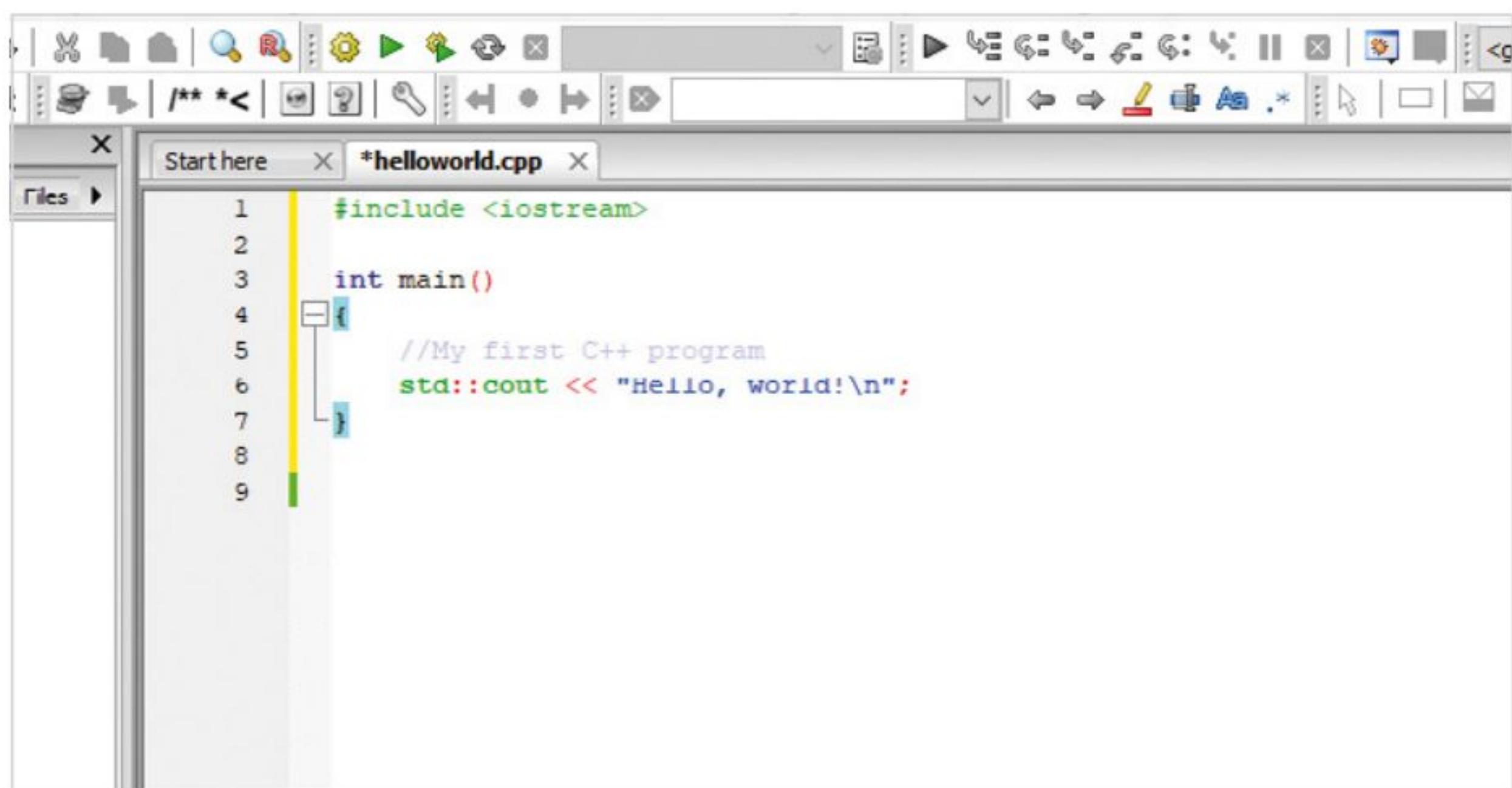


STEP 8 Note again the colour coding change. Press Return at the end of the previous step's line, and then enter:

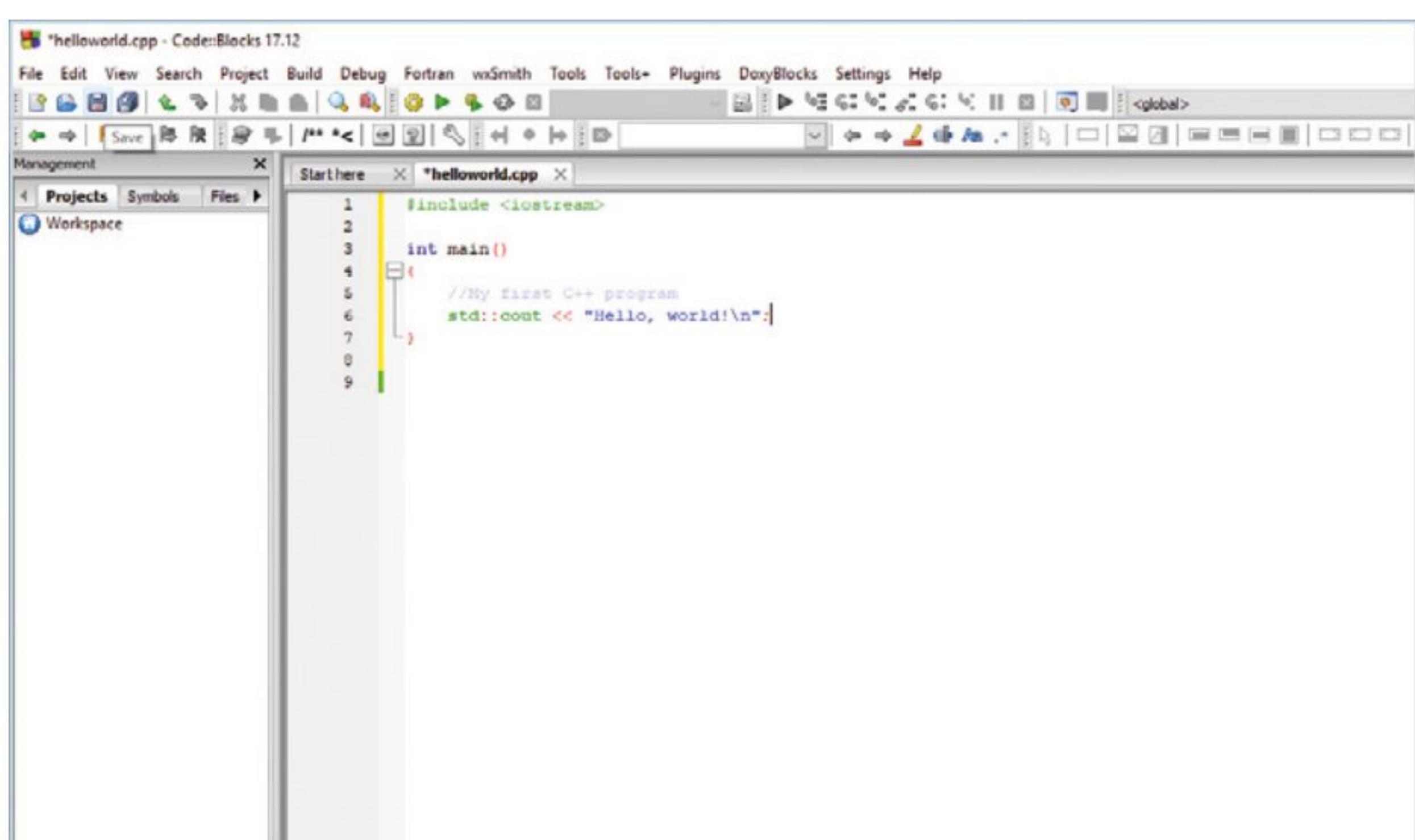
```
std::cout << "Hello, world!\n";
```



STEP 9 Just as before, Code::Blocks auto-completes the code you're entering, including placing a closing speech mark as soon as you enter the first. Don't forget the semicolon at the end of the line; this is one of the most important elements to a C++ program and we'll tell you why in the next section. For now, move the cursor down to the closing curly bracket and press Return.



STEP 10 That's all you need to do for the moment. It may not look terribly amazing but C++ is best absorbed in small chunks. Don't execute the code at the moment as you need to look at how a C++ program is structured first; then you can build and run the code. For now, click on Save, the single floppy disc icon.





Structure of a C++ Program

C++ has a very defined structure and way of doing things. Miss something out, even as small as a semicolon, and your entire program will fail to be compiled and executed. Many a professional programmer has fallen foul of sloppy structure.

#INCLUDE <C++ STRUCTURE>

Learning the basics of programming, you begin to understand the structure of a program. The commands may be different from one language to the next, but you will start to see how the code works.

C++

C++ was invented by Danish student Bjarne Stroustrup in 1979, as a part of his Ph.D. thesis. Initially C++ was called C with Classes, which added features to the already popular C programming language, while making it a more user-friendly environment through a new structure.

Bjarne Stroustrup, inventor of C++.



#INCLUDE

The structure of a C++ program is quite precise. Every C++ code begins with a directive: **#include** <>. The directive instructs the pre-processor to include a section of the standard C++ code. For example: **#include <iostream>** includes the iostream header to support input/output operations.

```
Start here × *helloworld.cpp ×
1 #include <iostream>
2
3
4
5
6
```

INT MAIN()

int main() initiates the declaration of a function, which is a group of code statements under the name 'main'. All C++ code begins at the main function, regardless of where it actually lies within the code.

```
Start here × *helloworld.cpp ×
1 #include <iostream>
2
3 int main()
4
5
6
```

BRACES

The open brace (curly brackets) is something that you may not have come across before, especially if you're used to Python. The open brace indicates the beginning of the main function and contains all the code that belongs to that function.

```
1 #include <iostream>
2
3 int main()
4 {
5
6 }
```

COMMENTS

Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line. Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line.

```

Start here *helloworld.cpp X
1 #include <iostream>
2
3 int main()
4 {
5     //My first C++ program
6 }
7
8
9

```

STD

While **std** stands for something quite different, in C++ it means Standard. It's part of the Standard Namespace in C++, which covers a number of different statements and commands. You can leave the **std::** part out of the code but it must be declared at the start with: **using namespace std;** not both. For example:

```
#include <iostream>
using namespace std;
```

```

Start here *helloworld.cpp X
1 #include <iostream>
2
3 int main()
4 {
5     //My first C++ program
6     std::cout << "Hello, world!\n"; //Remember: declare
7 }
8
9

```

COUT

In this example we're using cout, which is a part of the Standard Namespace, hence why it's there, as you're asking C++ to use it from that particular namespace. Cout means Character OUTput, which displays, or prints, something to the screen. If we leave **std::** out we have to declare it at the start of the code, as mentioned previously.

```

Start here *helloworld.cpp X
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout
8 }
9

```

<<

The two chevrons used here are insertion operators. This means that whatever follows the chevrons is to be inserted into the std::cout statement. In this case they're the words 'Hello, world', which are to be displayed on the screen when you compile and execute the code.

```

//My first C++ program
std::cout << "Hello, world!\n"

```

std
public ostream cout
(variable)
Open declaration
Close Top

OUTPUTS

Leading on, the "Hello, world!" part is what we want to appear on the screen when the code is executed. You can enter whatever you like, as long as it's inside the quotation marks. The brackets aren't needed but some compilers insist on them. The \n part indicates a new line is to be inserted.

```
//My first C++ program
cout << "Hello, world!\n"
```

; AND }

Finally you can see that lines within a function code block (except comments) end with a semicolon. This marks the end of the statement and all statements in C++ must have one at the end or the compiler fails to build the code. The very last line has the closing brace to indicate the end of the main function.

```

Start here *helloworld.cpp X
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout << "Hello, world!\n";
8 }
9

```



Compile and Execute

You've created your first C++ program and you now understand the basics behind the structure of one. Let's actually get things moving and compile and execute, or run if you prefer, the program and see how it looks.

GREETINGS FROM C++

Compiling and executing C++ code from Code::Blocks is extraordinarily easy; just a matter of clicking an icon and seeing the result. Here's how it's done.

STEP 1

Open Code::Blocks, if you haven't already, and load up the previously saved Hello World code you created. Ensure that there are no visible errors, such as missing semicolons at the end of the std::cout line.

```
#include <iostream>
int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n";
}
```

STEP 2

If your code is looking similar to the one in our screenshot, then look to the menu bar along the top of the screen. Under the Fortran entry in the topmost menu you can see a group of icons: a yellow cog, green play button and a cog/play button together. These are Build, Run, Build and Run functions.

```
#include <iostream>
int main()
{
    //My first C++ program
}
```

STEP 3

Start by clicking on the Build icon, the yellow cog. At this point, your code has now been run through the Code::Blocks compiler and checked for any errors. You can see the results of the Build by looking to the bottom window pane. Any messages regarding the quality of the code are displayed here.

```
in "no project" (compiler: unknown) ===
0 warning(s) (0 minute(s), 0 second(s))
```

STEP 4

Now click on the Run icon, the green play button. A command line box appears on your screen displaying the words: Hello, world!, followed by the time it's taken to execute the code, and asking you press a key to continue. Well done, you just compiled and executed your first C++ program.

```
Hello, world!
Process returned 0 (0x0) execution time : 0.039 s
Press any key to continue.
```

STEP 5 Pressing any key in the command line box closes it, returning you to Code::Blocks. Let's alter the code slightly. Under the #include line, enter:

```
using namespace std;
```

Then, delete the std:: part of the Cout line; like so:

```
cout << "Hello, world\n";
```

```
Start here * helloworld.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout << "Hello, world!\n";
8 }
9
10
11
12
```

STEP 6 In order to apply the new changes to the code, you need to re-compile, build, and run it again. This time, however, you can simply click the Build/Run icon, the combined yellow cog and green play button.

```
Build Debug Fortran wxSmith Tools Tools+ Plugins Doxygen Settings Help
Start here * helloworld.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout << "Hello, world!\n";
8 }
9
10
11
12
```

C:\Users\david\Documents\C++\helloworld.exe
Hello, world!
Process returned 0 (0x0) execution time : 0.041 s
Press any key to continue.

STEP 7 Just as we mentioned in the previous pages, you don't need to have std::cout if you already declare using namespace std; at the beginning of the code. We could have easily clicked the Build/Run icon to begin with but it's worth going through the available options. You can also see that by building and running, the file has been saved.

```
helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins Doxygen Settings Help
Management Start here * helloworld.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout << "Hello, world!\n";
8 }
9
10
11
12
```

STEP 8 Create a deliberate error in the code. Remove the semicolon from the cout line, so it reads:

```
cout << "Hello, world!\n"
```

```
Start here * helloworld.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout << "Hello, world!\n"
8 }
9
10
11
12
```

STEP 9 Now click the Build and Run icon again to apply the changes to the code. This time Code::Blocks refuses to execute the code, due to the error you put in. In the Log pane at the bottom of the screen you are informed of the error, in this case: Expected ';' before '}' token, indicating the missing semicolon.

```
Start here * helloworld.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout << "Hello, world!\n"
8 }
9
10
11
12
```

Log pane: Expected ';' before '}' token

STEP 10 Replace the semicolon and under the cout line, enter a new line to your code:

```
cout << "And greetings from C++!\n";
```

The \n simply adds a new line under the last line of outputted text. Build and Run the code, to display your handiwork.

```
helloworld - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins Doxygen Settings Help
Management Start here * helloworld.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     //My first C++ program
7     cout << "Hello, world!\n";
8     cout << "And greetings from C++!\n";
9 }
10
11
12
```

C:\Users\david\Documents\C++\helloworld.exe
Hello, world!
And greetings from C++!
Process returned 0 (0x0) execution time : 0.046 s
Press any key to continue.

Log pane: No errors or warnings found.