# C++ Basics

Each C++ program consists of two parts: the preprocessor directives and the main function. Let’s look at a short program and analyze its different sections.

#include <iostream>

int main()

{

std::cout << "Hello world, I am ready for C++";

return 0;

}

The first line we encounter has a hash at the start of the line. Any line that has a hash sign at the start is a **preprocessor directive**.

#include <iostream>

After the hash sign we have the word include. There are several preprocessor directives available in C++, but include is the one we see and use the most often.

Include means add the declarations of the given library. In this case we are adding the declarations of the iostream library.

The final detail of this line …. The brackets.

The brackets say “Look for this file in the directory where all the standard libraries are stored”. C++ also allows us to specify the library name using double quotes.

#include "main.hpp"

For more information on the standard library visit: [**C++ Standard Library**](http://www.cplusplus.com/reference/)

The double quotes say “look in the current directory, if the file is not there, then look in the directory where the standard libraries are stored”.

If we change the brackets to double quotes, for this case, the program still compiles. We will see later in the course a situation where using the correct enclosure (Brackets or quotes) makes a difference.

The next sections is the main section of the program. We will discuss this in more detail in the next concept.

int main()

{

std::cout << "Hello world, I am ready for C++";

return 0;

}

**Comments can be added in two ways:**

* **As a comment block**
* **As a single line**

In the next node you will see the prompt for a "Hello World" program.

The prompt is enclosed in the symbols: '/\*' and '\*/'. This is how we signify comment block. For example:

/\*The start of a comment block.

Everything between "/\* "and " \*/ "

is in the comment block.

The end of a comment block.\*/

For this course we are adding asterisks for each line of the comment block. It is not necessary, but we think it draws your attention to the comment block.

For example:

/\*write a C++ program that outputs the following statement:

\*\*\* "Hello world, I am ready for C++"

\*/

A comment can be added as a single line by preceding the comment with two slash marks.

For example the code snippet below has two comments. Each comment begins with two slash marks. :

int main()

{

int year = 0;

int age = 0;

std::string name = " ";

//print a message to the user <- this is a comment

std::cout<<"What year is your favorite? ";

//get the user response and assign it to the year<- this line is a comment

std::cin >> year;

...

As with all programming languages, there are many different ways to style the code we write. For this course, we are not using a standard style. We are trying to remain consistent throughout the course, while making the codes as readable as possible.

There are a number of style guides available, **the best one is the one used by the people who are paying you**.

A straightforward style guide is:

[**Modern C++ Coding Guidelines**](https://github.com/Microsoft/AirSim/blob/master/docs/coding_guidelines.md)

For a more detailed guideline:

[**Google C++ Style Guideline**](https://google.github.io/styleguide/cppguide.html)

**Programming Quiz Checker**

Now the question you are asking is … what does it mean to get it right? What did I get right?

When you click on the submit button, the program is compiled and run.

The command to compile is :

g++ main.cpp -o main.out

G++ for the C++ compiler. The file name is main.cpp. -o is for the name of the output. Main.out is the name of the output.

To run the output file, main.out.

We run:

./main.out

The grading program saves the output from main.out in a file. Then it checks this output against what was asked. If they match, then your quiz is graded correctly. Otherwise, the quiz is graded incorrectly.

So, in this case, we are not checking that you have written the code correctly, just that the output is correct.

In later quizzes the grader behaves in a slightly different manner, but I will explain it then.

The programming quiz grader uses a standard compile and link command for C++.

You can use these same commands to compile and link and run you the code on your machine, outside of the Udacity classroom.

There is an optional lesson on how to do this in more detail.

We wrote this in our last program:

std::cout<<"Hello World";

Writing std:: can be a pain. So C++ actually offers a shortcut for writing cout.

Before the start of the main function, put in the command “using namespace std;”

using namespace std;

int main()

{

}

This tells the compiler to assume we are using the standard library, so we don’t have to write std::.

**NOTE: I will add there is some controversy about using namespace.**

When the commands are not explicitly defined, there is a possibility that when your code is added to a large project, your code might reference a command from a different library.

In this class, its use is up to you. Sometimes I will use namespace and sometimes I will not.

/\*Use the namespace keyword to simplify typing\*/

#include <iostream>

using namespace std;

int main()

{

cout << "Hey, writing std:: is pain, ";

cout << "change the program so I don't have to write it.";

return 0;

}

As you may have noticed, we used “cout” to write to the console.

Any string literal must be enclosed in double quotes. Numbers do not require double quotes. If a variable name is used, the value of the variable is printed.

We can have multiple insertions to cout.

A program snippet:

int integer = 4543;

std::cout<<”The value of integer is “<<integer;

Output: The value of integer is 4543

Also, note that cout does not automatically add newlines. You must add them using the escape sequence **“\n”** as shown below:

int integer = 4543;

std::cout<<”The value of integer is “<<integer<<”\n”;

For this programming assignment, you will be printing the size of different variables. As with other programming languages, the size a variable is allocated in memory is dependent upon its type. To determine how many bytes each variable type uses, C++ provides the function sizeof(variableType).

You will use the command :

sizeof(variable type) ie: sizeof(int)

When printing out the variables, you may want to print them out, one per line.

To do this, add "\n" or "endl" to the end of the text to be printed.

For example:

This is the new line character.

cout<<"int size = "<<sizeof(int)<<"\n";

A second option is:

"endl"

cout<<"int size = "<<sizeof(int)<<endl;

In many instances endl and "\n" will perform the same function. There are cases where they will be different.

You can read a quick discussion about their differences here:

[**newline and endline differences**](http://stackoverflow.com/questions/7324843/why-use-endl-when-i-can-use-a-newline-character)

Go here to get more information about [**sizeof**](https://www.tutorialspoint.com/cplusplus/cpp_sizeof_operator.htm)

/\*GOAL: Practice writing to the console and learn

\*\*the variables types available in C++

\*\*Print the sizes of each variable to the console.

\*\*Print them in the following order:

\*\*int, short, long, char, float, double, bool

\*/

#include <iostream>

int main()

{

using namespace std;

cout<<"int size = "<<sizeof(int)<<"\n";

cout<<"short size = "<<sizeof(short)<<"\n";

cout<<"long size = "<<sizeof(long)<<"\n";

cout<<"char size = "<<sizeof(char)<<"\n";

cout<<"float size = "<<sizeof(float)<<"\n";

cout<<"double size = "<<sizeof(double)<<"\n";

cout<<"bool size = "<<sizeof(bool)<<"\n";

return 0;

}

In C++ we can define a variable as a constant. Meaning, its value does not change for the life of the program.

We use the keyword 'const' to define a constant.

For example:

const int weightGoal = 100;

With this statement we have set the integer weightGoal to 100. It cannot be changed during the program. If you want to change the value of weightGoal, you will have to edit the source code and recompile it.

C++ also allows for enumerated constants. This means the programmer can create a new variable type and then assign a finite number of values to it. Here is the form of the enum keyword:

enum type\_name {

value1,

value2,

value3,

.

.

} object\_names;

For example:

enum MONTH {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

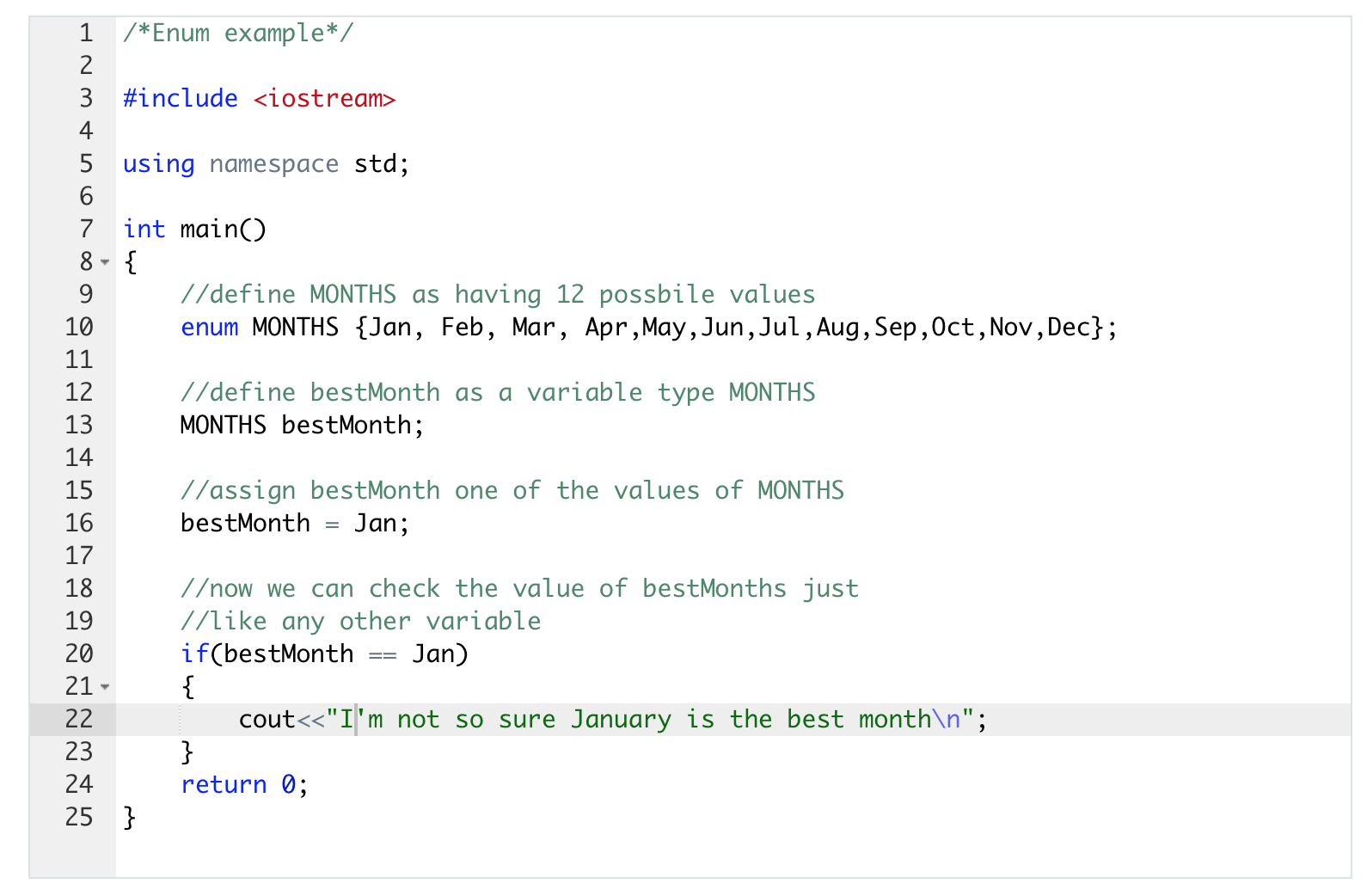
In this example the enum variable MONTH has twelve possible values. These 12 values translate into 12 integer values.

Jan = 0

Feb = 1

etc.

You can read more about enumerated variables at [**C++ Data Types**](http://www.cplusplus.com/doc/tutorial/other_data_types/)



To format data we can use escape sequences. These do not require any additional libraries.

The C++ escape sequences are pretty similar to other languages. The most common ones are: \n - newline \t - tab

We can also format the output by using the iomanip library.

Include it as #include .

Once it is included, you can format output using the iomanip library. For example, we can set the width of an output using the setw command.

#include <iomanip>

std::cout<<"\n\nThe text without any formating\n";

std::cout<<"Ints"<<"Floats"<<"Doubles"<< "\n";

std::cout<<"\nThe text with setw(15)\n";

std::cout<<"Ints"<<std::setw(15)<<"Floats"<<std::setw(15)<<"Doubles"<< "\n";

std::cout<<"\n\nThe text with tabs\n";

std::cout<<"Ints\t"<<"Floats\t"<<"Doubles"<< "\n";

Output will be:

The text without any formating

IntsFloatsDoubles

The text with setw(15)

Ints Floats Doubles

The text with tabs

Ints Floats Doubles

A sample solution to the Format Output Program.

/*Formatting Output Goal: practice using cout to format output to console Print the variables in three columns: \*\*Ints, Floats, Doubles* /

#include <iostream>

#include <iomanip>

int main()

{

int a = 45;

float b = 45.323;

double c = 45.5468;

int aa = a + 9;

float bb = b + 9;

double cc = c + 9;

int aaa = aa + 9;

float bbb = bb + 9;

double ccc = cc + 9;

std::cout<<"print with set width = 10\n";

std::cout<<"Ints"<<std::setw(10);

std::cout<<"Floats"<<std::setw(10);

std::cout<<"Doubles"<<std::setw(10) << "\n";

std::cout<< a;

std::cout<< std::setw(12)<< b;

std::cout<< std::setw(10)<< c << "\n";

std::cout<< aa;

std::cout<< std::setw(12)<< bb;

std::cout<< std::setw(10)<< cc << "\n";

std::cout<< aaa;

std::cout<< std::setw(12)<< bbb;

std::cout<< std::setw(10)<< ccc << "\n\n";

std::cout<<"print with tabs\n";

std::cout<<"Int"<<"\tFloats"<<"\tDoubles\n";

std::cout<< aaa<<"\t"<< bbb<<"\t"<< ccc <<"\n";

return 0;

}

The output for the program:

print with set width = 10

Ints Floats Doubles

45 45.323 45.5468

54 54.323 54.5468

63 63.323 63.5468

print with tabs

Ints Floats Doubles

63 63.323 63.5468

As with other programming languages, we can read and write files.

File IO Steps:

- Include the <fstream> library

- Create a stream (input, output, both)

- ofstream myfile; (for writing to a file)

- ifstream myfile; (for reading a file)

- fstream myfile; (for reading and writing a file)

- Open the file myfile.open(“filename”);

- Write or read the file

- Close the file myfile.close();

In the next quiz you are going to see the output stream and the input stream in action. Then I want you to play with this program by adding text to the input.txt file and then changing the ifstream and ofstream commands to fstream . Make sure that the text you write is read by the program.

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main () {

string line;

//create an output stream to write to the file

//append the new lines to the end of the file

ofstream myfileI ("input.txt", ios::app);

if (myfileI.is\_open())

{

myfileI << "\nI am adding a line.\n";

myfileI << "I am adding another line.\n";

myfileI.close();

}

else cout << "Unable to open file for writing";

//create an input stream to read the file

ifstream myfileO ("input.txt");

//During the creation of ifstream, the file is opened.

//So we do not have explicitly open the file.

if (myfileO.is\_open())

{

while ( getline (myfileO,line) )

{

cout << line << '\n';

}

myfileO.close();

}

else cout << "Unable to open file for reading";

return 0;

}

Our program reads and writes to this file. You can look at this file, you will see it has text in it. Our program will first write to this file, then read from it. Normally, if a program writes to a file we can open that file and see the added text in the file. We can’t do that for input.txt. We won’t see the added text in the file.

But we know it is writing to the file because we can see it being read by the program. We can also change input.txt and we will see the program reading the new text.

As we have seen we can include additional libraries in C++, we can also include our own libraries.

Traditionally, these files are called header files and they have an .hpp extension. Although any extension will work.

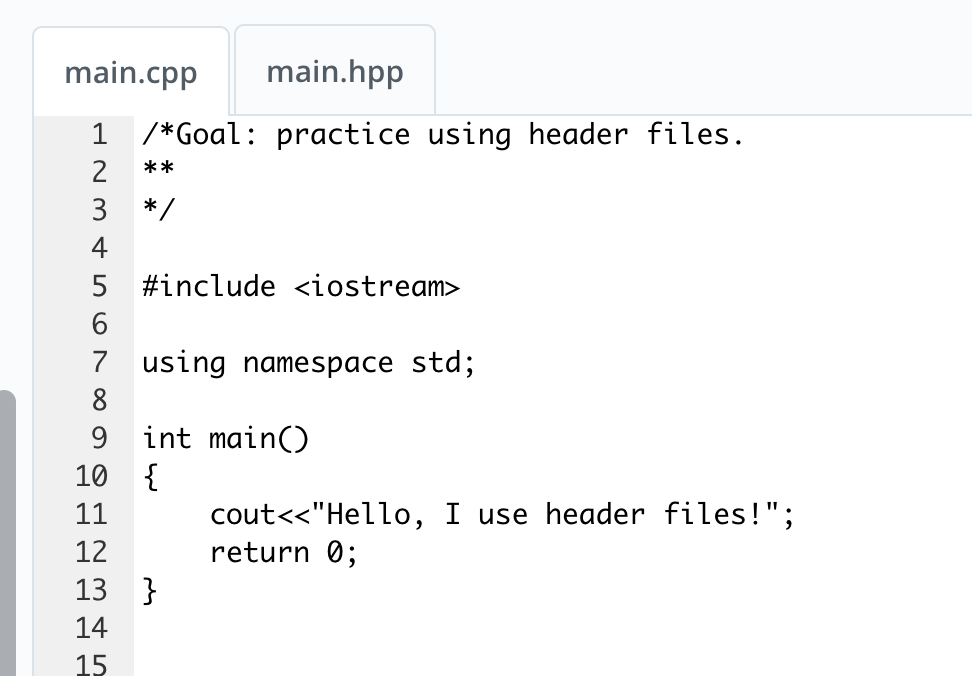
* Header files contain information about how to do a task.
* The main program contains information about what to do.

Let’s see how a header file works with a simple hello world program.

We have a simple hello world program. We can test this, and the program runs.

Below I have included a screenshot of the programming quiz interface. Note that there is a main.cpp and a main.hpp.

* main.cpp: all the code on **what** the program does goes in this file.
* main.hpp: all the code on **how** the program does a task goes in this file.



Let’s move out everything that is not directly related to doing the task and let’s put it in the header file.

So we deleted from main.cpp:

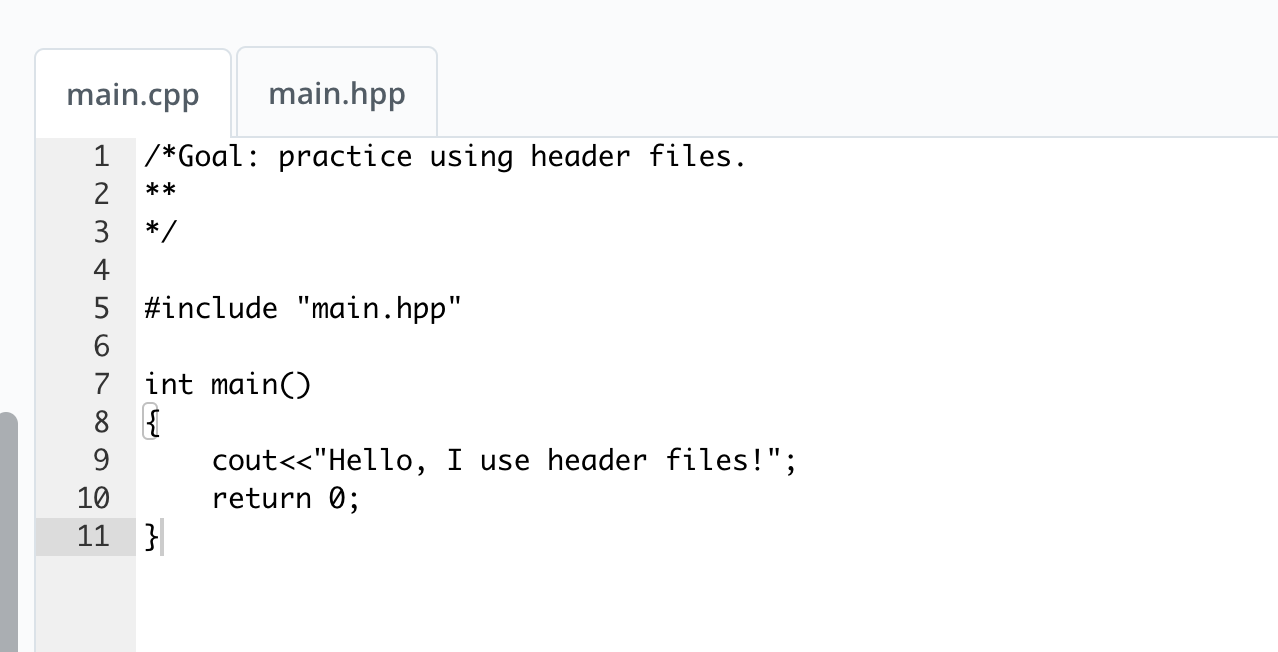
#include <iostream>

using namespace std;

Then added a statement to include main.hpp

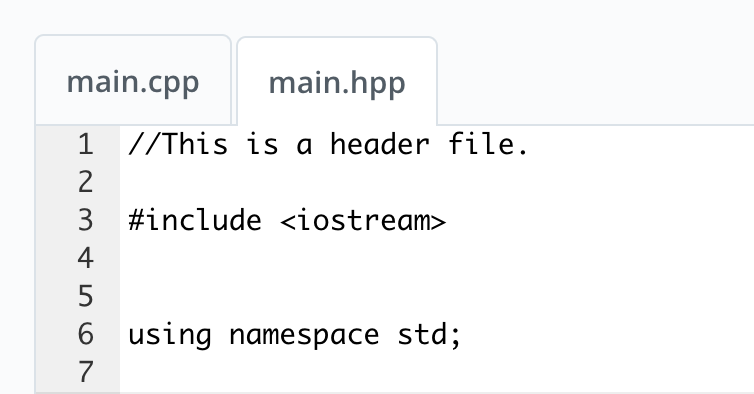
#include "main.hpp"

I have included a screenshot of the modified main.cpp. When you modify your code, make sure you are modifying the correct file, main.cpp. I, for example, have on more than one occasion modified the wrong file.



Then we create a header file. We moved the include and the using statements from the main.ccp and put them in the header file.

Here's a screenshot of the main.hpp file. Again, make sure you are modifying the correct file.



Move the include statement and the using namespace statement to the header file. Now we have to tell the compiler to include the header file. So we add the line #include main.hpp to the main.cpp file.

#include "main.hpp"

**Please note … I’m using double quotes here.**

If you use the angle brackets you will get a compile error!!

We can run this program, and it works just as if all the header information were still in the main.cpp program.