C++ Input/ Output

C++ Input/Output

There's a satisfying feeling when you program code that asks the user for input, before using that input to produce something that the user can see. Even if it's simply asking for someone's name, and displaying a personal welcome message, it's a big leap forward.

User interaction, character literals, defining constants and file input and output are all covered in the following pages. All of which help you to understand more thoroughly how a C++ program works, and enables you to add that personal touch with improved user interaction in your own code.

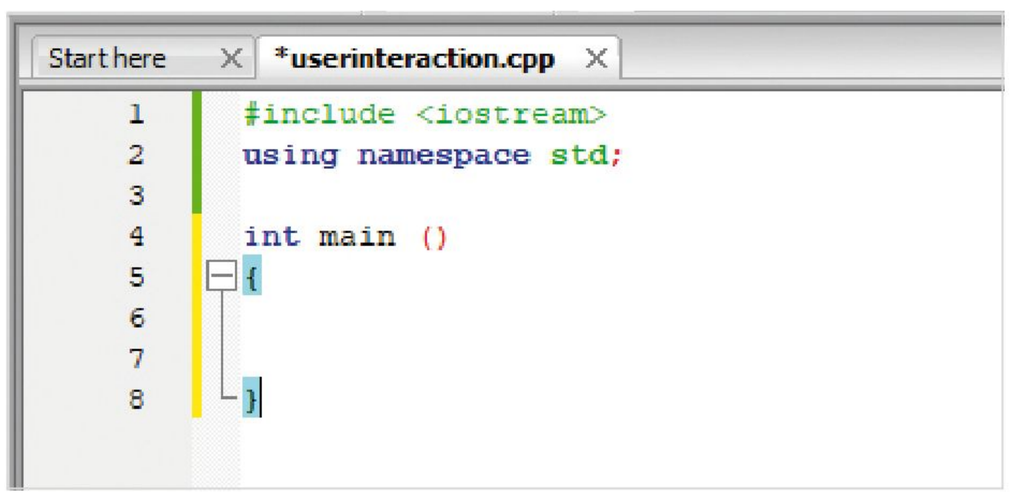
User Interaction

There's nothing quite as satisfying as creating a program that responds to you. This basic user interaction is one of the most taught aspects of any language and with it you're able to do much more than simply greet the user by name.

HELLO, DAVE

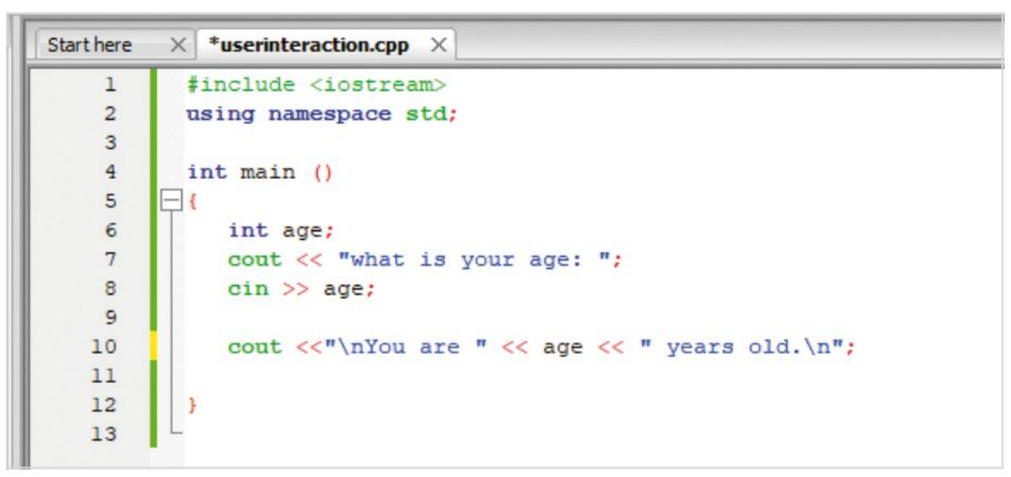
You have already used cout, the standard output stream, throughout our code. Now you're going to be using cin, the standard input stream, to prompt a user response.

STEP 1 Anything that you want the user to input into the program needs to be stored somewhere in the system memory, so it can be retrieved and used. Therefore, any input must first be declared as a variable, so it's ready to be used by the user. Start by creating a blank C++ file with headers.

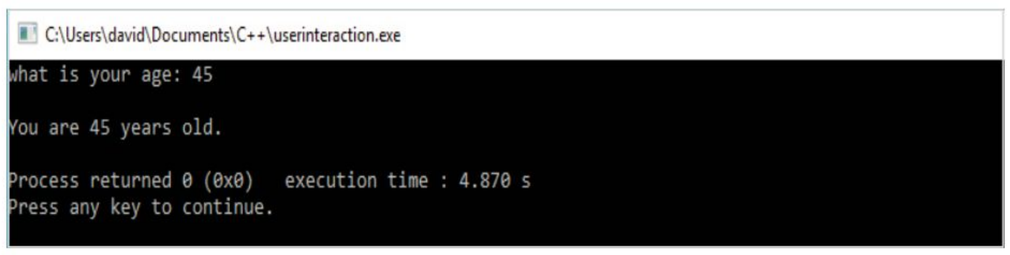


STEP 2 The data type of the variable must also match the type of input you want from the user. For example, to ask a user their age, you would use an integer like this:

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

int main ()
{
   int age;
   cout << "what is your age: ";
   cin >> age;

   cout <<"\nYou are " << age << " years old.\n";

}
```

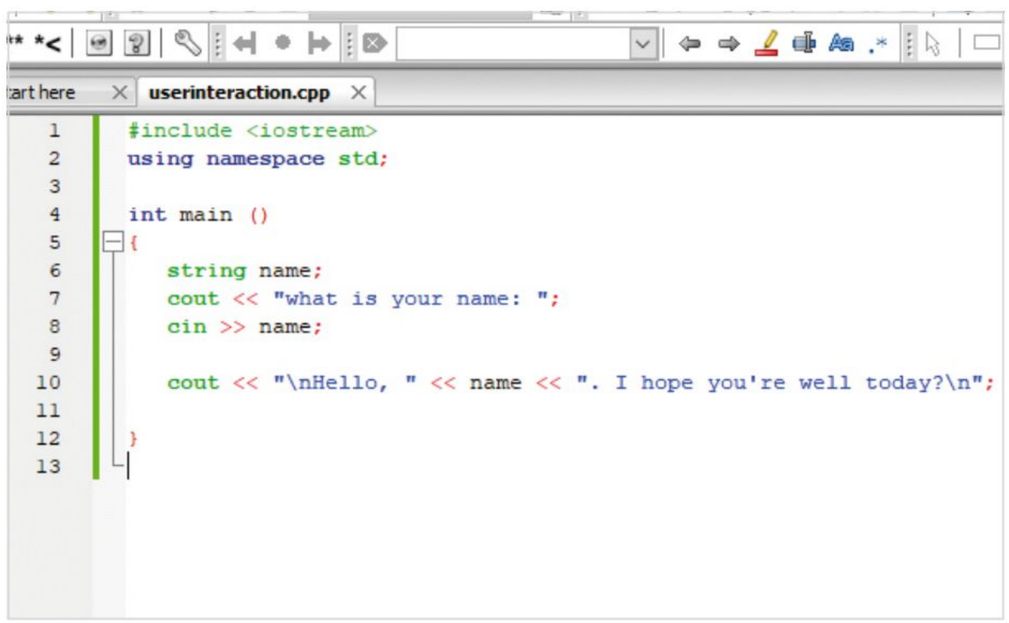


STEP 3 The cin command works in the opposite way from the cout command. With the first cout line you're outputting 'What is your age' to the screen, as indicated with the chevrons. Cin uses opposite facing chevrons, indicating an input. The input is put into the integer age and called up in the second cout command. Build and run the code.

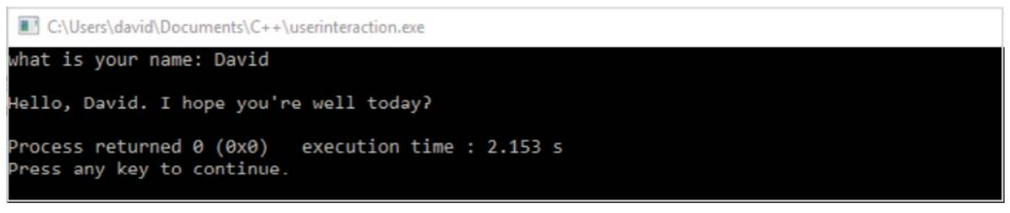


STEP 4 If you're asking a question, you need to store the input as a string; to ask the user their name, you would use:

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

int main ()
{
   string name;
   cout << "what is your name: ";
   cin >> name;

   cout << "\nHello, " << name << ". I hope you're well today?\n";

}
```

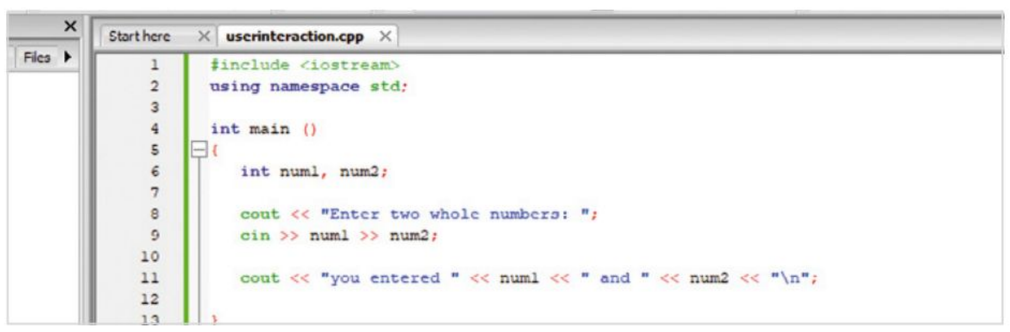


STEP 5 The principal works the same as the previous code. The user's input, their name, is stored in a string, because it contains multiple characters, and retrieved in the second cout line. As long as the variable 'name' doesn't change, then you can recall it wherever you like in your code.



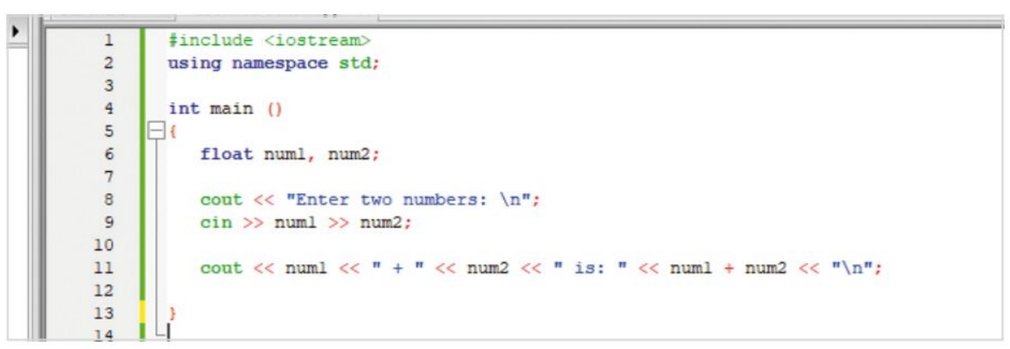
STEP 6 You can chain input requests to the user but just make sure you have a valid variable to store the input to begin with. Let's assume you want the user to enter two whole numbers:

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

int main ()
{
   int num1, num2;

   cout << "Enter two whole numbers: ";
   cin >> num1 >> num2;

   cout << "you entered " << num1 << " and " << num2 << "\n";

}
```



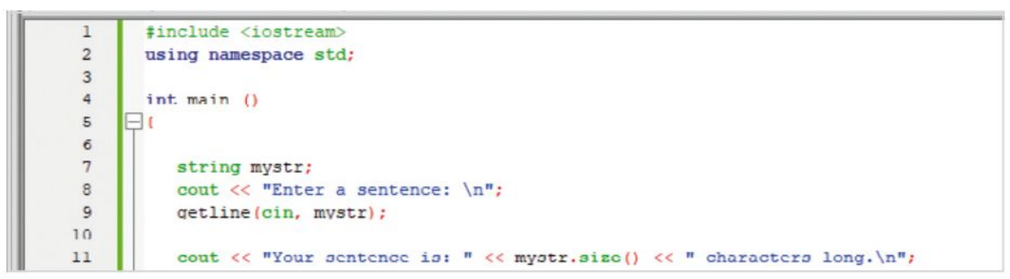
STEP 7 Likewise, inputted data can be manipulated once you have it stored in a variable. For instance, ask the user for two numbers and do some maths on them:

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

int main ()
{
   float num1, num2;

   cout << "Enter two numbers: \n";
   cin >> num1 >> num2;

   cout << num1 << " + " << num2 << " is: " <<
num1 + num2 << "\n";

}
```

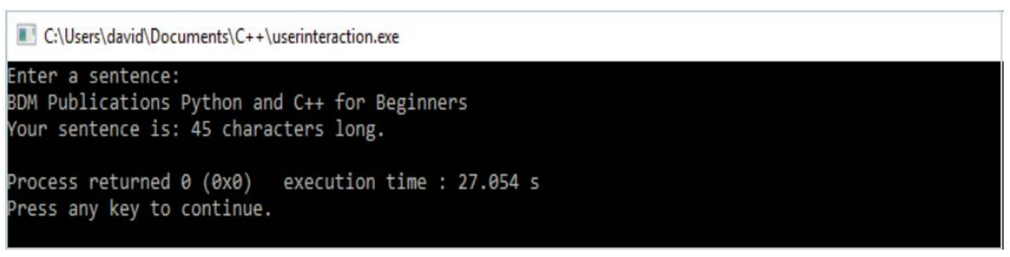


STEP 8 While cin works well for most input tasks, it does have a limitation. Cin always considers spaces as a terminator, so it's designed for just single words not multiple words. However, getline takes cin as the first argument and the variable as the second:

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

int main ()
{

   string mystr;
   cout << "Enter a sentence: \n";
   getline(cin, mystr);

   cout << "Your sentence is: " << mystr.size() <<
" characters long.\n";

}
```

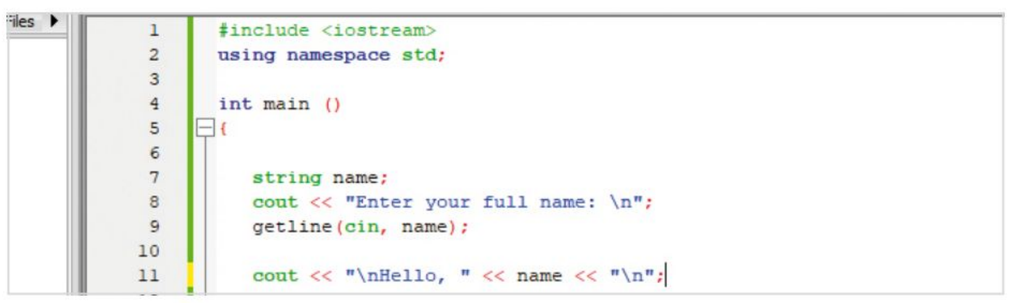


STEP 9 Build and execute the code, then enter a sentence with spaces. When you're done the code reads the number of characters. If you remove the getline line and replace it with cin >> mystr and try again, the result displays the number of characters up to the first space.



STEP 10 Getline is usually a command that new C++ programmers forget to include. The terminating white space is annoying when you can't figure out why your code isn't working. In short, it's best to use getline(cin, variable) in future:

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

int main ()
{

   string name;
   cout << "Enter your full name: \n";
   getline(cin, name);

   cout << "\nHello, " << name << "\n";

}
```



Character Literals

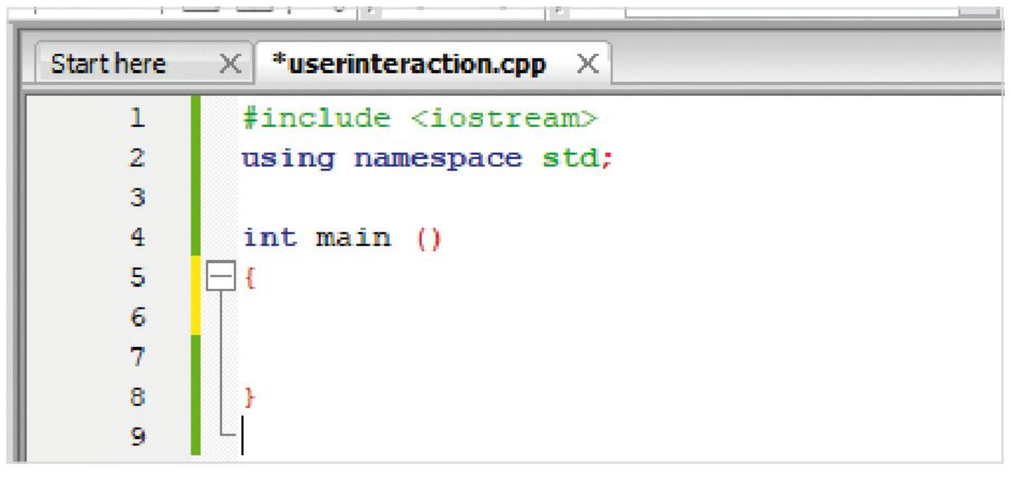
In C++ a literal is an object or variable that once defined remains the same throughout the code. However, a character literal is defined by a backslash, such as the \n you've been using at the end of a cout statement to signify a new line.

ESCAPE SEQUENCE

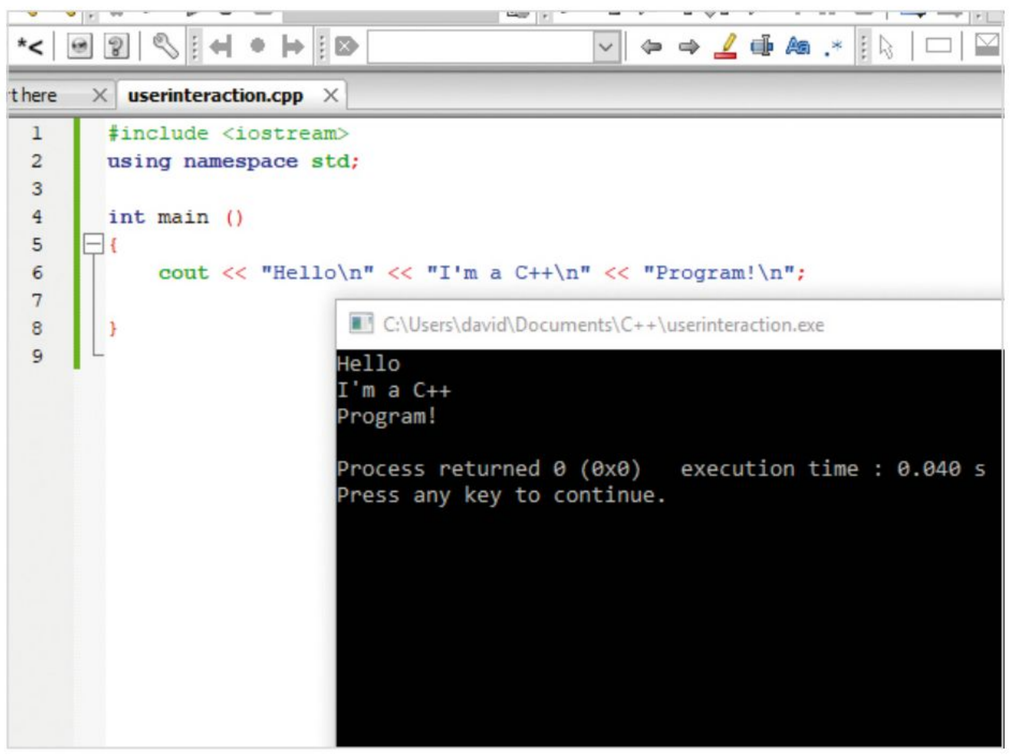
When used in something like a cout statement, character literals are also called escape sequence codes. They allow you to insert a quote, an alert, new line and much more.

STEP 1 Create a new C++ file and enter the relevant headers:

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

int main ()
{

}
```

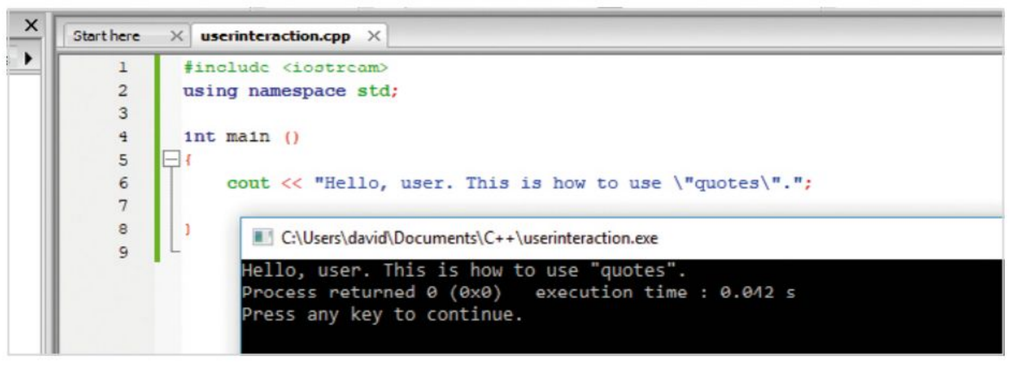


STEP 2 You've already experienced the \n character literal placing a new line wherever it's called. The line: **cout << "Hello\n" << "I'm a C++\n" << "Program\n";** outputs three lines of text, each starting after the last \n.



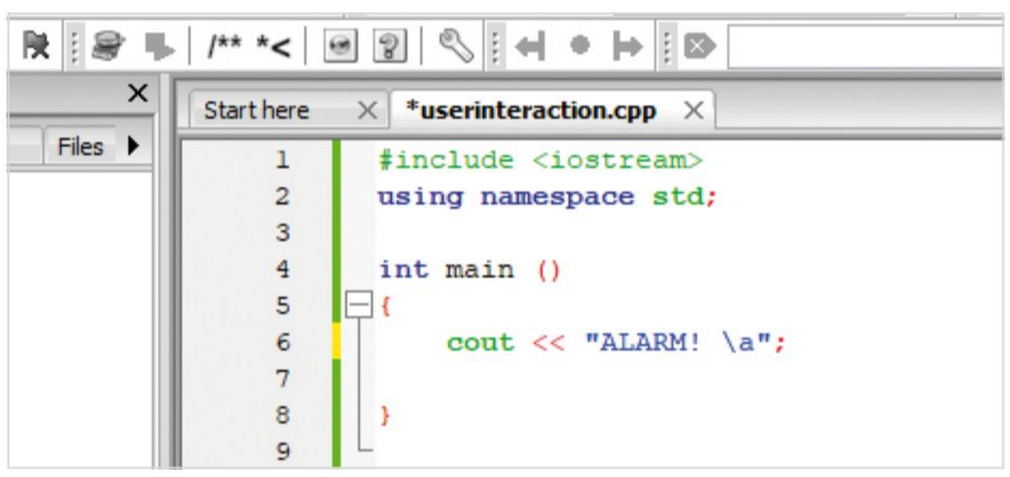
STEP 3 If you wanted to insert speech quotes inside a cout statement, you would have to use a backslash as it already uses quotes:

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

int main ()
{
    cout << "Hello, user. This is how to use
\"quotes\".";

}
```



STEP 4 There's even a character literal that can trigger an alarm. In Windows 10, it's the notification sound that chimes when you use \a. Try this code, and turn up your sound.

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

int main ()
{
    cout << "ALARM! \a";

}
```



A HANDY CHART

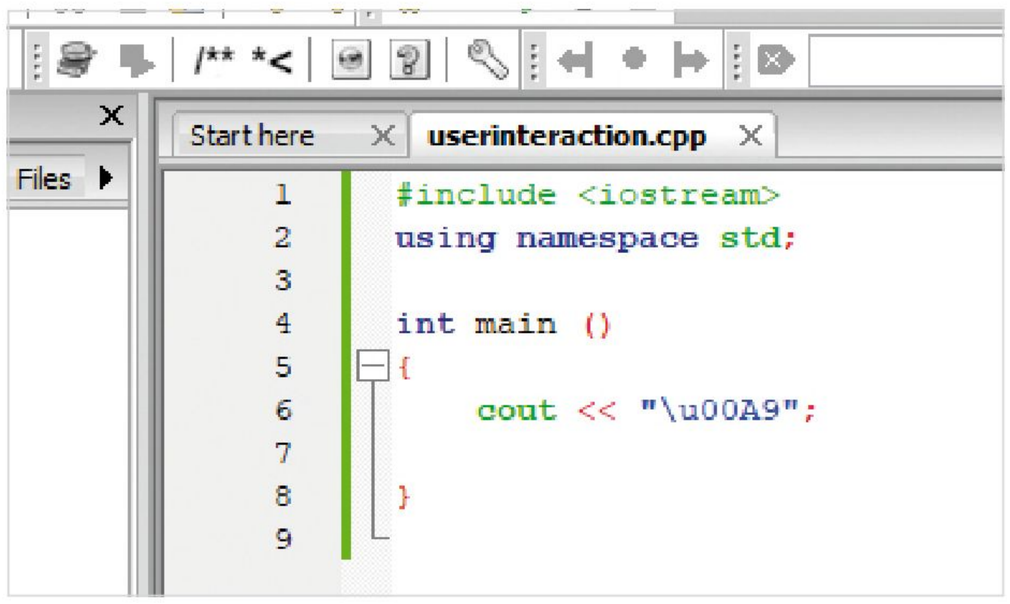
There are numerous character literals, or escape sequence codes, to choose from. We therefore thought it would be good for you to have a handy chart available, for those times when you need to insert a code.

ESCAPE SEQUENCE CODE	CHARACTER
\\	Backslash
\’	Single Quote
\”	Double Quote (Speech Marks)
\?	Question Mark
\a	Alert/Alarm
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab
\0	Null Character
\uxxxx	Unicode (UTF-8)
\Uxxxxxxxx	Unicode (UTF-16)

UNICODE CHARACTERS (UTF-8)

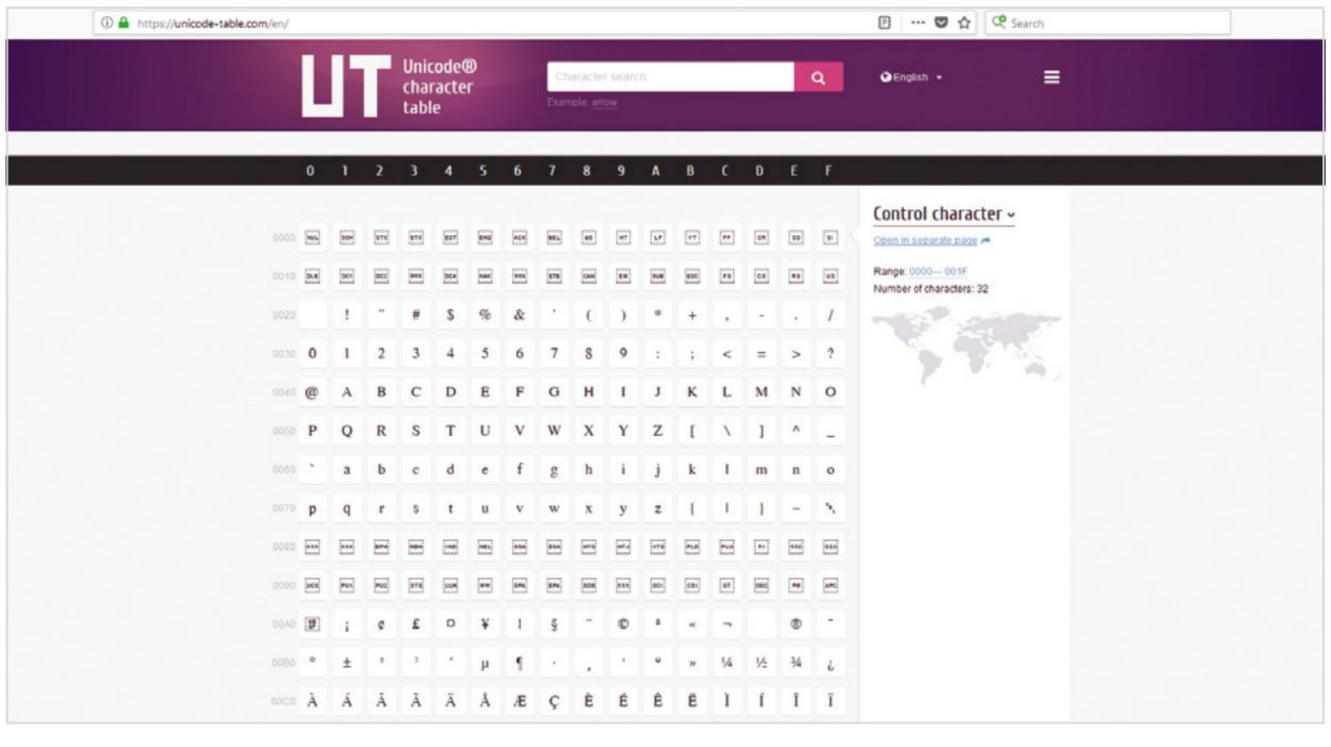
Unicode characters are symbols or characters that are standard across all platforms. For example, the copyright symbol, that can be entered via the keyboard by entering the Unicode code, followed by ALT+X. In the case of the copyright symbol enter: 00A9 Alt+X. In C++ code, you would enter:

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

int main ()
{
    cout << “\u00A9”;
```



UNICODE CHARACTER TABLE

A complete list of the available Unicode characters can be found at **www.unicode-table.com/en/**. Hover your mouse over the character to see the unique code to enter in C++. While it may be a little overwhelming to look at to begin with, bookmark the page as you will probably need to come back to it for reference as you dig deeper into C++, and indeed character literals. One more thing, the table will also display characters from different languages, such as Tibetan or Sudanese. This means your code can be truly universal.



Defining Constants

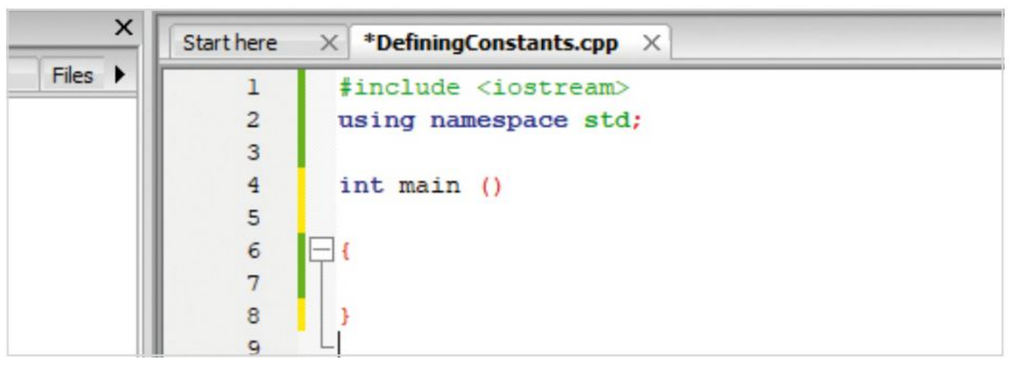
Constants are fixed values in your code. They can be any basic data type but as the name suggests their value remains constant throughout the entire code. There are two separate ways to define a constant in C++, the #define pre-processor and const.

#DEFINE

The pre-processors are instructions to the compiler to pre-process the information before it goes ahead and compiles the code. #include is a pre-processor as is #define.

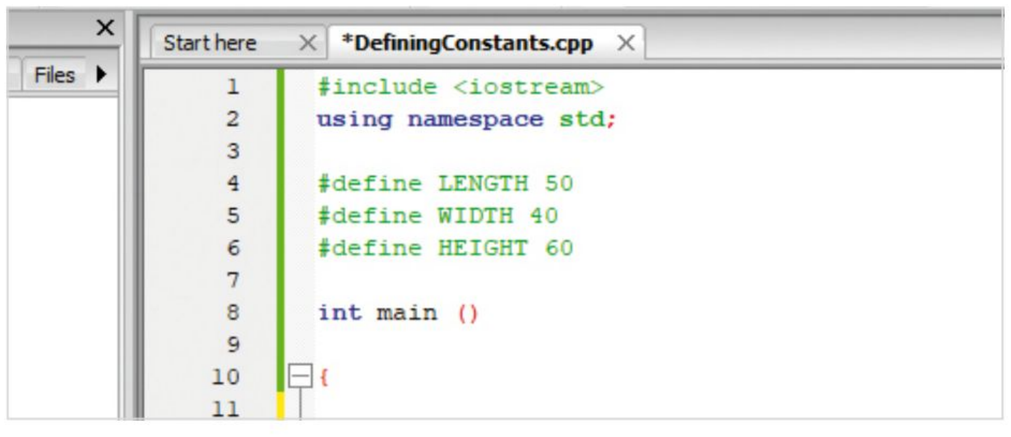
STEP 1 You can use the #define pre-processor to define any constants you want in our code. Start by creating a new C++ file complete with the usual headers:

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

int main ()

{

}
```



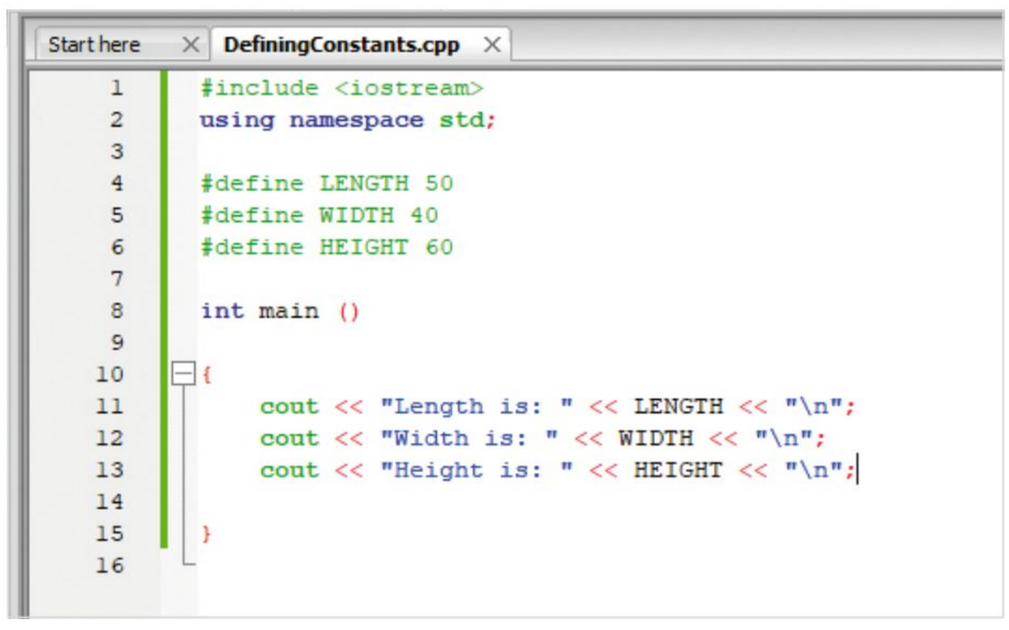
STEP 2 Now let's assume your code has three different constants: length, width and height. You can define them with:

```
#include <iostream>
using namespace std;
#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{

}
```

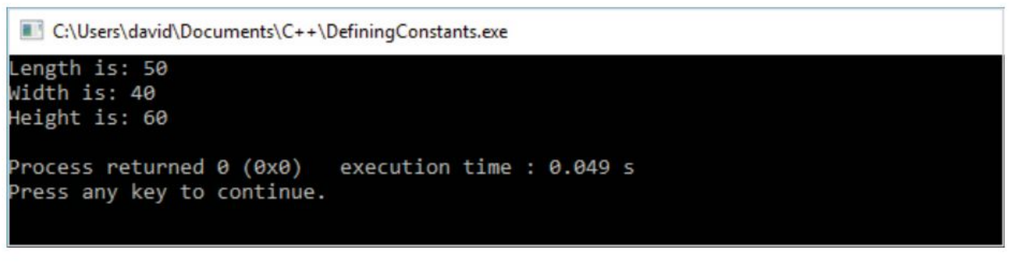


STEP 3 Note the capitals for defined constants, it's considered good programming practise to define all constants in capitals. Here, the assigned values are 50, 40 and 60, so let's call them up:

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

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{
    cout << “Length is: “ << LENGTH << “\n”;
    cout << “Width is: “ << WIDTH << “\n”;
    cout << “Height is: “ << HEIGHT << “\n”;
}
```



STEP 4 Build and run the code. Just as expected, it displays the values for each of the constants created. It's worth noting that you don't need a semicolon when you're defining a constant with the #define keyword.



STEP 5 You can also define other elements as a constant. For example, instead of using \n for a newline in the cout statement, you can define it at the start of the code:

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

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60
#define NEWLINE ‘\n’

int main ()

{
    cout << “Length is: “ << LENGTH << NEWLINE;
    cout << “Width is: “ << WIDTH << NEWLINE;
    cout << “Height is: “ << HEIGHT << NEWLINE;
}
```

```
1  #include <iostream>
2  using namespace std;
3
4  #define LENGTH 50
5  #define WIDTH 40
6  #define HEIGHT 60
7  #define NEWLINE '\n'
8
9  int main ()
10
```

STEP 6 The code, when built and executed, does exactly the same as before, using the new constant NEWLINE to insert a newline in the cout statement. Incidentally, creating a newline constant isn’t a good idea unless you’re making it smaller than \n or even the endl command.

```
16
C:\Users\david\Documents\C++\DefiningConstants.exe
Length is: 50
Width is: 40
Height is: 60

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

STEP 7 Defining a constant is a good way of initialising your base values at the start of your code. You can define that your game has three lives, or even the value of PI without having to call up the C++ math library:

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

#define PI 3.14159

int main ()

{
    cout << “The value of Pi is: “ << PI << endl;

}
```

```
Files
Start here | DefiningConstants.cpp
1  #include <iostream>
2  using namespace std;
3
4  #define PI 3.14159
5
6  int main ()
7
8  {
9      cout << "The value of Pi is: " << PI << endl;
10
11 }
12
```

STEP 8 Another method of defining a constant is with the const keyword. Use const together with a data type, variable and value: const type variable = value. Using Pi as an example:

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

int main ()

{
    const double PI = 3.14159;
    cout << “The value of Pi is: “ << PI << endl;
}
```

```
Files
Start here | *DefiningConstants.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5
6  {
7      const double PI = 3.14159;
8      cout << "The value of Pi is: " << PI << endl;
9  }
10
```

STEP 9 Because you’re using const within the main block of code, you need to finish the line with a semicolon. You can use either, as long as the names and values don’t clash, but it’s worth mentioning that #define requires no memory, so if you’re coding to a set amount of memory, #define is your best bet.

```
5
6  {
7      const double PI = 3.14159;
8      cout << "The value of Pi is: " << PI << endl;
9  }
10
C:\Users\david\Documents\C++\DefiningConstants.exe
The value of Pi is: 3.14159

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

STEP 10 Const works in much the same way as #define. You can create static integers and even newlines:

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

int main()

{
    const int LENGTH = 50;
    const int WIDTH = 40;
    const char NEWLINE = ‘\n’;

    int area;
    area = LENGTH * WIDTH;

    cout << “Area is: “ << area << NEWLINE;

}
```

```
Files
Start here | DefiningConstants.cpp
1  #include <iostream>
2  using namespace std;
3
4  #define PI 3.14159
5
6  int main ()
7
8  {
9      cout << "The value of Pi is: " << PI << endl;
10
11 }
12
```

```
iles
Start here | *DefiningConstants.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main()
5
6  {
7      const int LENGTH = 50;
8      const int WIDTH = 40;
9      const char NEWLINE = '\n';
10
11     int area;
12     area = LENGTH * WIDTH;
```

File Input/Output

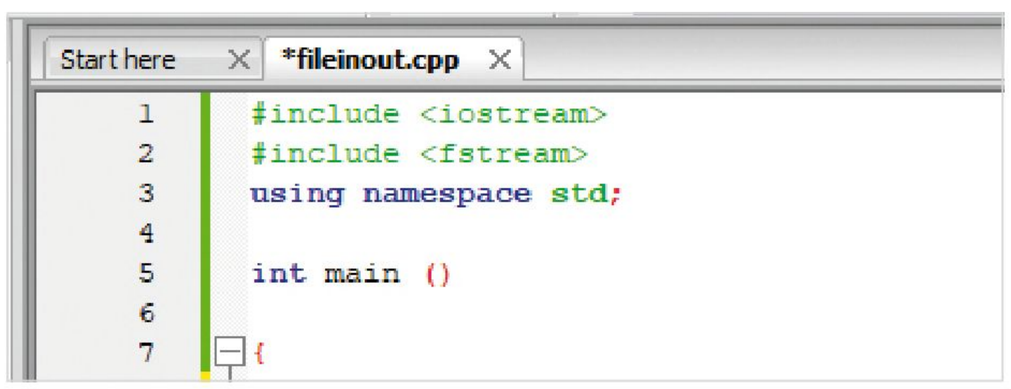
The standard iostream library provides C++ coders with the cin and cout input and output functionality. However, to be able to read and write from a file you need to utilise another C++ library, called fstream.

FSTREAMS

There are two main data types within the fstream library that are used to open a file, read from it and write to it; these are ofstream and ifstream. Here's how they work.

STEP 1 The first task is to create a new C++ file and along with the usual headers you need to include the new fstream header:

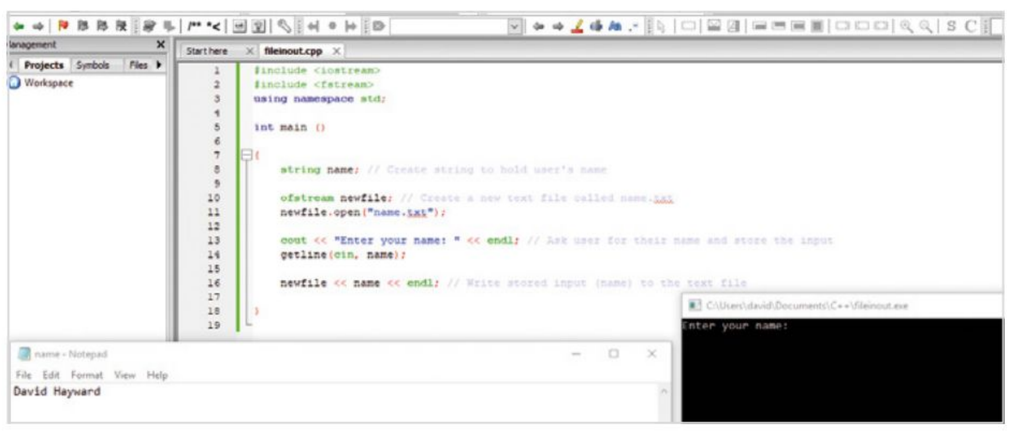
```
#include <iostream>
#include <fstream>
Using namespace std;

int main ()

{

}
```



STEP 2 Begin by asking a user for their name and writing that information to a file. You need the usual string to store the name, and getline to accept the input from the user.

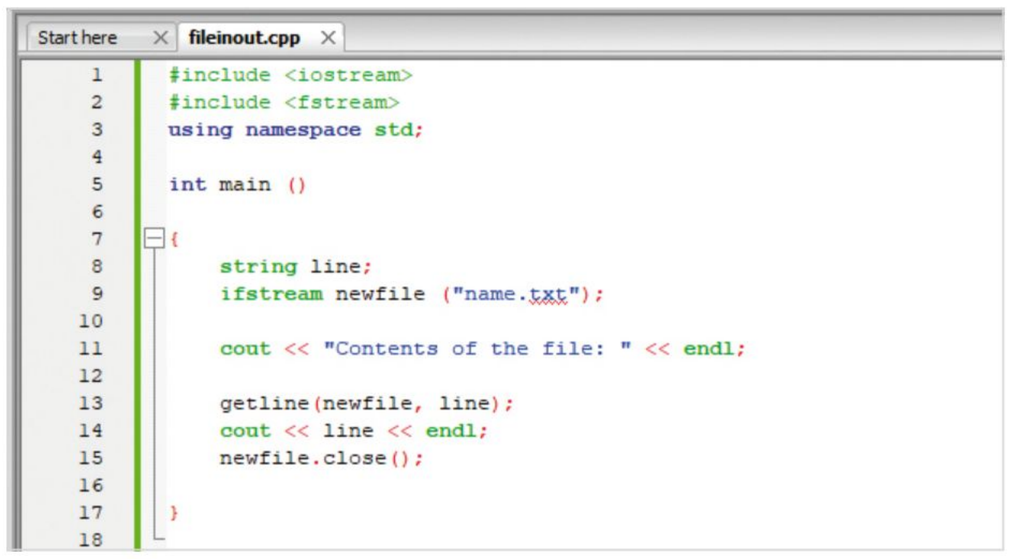
```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string name;

    ofstream newfile;
    newfile.open("name.txt");

    cout << "Enter your name: " << endl;
    getline(cin, name);

    newfile << name << endl;

    newfile.close();

}
```

STEP 3 We've included comments in the screenshot of step 2 to help you understand the process. You created a string called name, to store the user's inputted name. You also created a text file called name.txt (with the ofstream newfile and newfile.open lines), asked the user for their name and stored it and then written the data to the file.



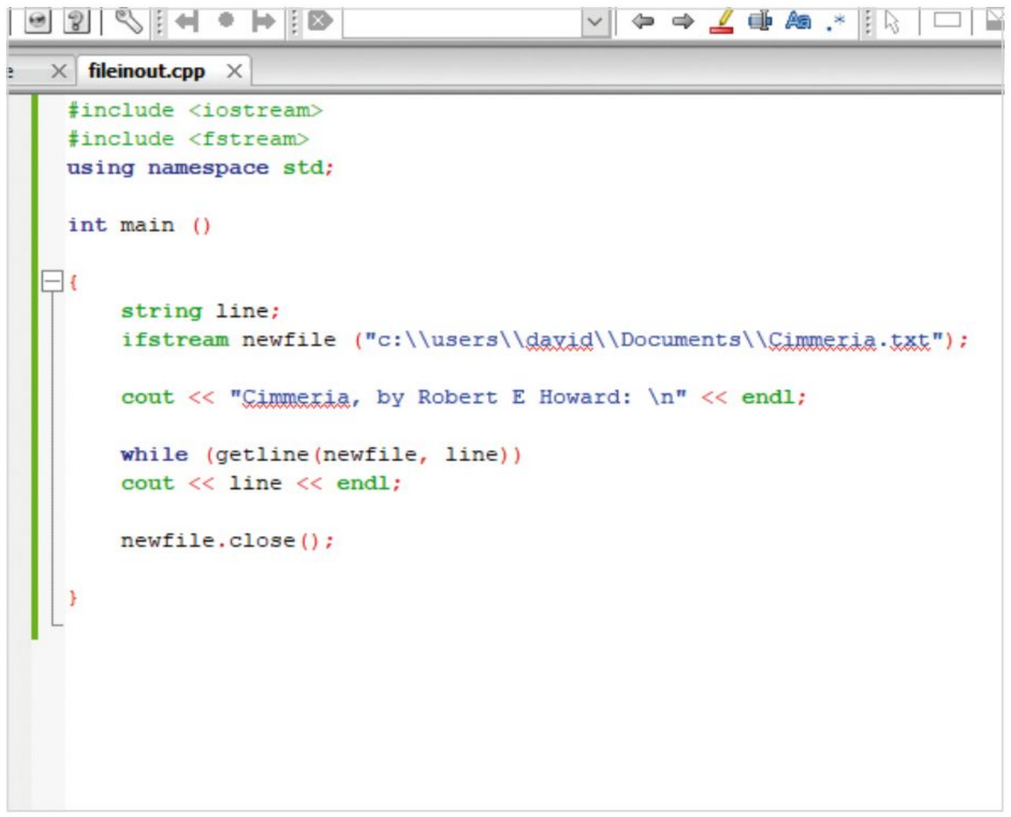
STEP 4 To read the contents of a file, and output it to the screen, you need to do things slightly differently. First you need to create a string variable to store the file's contents (line by line), then open the file, use getline to read the file line by line and output those lines to the screen. Finally, close the file.

```
    string line;
    ifstream newfile ("name.txt");

    cout << "Contents of the file: " << endl;

    getline(newfile, line);
    cout << line << endl;
    newfile.close();
```

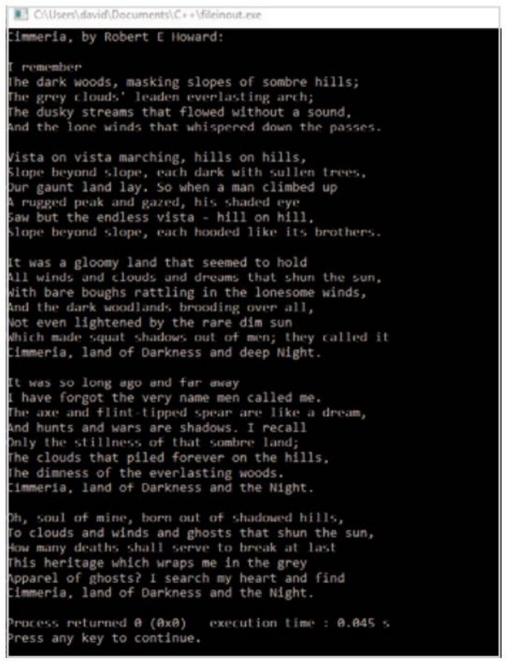


STEP 5 The code above is great for opening a file with one or two lines but what if there are multiple lines? Here we opened a text file of the poem Cimmeria, by Robert E Howard:

```
    string line;
    ifstream newfile ("c:\\users\\david\\
Documents\\Cimmeria.txt");

    cout << "Cimmeria, by Robert E Howard: \n" <<
endl;

    while (getline(newfile, line))
    cout << line << endl;

    newfile.close();
```



STEP 6 You can no doubt see that we've included a while loop, which we cover in a few pages time. It means that while there are lines to be read from the text file, C++ getlines them. Once all the lines are read, the output is displayed on the screen and the file is closed.

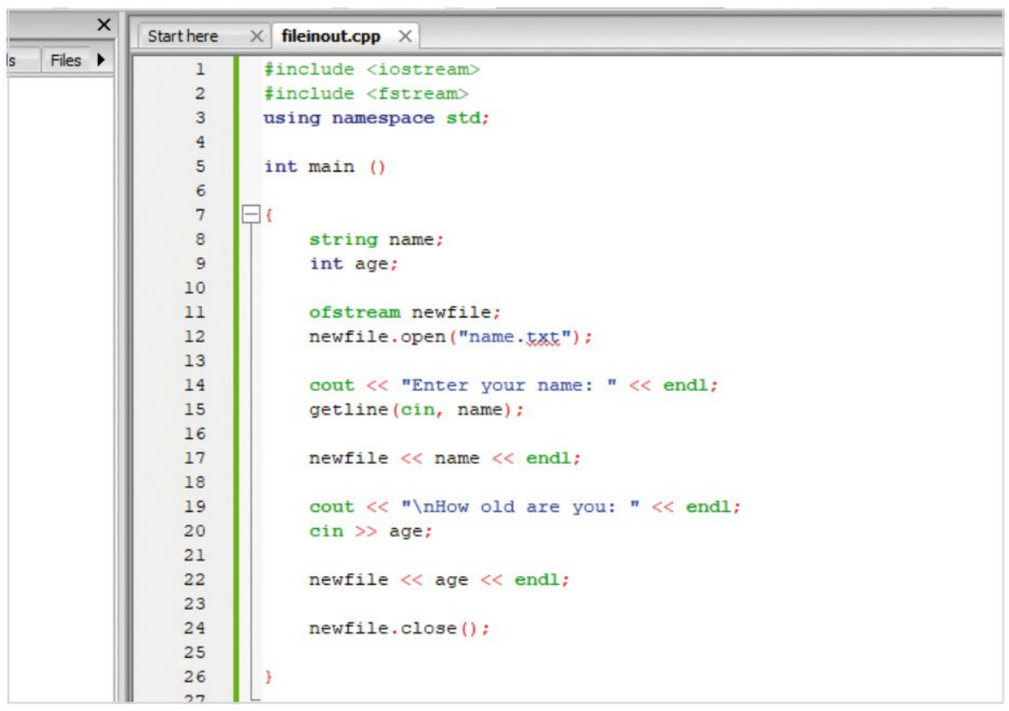


STEP 7 You can also see that the location of the text file Cimmeria.txt isn't in the same folder as the C++ program. When we created the first name.txt file, it was written to the same folder where the code was located; this is done by default. To specify another folder, you need to use double-back slashes, as per the character literals/escape sequence code.

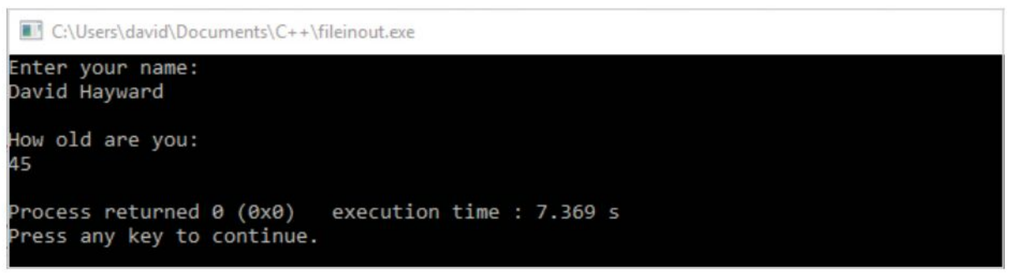
```
6
7   {
8       string line;
9       ifstream newfile ("c:\\users\\david\\Documents\\Cimmeria.txt");
10
11      cout << "Cimmeria, by Robert E Howard: \n" << endl;
12
```

STEP 8 Just as you might expect, you can write almost anything you like to a file, for reading either in Notepad or via the console through the C++ code:

```
    string name;
    int age;

    ofstream newfile;
    newfile.open("name.txt");

    cout << "Enter your name: " << endl;
    getline(cin, name);

    newfile << name << endl;

    cout << "\nHow old are you: " << endl;
    cin >> age;

    newfile << age << endl;

    newfile.close();
```



STEP 9 The code from step 8 differs again, but only where it comes to adding the age integer. Notice that we used cin >> age, instead of the previous getline(cin, variable). The reason for this is that the getline function handles strings, not integers; so when you're using a data type other than a string, use the standard cin.



STEP 10 Here's an exercise: see if you can create code to write several different elements to a text file. You can have a user's name, age, phone number etc. Maybe even the value of Pi and various mathematical elements. It's all good practice.