

# Variables

Variables differ slightly when using C++ as opposed to Python. In Python, you can simply state that 'a' equals 10 and a variable is assigned. However, in C++ a variable has to be declared with its type before it can be used.

## THE DECLARATION OF VARIABLES

You can declare a C++ variable by using statements within the code. There are several distinct types of variables you can declare. Here's how it works.

### STEP 1

Open up a new, blank C++ file and enter the usual code headers:

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

int main()
{}
```

```
Start here X Variables.cpp X
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6 }
```

### STEP 2

Start simple by creating two variables, a and b, with one having a value of 10 and the other 5. You can use the data type int to declare these variables. Within the curly brackets, enter:

```
int a;
int b;
a = 10;
b = 5;
```

```
Start here X *Variables.cpp X
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int a;
7     int b;
8
9     a = 10;
10    b = 5;
11 }
```

### STEP 3

You can build and run the code but it won't do much, other than store the values 10 and 5 to the integers a and b. To output the contents of the variables, add:

```
cout << a;
cout << "\n";
cout << b;
```

The cout << "\n"; part simply places a new line between the output of 10 and 5.

```
Start here X Variables.cpp X
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int a;
7     int b;
8
9     a = 10;
10    b = 5;
11
12    cout << a;
13    cout << "\n";
14    cout << b;
15
16 }
17
18 }
```

### STEP 4

Naturally you can declare a new variable, call it result and output some simple arithmetic:

```
int result;
result = a + b;
cout << result;
```

Insert the above into the code as per the screenshot.

```
Start here X Variables.cpp X
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int a;
7     int b;
8     int result;
9
10    a = 10;
11    b = 5;
12    result = a + b;
13
14    cout << result;
15
16 }
17
18
```

C:\Users\David\Documents\C++\Variables.exe  
15  
Process returned 0 (0x0) execution time : 0.045 s  
Press any key to continue.

**STEP 5**

You can assign a value to a variable as soon as you declare it. The code you've typed in could look like this, instead:

```
int a = 10;
int b = 5;
int result = a + b;

cout << result;
```

**STEP 6**

Specific to C++, you can also use the following to assign values to a variable as soon as you declare them:

```
int a (10);
int b (5);
```

Then, from the C++ 2011 standard, using curly brackets:

```
int result {a+b};
```

**STEP 7**

You can create global variables, which are variables that are declared outside any function and used in any function within the entire code. What you've used so far are local variables: variables used inside the function. For example:

```
#include <iostream>
using namespace std;
int StartLives = 3;

int main ()
{
    startLives = StartLives - 1;
    cout << StartLives;
}
```

**STEP 8**

The previous step creates the variable StartLives, which is a global variable. In a game, for example, a player's lives go up or down depending on how well or how bad they're doing. When the player restarts the game, the StartLives returns to its default state: 3. Here we've assigned 3 lives, then subtracted 1, leaving 2 lives left.

**STEP 9**

The modern C++ compiler is far more intelligent than most programmers give it credit. While there are numerous data types you can declare for variables, you can in fact use the auto feature:

```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()
{
    double area, radius = 1.5;
    area = pi * radius * radius;
    cout << area;
}
```

**STEP 10**

A couple of new elements here: first, auto won't work unless you go to Settings > Compiler and tick the box labelled 'Have G++ follow the C++11 ISO C++ Language Standard [-std=c++11]'. Then, the new data type, double, which means double-precision floating point value. Enable C++11, then build and run the code. The result should be 7.06858.

# Data Types

Variables, as we've seen, store information that the programmer can then later call up, and manipulate if required. Variables are simply reserved memory locations that store the values the programmer assigns, depending on the data type used.

## THE VALUE OF DATA

There are many different data types available for the programmer in C++, such as an integer, floating point, Boolean, character and so on. It's widely accepted that there are seven basic data types, often called Primitive Built-in Types; however, you can create your own data types should the need ever arise within your code.

The seven basic data types are:

TYPE	COMMAND
Integer	Integer
Floating Point	float
Character	char
Boolean	bool
Double Floating Point	double
Wide Character	wchar_t
No Value	void

These basic types can also be extended using the following modifiers: Long, Short, Signed and Unsigned. Basically this means the modifiers can expand the minimum and maximum range values for each data type. For example, the int data type has a default value range of -2147483648 to 2147483647, a fair value, you would agree.

Now, if you were to use one of the modifiers, the range alters:

Unsigned int = 0 to 4294967295

Signed int = -2147483648 to 2147483647

Short int = -32768 to 32767

Unsigned Short int = 0 to 65,535

Signed Short int = -32768 to 32767

Long int = -2147483647 to 2147483647

Signed Long int = -2147483647 to 2147483647

Unsigned Long int = 0 to 4294967295

Naturally you can get away with using the basic type without the modifier, as there's plenty of range provided with each data type. However, it's considered good C++ programming practise to use the modifiers when possible.

There are issues when using the modifiers though. Double represents a double-floating point value, which you can use for

incredibly accurate numbers but those numbers are only accurate up to the fifteenth decimal place. There's also the problem when displaying such numbers in C++ using the cout function, in that cout by default only outputs the first five decimal places. You can combat that by adding a cout.precision () function and adding a value inside the brackets, but even then you're still limited by the accuracy of the double data type. For example, try this code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;
}
```

```
Start here X DataTypes.cpp X
1 #include <iostream>
2 using namespace std;
3 double PI = 3.141592653589793238463;
4
5 int main()
6 {
7     cout << PI;
8 }
9
10
```

```
C:\Users\david\Documents\C++\DataTypes.exe
3.14159
Process returned 0 (0x0) execution time : 0.054 s
Press any key to continue.
```

Build and run the code and as you can see the output is only 3.14159, representing cout's limitations in this example.

You can alter the code including the aforementioned cout.precision function, for greater accuracy. Take precision all the way up to 22 decimal places, with the following code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
```

```
cout.precision(22);
cout << PI;
}

Start here X DataTypes.cpp X

1 #include <iostream>
2 using namespace std;
3 double PI = 3.14159265358979323846;
4
5 int main()
6 {
7     cout.precision(22);
8     cout << PI;
9
10 }
11
```

```
C:\Users\david\Documents\C++\DataTypes.exe
3.141592653589793115998
Process returned 0 (0x0)    execution time : 0.047 s
Press any key to continue.
```

Again, build and run the code; as you can see from the command line window, the number represented by the variable PI is different to the number you've told C++ to use in the variable. The output reads the value of PI as 3.141592653589793115998, with the numbers going awry from the fifteenth decimal place.

This is mainly due to the conversion from binary in the compiler and that the IEEE 754 double precision standard occupies 64-bits of data, of which 52-bits are dedicated to the significant (the significant digits in a floating-point number) and roughly 3.5-bits are taken holding the values 0 to 9. If you divide 53 by 3.5, then you arrive at 15.142857 recurring, which is 15-digits of precision.

To be honest, if you're creating code that needs to be accurate to more than fifteen decimal places, then you wouldn't be using C++, you would use some scientific specific language with C++ as the connective tissue between the two languages.

You can create your own data types, using an alias-like system called `typedef`. For example:

The screenshot shows a C++ development environment with the following details:

- Toolbar:** Includes standard icons for file operations (New, Open, Save, Print), project management (Build, Run, Stop), and code navigation.
- Tab Bar:** Shows the current file is "DataTypes.cpp".
- Code Editor:** Displays the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3 typedef int metres;
4
5 int main()
6 {
7     metres distance;
8     distance = 15;
9     cout << "distance in metres is: " << distance;
10
11 }
```

The code defines a type alias `metres` for `int`. It then declares a variable `distance` of type `metres`, initializes it to 15, and prints its value using `cout`.

```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << "distance in metres is: " << distance;
}
```

```
C:\Users\david\Documents\C++\ DataTypes.exe
distance in metres is: 15
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

This code when executed creates a new int data type called metres. Then, in the main code block, there's a new variable called distance, which is an integer; so you're basically telling the compiler that there's another name for int. We assigned the value 15 to distance and displayed the output: distance in metres is 15.

It might sound a little confusing to begin with but the more you use C++ and create your own code, the easier it becomes.



# Strings

Strings are objects that represent and hold sequences of characters. For example, you could have a universal greeting in your code 'Welcome' and assign that as a string to be called up wherever you like in the program.

## STRING THEORY

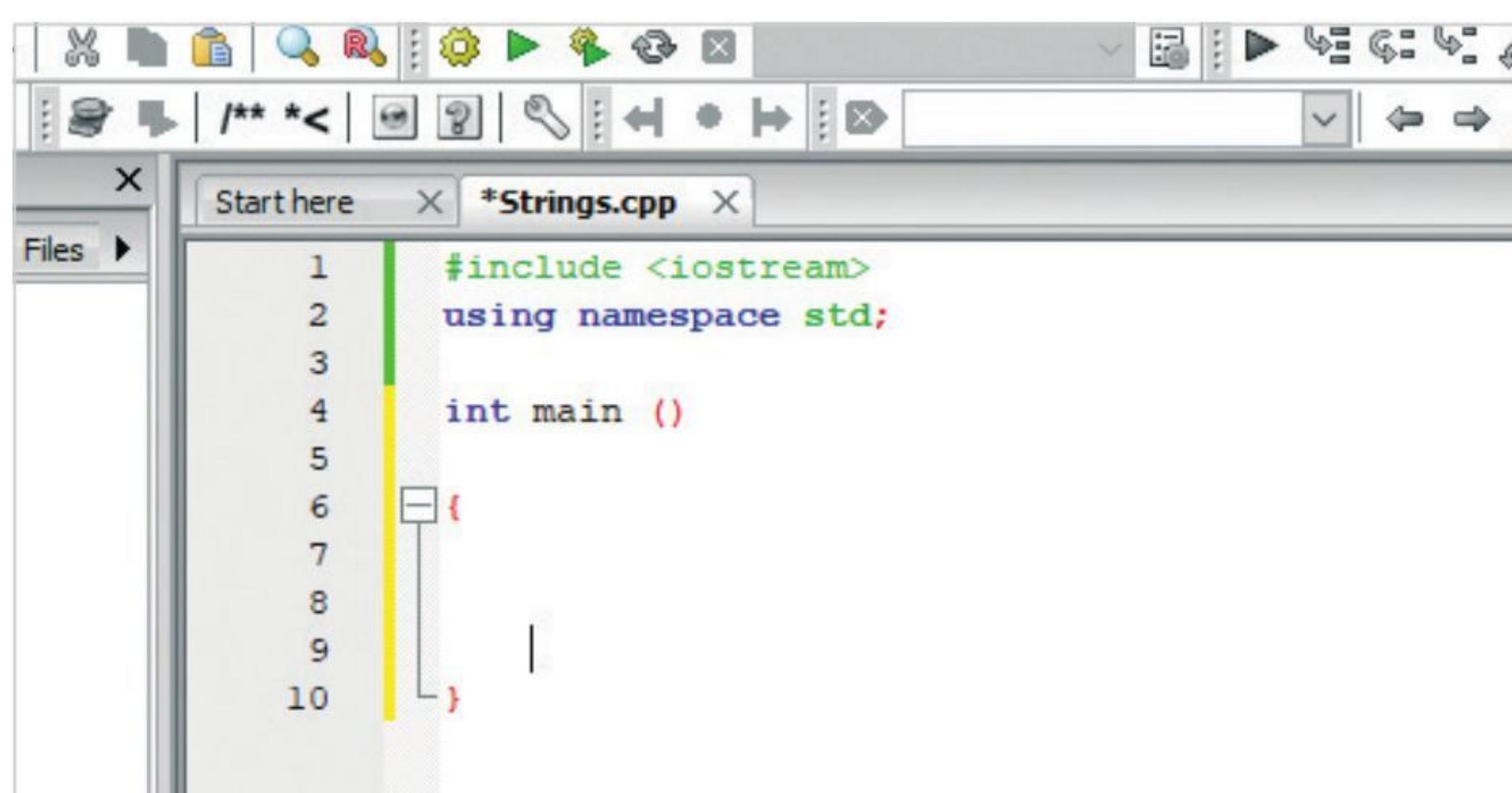
There are different ways in which you can create a string of characters, which historically are all carried over from the original C language, and are still supported by C++.

### STEP 1

To create a string you use the `char` function. Open a new C++ file and begin with the usual header:

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

int main ()
{
}
```



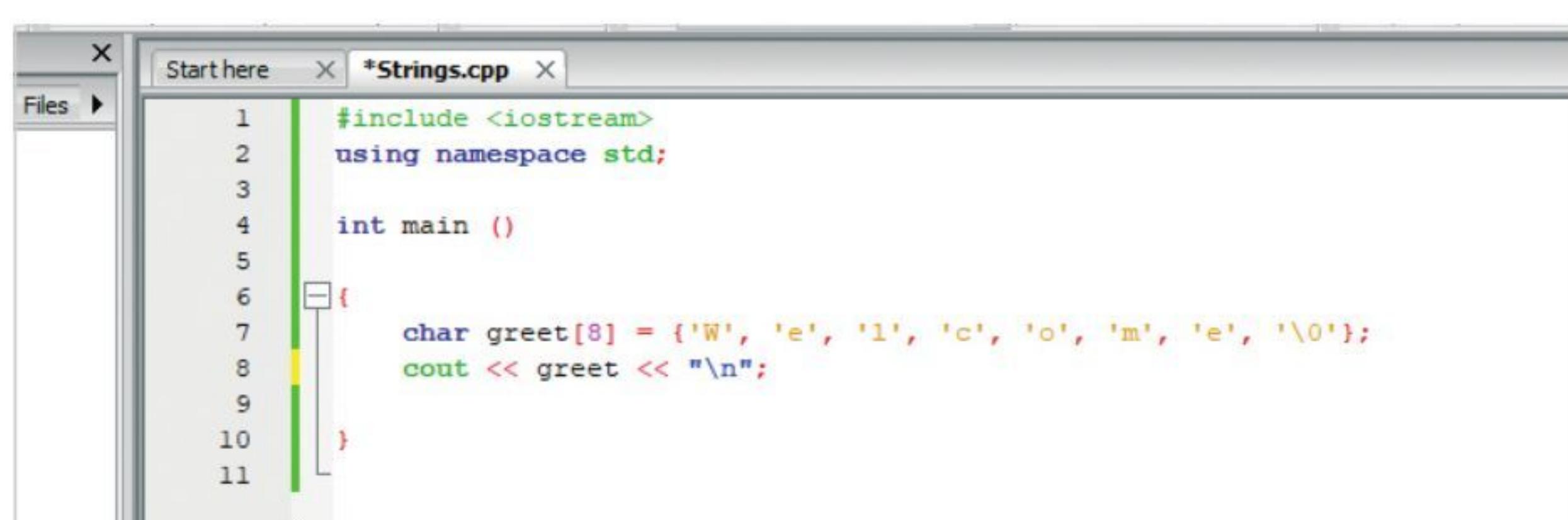
A screenshot of a C++ IDE showing the code for Step 1. The code consists of the standard header and the `int main()` function definition. The cursor is positioned at the end of the opening brace of the main function.

### STEP 2

It's easy to confuse a string with an array. Here's an array, which can be terminated with a null character:

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

int main ()
{
    char greet[8] = {'W', 'e', 'l', 'c', 'o', 'm',
                     'e', '\0'};
    cout << greet << "\n";
}
```



A screenshot of a C++ IDE showing the code for Step 2. It includes a character array `greet` initialized with the letters of "Welcome" followed by a null terminator. The output is directed to the console using `cout`.

### STEP 3

Build and run the code, and 'Welcome' appears on the screen. While this is perfectly fine, it's not a string. A string is a class, which defines objects that can be represented as a stream of characters and doesn't need to be terminated like an array. The code can therefore be represented as:

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

int main ()
{
    char greet[] = "Welcome";
    cout << greet << "\n";
}
```



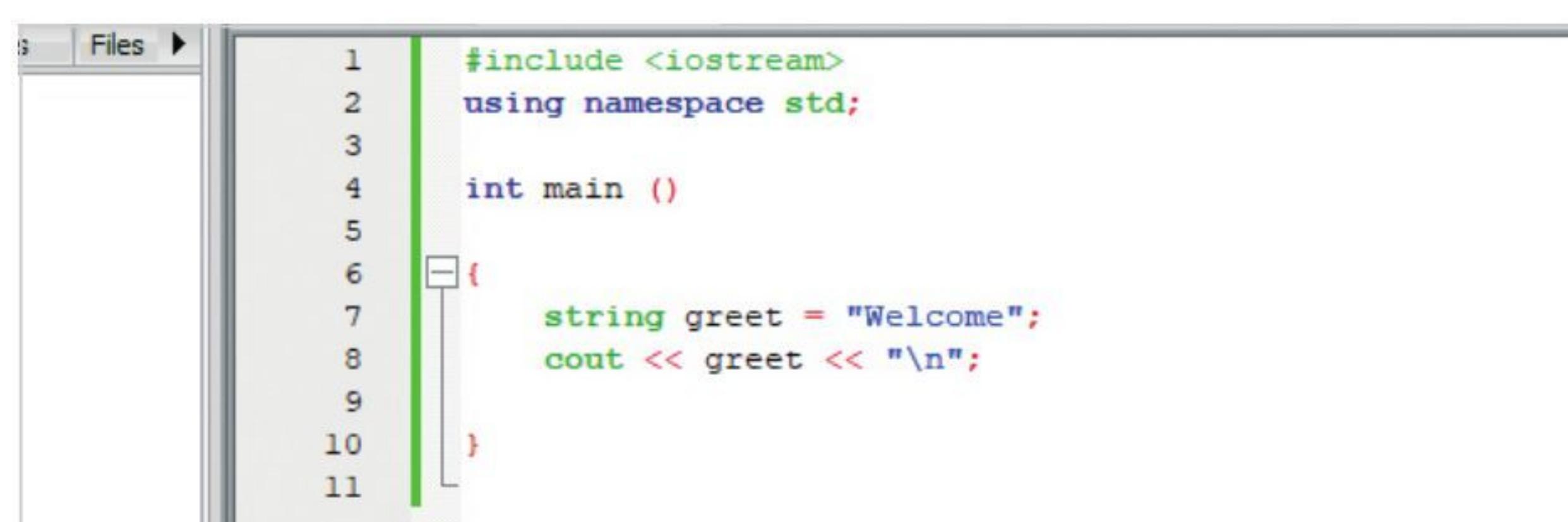
A screenshot of a C++ IDE showing the code for Step 3. It uses a character array `greet` initialized with the string "Welcome". The output is directed to the console using `cout`.

### STEP 4

In C++ there's also a `string` function, which works in much the same way. Using the greeting code again, you can enter:

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

int main ()
{
    string greet = "Welcome";
    cout << greet << "\n";
}
```



A screenshot of a C++ IDE showing the code for Step 4. It uses the `string` class to store the greeting "Welcome". The output is directed to the console using `cout`.

**STEP 5**

There are also many different operations that you can apply with the string function. For instance, to get the length of a string you can use:

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

int main ()
{
    string greet = "Welcome";
    cout << "The length of the string is: ";
    cout << greet.size() << "\n";
}
```

A screenshot of a code editor window titled 'Strings.cpp'. The code is identical to the one provided in the text block above, demonstrating how to calculate the size of a string.

**STEP 6**

You can see that we used `greet.size()` to output the length, the number of characters there are, of the contents of the string. Naturally, if you call your string something other than `greet`, then you need to change the command to reflect this. It's always `stringname.operation`. Build and run the code to see the results.

A screenshot of a terminal window showing the output of the program. The output is: 'The length of the string is: 7'. This indicates that the string 'Welcome' has a length of 7 characters.

**STEP 7**

You can of course add strings together, or rather combine them to form longer strings:

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

int main ()
{
    string greet1 = "Hello";
    string greet2 = ", world!";
    string greet3 = greet1 + greet2;

    cout << greet3 << "\n";
}
```

A screenshot of a code editor window titled 'Strings.cpp'. The code is identical to the one provided in the text block above, demonstrating how to concatenate two strings.

**STEP 8**

Just as you might expect, you can mix in an integer and store something to do with the string. In this example, we created `int length`, which stores the result of `string.size()` and outputs it to the user:

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

int main ()
{
    int length;
    string greet1 = "Hello";
    string greet2 = ", world!";
    string greet3 = greet1 + greet2;

    length = greet3.size();

    cout << "The length of the combined strings
is: " << length << "\n";
}
```

**STEP 9**

Using the available operations that come with the `string` function, you can manipulate the contents of a string. For example, to remove characters from a string you could use:

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

int main ()
{
    string strg ("Here is a long sentence in a
string.");
    cout << strg << '\n';

    strg.erase (10,5);
    cout << strg << '\n';

    strg.erase (strg.begin()+8);
    cout << strg << '\n';

    strg.erase (strg.begin()+9, strg.end()-9);
    cout << strg << '\n';
}
```

**STEP 10**

It's worth spending some time playing around with the numbers, which are the character positions in the string. Occasionally, it can be hit and miss whether you get it right, so practice makes perfect. Take a look at the screenshot to see the result of the code.

A screenshot of a terminal window showing the output of the program. The output is: 'Here is a long sentence in a string.' followed by three blank lines. This indicates that the first 10 characters ('Here is a long') were removed from the original string.



# C++ Maths

Programming is mathematical in nature and as you might expect, there's plenty of built-in scope for some quite intense maths. C++ has a lot to offer someone who's implementing mathematical models into their code. It can be extremely complex or relatively simple.

## C++ = MC<sup>2</sup>

The basic mathematical symbols apply in C++ as they do in most other programming languages. However, by using the C++ Math Library, you can also calculate square roots, powers, trig and more.

**STEP 1** C++'s mathematical operations follow the same patterns as those taught in school, in that multiplication and division take precedence over addition and subtraction. You can alter that though. For now, create a new file and enter:

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

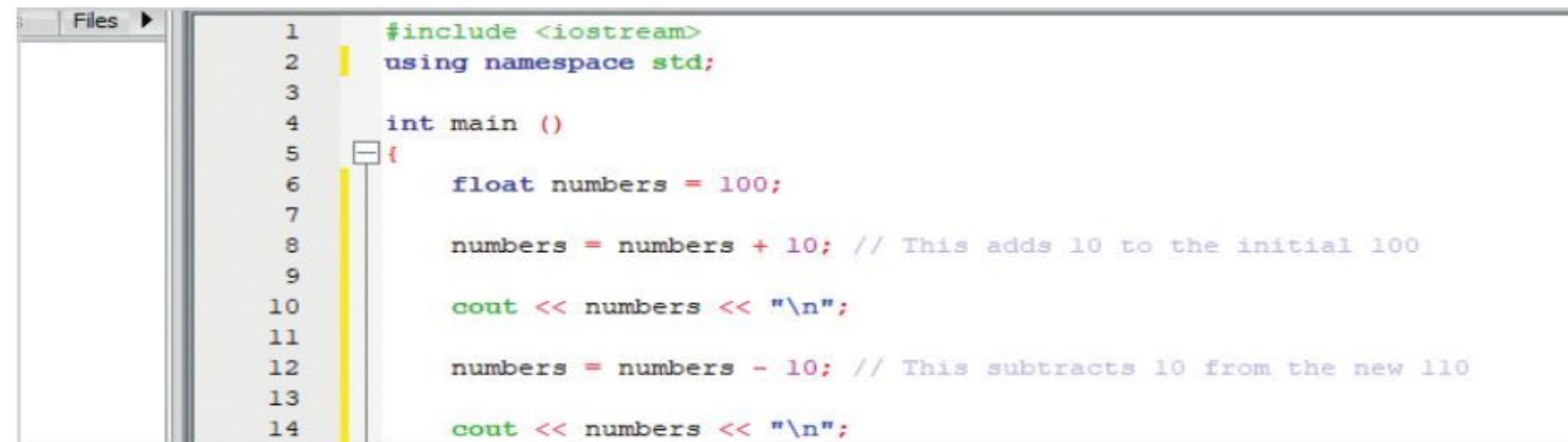
int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the
initial 100

    cout << numbers << "\n";

    numbers = numbers - 10; // This subtracts 10
from the new 110

    cout << numbers << "\n";
}
```



**STEP 2** While simple, it does get the old maths muscle warmed up. Note that we used a float for the numbers variable. While you can happily use an integer, if you suddenly started to use decimals, you would need to change to a float or a double, depending on the accuracy needed. Run the code and see the results.

```
C:\Users\david\Documents\C++\Maths.exe
110
100

Process returned 0 (0x0) execution time : 0.043 s
Press any key to continue.
```

**STEP 3** Multiplication and division can be applied as such:

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

int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100
by 10

    cout << numbers << "\n";

    numbers = numbers / 10; // And this divides
1000 by 10

    cout << numbers << "\n";
}
```

```
Start here x Maths.cpp x
1 #include <iostream>
2 using namespace std;
3
4
5 int main ()
6 {
7     float numbers = 100;
8
9     numbers = numbers * 10; // This multiplies 100 by 10
10
11     cout << numbers << "\n";
12
13     numbers = numbers / 10; // And this divides 1000 by 10
14
15     cout << numbers << "\n";
16 }
```

**STEP 4** Again, execute the simple code and see the results. While not particularly interesting, it's a start into C++ maths. We used a float here, so you can play around with the code and multiply by decimal places, as well as divide, add and subtract.

```
C:\Users\david\Documents\C++\Maths.exe
1000
100

Process returned 0 (0x0) execution time : 0.050 s
Press any key to continue.
```

**STEP 5**

The interesting maths content comes when you call upon the C++ Math Library. Within this header are dozens of mathematical functions along with further operations. Everything from computing cosine to arc tangent with two parameters, to the value of PI. You can call the header with:

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

int main ()
{
}
```

```
Start here *Maths.cpp x
Files >
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main ()
6 {
7
8
9 }
10
```

**STEP 6**

Start by getting the square root of a number:

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

int main ()
{
    float number = 134;
    cout << "The square root of " << number << "
is: " << sqrt(number) << "\n";
}
```

```
Start here *Maths.cpp x
Files >
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main ()
6 {
7     float number = 134;
8     cout << "The square root of " << number << "
9         is: " << sqrt(number) << "\n";
10
11 }
12
```

**STEP 7**

Here we created a new float called number and used the `sqrt(number)` function to display the square root of 134, the value of the variable, number. Build and run the code, and your answer reads 11.5758.

```
C:\Users\david\Documents\C++\Maths.exe
The square root of 134 is: 11.5758
Process returned 0 (0x0) execution time : 0.046 s
Press any key to continue.
```

**STEP 8**

Calculating powers of numbers can be done with:

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

int main ()
{
    float number = 12;
    cout << number << " to the power of 2 is " <<
        pow(number, 2) << "\n";
    cout << number << " to the power of 3 is " <<
        pow(number, 3) << "\n";
    cout << number << " to the power of .08 is " <<
        pow(number, 0.8) << "\n";
}
```

```
Start here *Maths.cpp x
Files >
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main ()
6 {
7     float number = 12;
8
9     cout << number << " to the power of 2 is " << pow(number, 2) << "\n";
10    cout << number << " to the power of 3 is " << pow(number, 3) << "\n";
11    cout << number << " to the power of .08 is " << pow(number, 0.8) << "\n";
12
13 }
14
```

**STEP 9**

Here we created a float called number with the value of 12, and the `pow(variable, power)` is where the calculation happens. Of course, you can calculate powers and square roots without using variables. For example, `pow(12, 2)` outputs the same value as the first cout line in the code.

```
C:\Users\david\Documents\C++\Maths.exe
12 to the power of 2 is 144
12 to the power of 3 is 1728
12 to the power of .08 is 7.30037

Process returned 0 (0x0) execution time : 0.049 s
Press any key to continue.
```

**STEP 10**

The value of Pi is also stored in the cmath header library. It can be called up with the `M_PI` function. Enter `cout << M_PI;` into the code and you get 3.14159; or you can use it to calculate:

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

int main ()
{
    double area, radius = 1.5;
    area = M_PI * radius * radius;
    cout << area << "\n";
}
```

```
Start here *Maths.cpp x
Files >
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main ()
6 {
7     double area, radius = 1.5;
8     area = M_PI * radius * radius;
9     cout << area << "\n";
10
11 }
12
13
14
```