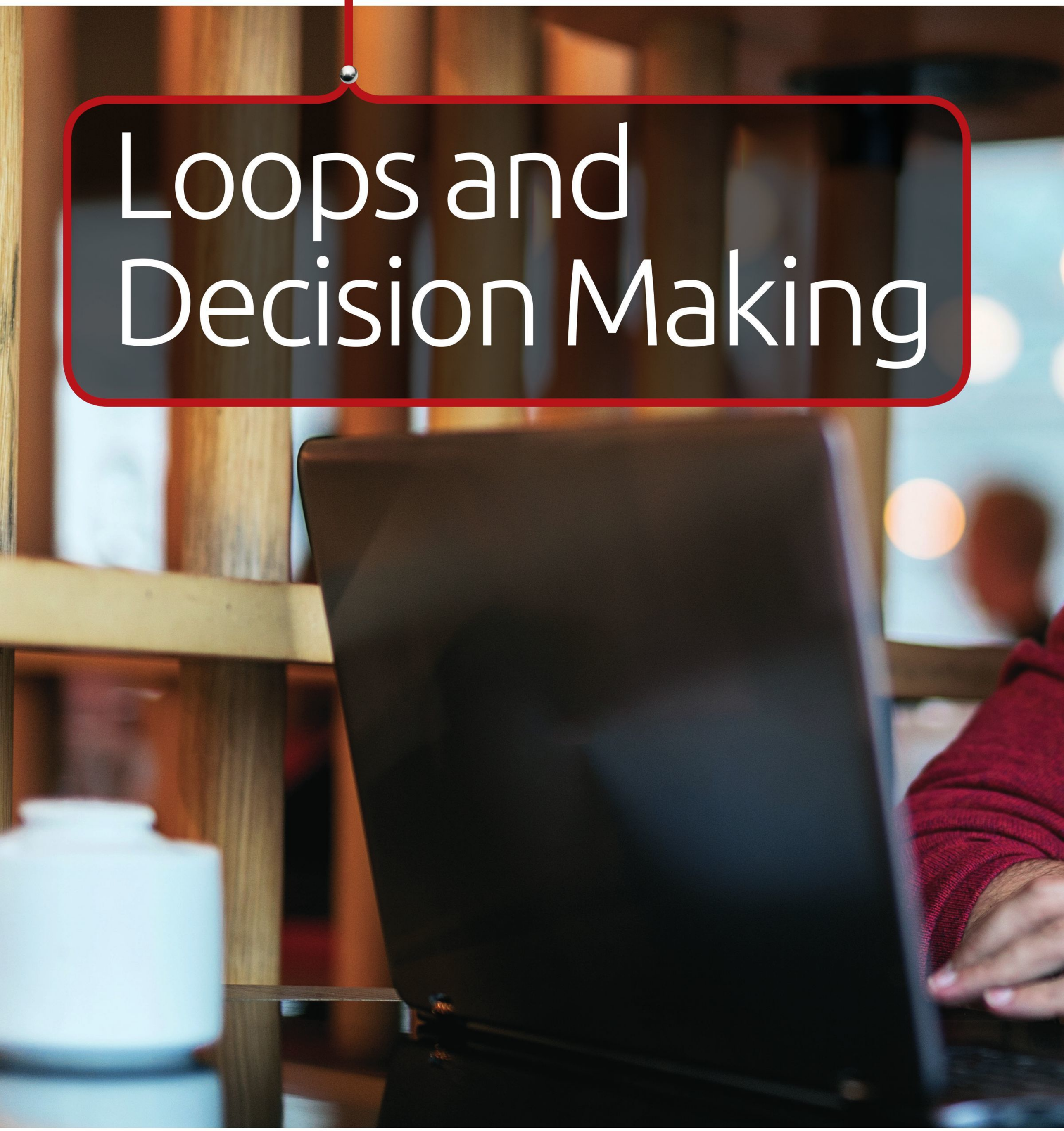
Loops and Decision Making



Loops and repetition are one of the most important factors of any programming language. Good use of a loop creates a program that does exactly what you want it to and delivers the desired outcome without issues or errors.

Without loops and decision-making events within the code, your program will never be able to offer the user any choice. It's this understanding of choice that elevates your skills as a programmer and makes for much better code.

Creating a successful loop gives the code a beginning, middle and end; ready to start again should the user wish to.

While Loop

A while loop's function is to repeat a statement, or a group of statements, while a certain condition remains true. When the while loop starts, it initialises itself by testing the condition of the loop and the statements within, before executing the rest of the loop.

TRUE OR FALSE?

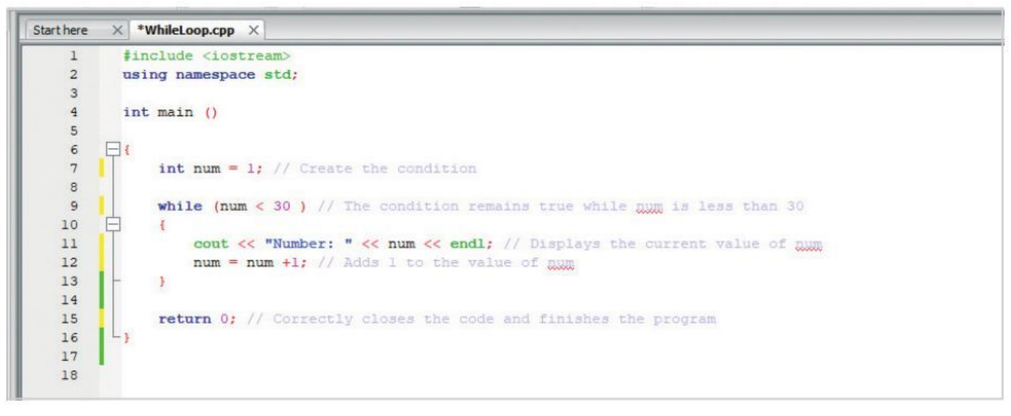
While loops are one of the most popular form of C++ code looping. They repeatedly run the code contained within the loop while the condition is true. Once it proves false, the code continues as normal.

STEP 1 Clear what you've done so far and create a new C++ file. There's no need for any extra headers at the moment, so add the standard headers as per usual:

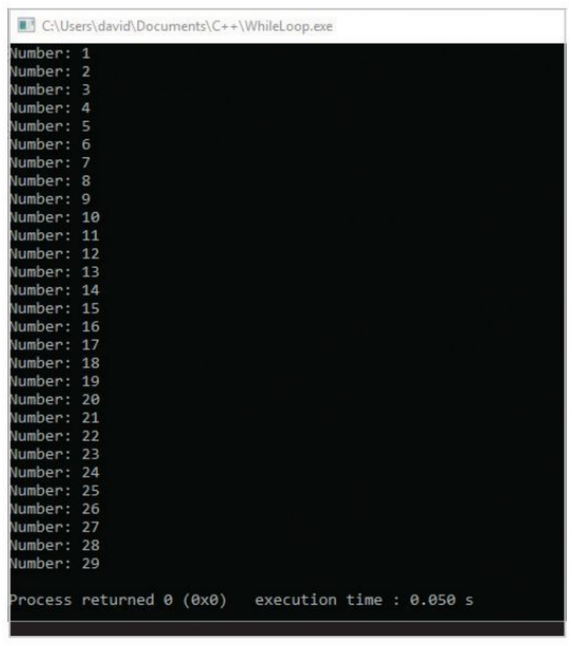
```
#include <iostream>
using namespace std;

int main ()

{

}
```

STEP 2 Create a simple while loop. Enter the code below, build and run (we've added comments to the screen shot):

```
{
    int num = 1;

    while (num < 30 )
    {
        cout << "Number: " << num << endl;
        num = num +1;
    }

    return 0;
}
```



STEP 3 First you need to create a condition, so use a variable called num and give it the value 1. Now create the while loop, stating that as long as num is less than 30, the loop is true. Within the loop the value of num is displayed and adds 1 until it's more than 30.



STEP 4 We're introducing a few new elements here. The first are the opening and closing braces for the while loop. This is because our loop is a compound statement, meaning a group of statements; note also, there's no semicolon after the while statement. You now also have return 0, which is a clean and preferred way of ending the code.

```
{
    int num = 1; // Create the condition

    while (num < 30 ) // The condition remains true while num is less than 30
    {
        cout << "Number: " << num << endl; // Displays the current value of num
        num = num +1; // Adds 1 to the value of num
    }

    return 0; // Correctly closes the code and finishes the program
}
```

STEP 5 If you didn't need to see the continually increasing value of num, you could have done away with the compound while statement and instead just added num by itself until it reached 30, and then displayed the value:

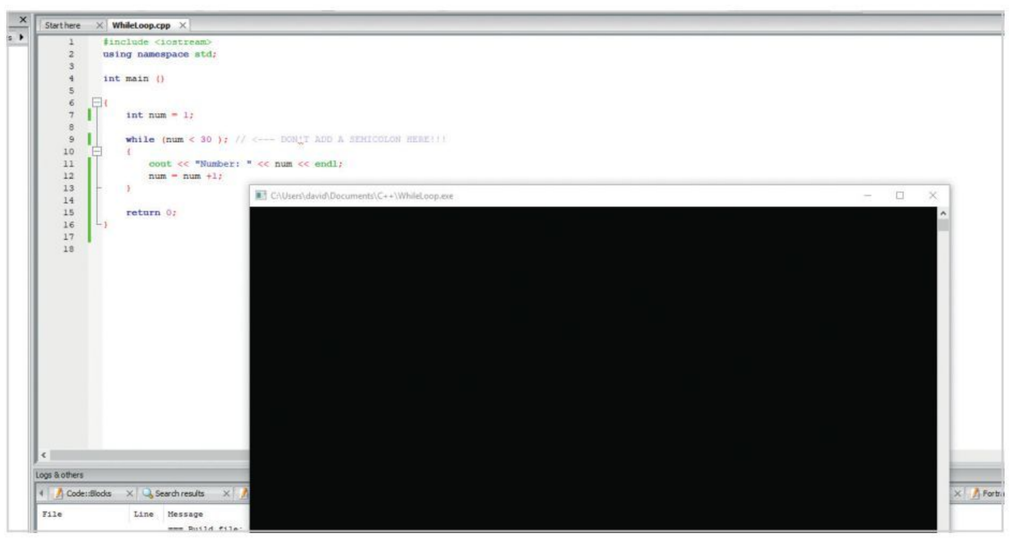
```
{
    int num = 1;

    while (num < 30 )
        num++;
    cout << "Number: " << num << endl;

    return 0;
}
```

STEP 6

It's important to remember not to add a semicolon at the end of a while statement. Why? Well, as you know, the semicolon represents the end of a C++ line of code. If you place one at the end of a while statement, your loop will be permanently stuck until you close the program.

STEP 7

In our example, if we were to execute the code the value of num would be 1, as set by the int statement. When the code hits the while statement it reads that while the condition of 1 being less than 30 is true, loop. The semicolon closes the line, so the loop repeats; but it never adds 1 to num, as it won't continue through the compound statement.



STEP 8

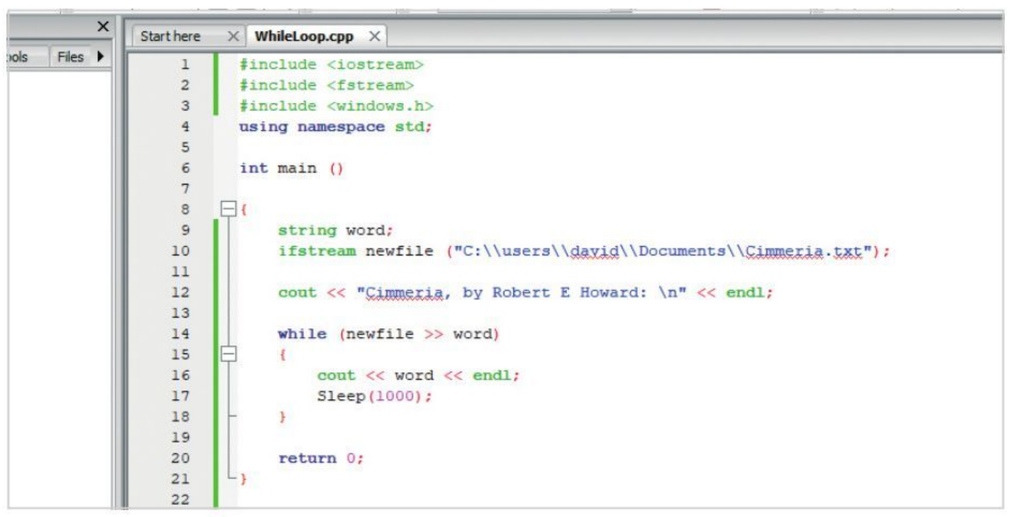
You can manipulate the while statement to display different results depending on what code lies within the loop. For example, to read the poem, Cimmeria, word by word, you would enter:

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

int main ()

{
    string word;
    ifstream newfile ("C:\\users\\david\\
Documents\\Cimmeria.txt");

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

    while (newfile >> word)
    {
        cout << word << endl;
    }

    return 0;
}
```

STEP 9

You can further expand the code to enable each word of the poem to appear every second. To do so, you need to pull in a new library, <windows.h>. This is a Windows only library and within it you can use the Sleep() function:

```
#include <iostream>
#include <fstream>
#include <windows.h>
using namespace std;

int main ()

{
    string word;
    ifstream newfile ("C:\\users\\david\\
Documents\\Cimmeria.txt");

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

    while (newfile >> word)
    {
        cout << word << endl;
        Sleep(1000);
    }

    return 0;
}
```



STEP 10

Sleep() works in milliseconds, so Sleep(1000) is one second, Sleep(10000) is ten seconds and so on. Combining the sleep function (along with the header it needs) and a while loop enables you to come up with some interesting countdown code.

```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    int a = 10;

    while (a != 0)
    {
        cout << a << endl;
        a = a - 1;
        Sleep(1000);
    }

    cout << "\nBlast Off!" << endl;

    return 0;
}
```

For Loop

In some respects, a for loop works in a very similar way to that of a while loop, although it's structure is different. A for loop is split into three stages: an initialiser, a condition and an incremental step. Once set up, the loop repeats itself until the condition becomes false.

LOOPY LOOPS

The initialise stage of a for loop is executed only once and this sets the point reference for the loop. The condition is evaluated by the loop to see if it's true or false and then the increment is executed. The loop then repeats the second and third stage.

STEP 1 Create a new C++ file, with the standard headers:

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

int main ()

{


}
```

STEP 2 Start simple and create a for loop that counts from 1 to 30, displaying the value to the screen with each increment:

```
{
    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
    {
        cout << "Number: " << num << endl;
    }

    return 0;

}
```

STEP 3 Working through the process of the for loop, begin by creating an integer called num and assigning it a value of 1. Next, set the condition, in this case num being less than 30. The last stage is where you create the increments; here it's the value of num being added by 1.

```
{
    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
```

STEP 4 After the loop, you created a compound statement in braces (curly brackets), that displays the current value of the integer num. Every time the for loop repeats itself, the second and third stages of the loop, it adds 1 until the condition <30 is false. The loop then ends and the code continues, ending neatly with return 0.

```
{
    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
    {
        cout << "Number: " << num << endl;
    }

    return 0;

}
```

STEP 5 A for loop is quite a neat package in C++, all contained within its own brackets, while the other elements outside of the loop are displayed below. If you want to create a 10-second countdown, you could use:

```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    //For Loop Begins
    for( int a = 10; a != 0; a = a -1 )
    {
        cout << a << endl;
        Sleep(1000);
    }

    cout << "\nBlast Off!" << endl;

    return 0;

}
```

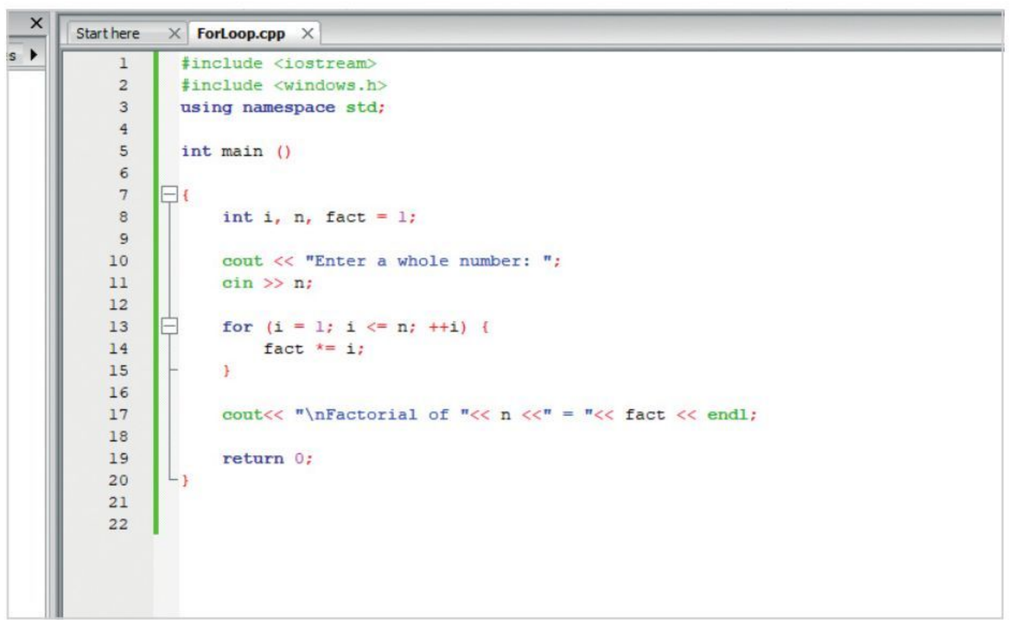
STEP 6

With the countdown code, don't forget to include the windows.h library, so you can use the Sleep command. Build and run the code; in the command console you can see the numbers 10 to 1 countdown in one second increments, until it reaches zero and Blast Off! appears.

STEP 7

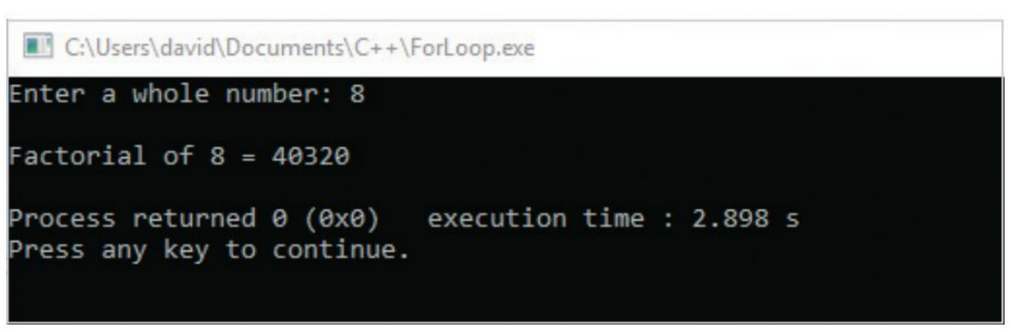
Naturally you can include a lot more content into a for loop, including some user input:

```
    int i, n, fact = 1;

    cout << “Enter a whole number: “;
    cin >> n;

    for (i = 1; i <= n; ++i) {
        fact *= i;
    }

    cout<< “\nFactorial of “<< n <<” = “<< fact <<
endl;

    return 0;
```



STEP 8

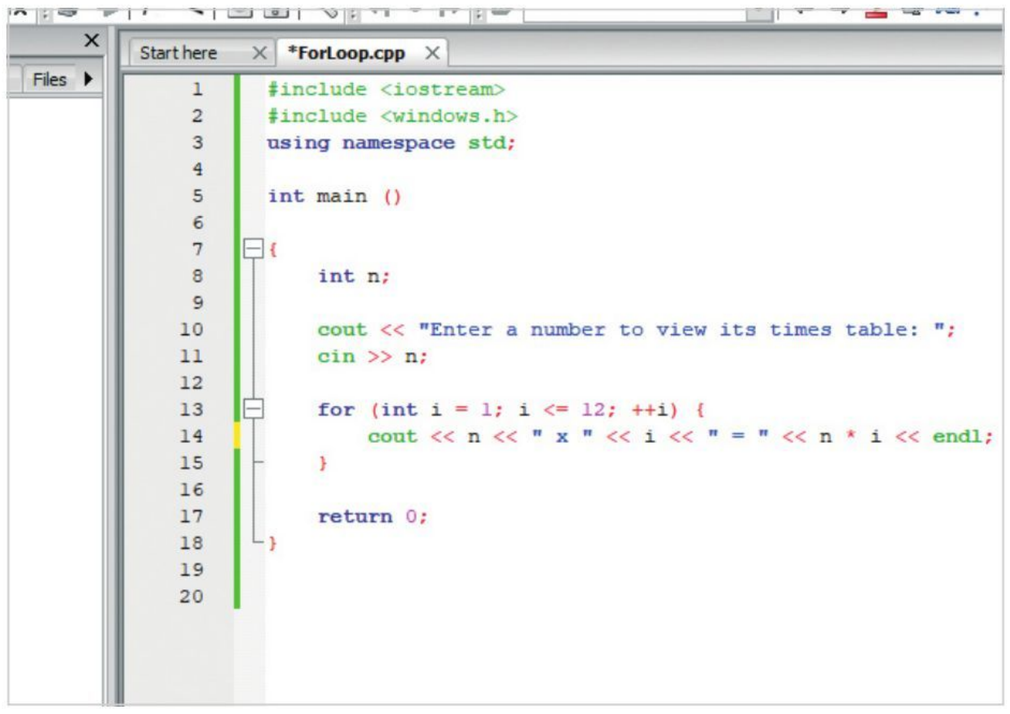
The code from step 7, when built and run, asks for a number, then displays the factorial of that number through the for loop. The user's number is stored in the integer n, followed by the integer I which is used to check if the condition is true or false, adding 1 each time and comparing it to the user's number, n.



STEP 9

Here's an example of a for loop displaying the multiplication tables of a user inputted number. Handy for students:

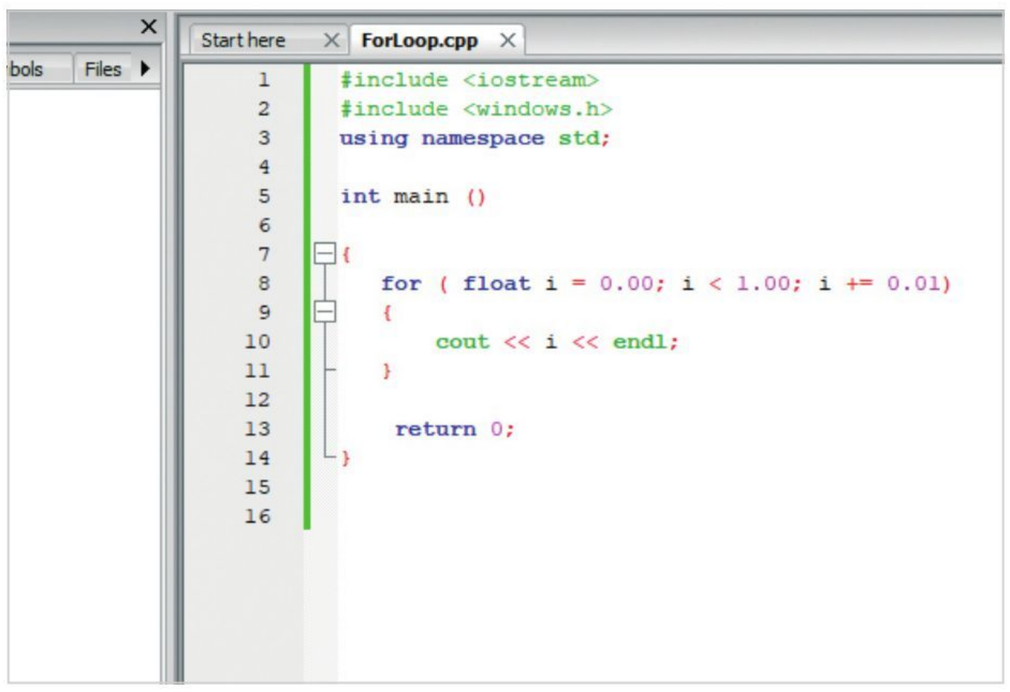
```
{
    int n;

    cout << “Enter a number to view its times
table: “;
    cin >> n;

    for (int i = 1; i <= 12; ++i) {
        cout << n << “ x “ << i << “ = “ << n * i
<< endl;
    }

    return 0;
}
```



STEP 10

The value of the integer i can be expanded from 12 to whatever number you want, displaying a very large multiplication table in the process (or a small one). Of course the data type within a for loop doesn't have to be an integer; as long as it's valid, it works.

```
for ( float i = 0.00; i < 1.00; i += 0.01)
    {
        cout << i << endl;
    }

return 0;
```



Do… While Loop

A do… while loop differs slightly from that of a for or even a while loop. Both for and while set and examine the state of the condition at the start of the loop, or the top of the loop if you prefer. However, a do… while loop, is similar to a while loop but instead checks the condition at the bottom of the loop.

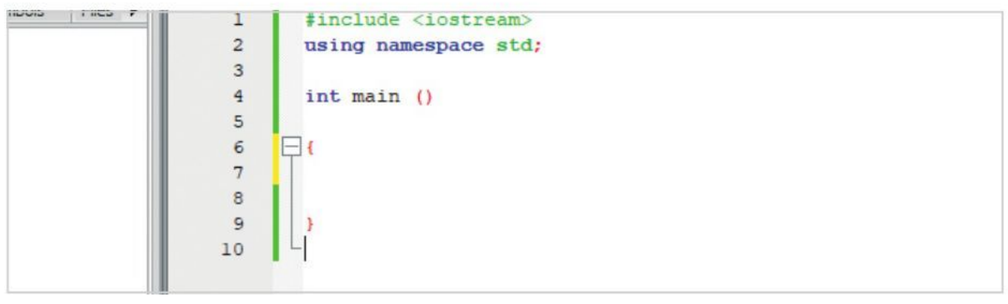
DO LOOPS

The good thing about a do… while loop is that it's guaranteed to run through at least once. It's structure is: do, followed by statements, while condition is true. This is how it works.

STEP 1

Begin with a new, blank C++ file and enter the standard headers:

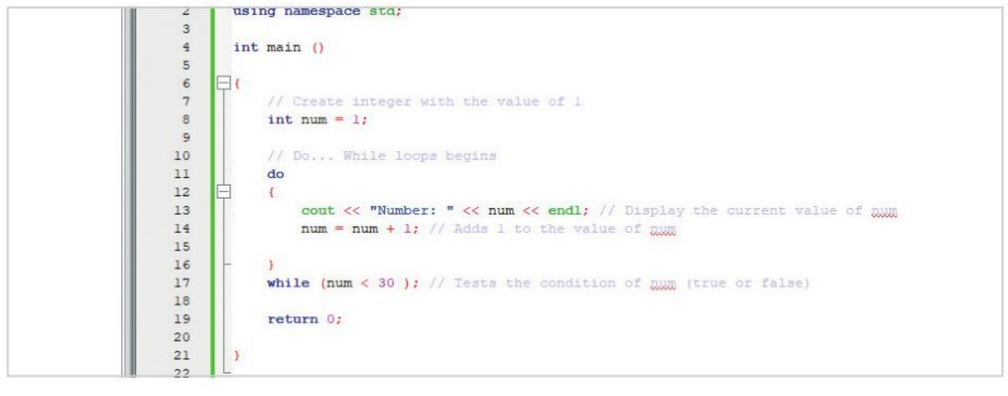
```
#include <iostream>
using namespace std;

int main ()

{


}
```



STEP 2

Begin with a simple number count:

```
{
    int num = 1;

    do
    {
        cout << “Number: “ << num << endl;
        num = num + 1;

    }
    while (num < 30 );

    return 0;

}
```



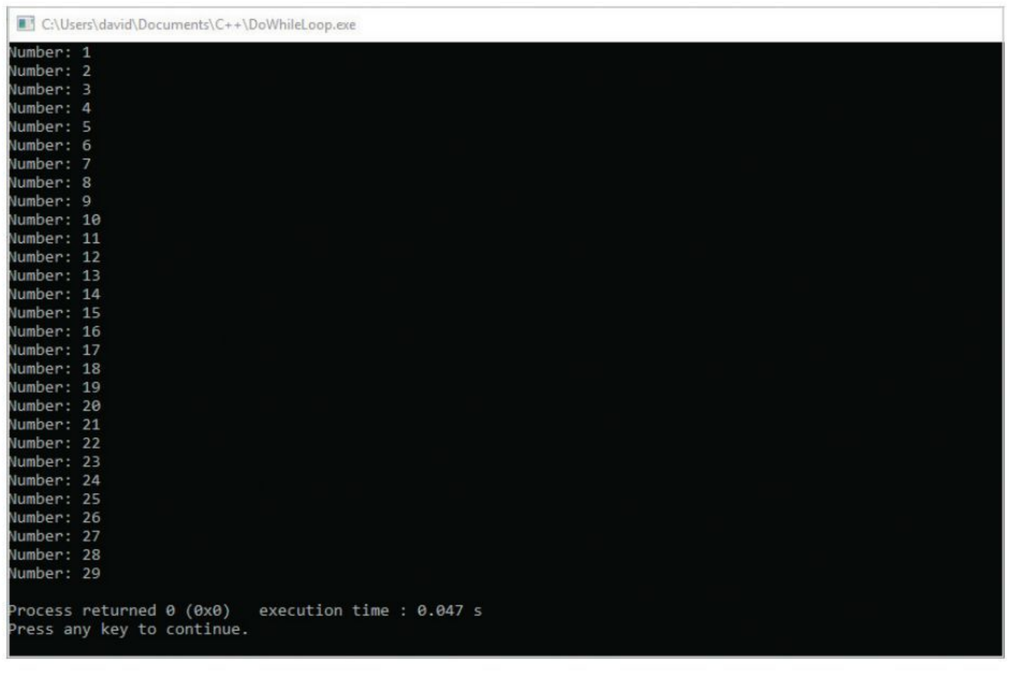
STEP 3

Now, here's a look at the structure of a do… while loop. First you create an integer called num, with the value of 1. Now the do… while loops begins. The code inside the body of the loop is executed at least once, then the condition is checked for either true or false.

```
// Create integer with the value of 1
int num = 1;

// Do... While loops begins
do
{
    cout << "Number: " << num << endl; // Display the current value of num
    num = num + 1; // Adds 1 to the value of num

}
while (num < 30 ); // Tests the condition of num (true or false)
```

STEP 4

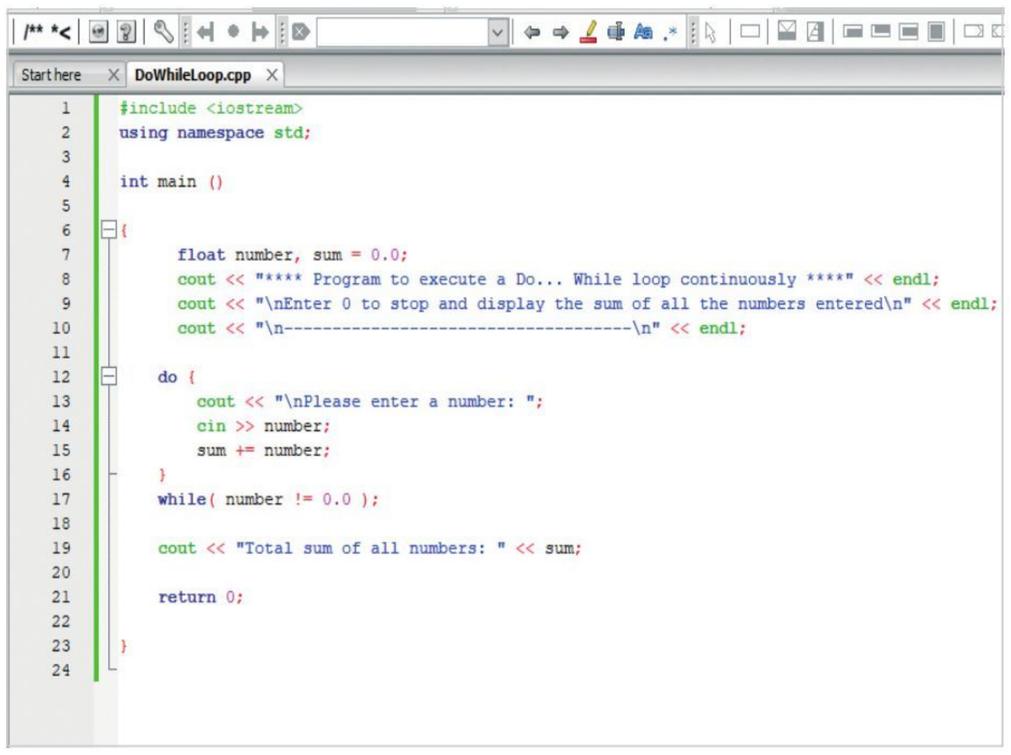
If the condition is true, the loop is executed. This continues until the condition is false. When the condition has been expressed as false, the loop terminates and the code continues. This means you can create a loop where the code continues until the user enters a certain character.



STEP 5

If you want code to add up user inputted numbers until the user enters zero:

```
{
      float number, sum = 0.0;
      cout << “**** Program to execute a Do...
While loop continuously ****” << endl;
      cout << “\nEnter 0 to stop and display the
sum of all the numbers entered\n” << endl;
      cout << “\n--------------------------------
---\n” << endl;

    do {
         cout<<”\nPlease enter a number: “;
         cin>>number;
         sum += number;
    }
    while(number != 0.0);
    cout<<”Total sum of all numbers: “<<sum;
    return 0;
}
```



STEP 6

The code from Step 5 works as follows: two floating point variables are assigned, number and sum, both with the value of 0.0. There is a brief set of instructions for the user, then the do… while loop begins.

```
{
      float number, sum = 0.0;
      cout << "**** Program to execute a Do... While loop continuously ****" << endl;
      cout << "\nEnter 0 to stop and display the sum of all the numbers entered\n" << endl;
      cout << "\n--------------------------------\n" << endl;
```

STEP 7

The do… while loop in this instance asks the user to input a number, which you assigned to the float variable, number. The calculation step uses the second floating point variable, sum, which adds the value of number every time the user enters a new value.

```
    do {
         cout << "\nPlease enter a number: ";
         cin >> number;
         sum += number;
```

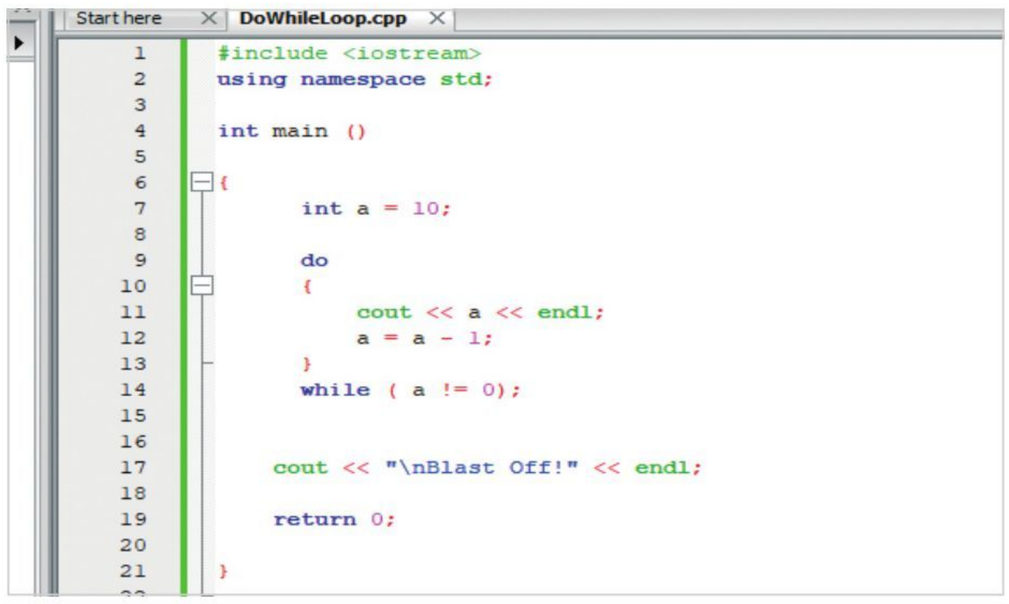
STEP 8

Finally, the while statement checks the condition of the variable number. If the user has entered zero, then the loop is terminated, if not then it continues indefinitely. When the user finally enters zero, the value of sum, the total value of all the user's input, is displayed. The loop, and the program, then ends.

STEP 9

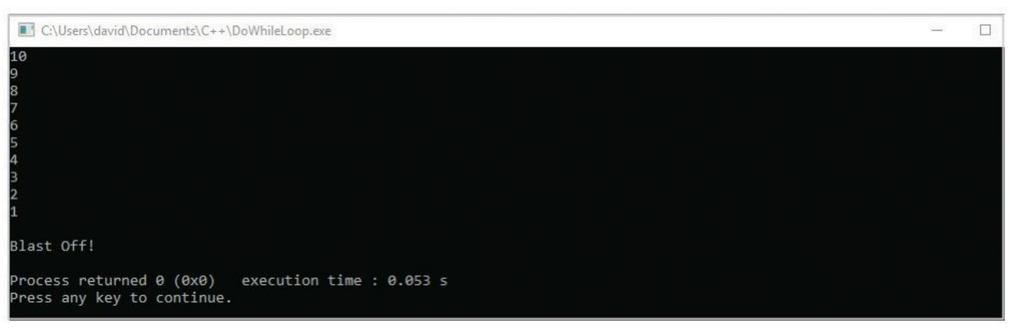
Using the countdown and Blast Off! code used previously, a do… while loop would look like:

```
{
      int a = 10;

      do
      {
            cout << a << endl;
            a = a - 1;
      }
      while ( a != 0);


    cout << “\nBlast Off!” << endl;

    return 0;
}
```



STEP 10

The main advantage of using a do… while loop is because it's an exit-condition loop; whereas a while loop is an entry-control loop. Therefore, if your code requires a loop that needs to be executed at least once (for example, to check the number of lives in a game), then a do… while loop is perfect.



If Statement

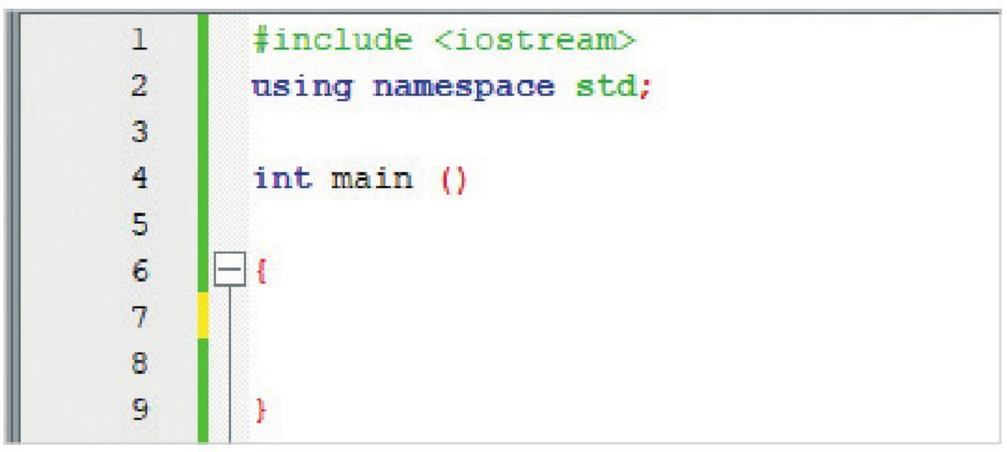
The decision making statement 'if' is probably one of the most used statements in any programming language, regardless of whether it's C++, Python, BASIC or anything else. It represents a junction in the code, where IF one condition is true, do this; or IF it's false, do that.

IF ONLY

If uses a Boolean expression within its statement. If the Boolean expression is true, the code within the statement is executed. If not, then the code after the statement is executed instead.

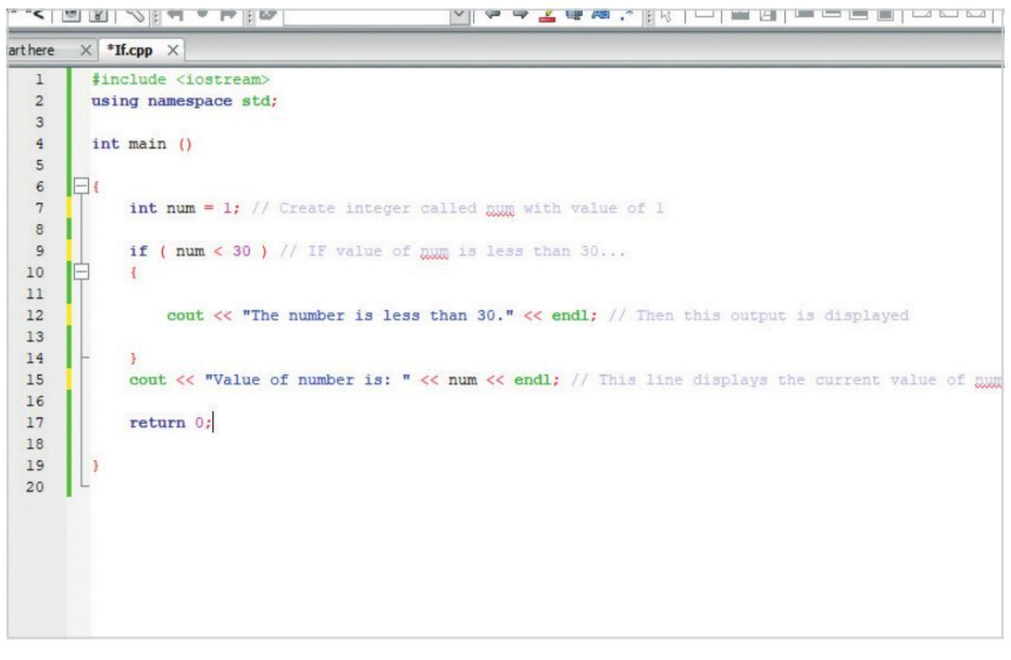
STEP 1 First, create a new C++ file and enter the relevant standard headers, as usual:

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

int main ()

{


}
```



STEP 2 If is best explained when you use a number-based condition:

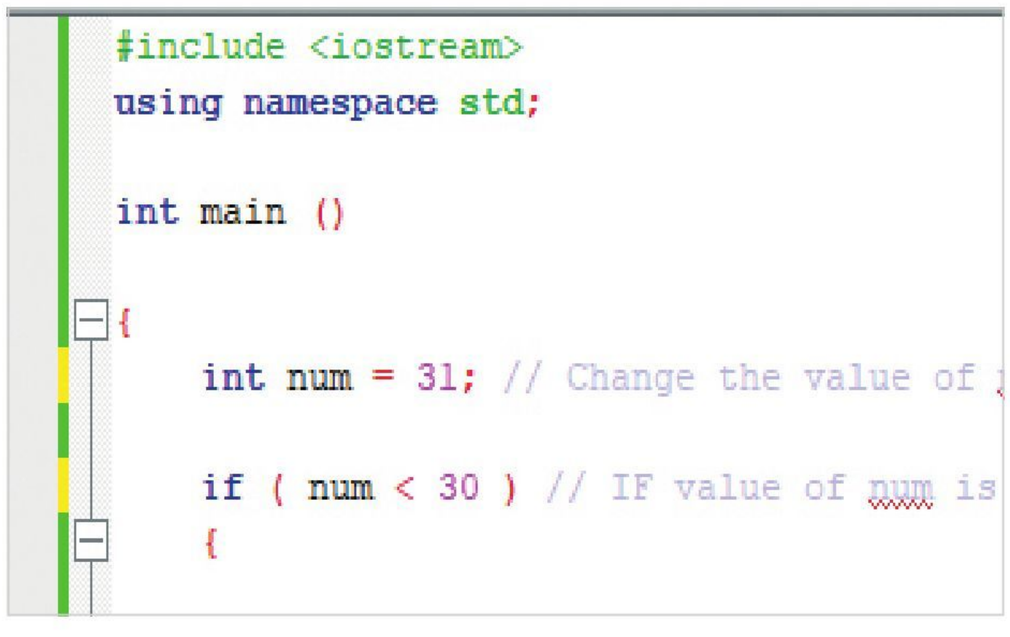
```
{
    int num = 1;

    if ( num < 30 )
    {

        cout << “The number is less than 30.” <<
endl;

    }
    cout << “Value of number is: “ << num << endl;

    return 0;

}
```



STEP 3 What's going on here? To begin, an integer called num was created and assigned with the value of 1. The if statement comes next, and in this case we've instructed the code that if the condition, the value, of num is less than 1, then the code within the braces should be executed.

```
int num = 1; // Create integer called num with value of 1

if ( num < 30 ) // IF value of num is less than 30...
{

    cout << "The number is less than 30." << endl; // Then this output is displayed
```

STEP 4 The second cout statement displays the current value of num and the program terminates safely. It's easy to see how the if statement works if you were to change the initial value of num from 1 to 31.



STEP 5

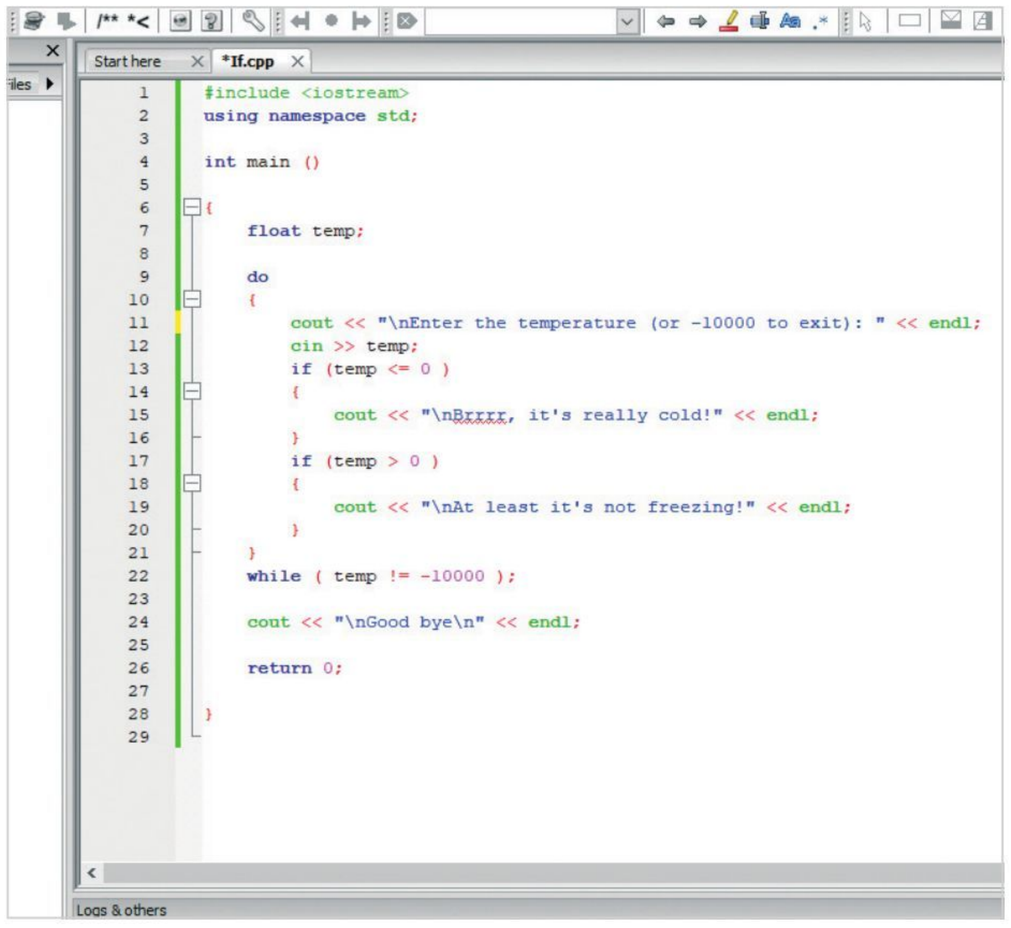
When you change the value to anything above 30, then build and run the code, you can see that the only line to be outputted to the screen is the second cout statement, displaying the current value of num. This is because the initial if statement is false, so it ignores the code within the braces.

```
C:\Users\david\Documents\C++\if.exe
Value of number is: 31

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

STEP 6

You can include an if statement within a do… while loop. For example:

```
{
    float temp;

    do
    {
        cout << “\nEnter the temperature (or
-10000 to exit): “ << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << “\nBrrrr, it’s really cold!”
<< endl;
        }
        if (temp > 0 )
        {
            cout << “\nAt least it’s not
freezing!” << endl;
        }
    }
    while ( temp != -10000 );

    cout << “\nGood bye\n” << endl;

    return 0;

}
```



STEP 7

The code in Step 6 is simplistic but effective. First we created a floating point integer called temp, then a do… while loop that asks the user to enter the current temperature.

```
{
    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
```

STEP 8

The first if statement checks to see if the user’s inputted value is less than or equal to zero. If it is, then the output is ‘Brrrr, it’s really cold!’. Otherwise, if the input is greater than zero, the code outputs ‘At least it’s not freezing!’.

```
    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << "\nBrrrr, it's really cold!" << endl;
        }
        if (temp > 0 )
        {
            cout << "\nAt least it's not freezing!" << endl;
        }
```

STEP 9

Finally, if the user enters the value -10000, which is impossibly cold so is therefore a unrealistic value, the do… while loop is terminated and a friendly ‘Good bye’ is displayed to the screen.

```
{
    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << "\nBrrrr, it's really cold!" << endl;
        }
        if (temp > 0 )
        {
            cout << "\nAt least it's not freezing!" << endl;
        }
    }
    while ( temp != -10000 );

    cout << "\nGood bye\n" << endl;

    return 0;

}
```

STEP 10

Using if is quite powerful, if it’s used correctly. Just remember that if the condition is true then the code executes what’s in the braces. If not, it continues on its merry way. See what else you can come up with using if and a combination of loops.

```
Enter the temperature (or -10000 to exit):
34

At least it's not freezing!

Enter the temperature (or -10000 to exit):
2

At least it's not freezing!

Enter the temperature (or -10000 to exit):
-110

Brrrr, it's really cold!

Enter the temperature (or -10000 to exit):
0

Brrrr, it's really cold!

Enter the temperature (or -10000 to exit):
-10000

Brrrr, it's really cold!

Good bye


Process returned 0 (0x0)   execution time : 17.323 s
Press any key to continue.
```

If… Else Statement

There is a much better way to use an if statement in your code, with if… else. If… else works in much the same way as a standard if statement. If the Boolean expression is true, the code within the braces is executed. Else, the code within the next set of braces is used instead.

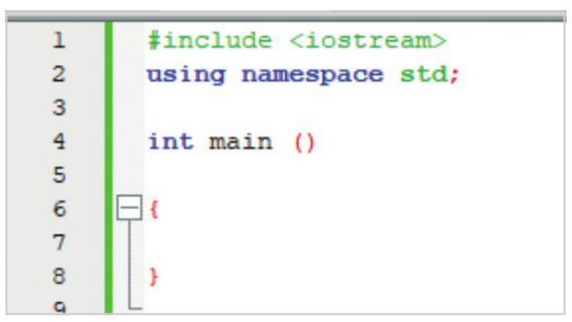
IF YES, ELSE NO

There are two sections of code that can be executed depending on the outcome in an if… else statement. It's quite easy to visualise once you get used to its structure.

STEP 1

Begin with a new C++ file and the standard headers:

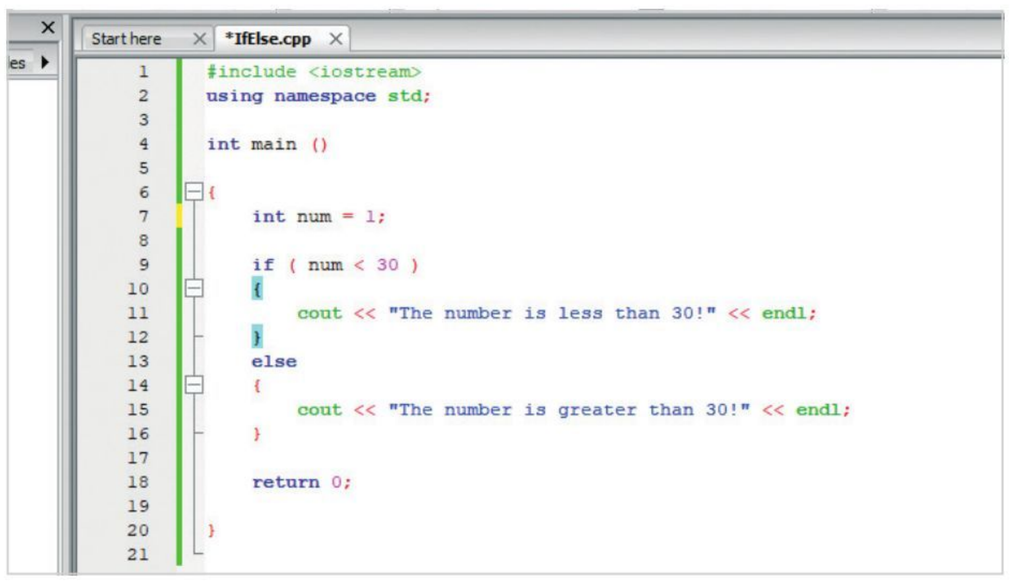
```
#include <iostream>
using namespace std;

int main ()

{

}
```



STEP 2

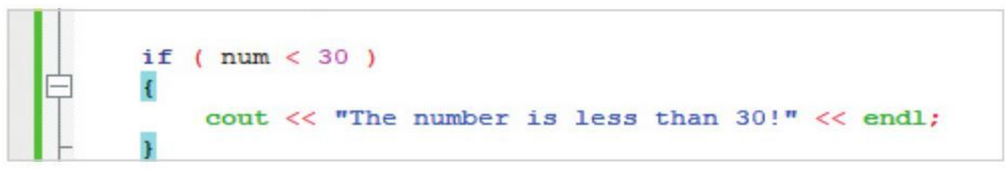
Let's expand the code from the if statement on the previous page:

```
{
    int num = 1;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else
    {
        cout << “The number is greater than 30!”
<< endl;
    }

    return 0;

}
```



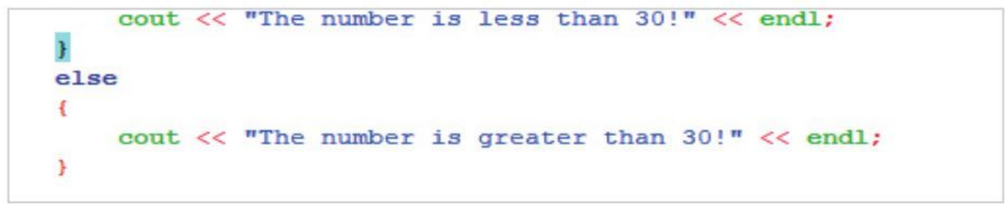
STEP 3

The first line in the code creates the integer called num and gives it a value of 1. The if statement checks to see if the value of num is less than thirty and if it is it outputs “The number is less than 30!” to the console.



STEP 4

The else companion to if checks if the number is greater than 30 and if so, then displays “The number is greater than 30!” to the console; and finally, the code is terminated satisfactorily.



STEP 5

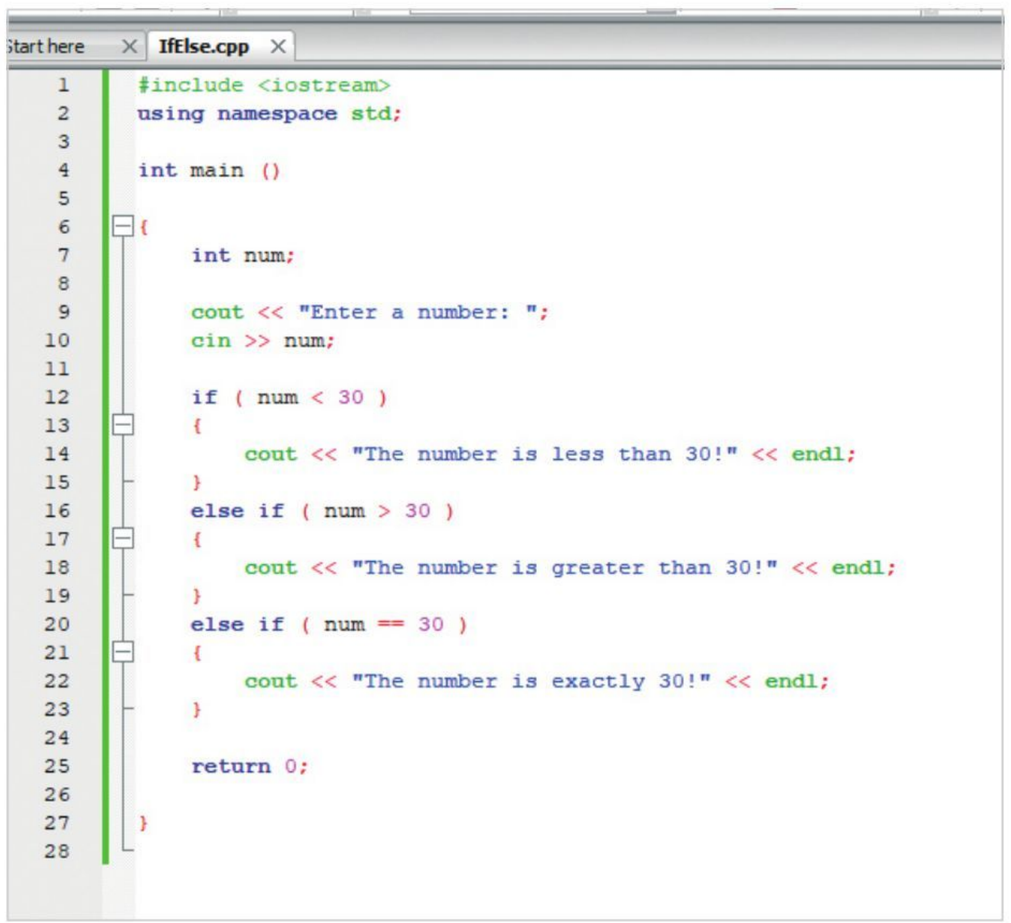
You can change the value of num in the code or you can improve the code by asking the user to enter a value:

```
{
    int num;

    cout << “Enter a number: “;
    cin >> num;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else
    {
        cout << “The number is greater than 30!”
<< endl;
    }

    return 0;

}
```

STEP 6

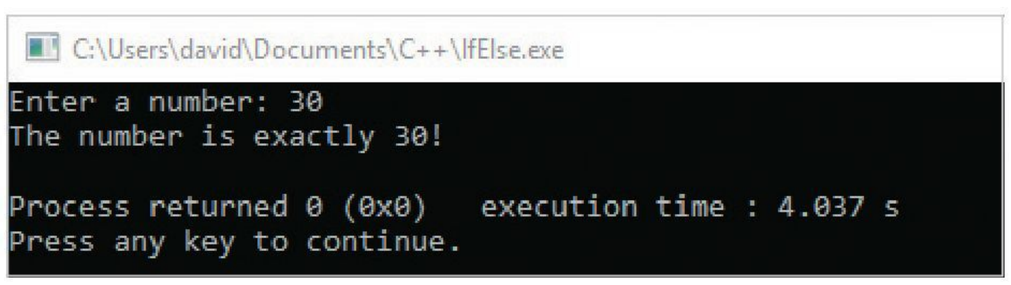
The code works the same way, as you would expect, but what if you wanted to display something if the user entered the number 30? Try this:

```
{
    int num;

    cout << “Enter a number: “;
    cin >> num;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else if ( num > 30 )
    {
        cout << “The number is greater than 30!”
<< endl;
    }
    else if ( num == 30 )
    {
        cout << “The number is exactly 30!” <<
endl;
    }

    return 0;

}
```



STEP 7

The new addition to the code is what’s known as a nested if… else statement. This allows you to check for multiple conditions. In this case, if the user enters a number less than 30, greater than 30 or actually 30 itself, a different outcome is presented to them.



STEP 8

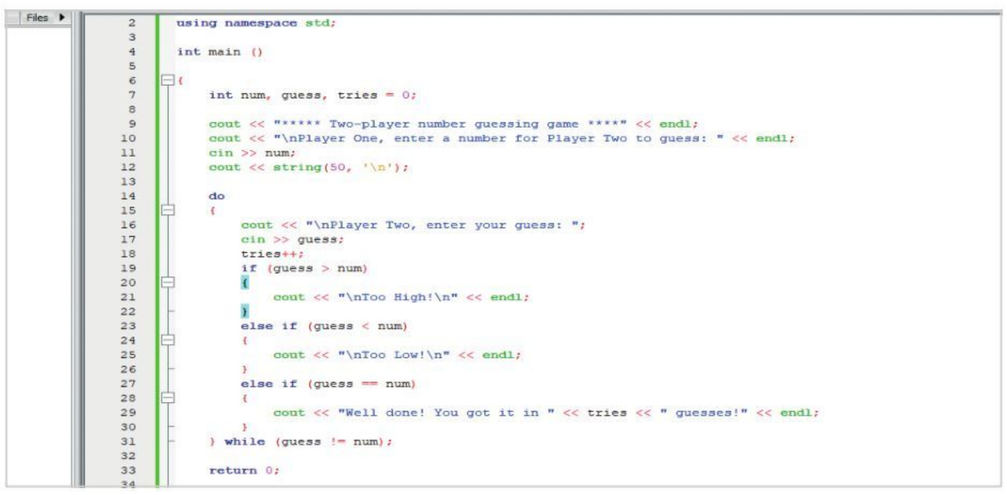
You can take this up a notch and create a two-player number guessing game. Begin by creating the variables:

```
int num, guess, tries = 0;

    cout << “***** Two-player number guessing game
****” << endl;
    cout << “\nPlayer One, enter a number for
Player Two to guess: “ << endl;
    cin >> num;
    cout << string(50, ‘\n’);
```

STEP 9

The cout << string(50, ‘\n’) line clears the screen so Player Two doesn’t see the entered number. Now you can create a do… while loop, together with if… else:

```
  do
    {
        cout << “\nPlayer Two, enter your guess: “;
        cin >> guess;
        tries++;
        if (guess > num)
        {
            cout << “\nToo High!\n” << endl;
        }
        else if (guess < num)
        {
            cout << “\nToo Low!\n” << endl;
        }
        else if (guess == num)
        {
            cout << “Well done! You got it in “ <<
tries << “ guesses!” << endl;
        }
    } while (guess != num);

    return 0;
```



STEP 10

Grab a second player, then build and run the code. Player One enters the number to be guessed, then Player Two can take as many guesses as they need to get the right number. Want to make it harder? Maybe use decimal numbers.