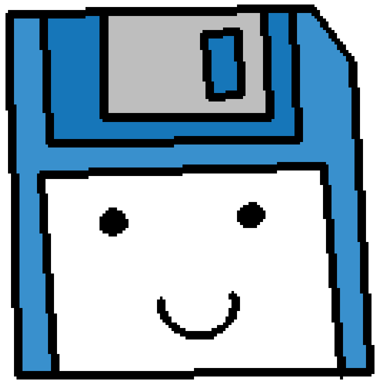
Flow of Control: Loops

Written by Rachel J. Morris, last updated 2015-01-14

Stay a-*while* and listen...



While Loops

Being able to loop through a task multiple times is part of what makes software so useful.

Ever had to go through a bunch of documents and add a tiny thing, over and over and over?

Computers handle repetitive tasks for us.

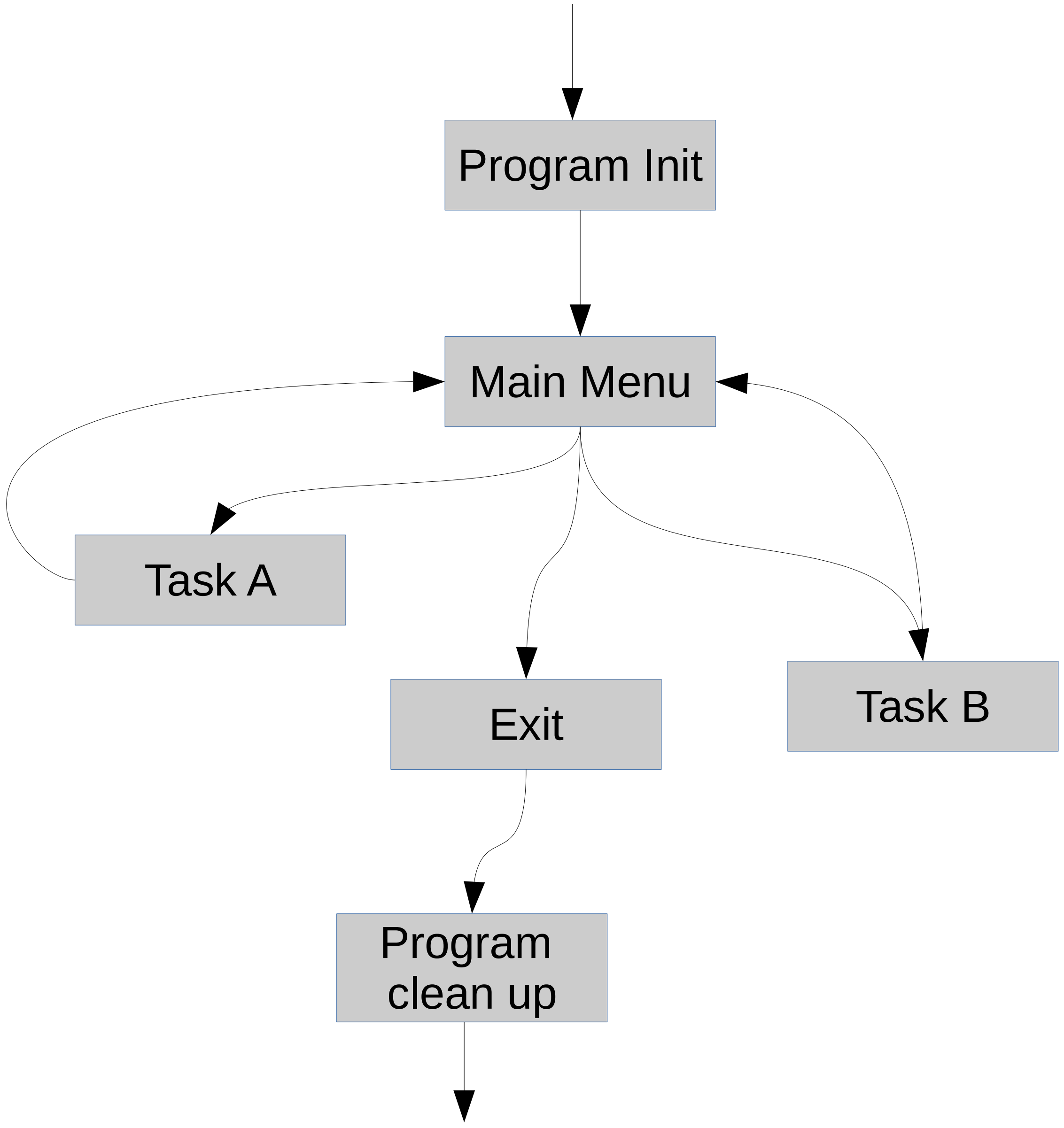
While Loops

Loops can also be used to section off portions of our program -

Without loops, a program would go from point A to point B and then quit.

But we can tell the program, “While the user has not quit”, Keep running.

While Loops



```
SAMPLE PROGRAM

OPTIONS
1. Do task A
2. Do task B
3. Quit

>> 1

((TASK A))

OPTIONS
1. Do task A
2. Do task B
3. Quit

>> 3

GOOD BYE
```

While Loops

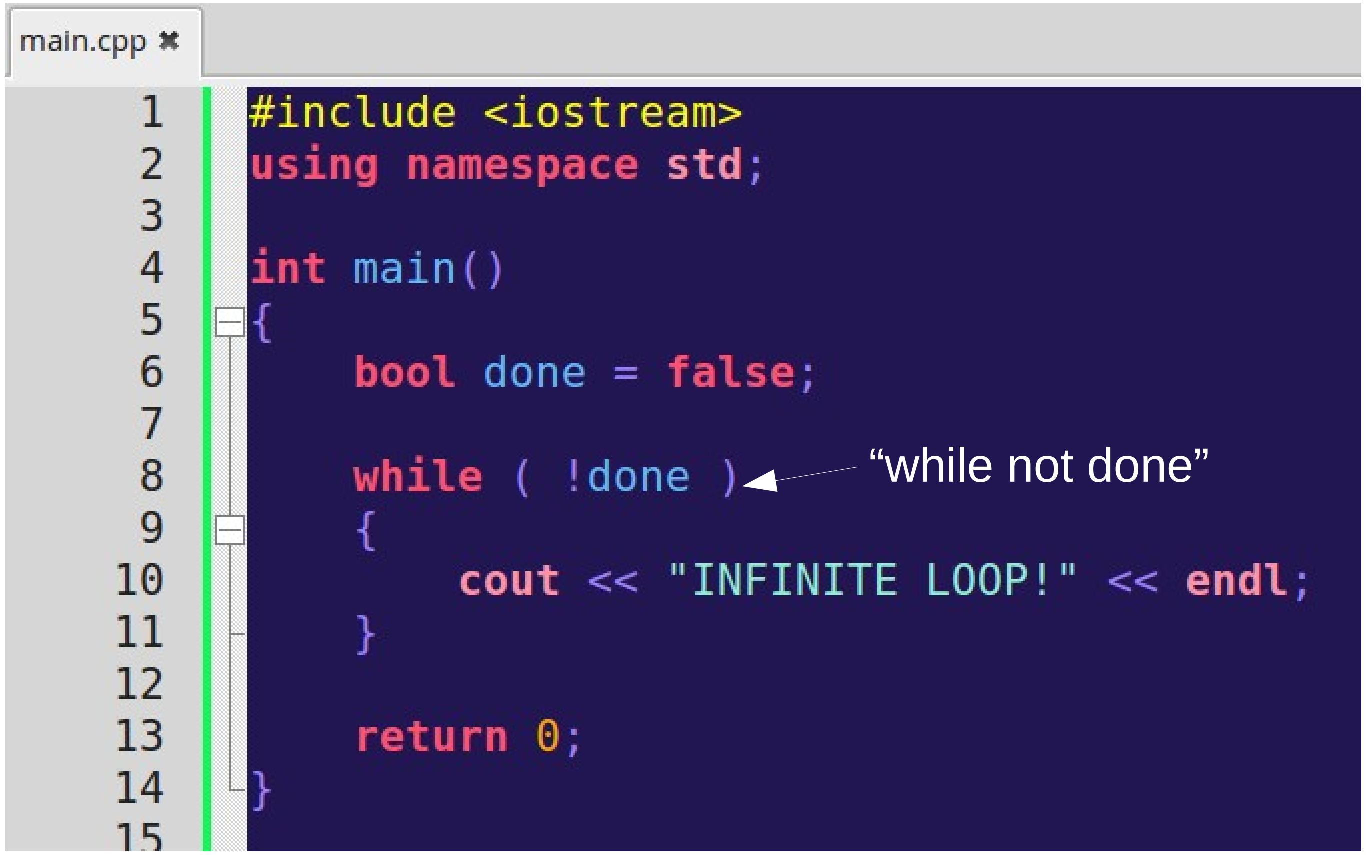
We use **Boolean Expressions** with **while loops**, to specify how it should keep looping. Once that expression is **`false`**, it will break out of the loop.

Conditional	C++
Equal to	`( var1 == var2 )`
Not equal to	`( var1 != var2 )`
Less than	`( var1 < var2 )`
Less than or equal to	`( var1 <= var2 )`
Greater than	`( var1 > var2 )`
Greater than or equal to	`( var1 >= var2 )`
True	`( var1 == true )`
True, shorthand	`( var1 )`
False	`( var1 == false )`
False, shorthand	`( !var1 )`

While Loops

Because a loop will keep looping until the given criteria is **false**, it can be very easy to write an infinite loop.

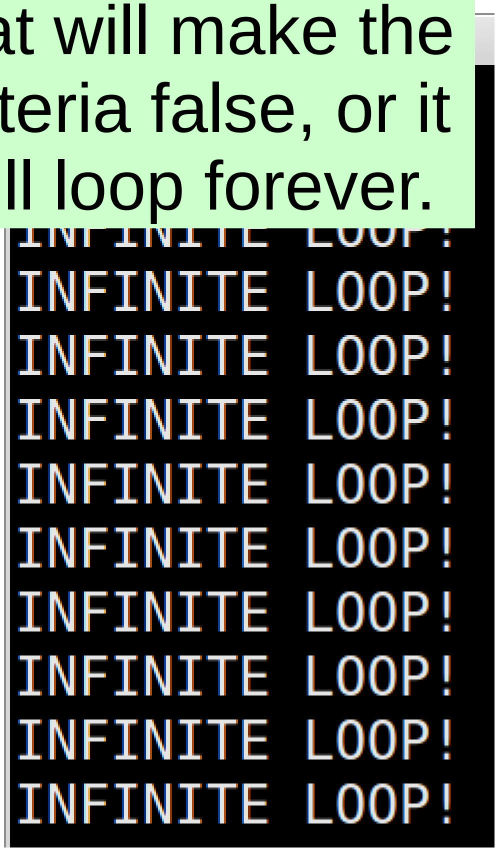
Somewhere within the **while** loop, you will need to add logic that will make the criteria false, or it will loop forever.



main.cpp

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

int main()
{
    bool done = false;

    while ( !done )
    {
        cout << "INFINITE LOOP!" << endl;
    }

    return 0;
}
```

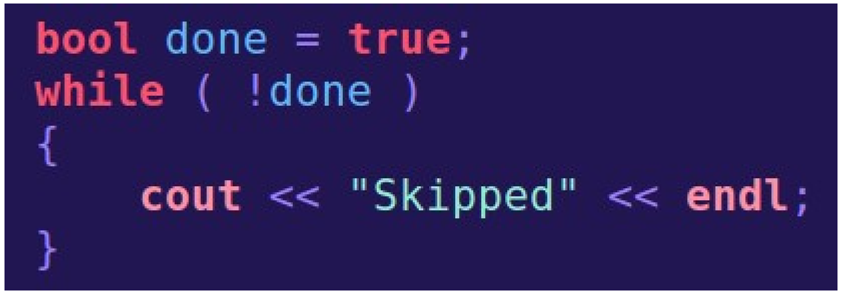
“while not done”



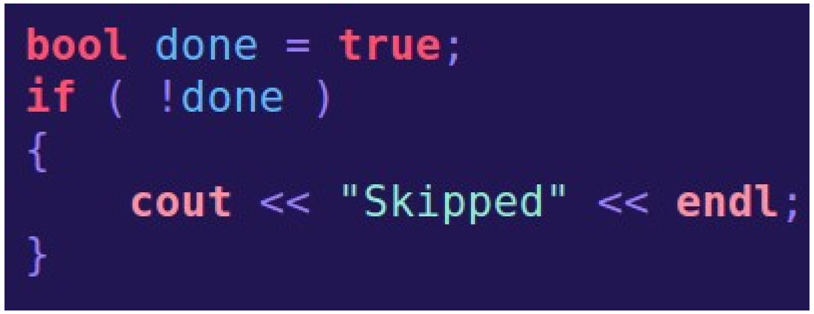
```
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
INFINITE LOOP!
```

While Loops

Like an **if statement**, if that criteria is **false** to begin with, the internal code block will be skipped.



```
bool done = true;
while ( !done )
{
    cout << "Skipped" << endl;
}
```

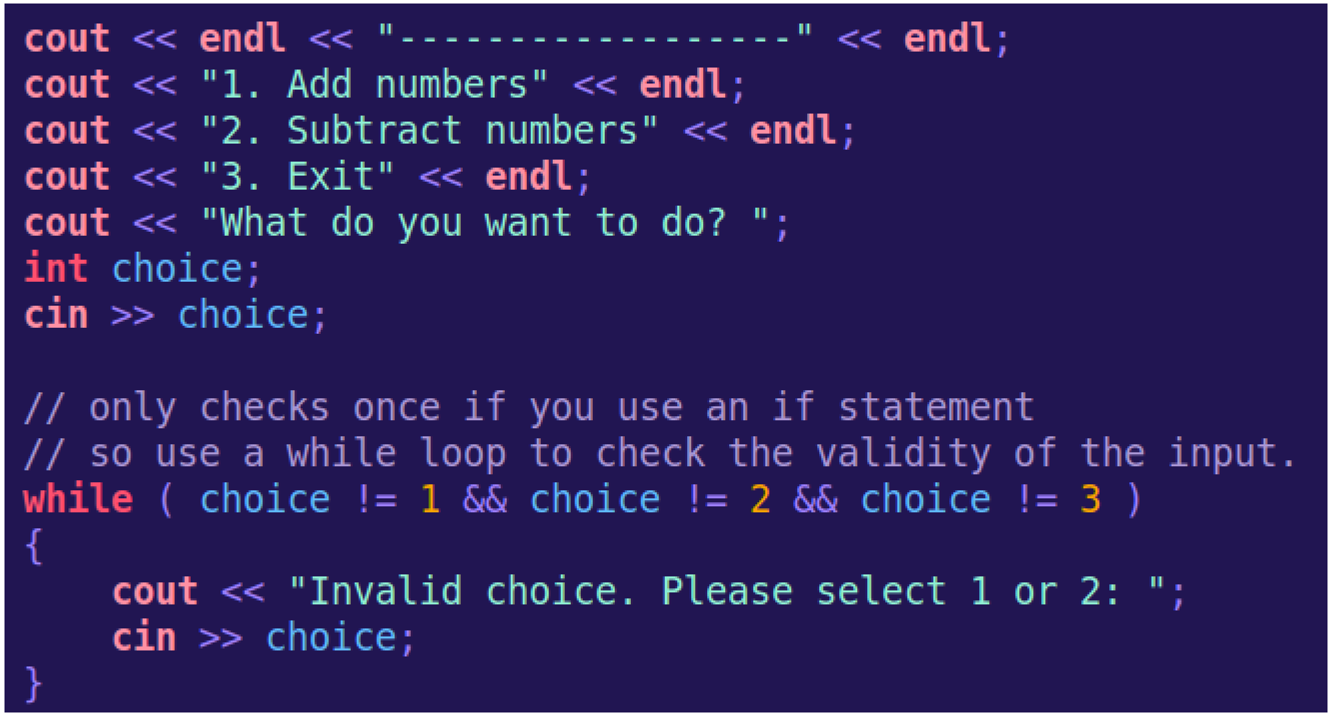


```
bool done = true;
if ( !done )
{
    cout << "Skipped" << endl;
}
```

But if the criteria is **true**, the **if statement** will only execute the internal code once, while the **while** loop will continue looping through the internal code.

While Loops

Application 1: Checking for valid input



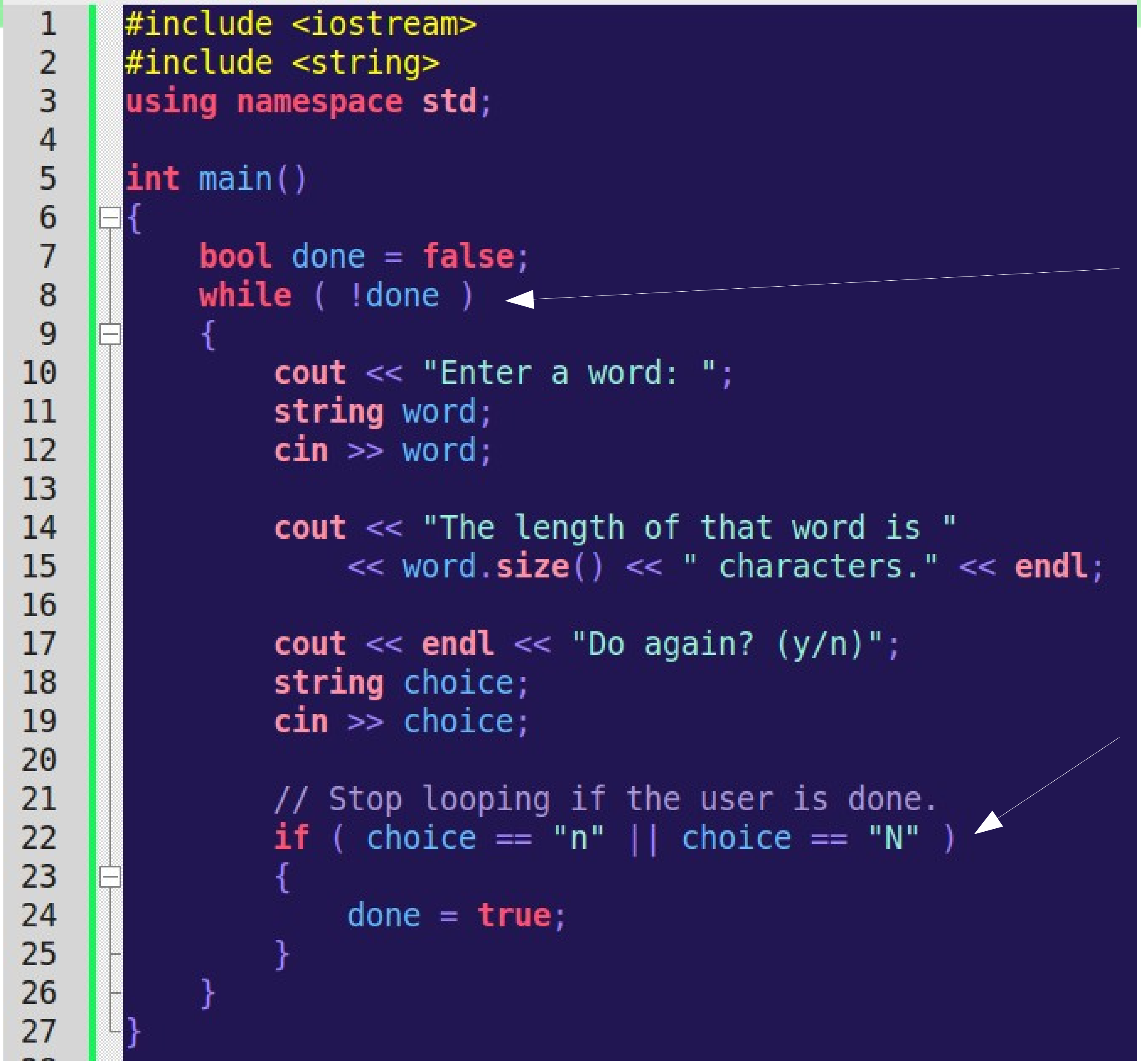
```cpp
cout << endl << "------------------" << endl;
cout << "1. Add numbers" << endl;
cout << "2. Subtract numbers" << endl;
cout << "3. Exit" << endl;
cout << "What do you want to do? ";
int choice;
cin >> choice;

// only checks once if you use an if statement
// so use a while loop to check the validity of the input.
while ( choice != 1 && choice != 2 && choice != 3 )
{
    cout << "Invalid choice. Please select 1 or 2: ";
    cin >> choice;
}
```

You can use a while loop to make sure the user inputted a valid option.

A while loop is best here, because an if statement would only check once.

If the user entered an invalid option on the 2nd try, an if statement wouldn't check it again. A while loop keeps looping until the user inputs a valid option.

While Loops



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

int main()
{
    bool done = false;
    while ( !done )
    {
        cout << "Enter a word: ";
        string word;
        cin >> word;

        cout << "The length of that word is "
            << word.size() << " characters." << endl;

        cout << endl << "Do again? (y/n)";
        string choice;
        cin >> choice;

        // Stop looping if the user is done.
        if ( choice == "n" || choice == "N" )
        {
            done = true;
        }
    }
}
```

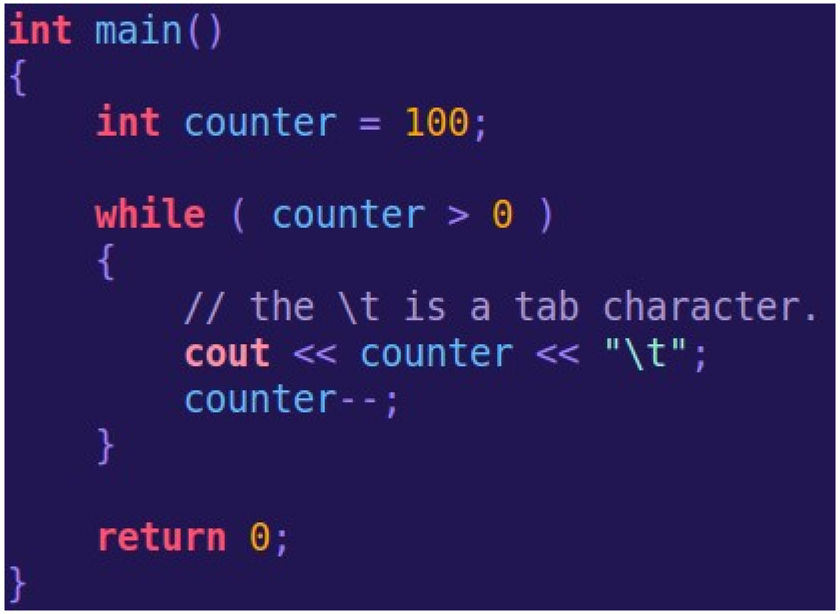
Application 2: Keep a program running

We can keep a program running until the user quits with a simple boolean.

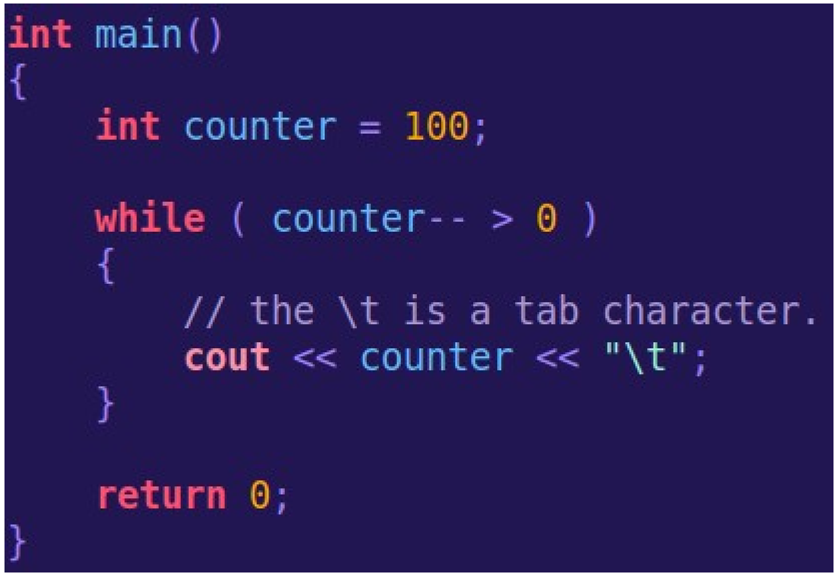
(Just make sure that you do set "done" to false at some point!)

While Loops

Application 3: Count



```
int main()
{
    int counter = 100;

    while ( counter > 0 )
    {
        // the \t is a tab character.
        cout << counter << "\t";
        counter--;
    }

    return 0;
}
```

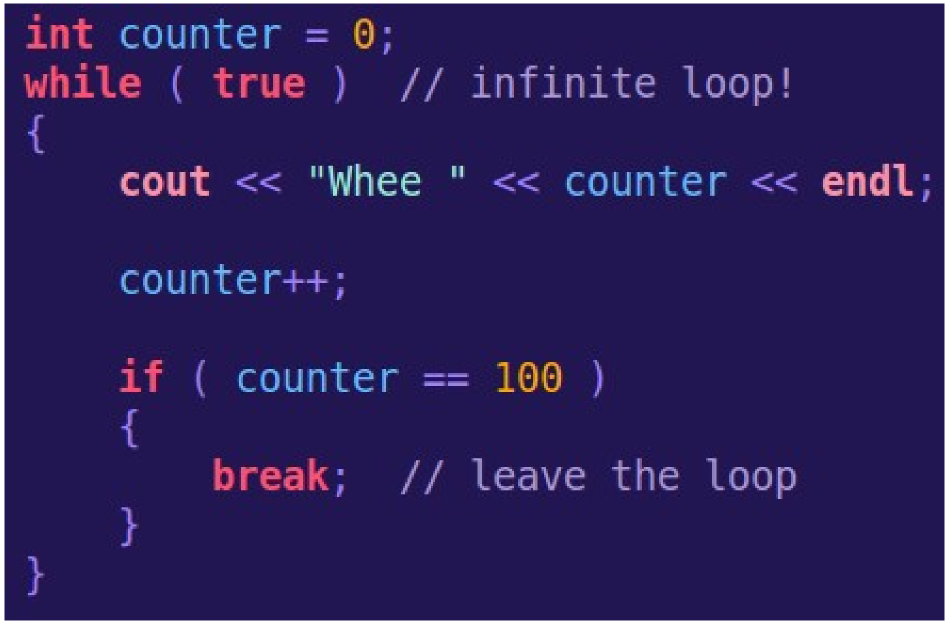


```
int main()
{
    int counter = 100;

    while ( counter-- > 0 )
    {
        // the \t is a tab character.
        cout << counter << "\t";
    }

    return 0;
}
```

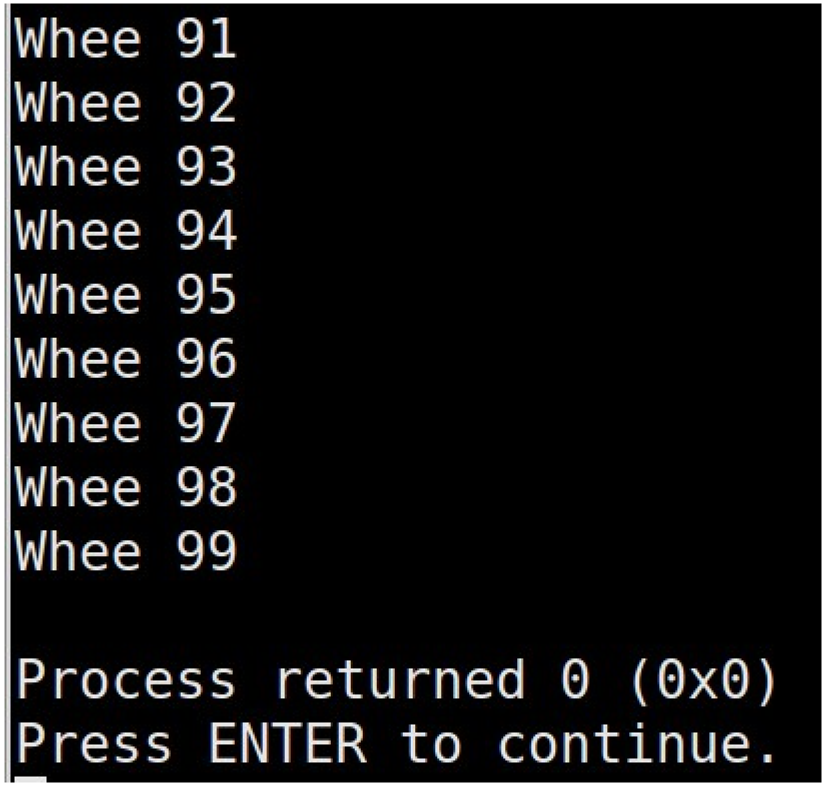
You can also use a **while** loop to do a certain task a specific amount of times.

While Loops

You can also use the **`break;`** command to break out of a **while** loop, rather than having to set its criteria to **false**.



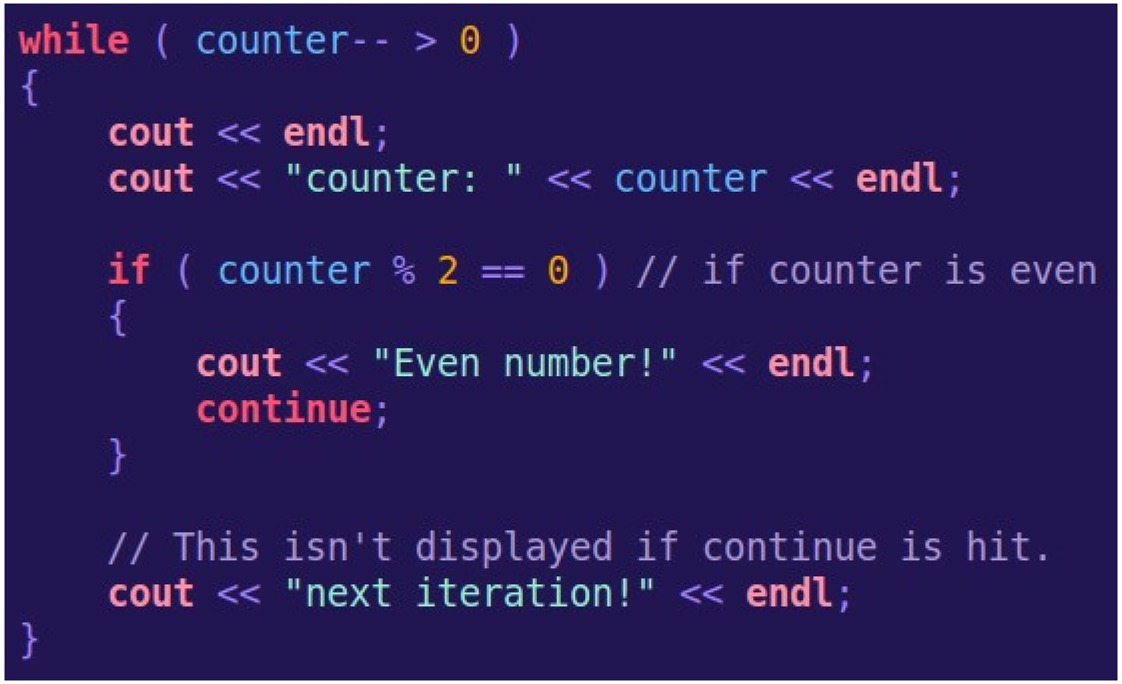
```cpp
int counter = 0;
while ( true )  // infinite loop!
{
    cout << "Whee " << counter << endl;

    counter++;

    if ( counter == 100 )
    {
        break;  // leave the loop
    }
}
```



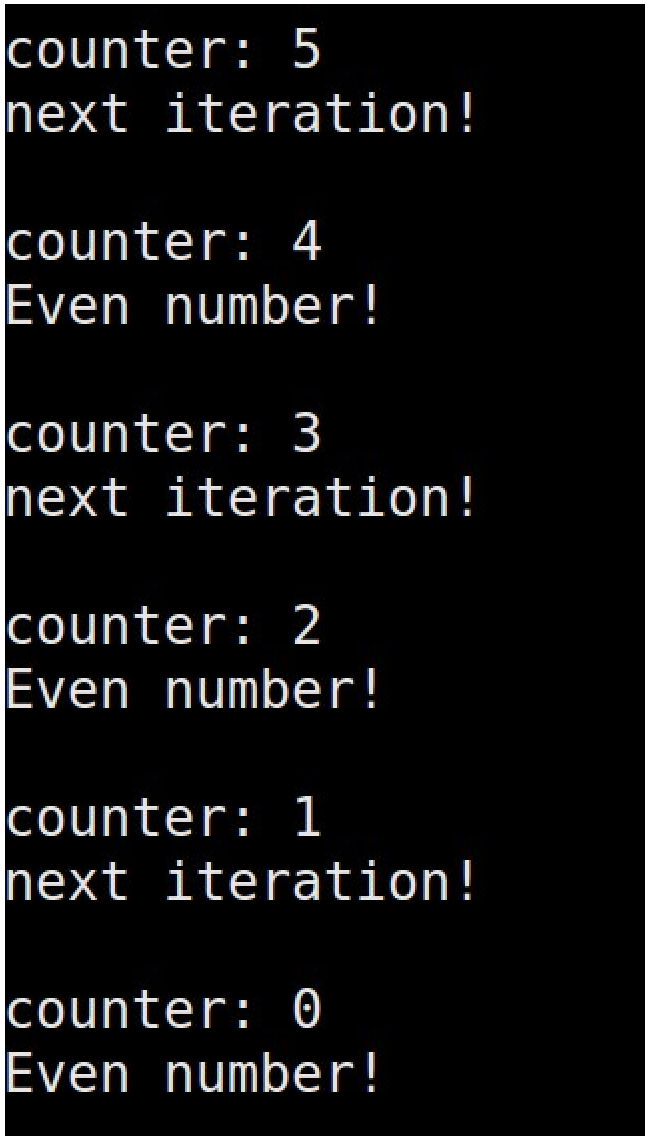
```
Whee 91
Whee 92
Whee 93
Whee 94
Whee 95
Whee 96
Whee 97
Whee 98
Whee 99

Process returned 0 (0x0)
Press ENTER to continue.
```

While Loops

There is also the **`continue`** keyword – it will skip the rest of the loop on the current iteration that you are on, but continue looping overall.



```cpp
while ( counter-- > 0 )
{
    cout << endl;
    cout << "counter: " << counter << endl;

    if ( counter % 2 == 0 ) // if counter is even
    {
        cout << "Even number!" << endl;
        continue;
    }

    // This isn't displayed if continue is hit.
    cout << "next iteration!" << endl;
}
```

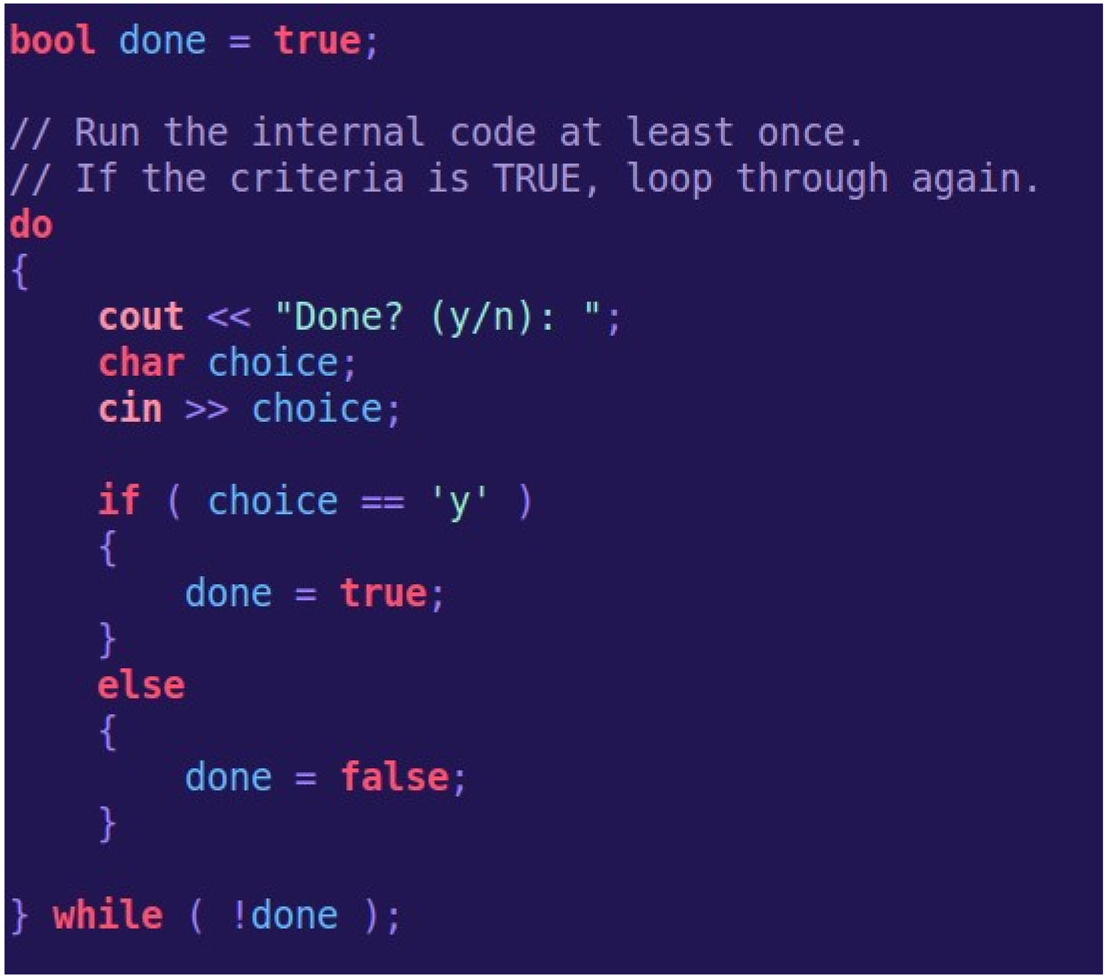


```
counter: 5
next iteration!

counter: 4
Even number!

counter: 3
next iteration!

counter: 2
Even number!

counter: 1
next iteration!

counter: 0
Even number!
```

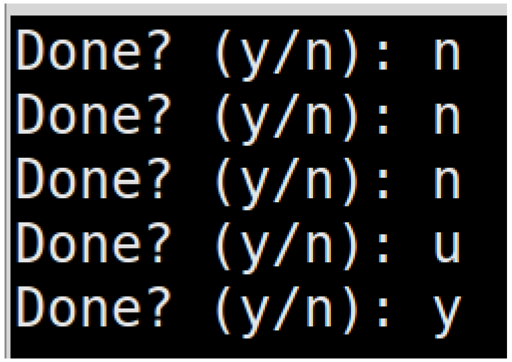
Do While Loops

A **do while** loop is similar to a **while** loop, except that a **do while** loop will always be executed ***at least once***.

After the first run through the loop, if the criteria is **false**, it will exit the loop.



```
bool done = true;

// Run the internal code at least once.
// If the criteria is TRUE, loop through again.
do
{
    cout << "Done? (y/n): ";
    char choice;
    cin >> choice;

    if ( choice == 'y' )
    {
        done = true;
    }
    else
    {
        done = false;
    }

} while ( !done );
```



```
Done? (y/n): n
Done? (y/n): n
Done? (y/n): n
Done? (y/n): u
Done? (y/n): y
```

For Loops

A **for loop** is a type of loop that is best suited for counting.

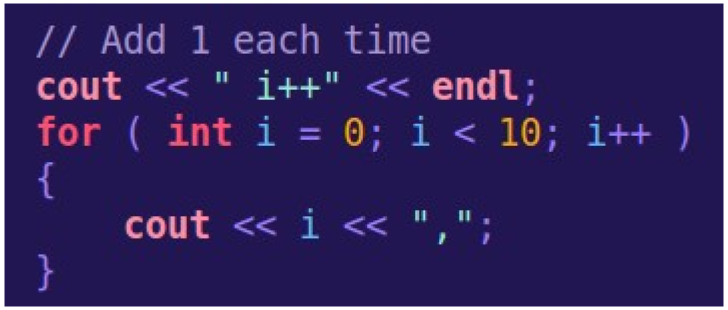
You specify a begin point, end point, and increment (or decrement) amount, and the for loop will loop for the appropriate amount of time.

Starting point　　Loop until　　Increment Amount

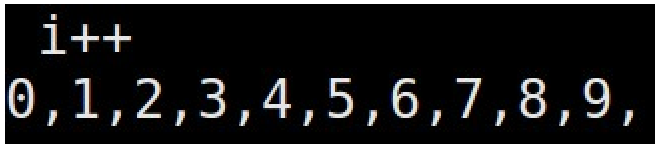
```
for ( int i = 0; i <= 10; i++ )
{
    cout << i << endl;
}
```

For Loops

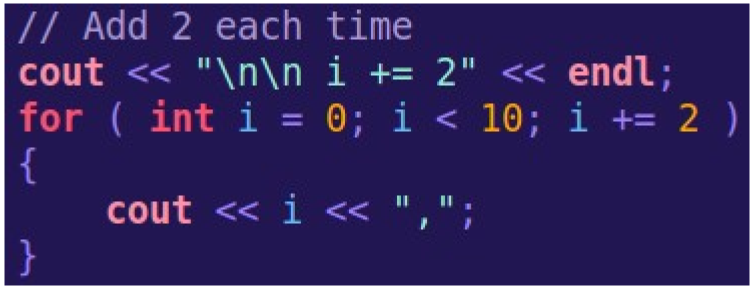
You can use **i++** or **++i** to add 1 to **i** each time through the loop,
Or you can use a **+=** to add a different amount



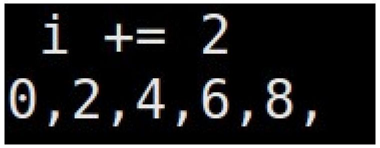
```
// Add 1 each time
cout << " i++" << endl;
for ( int i = 0; i < 10; i++ )
{
    cout << i << ",";
}
```



```
 i++
0,1,2,3,4,5,6,7,8,9,
```



```
// Add 2 each time
cout << "\n\n i += 2" << endl;
for ( int i = 0; i < 10; i += 2 )
{
    cout << i << ",";
}
```

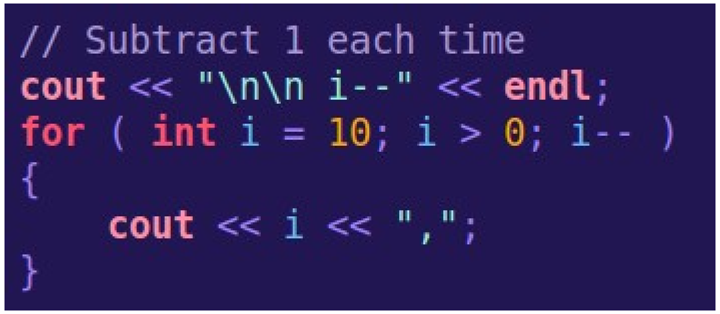


```
 i += 2
0,2,4,6,8,
```

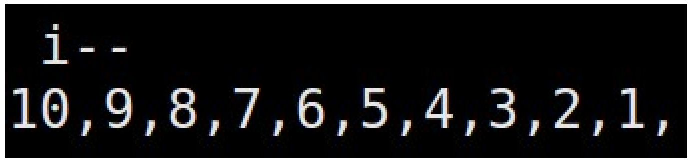
But you don't have to stick with just adding...

For Loops

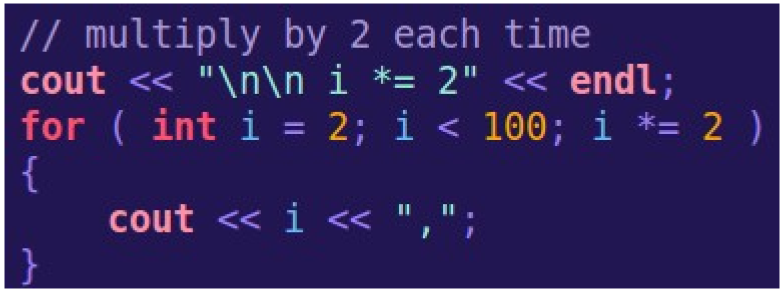
You can also use other math statements, such as
-= (subtract by), ***=** (multiply by), and **/=** (divide by)



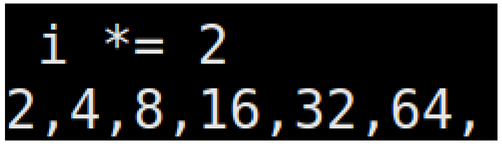
```
// Subtract 1 each time
cout << "\n\n i--" << endl;
for ( int i = 10; i > 0; i-- )
{
    cout << i << ",";
}
```



```
 i--
10,9,8,7,6,5,4,3,2,1,
```

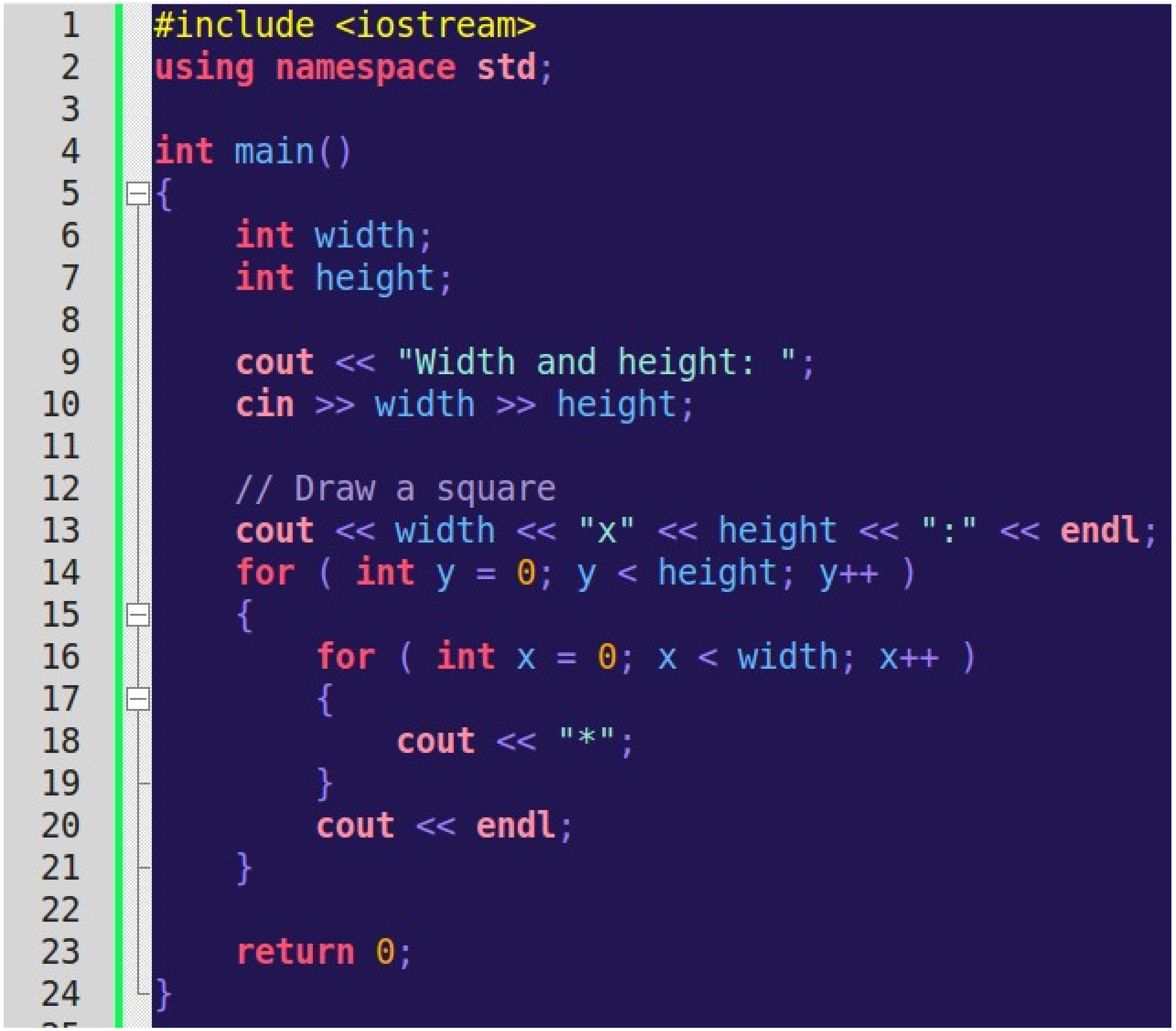


```
// multiply by 2 each time
cout << "\n\n i *= 2" << endl;
for ( int i = 2; i < 100; i *= 2 )
{
    cout << i << ",";
}
```



```
 i *= 2
2,4,8,16,32,64,
```

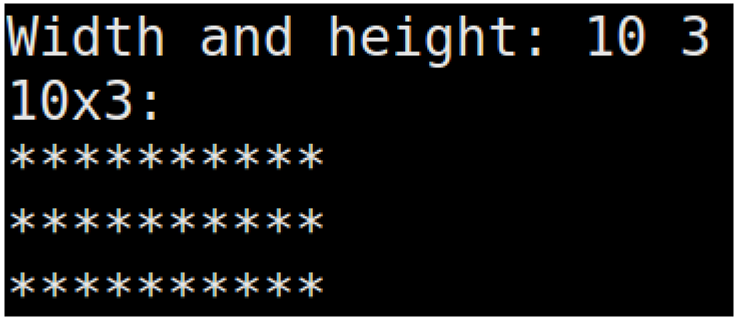
Nested Loops



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

int main()
{
    int width;
    int height;

    cout << "Width and height: ";
    cin >> width >> height;

    // Draw a square
    cout << width << "x" << height << ":" << endl;
    for ( int y = 0; y < height; y++ )
    {
        for ( int x = 0; x < width; x++ )
        {
            cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

And like if statements, you can also nest loops within loops.

You can also nest if statements within loops, and vice versa.



```
Width and height: 10 3
10x3:
**********
**********
**********
```

Got a lot of loops to make now!