Chapter 6: Loops

Repetition is a key element of any program, whether that’s just to repeat a function several times or encompass the entire program in, such as a game loop. This chapter is going to look at the 3 loops found within C++ and how they are used.

Before we get to some example code, look at Table 6.1: Loop Types. As you can see there are different types of loops. You will use different loops in different scenarios, but it is possible to use any of the listed loops with a little extra coding.

The main difference between a **for loop** and the others is that with a **for loop** you state how many iterations you wish to carry out, the others require a condition to be met to exit. The difference between a **while loop** and a **do while loop** is that a **do while loop** will always run at least once, whereas a **while loop** must meet a condition to run at all.

|  |  |
| --- | --- |
| **Loop Type** | **Description** |
| for loop | Execute a sequence of statements a set number of times. |
| while loop | Repeat a sequence of statements while a given condition is true. The condition is tested before executing the body of the loop. |
| do… while loop | Similar to the while loop, but the body of the loop is always executed once as the condition is not checked until the end of the loop. |

Table 6.1: Loop Types

**For Loop**

As described above, a **for loop** is used when you know the number of iterations required. If I want to do something 10 times, a **for loop** is what I need. If I want to loop until the player dies, a **for loop** is no use to you.

6.1 Example: for loop

A **for loop** takes the following format:

for ( Initialise count variable; Condition to meet to exit loop; Increment counter )

So lets break down those sections. It may look a little scary, but you have in fact already covered all of these components in previous chapters.

For the initialisation of the count variable we need to state the type, name the variable and give it a starting value. Something along these lines:

int i = 0;

Look familiar? Her we are saying we want a variable called ‘i’, which is on integer type and starts with the value 0.

Next we need a condition to exit the loop. In my example I said that I wanted to loop 10 times, so lets write the code for a conditional check for when my variable becomes greater than 10.

i > 10;

In the previous chapters we obviously used the **if statement**, but within a **for loop** this is implicit.

Finally we need to increment the counter. This could be by 1, 2, 10, whatever we like. My example wants to loop 10 times, so if the count starts at 0 I need to add 1 each time around the loop.

i++;

As I mentioned it, here is how you would increment by another value. 2 in this example.

i += 2;

Remember that the increment **operator ++** adds the value after use. This is the same here. The first pass through the loop will be its starting value. In our case it will be 0 on the first pass through the loop. At the end of the loop the count gets incremented.

Putting these components together into the f**or loop** looks like this:

for ( int i = 0; i < 10; i++ )

{

// Put the code you wish to repeat here.

}

We use curly braces to encapsulate the entire body of the loop. However, similar to **if statements**, if you only have one line of code in your loop you can omit the braces.

for ( int i = 0; i < 10; i++ )

cout << “i = “ << i << endl;

It is also possible to have your loop count down. You need to set your staring value to be higher than the condition and use the **-- operator**:

for ( int i = 10; i > 0; i-- )

cout << “i = “ << i << endl;

**While Loop**

A while loop differs from the **for loop** in that we don’t know how many loops we are going to do of the enclosed code. A while loop will only exit when a condition has been met.

6.2 Example: while loop

A **while loop** takes the following format:

while ( Condition is true )

{

//Do this code.

}

The condition can be in any of the formats covered in Chapter 5: Conditionals. The important point to remember is that you need to have some way of changing this condition from within the loop.

An example of a gameloop that goes until the player dies could look something like this:

bool playerAlive = true;

while ( playerAlive == true )

{

//Fictitious function to check if player dies.

if( PlayerDies() )

playerAlive = false;

}

As an example of how a while loop could be used in the same scenario as a for loop we are going to recreate the for loop example, but using a while loop.

int i = 0;

while ( i< 10 )

{

cout << “i = “ << i << endl;

i++;

}

**Do While Loop**

A do while loop differs from the **while loop** in that the condition is checked at the end of the loop. This means that it will execute the code at least once. It would be entirely possible for a **while loops** condition to be false and the body of the loop get skipped over and never executes.

6.3 Example: do while loop

A **do while loop** takes the following format:

do

{

//Do this code.

} while ( Condition is true ) ;

Recreating the player alive example from above, the following shows how a do while loop could be used. Remember though that this code will run at least once. What would happen if the player was in fact already dead when this loop was reached??

bool playerAlive = true;

do

{

//Fictitious function to check if player dies.

if( PlayerDies() )

playerAlive = false;

} while ( playerAlive == true ) ;

Note: Do not forget to add the semi-colon after the while condition. Syntax errors will result if it is missing.

**Nested Loops**

As loops are valid code statements it is entirely possible to have loops within loops. These are termed **nested loops**. You may think this unlikely, but imagine you are writing a game that has a game loop, it is more than likely that you will have other loops within this game loop.

This is an example of a **nested for loop**:

for ( int i = 0; i < 10; i++ )

{

// Output the current line.

cout << “Line: “ << i << endl;

// Nested loop to output 10 numbers.

for ( int j = 0; j < 10; j++ )

cout << j << “ “;

// Move on to the line below.

cout << endl;

}

**Controls Statements**

At times you may wish to exit a loop before a condition is met. For example, if you had a for loop that iterated 10 times, but on the 3rd iteration you need to exit the entire loop you could use the **break** **control statement**. Similarly, lets say you need to skip the remainder of the code in a loop and move on to the next iteration, there is a **control statement** for that too - **continue**. Take a look at Table 6.2: for an overview.

|  |  |
| --- | --- |
| **Control Statement** | **Description** |
| break | Terminates the loop or switch statement and moves execution to the statement immediately following the loop or switch. |
| continue | Causes the loop to skip the remainder of its body and immediately retest the loop condition before reiterating through the loop body. |

Table 6.2: Control Statements

6.1 Example: break statement

So we have decided that on a condition met within the loop we need to stop executing the loop. Simply put the **break** keyword at the required place. Remember the switch statement in Chapter 5: Conditionals? **Break** is used to get out of that statement at the required time too.

This example takes the **for loop** example from above and allows us to **break** out of the loop when i reaches the value of 3. There is no logical reason for this, it is purely to demonstrate how **break** works:

for ( int i = 0; i < 10; i++ )

{

// Break out of this loop

if ( i == 3 )

break;

}

6.1 Example: continue statement

For the **continue** example we will be writing the example posed earlier. We will construct a for loop that iterates 10 times, but if i is 3 we will skip that iteration.

for ( int i = 0; i < 10; i++ )

{

// Skip iteration 3 of this loop

if ( i == 3 )

continue;

//Output i so we can see if 3 gets output.

cout << “i = “ << i << endl;

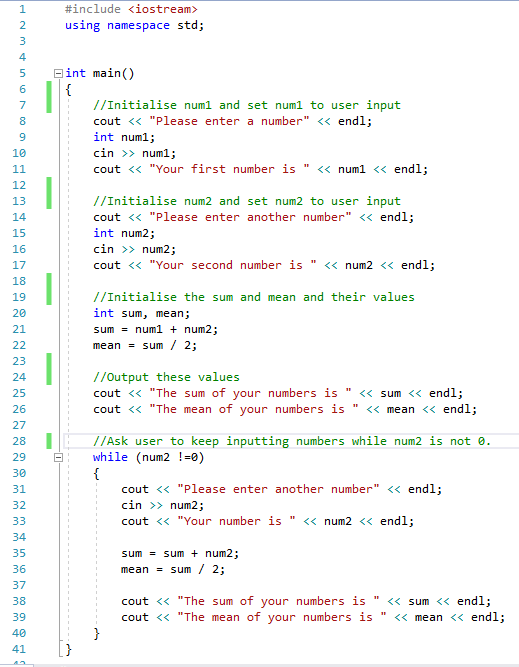
}

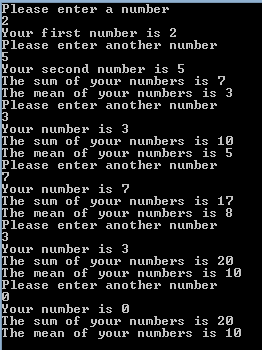
**Program 12: 10 Numbers**

This is for you to complete. You need to add the code listing and screen shot for this program here. It is crucial that you do this to keep your portfolio as a useful resource as the end of year test will require a selection of these programs.

Write a program

1. That asks to user to enter 2 numbers.
   1. Using those 2 numbers, calculate the *sum* (total) of the numbers and calculate the *mean* value of those numbers.
   2. Finally, print out the *sum* and *mean* values to the console.
2. Ask the user to enter another number.
   1. Calculate the *sum* and *mean* with the current total and output in the same way as in step 1.
3. Continue to ask the user indefinitely until the user enters a value of zero.





**Program 13: Starry Output**

This is for you to complete. You need to add the code listing and screen shot for this program here. It is crucial that you do this to keep your portfolio as a useful resource as the end of year test will require a selection of these programs.

Write a C++ program

1. Which asks the user for a number *n* between 1 and 10. The program should then print out *n* lines. Each should consist of a number of stars of the same number as the current line number. For example:

Please enter a number: 5

\*

\* \*

\* \* \*

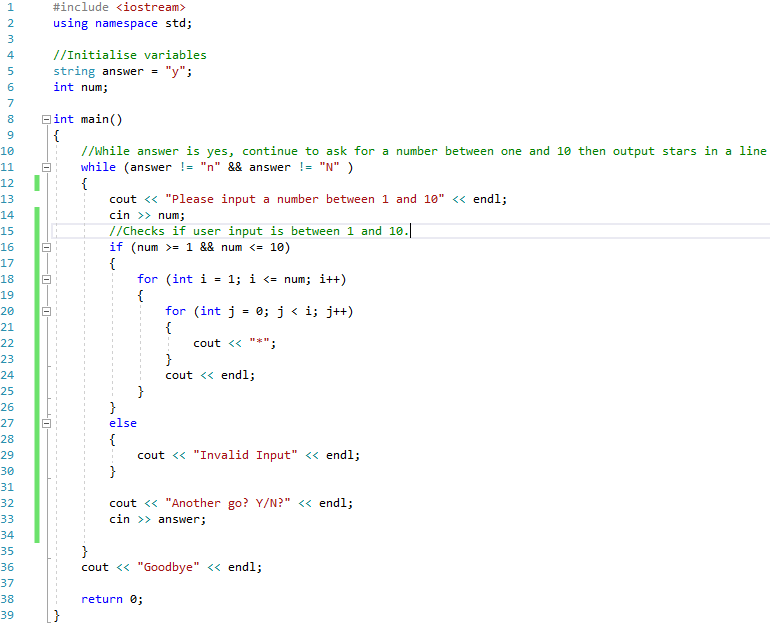
\* \* \* \*

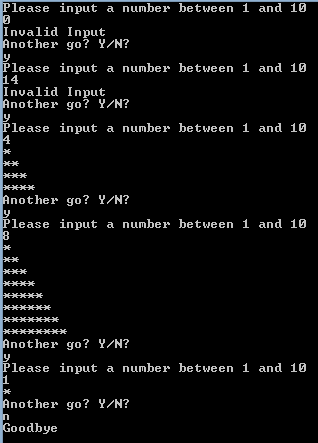
\* \* \* \* \*

1. The user should then be asked if they wish to have another go [Y or N].

* An input of ‘Y’ will
  + Loop the player back to number 1 above,
* An input of ‘N’ will
  + Output a line of text saying “Goodbye.”
  + Pause for a brief second so the user can read the text
  + Then quit.

Hint: you will need to use nested loops!





**Program 14: Early Exit**

This is for you to complete. You need to add the code listing and screen shot for this program here. It is crucial that you do this to keep your portfolio as a useful resource as the end of year test will require a selection of these programs.

Write a C++ program, which creates an integer called *playerLife*, sets it to a starting value of 100 and outputs this to the console screen.

The program should ask the user how much health to decrease the player life by with the following output:

cout << “Question “ << i << “: How much should we decrease the health by? “ endl;

It should ask this question 10 times, unless the health value goes below 0.

In the event of the *playerLife* variable going below zero before the question is asked 10 times, the following should be output

cout << “Early exit. “ << endl;

otherwise the program should output:

cout << “All 10 questions asked and player still has life remaining.” << endl;

