# Week 6: Repetition Structures Part Two

**Objectives**

Following completion of this topic, students should be able to:

* Understand for, while and do-while statements in C++
* Understand the relationship between the repetition structures
* Understand nested loops

When a segment of code needs to be executed 5 times, for example, rather than writing 5 blocks of the same code, a repetition structure can be used. Repetition structures allow the same code segment to be executed continuously. Repetition stops when some condition has been met.

The three repetition structures in C++ are for, while and do-while statements. It is possible to use any of the 3 repetition structures to perform the same task. However, most often one particular structure is *more* suited to the task at hand. For example;

* If the loop is a *counted* loop, i.e. it iterate *n* times, then a for loop is the ideal loop as this is its intent. In practice, it is also the least error prone loop to code.
* If termination of the loop were dependent upon code within the loop, then a while loop would be best suited. However, if your code *must* execute at least once, then a do-while loop would be the one to select.

**while Statement:**

The syntax of the while statement is:

while (condition)

statement;

where condition is a logical expression and statement is any executable statement.

* First, the condition is evaluated and if the result is **true** then the statement is executed. This continues repeatedly *while* the condition is **true**.
* If the condition evaluates to **false**, program flow falls to the next statement immediately following the while statement.
* If more than one statement is to be executed, they must be contained within braces {…}.

**do-while Statement:**

The syntax of the do-while statement is:

do

statement;

while (condition);

where condition is a logical expression and statement is any executable statement.

* First, the statement will be executed, and then the condition will be evaluated. This will continue repeatedly *while* the condition evaluates to **true**.
* If the condition evaluates to **false**, program execution falls to the next statement immediately following the do-while statement.
* The do-while statement works in a similar way to the while statement except that the condition is evaluated at the *end* of the loop instead of at the *beginning*. Hence, the statement will *always* be executed at least once.

**for Statement:**

The syntax of the for statement is:

for (initialisation; condition; update)

statement;

where initialisation, condition and update are optional expressions (*Note: the ‘****;****’ are still required though*) and statement is any executable statement. However, other loops should be used if all three expressions are not present. The three-part (initialisation; condition; update) controls the loop.

* The initialisation expression is used to declare and/or initialise the control variable(s) for the loop. It is evaluated *once* and is executed *first*.
* The condition expression is used to determine if the statement will be executed.
  + - It is evaluated immediately *after* initialisation.
    - If the condition evaluates to **true**, the statement will be executed.
    - If the condition evaluates to **false**, program flow will fall to the next statement immediately following the for statement.
    - The condition is *further* evaluated each time *after* the update expression has been executed.
* The update expression is used to update the control variable(s). It is evaluated immediately *after* the statement has been executed.
* If more than one statement is to be executed, they must be contained within braces {…}.
* *Style note: All code contained within a for statement* ***must*** *be indented.*

##### Task 1. The do-while Statement

1. Write a program ‘*task1a.cpp’* that accepts an integer value as the grade and displays the grade. The output from the program should be similar to: (*Text in* ***bold*** *is user input.)*

Enter the grade: **100**

The grade entered was 100.

1. Compile and test your program to ensure that its behaviour is correct.
2. Modify the program, using a do-while statement, to continuously get input and display the grade *while* a valid grade is entered. I.e. when an invalid grade is entered the loop terminates. An invalid grade is any value less than 0 or greater than 100. *Note:* *Ensure that your program only prints the grade* ***if*** *it is valid.*
3. Modify the program so that the user is alerted when an invalid grade has been entered then compile and test your program to ensure that it behaves correctly.
4. Copy the program to ‘*task1b.cpp’* and rewrite the loop as a while-do. You will find it easier to handle the termination condition.
5. Modify the program so that the loop is *sentinel* controlled. In this case, make it ‘999’. Thus other invalid grades will not terminate the program, but merely report an illegal grade. The sentinel should not be reported.

##### Task 2. The for Statement

1. Consider the following code. ***Note: Do not implement it yet.***

#include <iostream>

using namespace std;

const int MAX = 5;

int main()

{

int num = 0;

cout << "Number\tSquare\tCube" << endl

<< "------\t------\t----" << endl;

for(num = 1; num <= MAX; num++)

cout << num << '\t'

<< num \* num << '\t'

<< num \* num \* num << endl;

return 0;

}

1. What does the above program do? *Note: Explain in a few words*.
2. How many times will the for loop iterate? What do you think the output will be? Write it below.
3. Implement the code from step (a) as ‘*task2.cpp’* then compile and run the program. Compare the output to your answer from step (c). If there is a difference, first check the code to ensure that you have typed it correctly, otherwise review the code to ensure that you can read and interpret code effectively.
4. Modify the program to produce a table of numbers from 5 to 1, instead of 1 to 5.
5. Modify the program to produce a table of the numbers 0 through *N* with their squares and cubes, where *N* is input by the user.

##### Task 3. Nested Loops

In many situations, it is necessary or convenient to use a loop contained within another loop. Such loops are called nested loops and most often, though not always, a for loop is used. Many levels of nested-ness are allowed.

1. Consider the following code. **Do not implement it yet.**

#include <iostream>

using namespace std;

const int MAXI = 5;

const int MAXJ = 4;

int main()

{

int i, j;

for(i = 0; i <= MAXI; i++)

{

cout << i;

for (j = 0; j <= MAXJ; j++)

cout << j;

cout << endl;

}

return 0;

}

1. What will be the output from the above program?
2. How many times will the outer loop (i) iterate? How many times will the inner loop (j) iterate?
3. Implement the program as ‘*task3.cpp’* and compare the output to your answer in (b).
4. Modify the loop code, adding and initializing the variables in bold.

for(i = 0; i <= MAXI; i++)

{

**outerLoop++;**

cout << i;

for (j = 0; j <= MAXJ; j++)

{

**innerLoop++;**

cout << j;

}

cout << endl;

}

**cout << “Outer: ” << outerLoop << endl;**

**cout << “Inner: ” << innerLoop << endl;**

1. Compile and run the code to verify your answer in (c).

##### Task 4. Relationships Between Repetition Structures

It is possible to use any of the three repetition structures to perform the same task. Modification of the control variables is *often* all that is required.

1. Implement, compile and run the following program as ‘*task4a.cpp’*.

#include <iostream>

using namespace std;

int main()

{

int number = 0, total = 0;

cout << "Enter a number: ";

cin >> number;

while (number != 0)

{

if (number % 2 != 0)

total++;

number--;

}

cout << "Result: " << total << endl;

return 0;

}

1. What does the above program do?
2. Modify the program by replacing the while statement with a for statement. Run the program to ensure that it produces the same output as in step (a).
3. Copy the program to ‘*task4b.cpp’* and modify the program by replacing the while statement with a do-while statement. Run the program to ensure that it produces the same output as in steps (a) and (c).

##### Task 5. ASCII ART

1. Modify the program from TASK1 so that it automatically exits the loop after 5 consecutive invalid grades have been entered **or** the *sentinel* has been entered. Compile and test your program.
2. Write a program that accepts a number and produces a pattern as shown in the following sample outputs.

Line = 5

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

Line = 6

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

III. Write a program that accepts a number and produces a pattern as shown in the following sample outputs.

Line = 3

\*.\*

.\*

\*

Line = 8

.\*.\*.\*.\*

\*.\*.\*.\*

.\*.\*.\*

\*.\*.\*

.\*.\*

\*.\*

.\*

\*

IV. Write a program that accepts a number and produces a pattern as shown in the following sample outputs.

Line = 4

ABBA

B B

B B

ABBA



Line = 8

ABCDDCBA

BCD DCB

CD DC

D. D

D D

CD DC

BCD DCB

ABCDDCBA

Line = 5

Sorry, I need an even input!