ITERATION STATEMENTS

C Loops

In programming, we often have to perform an action repeatedly, with little or no variations in the details each time they are executed. This need is met by a mechanism known as a loop.

The versatility of the computer lies in its ability to perform a set of instructions repeatedly. This involves repeating some code in the program, either a specified number of times or until a particular condition is satisfied. Loop-controlled instructions are used to perform this repetitive operation efficiently, ensuring the program doesn't look redundant at the same time due to the repetitions.

Following are the three types of loops in C programming:

- For loop
- While loop
- Do-while loop

Types of Loops

Entry Controlled loops

In entry controlled loops, the test condition is evaluated before entering the loop body. The for loop and the while loop are examples of entry-controlled loops.

Exit Controlled Loops

In exit-controlled loops, the test condition is tested at the end of the loop. Regardless of whether the test condition is true or false, the loop body will execute at least once. The do-while loop is an example of an exit-controlled loop.

For Loop

A for loop is a repetition control structure that allows us to efficiently write a loop that will execute a specific number of times. The for loop working is as follows:

- The initialization statement is executed only once; in this statement, we initialize a variable to some value.
- In the second step, the test expression is evaluated. Suppose the test expression is evaluated to be true. In that case, the for loop keeps running, and the test expression is re-evaluated, but if the test expression is evaluated to false, then the for loop terminates.

- The loop keeps executing until the test expression is false. When the test expression is false, then the loop terminates.

While Loop

The while loop is called a pre-tested loop. The while loop allows code to be executed multiple times, depending upon a boolean condition that is given as a test expression. While introducing for loops, we saw that the number of iterations is known, whereas while loops are used in situations where we do not know the exact number of iterations of the loop. The while loop execution is terminated based on the test condition which evaluates to either true or false.

Do-while Loop

In do-while loops, the execution is terminated based on the test condition, very similar to how it is done in a while loop. The main difference between the do-while loop and while loop is that, in the do-while loop, the condition is tested at the end of the loop body, whereas the other two loops are entry-controlled. In a do-while loop, the loop body will execute at least once irrespective of the test condition.

Sometimes, while executing a loop, it becomes necessary to jump out of the loop. For this, we make use of the break statement and the continue statement.

Break Statement

Whenever a break statement is encountered in a loop, the loop is terminated regardless of what kind of loop we are in and the program continues with the statement following the loop.

Continue Statement

Whenever a continue statement is encountered in a loop, it will cause the control to go directly to the test condition which is where the loop starts, ignoring any piece of code following the continue statement.

while Loop

A While loop is also called a pre-tested loop. A while loop allows a piece of code in a program to be executed multiple times, depending upon a given test condition which evaluates to either true or false. The while loop is mostly used in cases where the number of iterations is not known. If the number of iterations is known, then we could also use a for loop.

The syntax for using a while loop:

```
while (condition test){    // Set of statements}
```

The body of a while loop can contain a single statement or a block of statements. The test condition may be any expression that should evaluate as either true or false. The loop iterates while the test condition evaluates to true. When the condition becomes false, the program control passes to the line immediately following the loop, which means it terminates.

One such example to demonstrate how a while loop works is:

```
#include <stdio.h>


int main()
{
   int i = 0;
   while (i <= 5)
   {
      printf("%d ", i);
      i++;
   }
   return 0;
}
```

Output

0 1 2 3 4 5

Properties of the while loop:

Following are some of the properties of the while loop.

- A conditional expression written in the brackets of while is used to check the condition. The set of statements defined inside the while loop will execute until the given condition returns false.
- The condition will return 0 if it is true. The condition will be false if it returns any nonzero number.
- In the while loop, we cannot execute the loop until we do not specify the condition expression.
- It is possible to execute a while loop without any statements. This will give no error.

- We can have multiple conditional expressions in a while loop.
- Braces are optional if the loop body contains only one statement.

do-while Loop

A do-while loop is a little different from a normal while loop. A do-while loop, unlike what happens in a while loop, executes the statements inside the body of the loop before checking the test condition.

So even if a condition is false in the first place, the do-while loop would have already run once. A do-while loop is very much similar to a while loop, except for the fact that it is guaranteed to execute the body at least once.

Unlike for and while loops, which test the loop condition first, then execute the code written inside the body of the loop, the do-while loop checks its condition at the end of the loop.

Following is the syntax of the do-while loop.

do{ statements;} while (test condition);

If the test condition returns true, the flow of control jumps back up to the do part, and the set of statements in the loop executes again. This process repeats until the given test condition becomes false.

How does the do-while loop work?

- First, the body of the do-while loop is executed once. Only then, the test condition is evaluated.
- If the test condition returns true, the set of instructions inside the body of the loop is executed again, and the test condition is evaluated.
- The same process goes on until the test condition becomes false.
- If the test condition returns false, then the loop terminates.

One such example to demonstrate how a do-while loop works is:

```
#include <stdio.h>


int main()

{

   int i = 5;
```

```
  do
  {
    printf("%d ", i);
    i++;
  } while (i < 5);

  return 0;
}
```

Output

5

As it was already mentioned at the beginning of this tutorial, a do-while loop runs for at least once even if the test condition returns false, because the test condition is evaluated only after the first execution of the instructions in the body of the loop.

Difference between a while and a do-while loop

A While loop is executed every time the given test condition returns true, whereas, a do-while loop is executed for the first time irrespective of the test condition being true or false because the test condition is checked only after executing the loop for the first time.

for Loop

The "For" loop is used to repeat a specific piece of code in a program until a specific condition is satisfied. The for loop statement is very specialized. We use a for loop when we already know the number of iterations of that particular piece of code we wish to execute. Although, when we do not know about the number of iterations, we use a while loop.

Here is the syntax of a for loop in C programming.

for (initialise counter; test counter; increment / decrement counter){ //set of statements}

Here,

- **initialize counter:** It will initialize the loop counter value. It is usually i=0.
- **test counter:** This is the test condition, which if found true, the loop continues, otherwise terminates.

- **Increment/decrement counter:** Incrementing or decrementing the counter.
- **Set of statements:** This is the body or the executable part of the for loop or the set of statements that has to repeat itself.

One such example to demonstrate how a for loop works is,

```
#include <stdio.h>


int main()
{
   int num = 10;
   int i;
   for (i = 0; i < num; i++)
   {
      printf("%d ", i);
   }
   return 0;
}
```

Output

0 1 2 3 4 5 6 7 8 9

- First, the initialization expression will initialize loop variables. The expression i=0 executes once when the loop starts. Then the condition i < num is checked. If the condition is true, then the statements inside the body of the loop are executed. After the statements inside the body are executed, the control of the program is transferred to the increment of the variable i by 1. The expression i++ modifies the loop variables. Iteratively, the condition i < num is evaluated again.
- If the condition is still evaluated true, the body of the loop will execute once again. The for loop terminates when i finally becomes less than num, therefore, making the condition i<num false.
- Just as if statement, we can have a for loop inside another for loop. This is known as a nested for loop. Similarly, while loop and do while loop can also be nested.

Why for loops if we already have while loops?

It is clear to a developer exactly how many times the loop will execute. So, if the developer has to dry run the code, it will become easier.

C Break/Continue

Break Statement

- Break statement is used to break the loop or switch case statements execution and brings the control to the next block of code after that particular loop or switch case it was used in.
- Break statements are used to bring the program control out of the loop it was encountered in.
- The break statement is used inside loops or switch statements in C Language.

One such example to demonstrate how a break statement works is:

```
#include <stdio.h>


int main()
{
   int i = 0;
   while (1)
   {
      if (i > 5)
      {
         break;
      }
      printf("%d ", i);
      i++;
   }


   return 0;
}
```

Output

0 1 2 3 4 5

Here, when i became 6, the break statement got executed and the program came out of the while loop.

Continue Statement

- The continue statement is used inside loops in C Language. When a continue statement is encountered inside the loop, the control jumps to the beginning of the loop for the next iteration, skipping the execution of statements inside the body of the loop after the continue statement.
- It is used to bring the control to the next iteration of the loop.
- Typically, the continue statement skips some code inside the loop and lets the program move on with the next iteration.
- It is mainly used for a condition so that we can skip some lines of code for a particular condition.
- It forces the next iteration to follow in the loop unlike a break statement, which terminates the loop itself the moment it is encountered.

One such example to demonstrate how a continue statement works is:

```
#include <stdio.h>


int main()
{
  for (int i = 0; i <= 10; i++)
  {
    if (i < 6)
    {
      continue;
    }
    printf("%d ", i);
  }
  return 0;
}
```

Output

6 7 8 9 10

Here, the continue statement was continuously executing while i remained less than 6. For all the other values of i, we got the print statement