Do-While Loop in C

The **do-while** loop is one of the most frequently used types of loops in C. The **do** and **while** keywords are used together to form a loop. The **do-while** is an exit-verified loop where the test condition is checked after executing the loop's body. Whereas the **while** loop is an **entry-verified**. The **for** loop, on the other hand, is an automatic loop.

Syntax of do while Loop

The syntax of do-while loop in C is –

```
do {
   statement(s);
} while(condition);
```

How do while Loop Works?

The loop construct starts with the keword **do**. It is then followed by a block of statements inside the curly brackets. The **while** keyword follows the right curly bracket. There is a parenthesis in front of **while**, in which there should be a Boolean expression.

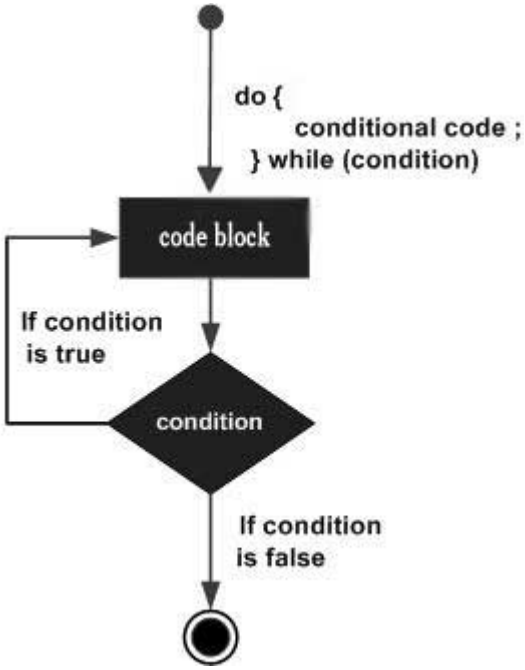
Now let's understand how the **while** loop works. As the C compiler encounters the **do** keyword, the program control enters and executes the code block marked by the curly brackets. As the end of the code block is reached, the expression in front of the **while** keyword is evaluated.

If the expression is true, the program control returns back to the top of loop. If the expression is false, the compiler stops going back to the top of loop block, and proceeds to the immediately next statement after the block. Note that there is a semicolon at the end of while statement.

Explore our **latest online courses** and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Flowchart of do while Loop

The following flowchart represents how the do-while loop works –



Since the expression that controls the loop is tested after the program runs the looping block for the first time, the do-while loop is called an "exit-verified loop". Here, the key point to note is that a do-while loop makes sure that the loop gets executed at least once.

The **while** keyword implies that the compiler continues to execute the ensuing block as long as the expression is true. However, since the condition sits at the end of the looping construct, it is checked after each iteration (rather than before each iteration as in the case of a **while** loop).

The program performs its first iteration unconditionally, and then tests the condition. If found to be true, the compiler performs the next iteration. As soon as the expression is found to be false, the loop body will be skipped and the first statement after the **while** loop will be executed.

Let us try to understand the behaviour of the **while** loop with a few examples.

Example of do while Loop

The following program prints the Hello world message five times.

</> Open Compiler

```
#include <stdio.h>

int main(){

   // local variable definition
   int a = 1;
```

```
   // while loop execution
   do{
      printf("Hello World\n");
      a++;
   } while(a <= 5);
   printf("End of loop");

   return 0;
}
```

Output

Here, the do-while loop acts as a counted loop. Run the code and check its output –

```
Hello World
Hello World
Hello World
Hello World
Hello World
End of loop
```

The variable "a" that controls the number of repetitions is initialized to 1. The program enters the loop unconditionally, prints the message, increments "a" by 1.

As it reaches the end of the loop, the condition in the **while** statement is tested. Since the condition "a <= 5" is true, the program goes back to the top of the loop and re-enters the loop.

Now "a" is 2, hence the condition is still true, hence the loop repeats again, and continues till the condition turns false. The loop stops repeating, and the program control goes to the step after the block.

Now, change the initial value of "a" to 10 and run the code again. It will produce the following output –

```
Hello World
End of loop
```

This is because the program enters the looping block unconditionally. Since the condition before the **while** keyword is false, hence the block is not repeated for the next time. Hence, the do-while loop takes at least one iteration as the test condition is at the end of the loop. For this reason, do-while loop is called an "exit-verified loop".

Difference Between while and do while Loops

The loops constructed with **while** and **do-while** appear similar. You can easily convert a **while** loop into a **do-while** loop and vice versa. However, there are certain key differences between the two.

The obvious syntactic difference is that the do-while construct starts with the **do** keyword and ends with the **while** keyword. The **while** loop doesn't need the **do** keyword. Secondly, you find a semicolon in front of **while** in case of a do-while loop. There is no semicolon in while loops.

Example

The location of the test condition that controls the loop is the major difference between the two. The test condition is at the beginning of a while loop, whereas it is at the end in case of a do-while loop. How does it affect the looping behaviour? Look at the following code –

</> Open Compiler

```
#include <stdio.h>

int main(){

   // local variable definition
   int a = 0, b = 0;

   // while loop execution
   printf("Output of while loop: \n");

   while(a < 5){
      a++;
      printf("a: %d\n", a);
   }

   printf("Output of do-while loop: \n");

   do{
      b++;
      printf("b: %d\n",b);
   } while(b < 5);
```

```
    return 0;
}
```

Output

Initially, "a" and "b" are initialized to "0" and the output of both the loops is same.

```
Output of while loop:
a: 1
a: 2
a: 3
a: 4
a: 5

Output of do-while loop:
b: 1
b: 2
b: 3
b: 4
b: 5
```

Now change the initial value of both the variables to 3 and run the code again. There's no change in the output of both the loops.

```
Output of while loop:
a: 4
a: 5

Output of do-while loop:
b: 4
b: 5
```

Now change the initial value of both the variables to 10 and run the code again. Here, you can observe the difference between the two loops –

```
Output of while loop:

Output of do-while loop:
b: 11
```

Note that the **while** loop doesn't take any iterations, but the **do-while** executes its body once. This is because the looping condition is verified at the top of the loop block in case of **while**, and since the condition is false, the program doesn't enter the loop.

In case of **do-while**, the program unconditionally enters the loop, increments "b" to 11 and then doesn't repeat as the condition is false. It shows that the **do-while** is guaranteed to take at least one repetition irrespective of the initial value of the looping variable.

The do-while loop can be used to construct a conditional loop as well. You can also use **break** and **continue** statements inside a **do-while loop**.