# C - While Loop

In C, **while** is one of the keywords with which we can form loops. The **while** loop is one of the most frequently used types of loops in C. The other looping keywords in C are **for** and **do-while**.

The **while** loop is often called the **entry verified loop**, whereas the **do-while** loop is an **exit verified loop**. The **for** loop, on the other hand, is an **automatic loop**.

## Syntax of C while Loop

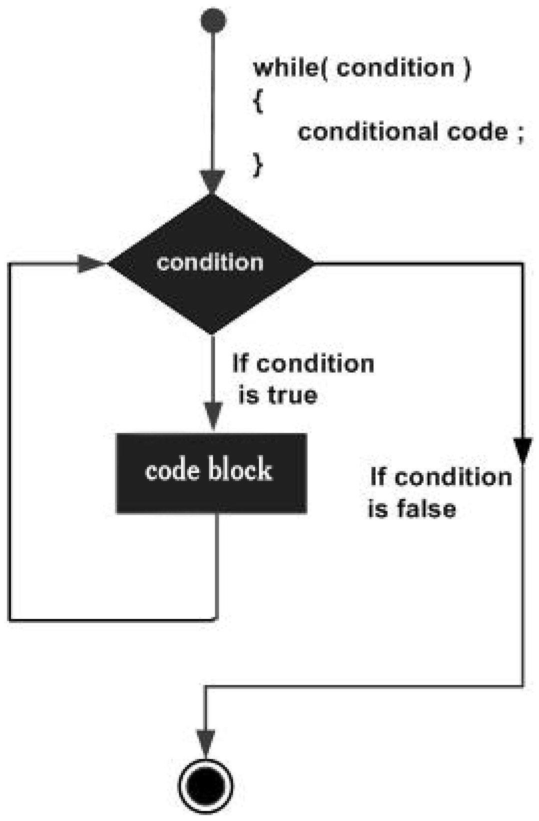
The syntax of constructing a **while** loop is as follows −

```
while(expression){
   statement(s);
}
```

The **while** keyword is followed by a parenthesis, in which there should be a Boolean expression. Followed by the parenthesis, there is a block of statements inside the curly brackets.

## Flowchart of C while Loop

The following flowchart represents how the **while** loop works −



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

## How while Loop Works in C?

The C compiler evaluates the expression. If the expression is true, the code block that follows, will be executed. If the expression is false, the compiler ignores the block next to the **while** keyword, and proceeds to the immediately next statement after the block.

Since the expression that controls the loop is tested before the program enters the loop, the **while** loop is called the **entry verified loop**. Here, the key point to note is that a **while** loop might not execute at all if the condition is found to be not true at the very first instance itself.

The **while** keyword implies that the compiler continues to execute the ensuing block as long as the expression is true. The condition sits at the top of the looping construct. After each iteration, the condition is tested. 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.

## Example of while Loop in C

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
   while(a <= 5){
      printf("Hello World \n");
      a++;
   }
   printf("End of loop");
   return 0;
}
```

## Output

Here, the **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
```

### Example Explanation

The variable "a" that controls the number of repetitions is initialized to 1, before the **while** statement. Since the condition "a <= 5" is true, the program enters the loop, prints the message, increments "a" by 1, and goes back to the top of the loop.

In the next iteration, "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. Now the output will show the following −

```
End of loop
```

This is because the condition before the **while** keyword is false in the very first iteration itself, hence the block is not repeated.

A "char" variable represents a character corresponding to its ASCII value. Hence, it can be incremented. Hence, we increment the value of the variable from "a" till it reaches "z".

## Using while as Conditional Loop

You can use a while loop as a conditional loop where the loop will be executed till the given condition is satisfied.

### Example

In this example, the **while** loop is used as a **conditional loop**. The loop continues to repeat till the input received is non-negative.

</> Open Compiler

```c
#include <stdio.h>

int main(){

   // local variable definition
   char choice = 'a';

   int x = 0;

   // while loop execution
   while(x >= 0){
      (x % 2 == 0) ? printf("%d is Even \n", x) : printf("%d is Odd \n", x);

      printf("\n Enter a positive number: ");
      scanf("%d", &x);
   }
   printf("\n End of loop");
   return 0;
}
```

### Output

Run the code and check its output −

```
0 is Even

Enter a positive number: 12
12 is Even

Enter a positive number: 25
25 is Odd

Enter a positive number: -1

End of loop
```

## While Loop with break and continue

In all the examples above, the **while** loop is designed to repeat for a number of times, or till a certain condition is found. C has break and continue statements to control the loop. These keywords can be used inside the **while** loop.

### Example

The **break** statement causes a loop to terminate −

```
while (expr){
   . . .
   . . .
   if (condition)
      break;
   . . .
}
```

### Example

The **continue** statement makes a loop repeat from the beginning −

```
while (expr){
   . . .
   . . .
   if (condition)
      continue;
   . . .
}
```

## More Examples of C while Loop

### Example: Printing Lowercase Alphabets

The following program prints all the lowercase alphabets with the help of a **while** loop.

</> Open Compiler

```
#include <stdio.h>

int main(){

   // local variable definition
   char a = 'a';

   // while loop execution
   while(a <= 'z') {
      printf("%c", a);
      a++;
   }
   printf("\n End of loop");
   return 0;
}
```

## Output

Run the code and check its output −

```
abcdefghijklmnopqrstuvwxyz
End of loop
```

## Example: Equate Two Variables

In the code given below, we have two variables "a" and "b" initialized to 10 and 0, respectively. Inside the loop, "b" is decremented and "a" is incremented on each iteration. The loop is designed to repeat till "a" and "b" are not equal. The loop ends when both reach 5.

Open Compiler

```
#include <stdio.h>

int main(){

   // local variable definition
   int a = 10, b = 0;

   // while loop execution
   while(a != b){
      a--;
      b++;
      printf("a: %d b: %d\n", a,b);
   }
   printf("\n End of loop");
   return 0;
}
```

## Output

When you run this code, it will produce the following output −

```
a: 9 b: 1
a: 8 b: 2
a: 7 b: 3
a: 6 b: 4
a: 5 b: 5

End of loop
```

## while Vs. do while Loops

The **do-while** loop appears similar to the **while** loop in most cases, although there is a difference in its syntax. The **do-while** is called the **exit verified loop**. In some cases, their behaviour is different. Difference between **while** and **do-while** loop is explained in the **do-while** chapter of this tutorial.