C while and do...while Loop

In this session, you will learn to create while and do...while loop in C programming with the help of examples.

In programming, loops are used to repeat a block of code until a specified condition is met.

Why need of loop :

Whenever we have to execute some statement or any general expression for different values number of times, then we are using loops.

Eg.

4^5 4! p=1; value i=1.... 5 is rotated in the term of loops .....condition

1*4=4 <= p*4=p ///this is general expression is inserted in loop...

4*4 =16 initialization 8! =8*7=*6=*5=*4=*3=*2=*1

16*4=64 loop with condition 8!= *1=*2

64*4=256 body or expression

256*4=1054 i++

- Basic structure using the **while** loops

  i) Initialization / value i=1.... 5 is rotated in the term of loops .....condition

  ii) loop with condition / while(i<=5)

  iii) body or expression / p=p*4

iv) i++/i--

C programming has three types of loops.

1. for loop
2. while loop
3. do...while loop

In the previous tutorial, we learned about `for` loop. In this tutorial, we will learn about `while` and `do..while` loop.

while loop

The syntax of the `while` loop is:

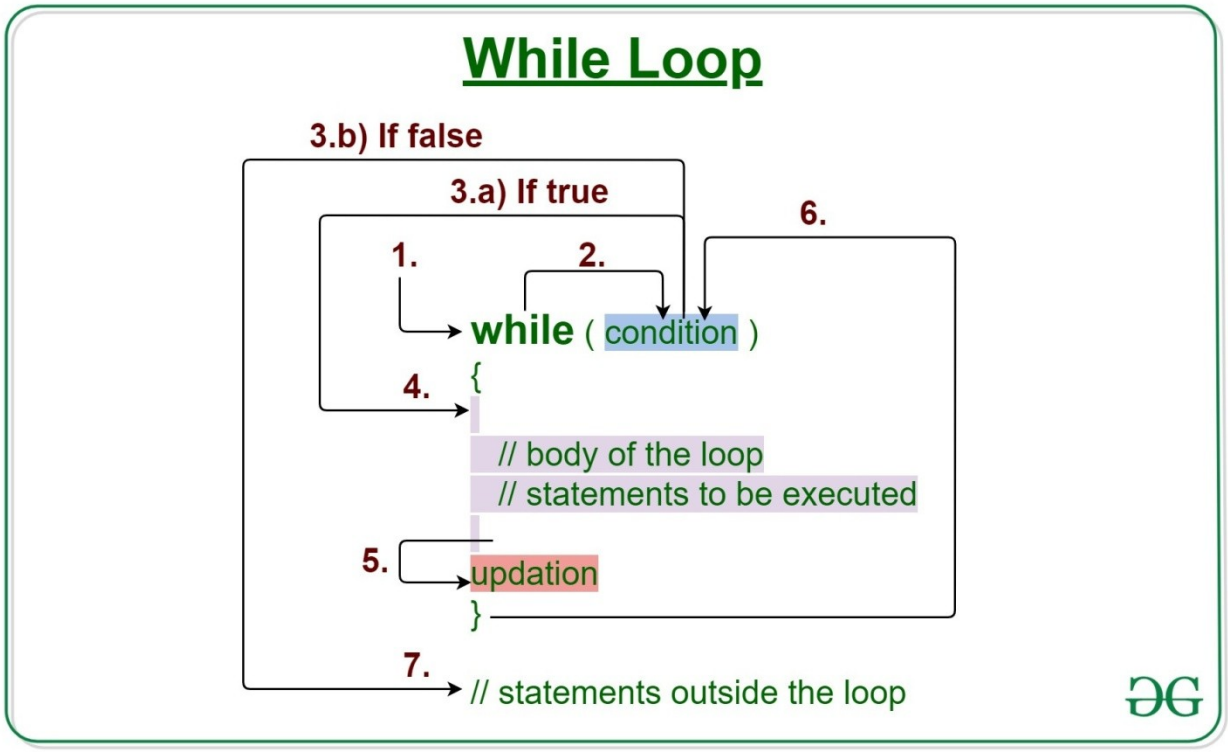
```
initialization
while (testExpression)
{
   // statements inside the body of the loop
Incre/decre
}
```

How while loop works?

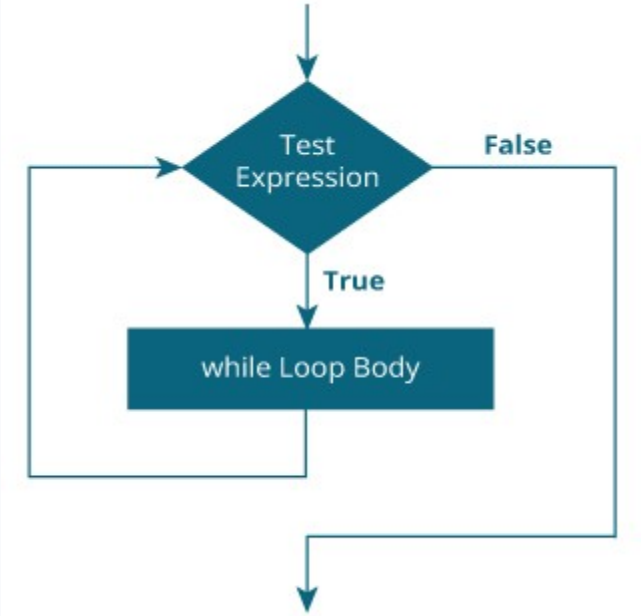
- The `while` loop evaluates the test expression inside the parenthesis `()`.
- If the test expression is true, statements inside the body of `while` loop are executed. Then, the test expression is evaluated again.
- The process goes on until the test expression is evaluated to false.

- If the test expression is false, the loop terminates (ends).

To learn more about test expression (when the test expression is evaluated to true and false), check out relational and logical operators.



Flowchart of while loop



Demo Example :

```
// Print " Om Shanti " 10 times  // in normal program we can do by writing printf statement 10 times, but using loops it will be very easier..

#include <stdio.h>
int main()
{
   int i = 1;   // initialization
         1 2 3 4 5 6 7 8 9 10 11
   while (i <= 10) // loop with condition
   {
      printf("\n Om Shanti ");   //body of loop
      ++i;  2 3 4 5 6 7 8 9 10 11                  // incre/decre
   }
    // condition becomes false
   return 0;
}

*Using decrement
#include <stdio.h>
int main()
{
   int i = 10;   // initialization
         10 9 8 7 6 5 4 3 2 1 0 false
   while (i >= 1) // loop with condition
```

```
    {
        printf(“\n Hellow world “);   //body of loop 1 2 3 4 5 6 7 8 9 10
        --i;  9 8 7 6 5 4 3 2 1 0                  // decre
    }
      // condition becomes false
    return 0;
}
```

Example 1: while loop

```
// Print numbers from 1 to 5

#include <stdio.h>
int main()
{
    int i = 1;

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

    return 0;
}
```

Output

```
1
2
3
4
5
```

Here, we have initialized `i` to 1.

1. When `i` is 1, the test expression `i <= 5` is true. Hence, the body of the `while` loop is executed. This prints 1 on the screen and the value of `i` is increased to 2.

2. Now, i is 2, the test expression i <= 5 is again true. The body of the while loop is executed again. This prints 2 on the screen and the value of i is increased to 3.
3. This process goes on until i becomes 6. When i is 6, the test expression i <= 5 will be false and the loop terminates.

Eg. To display number from 10 to 19.

```
#include <stdio.h>

void main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   //return 0;
}
```

When the above code is compiled and executed, it produces the following result

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Guess the output of this while loop

```
#include <stdio.h>
int main()
{
    int var=1;
    while (var <=2)
```

```
    {
      printf("%d ", var); //1 1 1 1111 1111 indefinite
      //no incre/decre tats why value of var should be always 1
only
    }
}
```

The program is an example of **infinite while loop**. Since the value of the variable var is same (there is no ++ or – operator used on this variable, inside the body of loop) the condition var<=2 will be true forever and the loop would never terminate.

Examples of infinite while loop

Example 1:

```
#include <stdio.h>
int main()
{
    int var = 6;
    while (var >=5) // var !=10
    {
      printf("%d", var); // 6 7 8
      var++;
    }
  return 0;
}
```

Infinite loop: var will always have value >=5 so the loop would never end.

Example 2:

```
#include <stdio.h>
int main()
{
   int var =5;
   while (var <=10)
   {
     printf("%d", var);
     var--;
   }
   return 0;
}
```

Infinite loop: var value will keep decreasing because of -- operator, hence it will always be <= 10.

Use of Logical operators in while loop

Just like relational operators (<, >, >=, <=, ! =, ==), we can also use logical operators in while loop. The following scenarios are valid :

```
while(num1<=10 && num2<=10)
```

-using AND(&&) operator, which means both the conditions should be true.

```
while(num1<=10||num2<=10)
```

– OR(||) operator, this loop will run until both conditions return false.

```
while(num1!=num2 &&num1 <=num2)
```

– Here we are using two logical operators NOT (!) and AND(&&).

```
while(num1!=10 ||num2>=num1)
```

Example of while loop using logical operator

In this example we are testing multiple conditions using logical operator inside while loop.

```
#include <stdio.h>
int main()
{
  int i=1, j=1;
  while (i <= 4 || j <= 3)   // T || T => T for i = 5 and j =5 F|| F =>
F
  {
        printf("%d\t %d\n",i, j); 1 1
        i++;
        j++;
  }
  return 0;
}
```

Output:

```
1 1
2 2
3 3
4 4
```

```
while(1)
```

It is an infinite loop which will run till a break statement is issued explicitly. Interestingly not while(1) but any integer which is non-zero will give a similar effect as while(1). Therefore, while(1), while(2) or while(-255), all will give infinite loop only.

```
while(1) or while(any non-zero integer)
{
   // loop runs infinitely
}
```

A simple usage of while(1) can be in the Client-Server program. In the program, the server runs in an infinite while loop to receive the packets sent from the clients.
But practically, it is not advisable to use while(1) in real-world because it increases the CPU usage and also blocks the code i.e one cannot come out from the while(1) until the program is closed manually. while(1) can be used at a place where condition needs to be true always.

More examples over while loop :

- Write a program to display table of any number.
  n , i=1 , 2*1 =2 2*2 = 4 2*3=6 2*4 =8 ..... n* i = r

```
#include<stdio.h>
int main( )
{
 int n, i;
printf("enter the number for which u want to display table\n");
scanf("%d", &n); //2
```

```
i=1;
while(i<=10)
{
  printf( " %d * %d =%d \n", n,i, n*i );
i++;
}
return 0;
}
```

do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

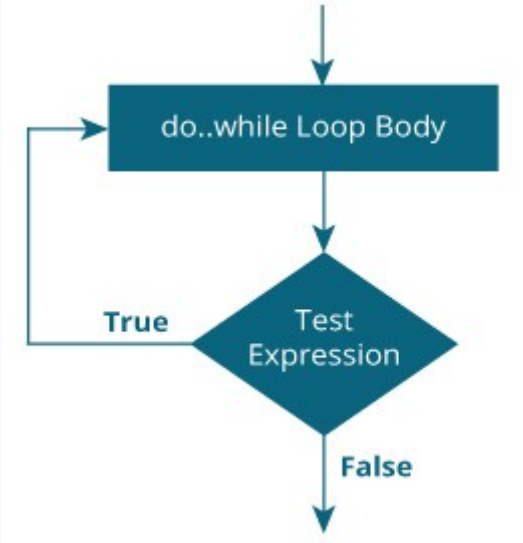
The syntax of the do...while loop is:

```
Initialization;
do
{
  // statements inside the body of the loop
}
while (testExpression);
```

How do...while loop works?

- The body of do...while loop is executed once. Only then, the test expression is evaluated.
- If the test expression is true, the body of the loop is executed again and the test expression is evaluated.
- This process goes on until the test expression becomes false.
- If the test expression is false, the loop ends.

Flowchart of do...while Loop



Example 2: do...while loop

```
// Program to add numbers until the user enters zero

#include <stdio.h>

int main()
{
   double number, sum = 0;

   // the body of the loop is executed at least once
   do
   {
      printf("Enter a number: ");
      scanf("%lf", &number); // 3 8 10 0
      sum += number;  // sum=sum+number => 0=0+3=3+8=11+10=21+0=21
   }
   while(number != 0.0); // T T T F

   printf("Sum = %.2lf",sum);
```

```
    return 0;
}
```

Output

```
Enter a number: 1.5
Enter a number: 2.4
Enter a number: -3.48
Enter a number: 4.25
Enter a number: 0
Sum = 4.70
```

Here is the difference table:

WHILE	DO-WHILE
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
If there is a single statement, brackets are not required.	Brackets are always required.
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
while(condition) { statement(s); }	do { statement(s); } while(condition);