Programming in C

(Loop , Nested loop)

Prepared By

Alok Haldar

Assistant professor

Department of Computer Science & BCA

Kharagpur College

In any programming language including C, loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

Types of Loop

There are 3 types of Loop in C language, namely:

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

`while` loop

`while` loop can be addressed as an **entry control** loop. It is completed in 3 steps.

- Variable initialization.(e.g `int x = 0;`)
- condition(e.g `while(x <= 10)`)
- Variable increment or decrement (`x++` or `x--` or `x = x + 2`)

Syntax :

variable initialization;
while(condition)
{
   statements;
   variable increment or decrement;
}

Example: Program to print first 10 natural numbers

```
#include<stdio.h>

void main( )
{
    int x;
    x = 1;
    while(x <= 10)
    {
        printf("%d\t", x);
        /* below statement means, do x = x+1, increment x by 1*/
        x++;
    }
}
```

`for` loop

`for` loop is used to execute a set of statements repeatedly until a particular condition is satisfied. We can say it is an **open ended loop.**. General format is,

```
for(initialization; condition; increment/decrement)
{
    statement-block;
}
```

In `for` loop we have exactly two semicolons, one after initialization and second after the condition. In this loop we can have more than one initialization or increment/decrement, separated using comma operator. But it can have only one **condition**.

The `for` loop is executed as follows:

1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is **true**, it executes the for-loop body.
4. Then it evaluate the increment/decrement condition and again follows from step 2.
5. When the condition expression becomes **false**, it exits the loop.

Example: Program to print first 10 natural numbers

```
#include<stdio.h>

void main( )
{
    int x;
    for(x = 1; x <= 10; x++)
    {
        printf("%d\t", x);
    }
}
```

Nested `for` loop

We can also have nested `for` loops, i.e one `for` loop inside another `for` loop. Basic syntax is,

```
for(initialization; condition; increment/decrement)
{
    for(initialization; condition; increment/decrement)
    {
        statement ;
    }
}
```

Example: Program to print half Pyramid of numbers

```
#include<stdio.h>

void main( )
{
    int i, j;
    /* first for loop */
    for(i = 1; i < 5; i++)
    {
        printf("\n");
        /* second for loop inside the first */
        for(j = i; j > 0; j--)
        {
            printf("%d", j);
        }
    }
}
```

`do while` loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of `do-while` loop. `do` statement evaluates the body of the loop first and at the end, the condition is checked using `while` statement. It means that the body of the loop will be executed at least once, even though the starting condition inside `while` is initialized to be **false**. General syntax is,

```
do
{
    .....
    .....
}
while(condition)
```

Example: Program to print first 10 multiples of 5.

```
#include<stdio.h>

void main()
{
    int a, i;
    a = 5;
    i = 1;
    do
    {
        printf("%d\t", a*i);
        i++;
    }
    while(i <= 10);
}
```

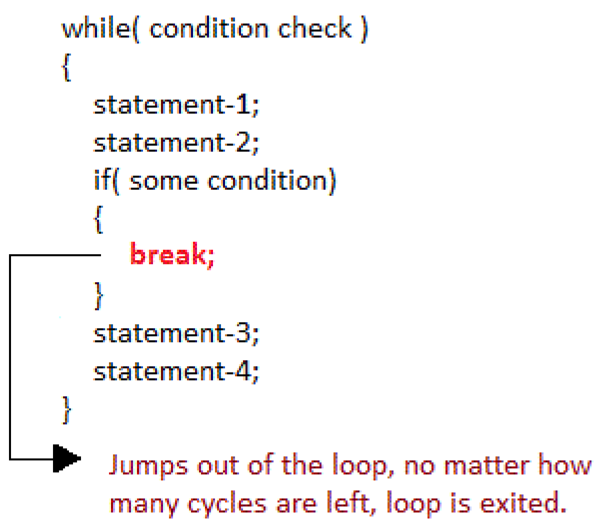
```
5 10 15 20 25 30 35 40 45 50
```

Jumping Out of Loops

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes **true**. This is known as jumping out of loop.

1) break statement

When `break` statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

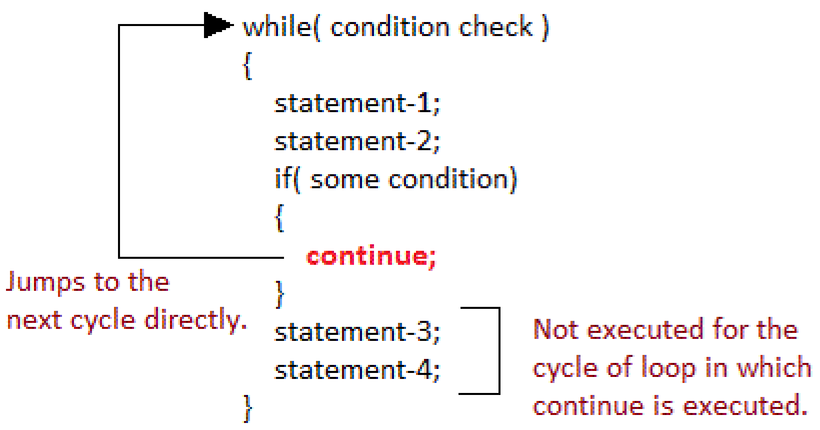


```
while( condition check )
{
   statement-1;
   statement-2;
   if( some condition)
   {
      break;
   }
   statement-3;
   statement-4;
}
```

Jumps out of the loop, no matter how many cycles are left, loop is exited.

2) continue statement

It causes the control to go directly to the test-condition and then continue the loop process. On encountering `continue`, cursor leave the current cycle of loop, and starts with the next cycle.



```
while( condition check )
{
   statement-1;
   statement-2;
   if( some condition)
   {
      continue;
   }
   statement-3;
   statement-4;
}
```

Jumps to the next cycle directly.
Not executed for the cycle of loop in which continue is executed.