Name :Asad ullah khan

Id: 6830

Clasa: BS SE b

Subject: Programming fundamental

1.This is one of the most frequently used loop in [C programming](https://beginnersbook.com/2014/01/c-tutorial-for-beginners-with-examples/).  
**Syntax of for loop:**

for (initialization; condition test; increment or decrement)

{

//Statements to be executed repeatedly

}

**Step 1:** First initialization happens and the counter variable gets initialized.  
**Step 2:** In the second step the condition is checked, where the counter variable is tested for the given condition, if the condition returns true then the C statements inside the body of for loop gets executed, if the condition returns false then the for loop gets terminated and the control comes out of the loop.  
**Step 3:** After successful execution of statements inside the body of loop, the counter variable is incremented or decremented, depending on the operation (++ or –).

**Example of For loop**

#include <stdio.h>

int main()

{

int i;

for (i=1; i<=3; i++)

{

printf("%d\n", i);

}

return 0;

}

B)

#include <iostream>

using namespace std;

int main()

{

int num1, num2;

cout<<"Enter first number:";

cin>>num1;

cout<<"Enter second number:";

cin>>num2;

if(num1>num2)

{

cout<<"First number "<<num1<<" is the largest";

}

else

{

cout<<"Second number "<<num2<<" is the largest";

}

return 0;

}

2.

# The Logical Operators

*Logical operators*are mainly used to control program flow. Usually, you will find them as part of an if, a while, or some other control statement (Chapter [6](https://users.cs.cf.ac.uk/Dave.Marshall/PERL/node41.html#ch:control))

The Logical operators are:

**op1 && op2**

-- Performs a logical AND of the two operands.

**op1 || op2**

-- Performs a logical OR of the two operands.

**!op1**

-- Performs a logical NOT of the operand.

The concept of logical operators is simple. They allow a program to make a decision based on multiple conditions. Each operand is considered a condition that can be evaluated to a true or false value. Then the value of the conditions is used to determine the overall value of the op1 operator op2 or !op1 grouping. The following examples demonstrate different ways that logical conditions can be used.

The && operator is used to determine whether both operands or conditions are true and.pl.

For example:

if ($firstVar == 10 && $secondVar == 9) {

print("Error!");

};

If either of the two conditions is false or incorrect, then the print command is bypassed.

The || operator is used to determine whether either of the conditions is true.

For example:

if ($firstVar == 9 || $firstVar == 10) {

print("Error!");

If either of the two conditions is true, then the print command is run.

**Caution** If the first operand of the || operator evaluates to true, the second operand will not be evaluated. This could be a source of bugs if you are not careful.

For instance, in the following code fragment:

if ($firstVar++ || $secondVar++) { print("\n"); }

variable $secondVar will not be incremented if $firstVar++ evaluates to true.

The ! operator is used to convert true values to false and false values to true. In other words, it inverts a value. Perl considers any non-zero value to be true-even string values. For example:

$firstVar = 10;

$secondVar = !$firstVar;

if ($secondVar == 0) {

print("zero\n");

};

is equal to *0-* and the program produces the following output:

zero

You could replace the 10 in the first line with "ten," 'ten,' or any non-zero, non-null value.

b.

//C++ program for converting degree Celsius into Fahrenheit and vice versa

#include<iostream>

using namespace std;

int main()

{

float fahr, cel;

char option;

cout << "Choose from following option:" << endl;

cout << "1. Celsius to Fahrenheit." << endl;

cout << "2. Fahrenheit to Celsius." << endl;

cin >> option;

//option for converting celsius into fahernheit

if (option == '1')

{

cout << "Enter the temperature in Celsius: ";

cin >> cel;

fahr = (1.8 \* cel) + 32.0; //temperature conversion formula

cout << "\nTemperature in degree Fahrenheit: " << fahr << " F" << endl;

}

//option for converting Fahrenheit into Celsius

else if (option == '2')

{

cout << "Enter the temperature in Fahrenheit: ";

cin >> fahr;

cel = (fahr - 32) / 1.8; //temperature conversion formula

cout << "\nTemperature in degree Celsius: " << cel << " C" << endl;

}

else

cout << "Error Wrong Input." << endl;

return 0;

}

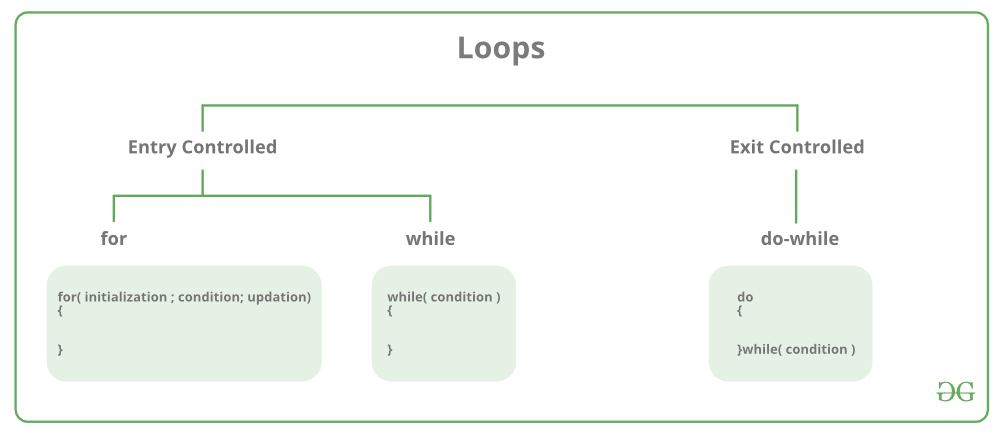
3 A.

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.  
In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

* An operation is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.
* **Counter not Reached:**If the counter has not reached the desired number, the next instruction in the sequence returns to the first instruction in the sequence and repeat it.
* **Counter reached:** If the condition has been reached, the next instruction “falls through” to the next sequential instruction or branches outside the loop.

There are mainly two types of loops:

1. **Entry Controlled loops**: In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.
2. **Exit Controlled Loops**: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. **do – while loop** is exit controlled loop.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20191128194516/Cpp-loops.png)

**for Loop**

A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.  
**Syntax:**

for (initialization expr; test expr; update expr)

{

// body of the loop

// statements we want to execute

}

In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated . Steps are repeated till exit condition comes.

* **Initialization Expression**: In this expression we have to initialize the loop counter to some value. for example: int i=1;
* **Test Expression**: In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: i <= 10;
* **Update Expression**: After executing loop body this expression increments/decrements the loop variable by some value. for example: i++;

Equivalent flow diagram for loop :  
/var/folders/wd/_lvv3z4d7kv1bf81s5nhmtsm0000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/loops.png

**Example:**

* C
* C++

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C program to illustrate for loop  #include <stdio.h>    int main()  {      int i=0;        for (i = 1; i <= 10; i++)      {          printf( "Hello World\n");      }        return 0;  } |

**Output:**

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

**While Loop**

While studying **for loop** we have seen that the number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where we do not know the exact number of iterations of loop beforehand. The loop execution is terminated on the basis of test condition.

**Syntax**:  
We have already stated that a loop is mainly consisted of three statements – initialization expression, test expression, update expression. The syntax of the three loops – For, while and do while mainly differs on the placement of these three statements.

**initialization expression;**

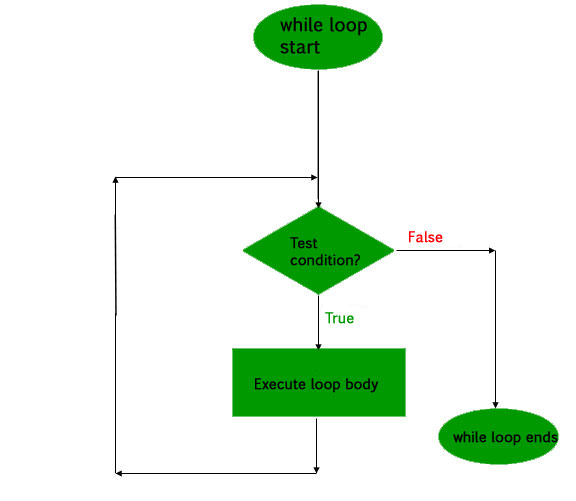
while (**test\_expression**)

{

// statements

**update\_expression;**

}

**Flow Diagram**:  


**Example:**

* C
* C++

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C program to illustrate while loop  #include <stdio.h>    int main()  {      // initialization expression      int i = 1;        // test expression      while (i < 6)      {          printf( "Hello World\n");            // update expression          i++;      }        return 0;  } |

**Output:**

Hello World

Hello World

Hello World

Hello World

Hello World

**do while loop**

In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body, i.e do while loop is exit controlled whereas the other two loops are entry controlled loops.  
**Note**: In do while loop the loop body will execute at least once irrespective of test condition.

**Syntax**:

**initialization expression;**

do

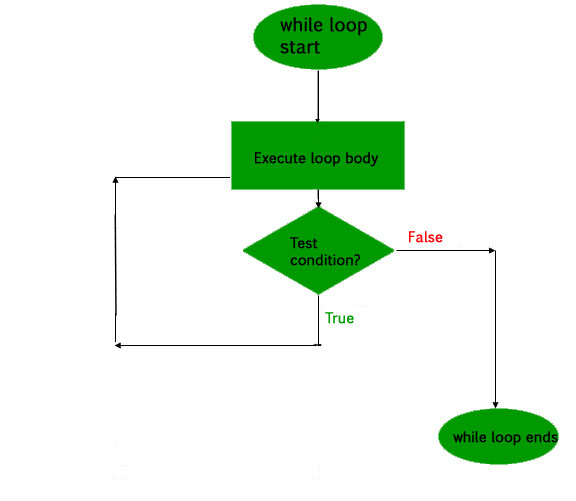
{

// statements

**update\_expression;**

} while (**test\_expression**);

**Note**: Notice the semi – colon(“;”) in the end of loop.

**Flow Diagram**:  


**Example:**

* C
* C++

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C program to illustrate do-while loop  #include <stdio.h>    int main()  {      int i = 2; // Initialization expression        do      {          // loop body          printf( "Hello World\n");            // update expression          i++;        }  while (i < 1);   // test expression        return 0;  } |

**Output:**

Hello World

In the above program the test condition (i<1) evaluates to false. But still as the loop is exit – controlled the loop body will execute once.

**What about an Infinite Loop?**

An infinite loop (sometimes called an endless loop ) is a piece of coding that lacks a functional exit so that it repeats indefinitely. An infinite loop occurs when a condition always evaluates to true. Usually, this is an error.

* C
* C++

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C program to demonstrate infinite loops  // using for and while  // Uncomment the  sections to see the output    #include <stdio.h>    int main ()  {      int i;        // This is an infinite for loop as the condition      // expression is blank      for ( ; ; )      {         printf("This loop will run forever.\n");      }        // This is an infinite for loop as the condition      // given in while loop will keep repeating infinitely      /\*      while (i != 0)      {          i-- ;          printf( "This loop will run forever.\n");      }      \*/        // This is an infinite for loop as the condition      // given in while loop is "true"      /\*      while (true)      {          printf( "This loop will run forever.\n");      }      \*/  } |

Output:

This loop will run forever.

This loop will run forever.

...................

B,

#include <iostream>

using namespace std;

int main()

{

int n;

cout << "Enter an integer: ";

cin >> n;

if ( n % 2 == 0)

cout << n << " is even.";

else

cout << n << " is odd.";

return 0;

}

4.

## **The break statement in C**

In any loop break is used to jump out of loop skipping the code below it without caring about the test condition.

It interrupts the flow of the program by breaking the loop and continues the execution of code which is outside the loop.

The common use of break statement is in switch case where it is used to skip remaining part of the code.

for (int-exp; test-exp; update-exp)

{

statement1;

if (condition)

break;

statement2;

}

## **The continue statement in C**

Like a break statement, continue statement is also used with [if](http://www.trytoprogram.com/c-programming/c-programming-if-statement/) condition inside the loop to alter the flow of control.

When used in [while](http://www.trytoprogram.com/c-programming/c-programming-while-and-do-while-loop/), [for](http://www.trytoprogram.com/c-programming/c-programming-for-loop/) or do...while loop, it skips the remaining statements in the body of that loop and performs the next iteration of the loop.

Unlike break statement, continue statement when encountered doesn’t terminate the loop, rather interrupts a particular iteration.

while (test\_condition)

{

statement1;

if (condition )

continue;

statement2;

}

Code to add 1+2+3….10

#include <iostream>

using namespace std;

int main()

{

int i,sum=0;

cout << "\n\n Find the first 10 natural numbers:\n";

cout << "---------------------------------------\n";

cout << " The natural numbers are: \n";

for (i = 1; i <= 10; i++)

{

cout << i << " ";

sum=sum+i;

}

cout << "\n The sum of first 10 natural numbers: "<<sum << endl;

}