**For loop in C++ with example**

A loop is used for executing a block of statements repeatedly until a particular condition is satisfied. For example, when you are displaying number from 1 to 100 you may want set the value of a variable to 1 and display it 100 times, increasing its value by 1 on each loop iteration.

In C++ we have three types of basic loops: for, while and do-while. In this tutorial we will learn how to use “for loop” in C++.

**Syntax of for loop**

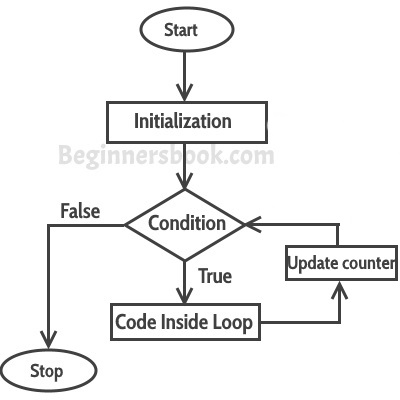
for(initialization; condition ; increment/decrement)

{

C++ statement(s);

}

**Flow of Execution of the for Loop**

As a program executes, the interpreter always keeps track of which statement is about to be executed. We call this the control flow, or the flow of execution of the program.  


**First step:** In for loop, initialization happens first and only once, which means that the initialization part of for loop only executes once.

**Second step:** Condition in for loop is evaluated on each loop iteration, if the condition is true then the statements inside for loop body gets executed. Once the condition returns false, the statements in for loop does not execute and the control gets transferred to the next statement in the program after for loop.

**Third step:** After every execution of for loop’s body, the increment/decrement part of for loop executes that updates the loop counter.

**Fourth step:** After third step, the control jumps to second step and condition is re-evaluated.

The steps from second to fourth repeats until the loop condition returns false.

**Example of a Simple For loop in C++**

Here in the loop initialization part I have set the value of variable i to 1, condition is i<=6 and on each loop iteration the value of i increments by 1.

#include <iostream>

using namespace std;

int main(){

for(int i=1; i<=6; i++){

/\* This statement would be executed

\* repeatedly until the condition

\* i<=6 returns false.

\*/

cout<<"Value of variable i is: "<<i<<endl;

}

return 0;

}

**Output:**

Value of variable i is: 1

Value of variable i is: 2

Value of variable i is: 3

Value of variable i is: 4

Value of variable i is: 5

Value of variable i is: 6

**Infinite for loop in C++**

A loop is said to be infinite when it executes repeatedly and never stops. This usually happens by mistake. When you set the condition in for loop in such a way that it never return false, it becomes infinite loop.

**For example:**

#include <iostream>

using namespace std;

int main(){

for(int i=1; i>=1; i++){

cout<<"Value of variable i is: "<<i<<endl;

}

return 0;

}

This is an infinite loop as we are incrementing the value of i so it would always satisfy the condition i>=1, the condition would never return false.

Here is another example of infinite for loop:

// infinite loop

for ( ; ; ) {

// statement(s)

}

**Example: display elements of array using for loop**

#include <iostream>

using namespace std;

int main(){

int arr[]={21,9,56,99, 202};

/\* We have set the value of variable i

\* to 0 as the array index starts with 0

\* which means the first element of array

\* starts with zero index.

\*/

for(int i=0; i<5; i++){

cout<<arr[i]<<endl;

}

return 0;

}

**Output:**

21

9

56

99

202

# While loop in C++ with example

In the last tutorial we discussed for loop. In this tutorial we will discuss while loop. As discussed earlier, loops are used for executing a block of program statements repeatedly until the given loop condition returns false.

#### Syntax of while loop

while(condition)

{

statement(s);

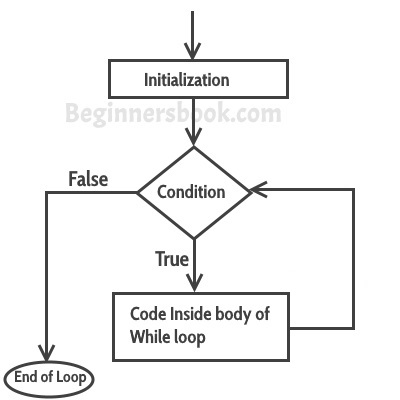
}

## How while Loop works?

In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute, this happens repeatedly until the condition returns false. When condition returns false, the control comes out of loop and jumps to the next statement in the program after while loop.

**Note:** The important point to note when using while loop is that we need to use increment or decrement statement inside while loop so that the loop variable gets changed on each iteration, and at some point condition returns false. This way we can end the execution of while loop otherwise the loop would execute indefinitely.

### Flow Diagram of While loop



## While Loop example in C++

#include <iostream>

using namespace std;

int main(){

int i=1;

/\* The loop would continue to print

\* the value of i until the given condition

\* i<=6 returns false.

\*/

while(i<=6){

cout<<"Value of variable i is: "<<i<<endl; i++;

}

}

**Output:**

Value of variable i is: 1

Value of variable i is: 2

Value of variable i is: 3

Value of variable i is: 4

Value of variable i is: 5

Value of variable i is: 6

## Infinite While loop

A while loop that never stops is said to be the infinite while loop, when we give the condition in such a way so that it never returns false, then the loops becomes infinite and repeats itself indefinitely.  
**An example of infinite while loop:**  
This loop would never end as I’m decrementing the value of i which is 1 so the condition i<=6 would never return false.

#include <iostream>

using namespace std;

int main(){

int i=1; while(i<=6) {

cout<<"Value of variable i is: "<<i<<endl; i--;

}

}

## Example: Displaying the elements of array using while loop

#include <iostream>

using namespace std;

int main(){

int arr[]={21,87,15,99, -12};

/\* The array index starts with 0, the

\* first element of array has 0 index

\* and represented as arr[0]

\*/

int i=0;

while(i<5){

cout<<arr[i]<<endl;

i++;

}

}

**Output:**

21

87

15

99

-12

**do-while loop in C++ with example**

As discussed in the last tutorial about while loop, a loop is used for repeating a block of statements until the given loop condition returns false. In this tutorial we will see do-while loop. do-while loop is similar to while loop, however there is a difference between them: In while loop, condition is evaluated first and then the statements inside loop body gets executed, on the other hand in do-while loop, statements inside do-while gets executed first and then the condition is evaluated.

**Syntax of do-while loop**

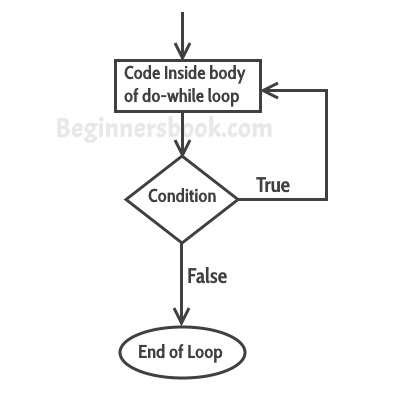
do

{

statement(s);

} while(condition);

**How do-while loop works?**

First, the statements inside loop execute and then the condition gets evaluated, if the condition returns true then the control jumps to the “do” for further repeated execution of it, this happens repeatedly until the condition returns false. Once condition returns false control jumps to the next statement in the program after do-while.  


**do-while loop example in C++**

#include <iostream>

using namespace std;

int main(){

int num=1;

do{

cout<<"Value of num: "<<num<<endl;

num++;

}while(num<=6);

return 0;

}

**Output:**

Value of num: 1

Value of num: 2

Value of num: 3

Value of num: 4

Value of num: 5

Value of num: 6

**Example: Displaying array elements using do-while loop**

Here we have an integer array which has four elements. We are displaying the elements of it using do-while loop.

#include <iostream>

using namespace std;

int main(){

int arr[]={21,99,15,109};

/\* Array index starts with 0, which

\* means the first element of array

\* is at index 0, arr[0]

\*/

int i=0;

do{

cout<<arr[i]<<endl;

i++;

}while(i<4);

return 0;

}

**Output:**

21

99

15

109

**Continue Statement in C++ with example**

Continue statement is used inside loops. Whenever a continue statement is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop’s body for the current iteration.

**Syntax of continue statement**

continue;

**Example: continue statement inside for loop**

As you can see that the output is missing the value 3, however the for loop iterate though the num value 0 to 6. This is because we have set a condition inside loop in such a way, that the continue statement is encountered when the num value is equal to 3. So for this iteration the loop skipped the cout statement and started the next iteration of loop.

#include <iostream>

using namespace std;

int main(){

for (int num=0; num<=6; num++) {

/\* This means that when the value of

\* num is equal to 3 this continue statement

\* would be encountered, which would make the

\* control to jump to the beginning of loop for

\* next iteration, skipping the current iteration

\*/

if (num==3) {

continue;

}

cout<<num<<" ";

}

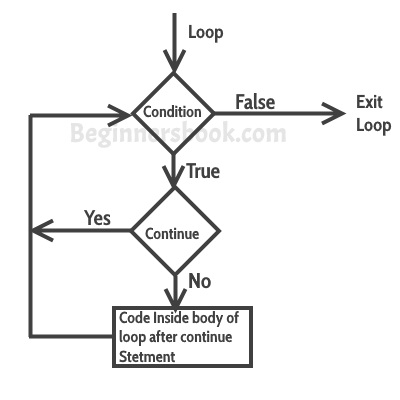
return 0;

}

**Output:**

0 1 2 4 5 6

**Flow Diagram of Continue Statement**



**Example: Use of continue in While loop**

#include <iostream>

using namespace std;

int main(){

int j=6;

while (j >=0) {

if (j==4) {

j--;

continue;

}

cout<<"Value of j: "<<j<<endl;

j--;

}

return 0;

}

**Output:**

Value of j: 6

Value of j: 5

Value of j: 3

Value of j: 2

Value of j: 1

Value of j: 0

**Example of continue in do-While loop**

#include <iostream>

using namespace std;

int main(){

int j=4;

do {

if (j==7) {

j++;

continue;

}

cout<<"j is: "<<j<<endl;

j++;

}while(j<10);

return 0;

}

**Output:**

j is: 4

j is: 5

j is: 6

j is: 8

j is: 9

**Break statement in C++ with example**

The **break statement** is used in following two scenarios:

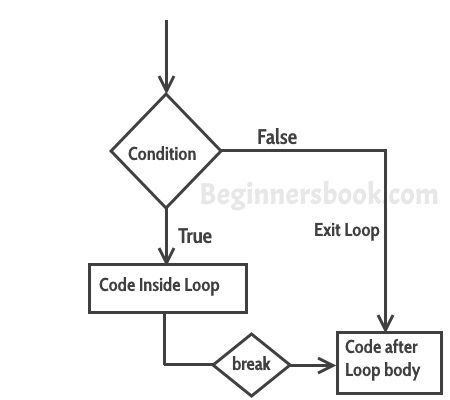
a) Use break statement to come out of the loop instantly. Whenever a break statement is encountered inside a loop, the control directly comes out of loop terminating it. It is used along with if statement, whenever used inside loop(see the example below) so that it occurs only for a particular condition.

b) It is used in switch case control structure after the case blocks. Generally all cases in switch case are followed by a break statement to avoid the subsequent cases (see the example below) execution. Whenever it is encountered in switch-case block, the control comes out of the switch-case body.

**Syntax of break statement**

break;

**break statement flow diagram**



**Example – Use of break statement in a while loop**

In the example below, we have a while loop running from 10 to 200 but since we have a break statement that gets encountered when the loop counter variable value reaches 12, the loop gets terminated and the control jumps to the next statement in program after the loop body.

#include <iostream>

using namespace std;

int main(){

int num =10;

while(num<=200) {

cout<<"Value of num is: "<<num<<endl;

if (num==12) {

break;

}

num++;

}

cout<<"Hey, I'm out of the loop";

return 0;

}

**Output:**

Value of num is: 10

Value of num is: 11

Value of num is: 12

Hey, I'm out of the loop

**Example: break statement in for loop**

#include <iostream>

using namespace std;

int main(){

int var;

for (var =200; var>=10; var --) {

cout<<"var: "<<var<<endl;

if (var==197) {

break;

}

}

cout<<"Hey, I'm out of the loop";

return 0;

}

**Output:**

var: 200

var: 199

var: 198

var: 197

Hey, I'm out of the loop

**Example: break statement in Switch Case**

#include <iostream>

using namespace std;

int main(){

int num=2;

switch (num) {

case 1: cout<<"Case 1 "<<endl;

break;

case 2: cout<<"Case 2 "<<endl;

break;

case 3: cout<<"Case 3 "<<endl;

break;

default: cout<<"Default "<<endl;

}

cout<<"Hey, I'm out of the switch case";

return 0;

}

**Output:**

Case 2

Hey, I'm out of the switch case

In this example, we have break statement after each Case block, this is because if we don’t have it then the subsequent case block would also execute. The output of the same program without break would be:

Case 2

Case 3

Default

Hey, I'm out of the switch case

**goto statement in C++ with example**

The goto statement is used for transferring the control of a program to a given label. The syntax of goto statement looks like this:

goto label\_name;

**Program structure:**

label1:

...

...

goto label2;

...

..

label2:

...

In a program we have any number of goto and label statements, the goto statement is followed by a label name, whenever goto statement is encountered, the control of the program jumps to the label specified in the goto statement.

goto statements are almost never used in any development as they are complex and makes your program much less readable and more error prone. In place of goto, you can use continue and break statement.

**Example of goto statement in C++**

#include <iostream>

using namespace std;

int main(){

int num; cout<<"Enter a number: "; cin>>num;

if (num % 2==0){

goto print;

}

else {

cout<<"Odd Number";

}

print:

cout<<"Even Number";

return 0;

}

**Output:**

Enter a number: 42

Even Number