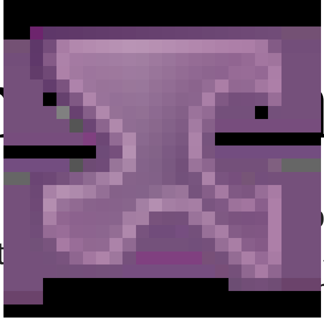
Con[illegible]tement



A program is usually not limited [illegible]f instructions. During its process it may bifurcate, repeat code or take decisions. For that [illegible]s **control structures** that serve to specify what has to be done by our program, when [illegible]mstances.

There are two types of Control Statement :

- Decision making/Branching Statement
- Looping Statement

Branching Statement

There are three types of Branching Statement, which are following :

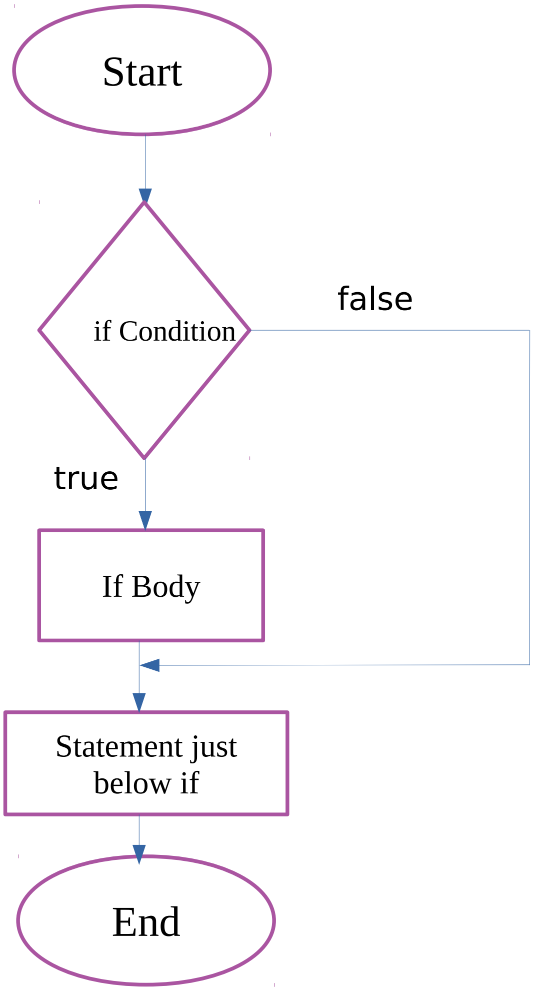
(1) if Statement

if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
{
     // Statements to execute if
     // condition is true
}
```

Flowchart of if Statement :



(2)

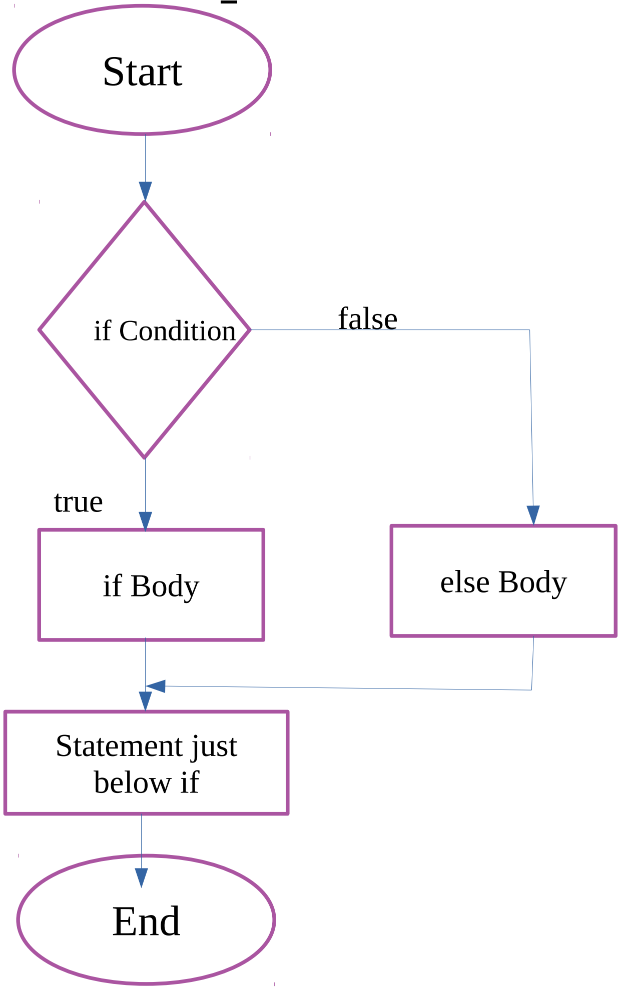
if_else Statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false. Here comes the C else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
if(condition)
{
     // Statements to execute if
     // condition is true
}
else
{
     // Statements to execute else
     // condition is true
}
```

Flowchart of if_else Statement :



(3) **switch Statement**

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax

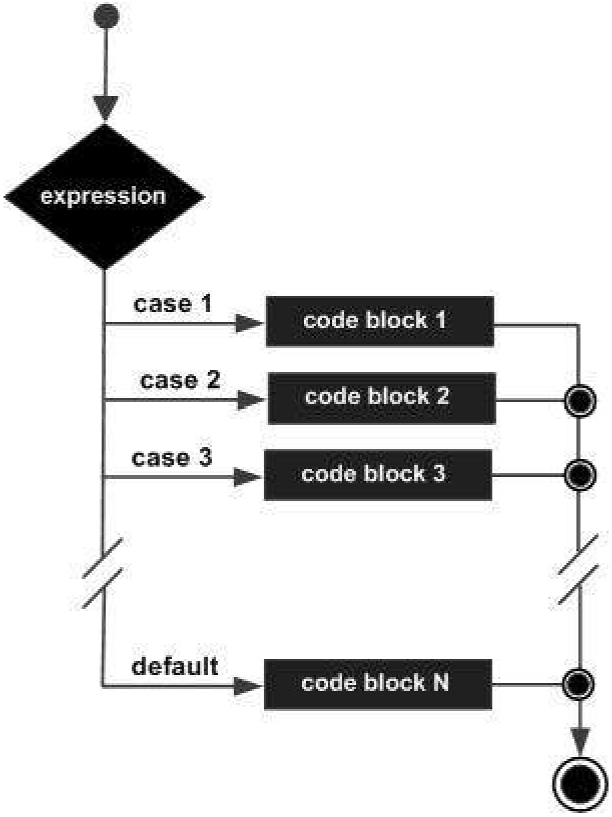
The syntax for a **switch** statement in C++ is as follows −

```
switch(expression) {
   case constant-expression  :
      statement(s);
      break; //optional
   case constant-expression  :
      statement(s);
      break; //optional

   // you can have any number of case statements.
   default : //Optional
      statement(s);
}
```

The following rules apply to a switch statement −

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Flowchart of switch Statement :



Compound Decision Making Statements

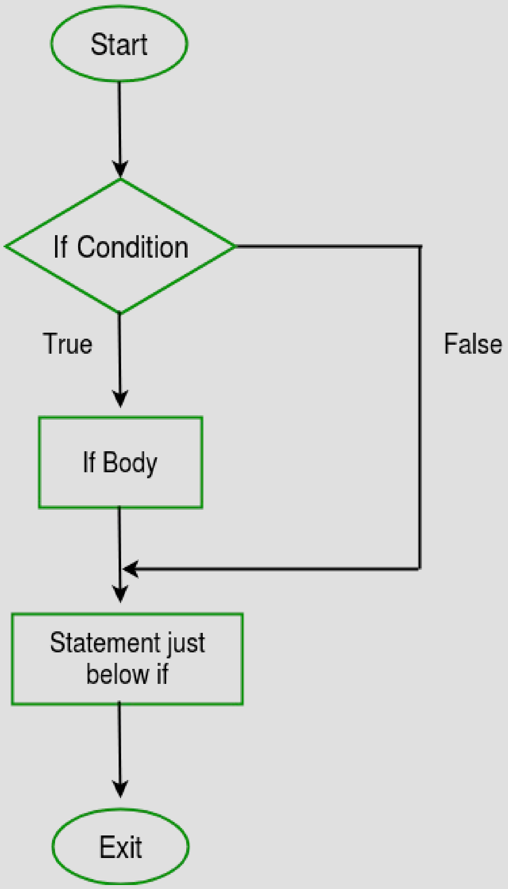
Compound Statement with if

The multiple time “if” condition are used in compound if statement is similar like “if statement”.

Syntax:

```
if(condition1)
{
     // Statements to execute if
     // condition1 is true
}
if(condition2)
{
     // Statements to execute else
     // condition2 is true
}
```

Flowchart



Note: The flowchart of compound if statement is **similar** to if statement.

Looping Statement

Loops in programming come into use when we need to repeatedly execute a block of statements.

There are four types of Looping Statement, which are following :

(1) while Loop Statement

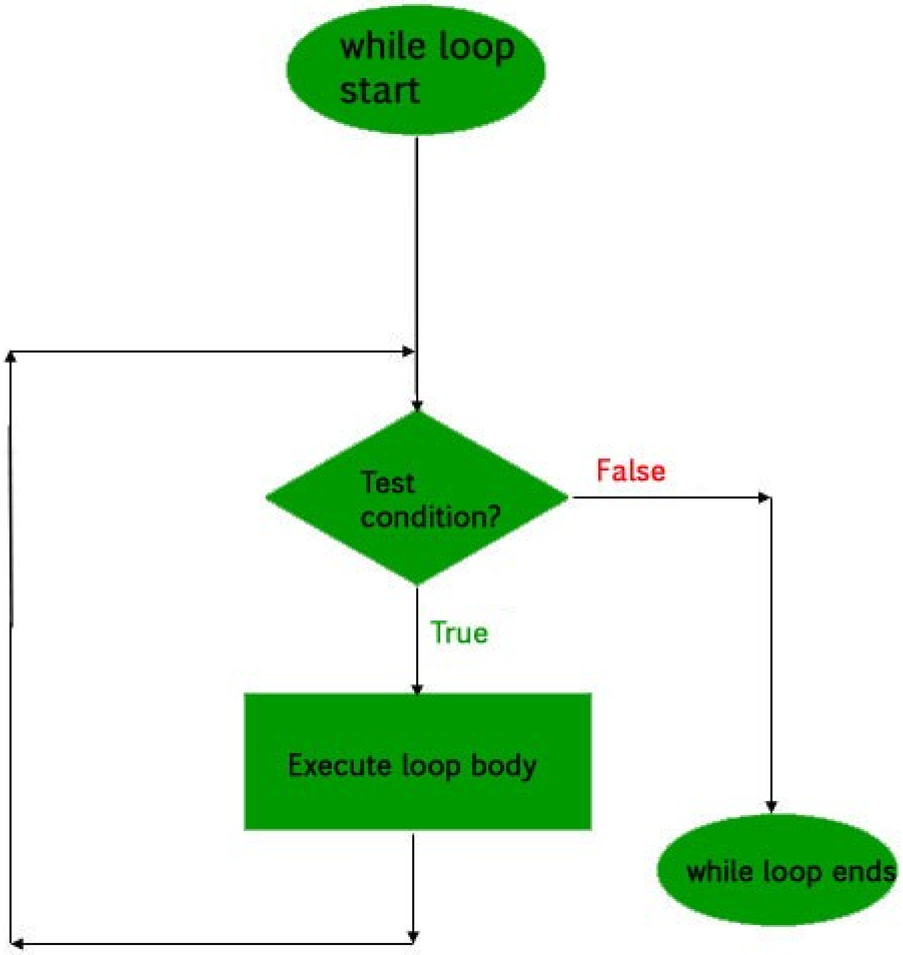
While studying for loop we have seen that the number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where we do not know the exact number of iterations of loop beforehand. The loop execution is terminated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



(2) do_while Loop Statement

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.

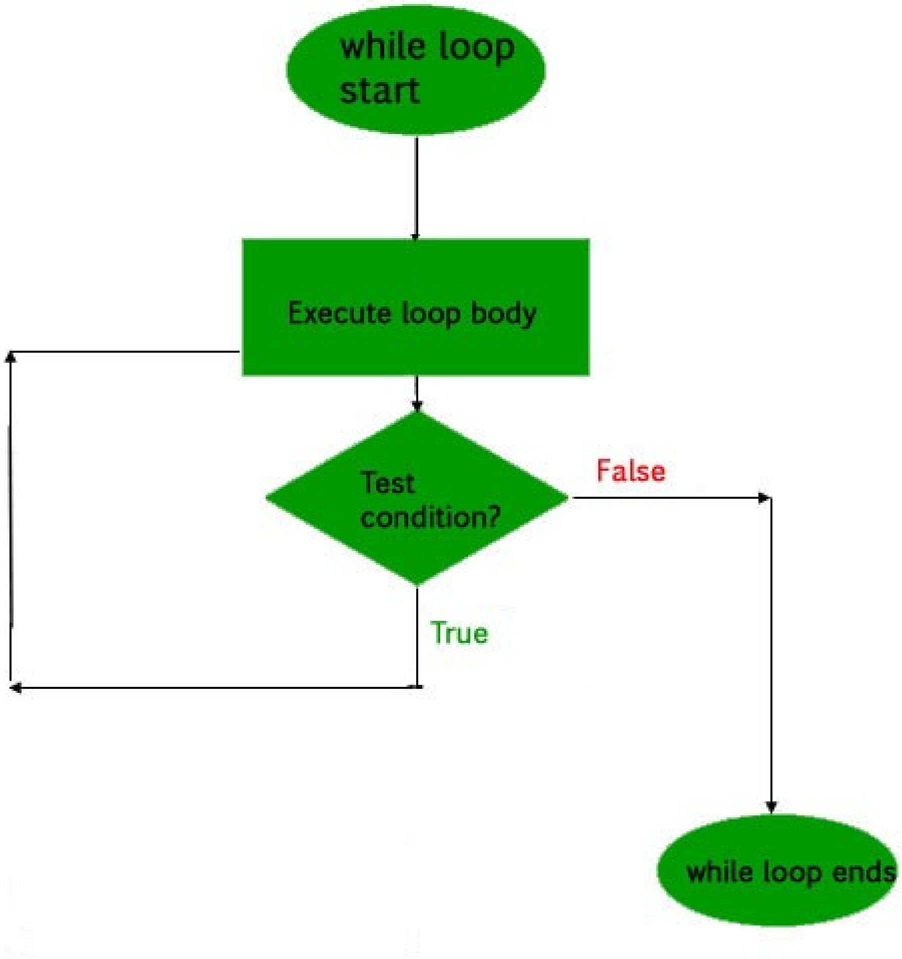
Note: In do while loop the loop body will execute at least once irrespective of test condition.

Syntax:

```
initialization expression;
do
{
   // statements


   update_expression;
} while (test_expression);
```

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

Flow Diagram:



(3) for Loop Statement

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.

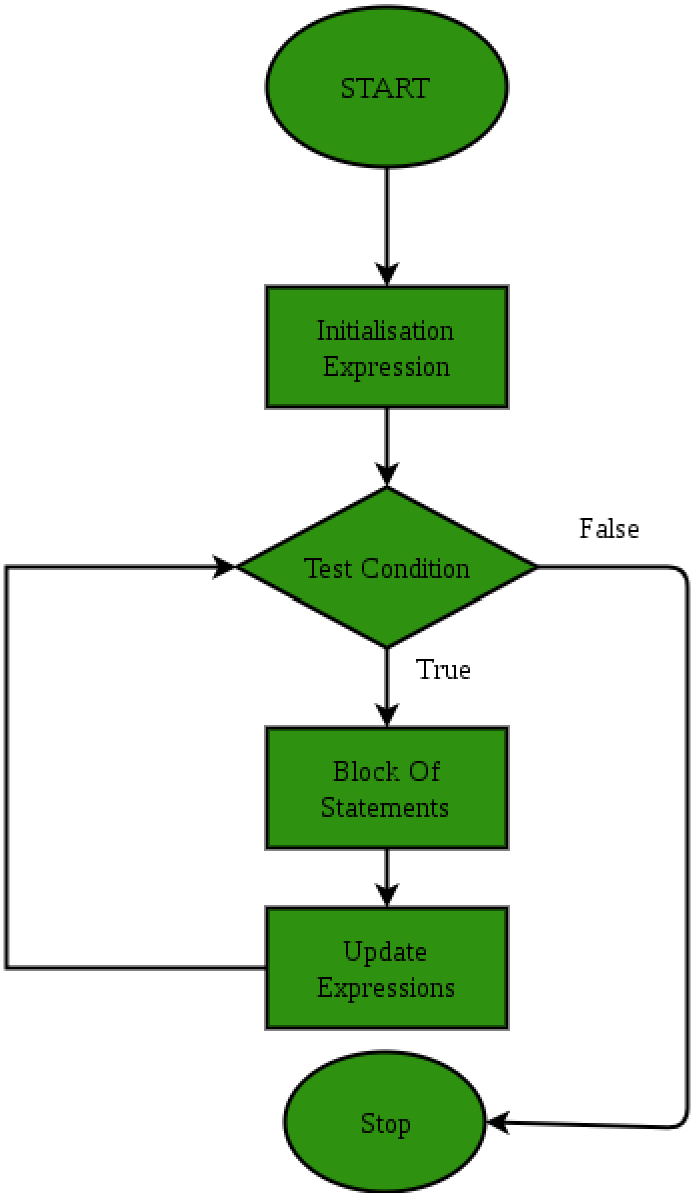
Syntax:

```
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 :



(4) Jump Statement

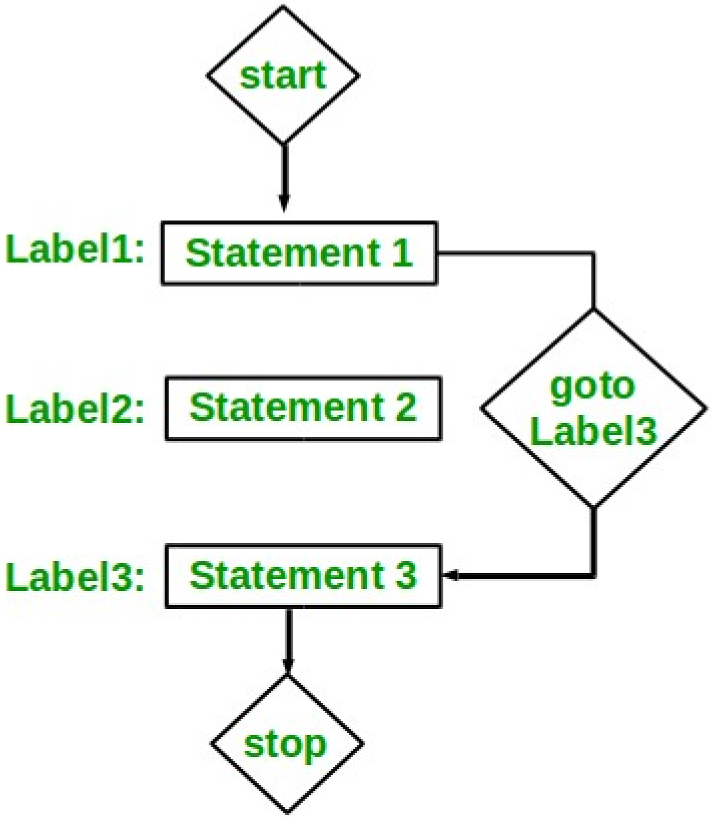
These statements are used in C or C++ for unconditional flow of control through out the funtions in a program. They support four type of jump statements:

a **goto:** The goto statement in C/C++ also referred to as unconditional jump statement can be used to jump from one point to another within a function.

Syntax:

```
Syntax1      |   Syntax2

----------------------------

goto label;  |    label:

.            |    .

.            |    .
.              |    .

label:         |    goto label;
```

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here label is a user-defined identifier which indicates the target statement. The statement immediately followed after ‘label:’ is the destination statement. The ‘label:’ can also appear before the ‘goto label;’ statement in the above syntax.



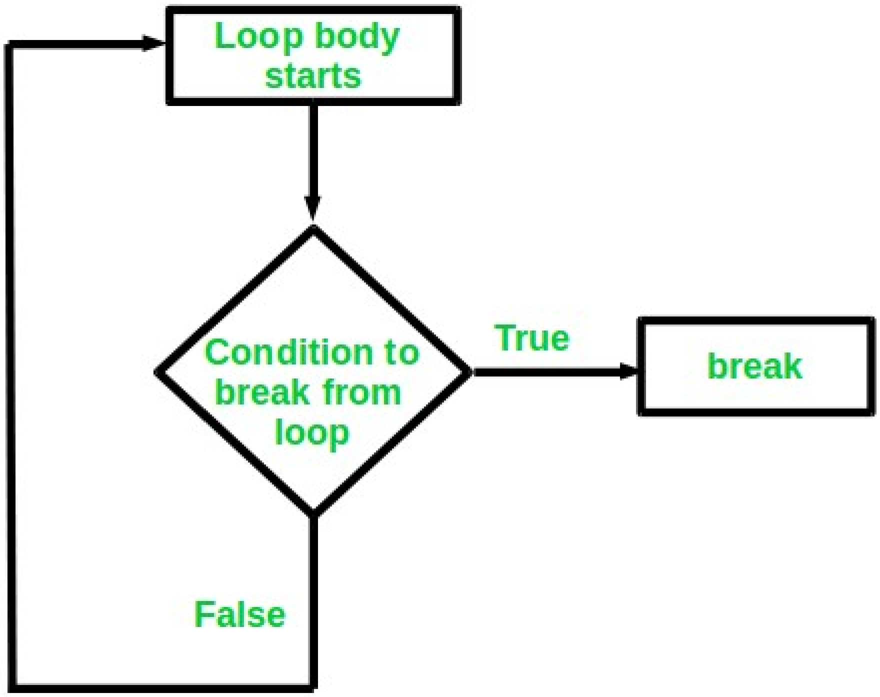
Note : "goto" is unconditional looping statement.

C break: This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

Syntax:

```
break;
```

Basically break statements are used in the situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.

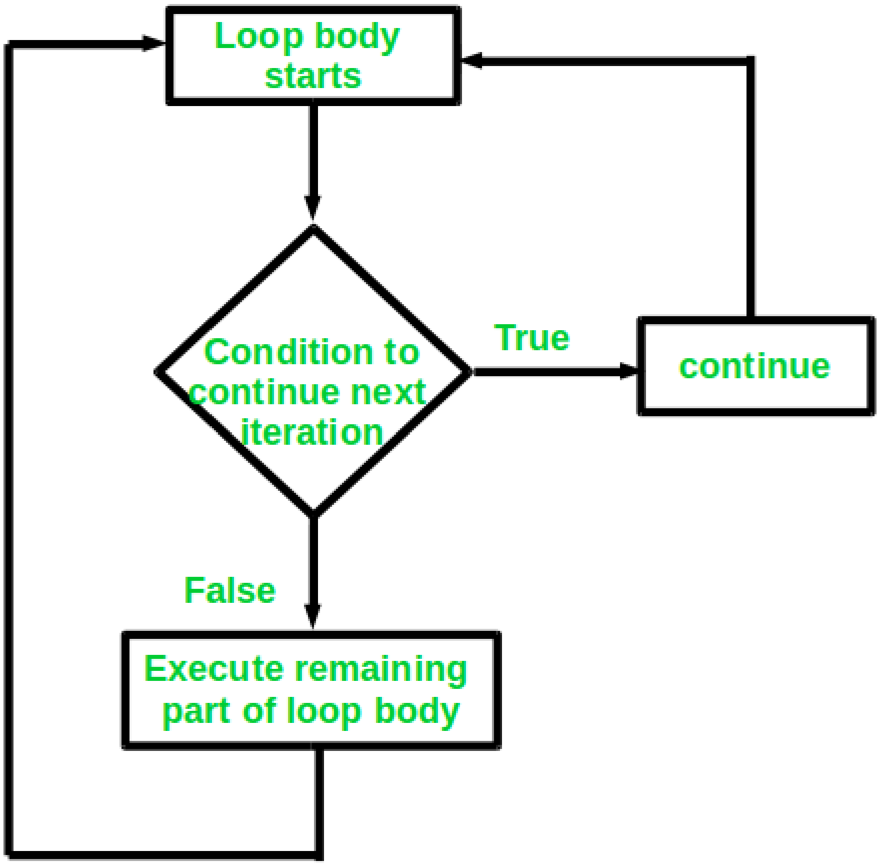


C continue: This loop control statement is just like the break statement.
The *continue* statement is opposite to that of break *statement*, instead of terminating the loop,

it forces to execute the next iteration of the loop.
As the name suggest the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and next iteration of the loop will begin.

Syntax:

```
continue;
```


Loop body starts
Condition to continue next iteration
True
continue
False
Execute remaining part of loop body