C if else Statement:

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.

There are the following variants of if statement in C language.

- If statement
- If-else statement
- If else-if ladder
- Nested if

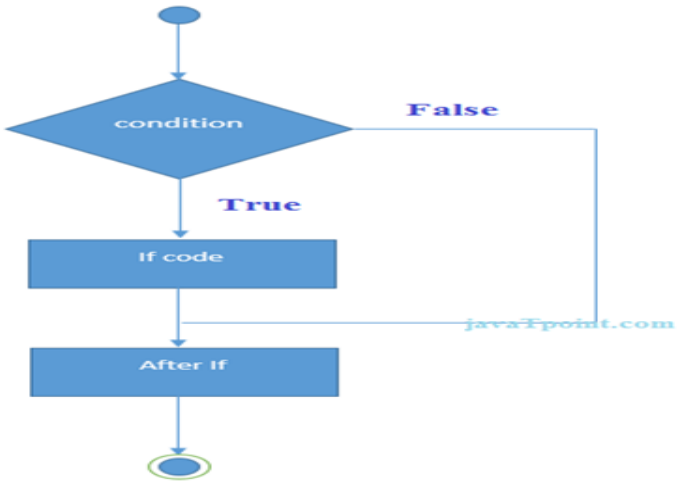
If Statement

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions.

syntax of the if statement:

```
if(expression)
{
        //code to be executed
}
```

Flowchart of if statement in C



```
#include<stdio.h>
int main()
{
int number=0;
if(number<=5)
{
printf (“0 is less than 5”);
}
```

Output

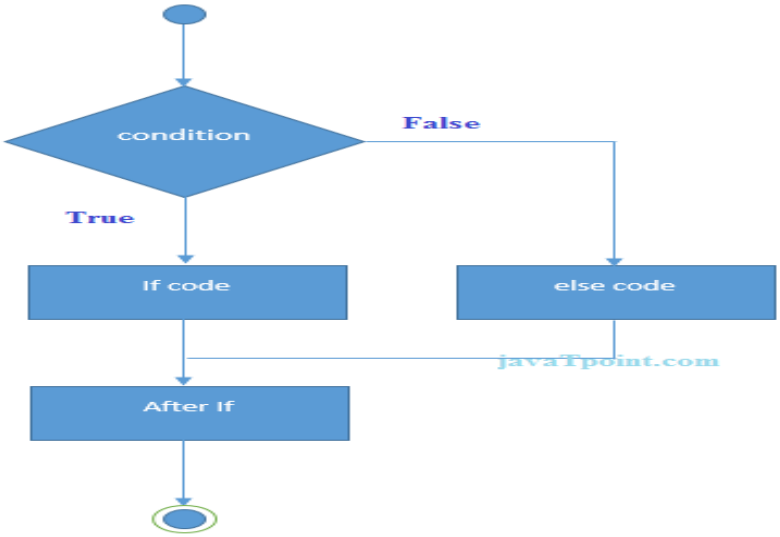
```
0 is less than 5
```

If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simultaneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition.

syntax of the if-else statement:

```
if(expression)
{
        //code to be executed if condition is true
}
else
{
        //code to be executed if condition is false
}
```




```
#include<stdio.h>
int main()
{
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0)
{
printf("%d is even number",number);
}
Else
{
printf("%d is odd number",number);
}
return 0;
}
```

Output

```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

Nested If else Statement

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

Syntax of nested id else statement:

```
if(condition1)
{
        //code to be executed if condition1 is true
}
else if(condition2)
{
        //code to be executed if condition2 is true
}
else if(condition3)
{
        //code to be executed if condition3 is true
}
...
else
{
    //code to be executed if all the conditions are false
}
```

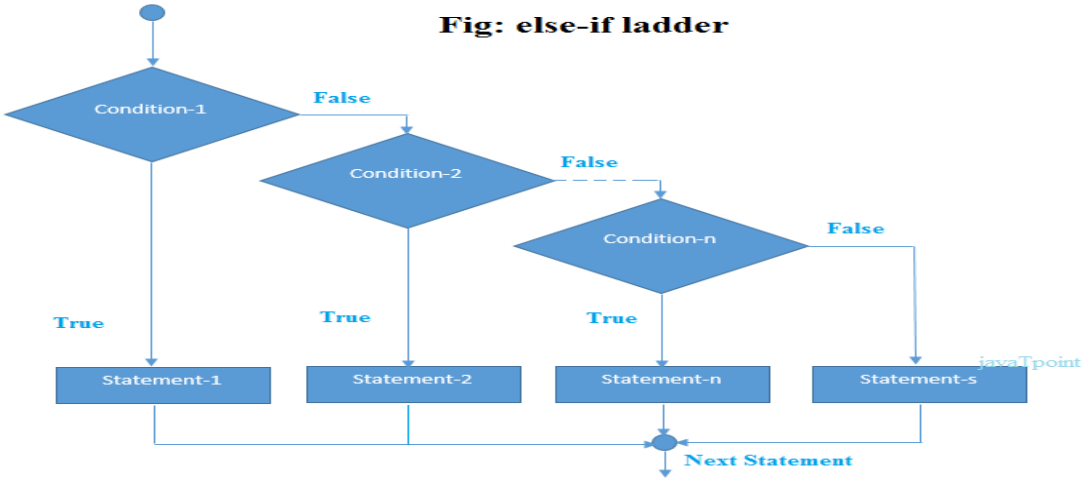


Fig: else-if ladder

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number==10)
{
        printf("number is equals to 10");
}
else if(number==50)
{
    printf("number is equal to 50");
}
else if(number==100)
{
    printf("number is equal to 100");
}
else
{
    printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

C Switch Statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable. Here, we can define various statements in the multiple cases for the different values of a single variable.

Syntax of switch statement:

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....
default:
 code to be executed if all cases are not matched;
}
```

Rules for switch statement in C language:

1) The switch expression must be of an integer or character type.

2) The case value must be an integer or character constant.

3) The case value can be used only inside the switch statement.

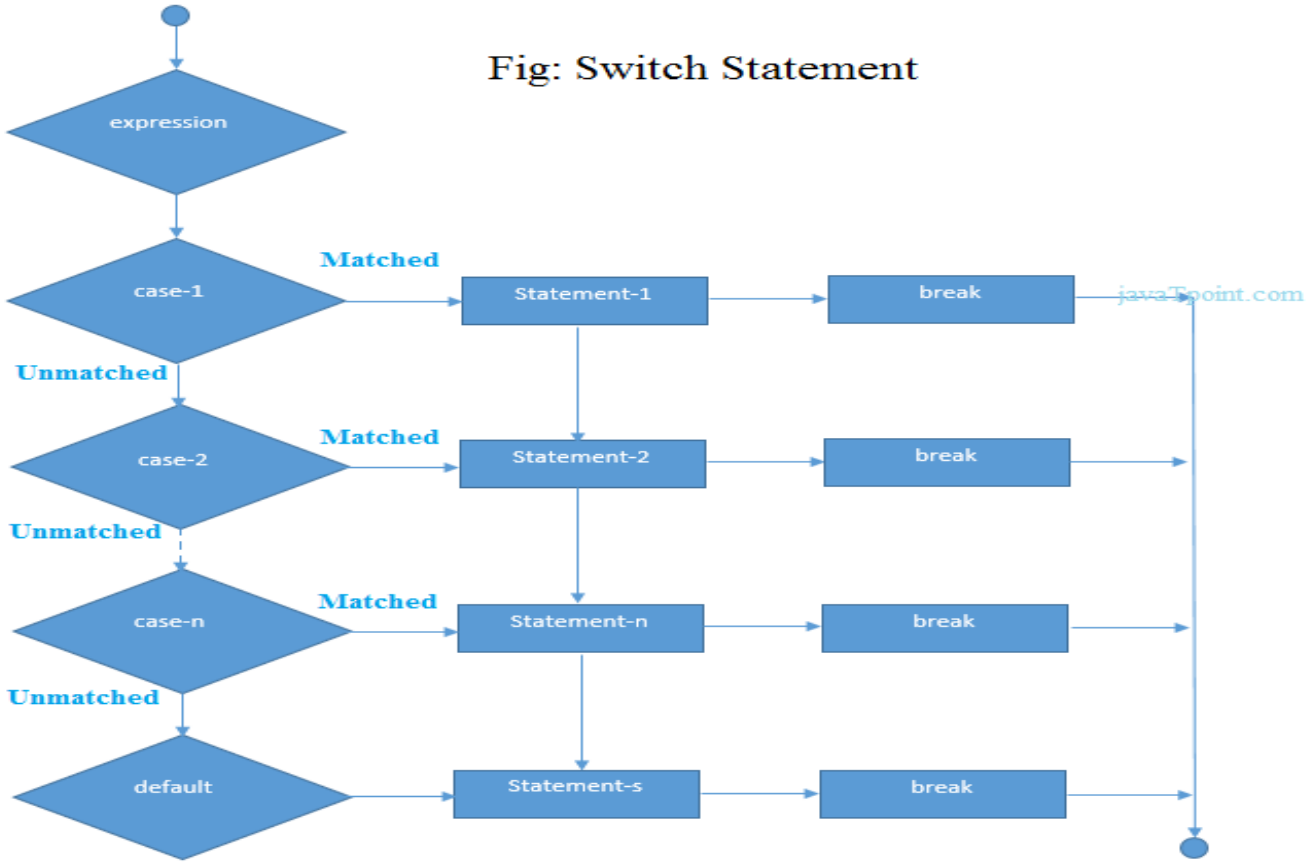
4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.

Let's try to understand it by the examples. We are assuming that there are following variables.

1. int x,y,z;
2. char a,b;
3. float f;

Valid Switch	Invalid Switch	Valid Case	Invalid Case
switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x,y))		case 'x'>'y';	case 1,2,3;

Flowchart of switch statement in C



Functioning of switch case statement

First, the integer expression specified in the switch statement is evaluated. This value is then matched one by one with the constant values given in the different cases. If a match is found, then all the statements specified in that case are executed along with the all the cases present after that case including the default statement. No two cases can have similar values. If the matched case contains a break statement, then all the cases present after that will be skipped, and the control comes out of the switch. Otherwise, all the cases following the matched case will be executed.

example of switch statement.

#include<stdio.h>

```
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
switch(number){
case 10:
printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output:

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

C Switch statement is fall-through

In C language, the switch statement is fall through; it means if you don't use a break statement in the switch case, all the cases after the matching case will be executed.

Let's try to understand the fall through state of switch statement by the example given below.

```
1. #include<stdio.h>
2. int main(){
3. int number=0;
4. printf("enter a number:");
```

```
5. scanf("%d",&number);
6. switch(number){
7. case 10:
8. printf("number is equal to 10\n");
9. case 50:
10.printf("number is equal to 50\n");
11.case 100:
12.printf("number is equal to 100\n");
13.default:
14.printf("number is not equal to 10, 50 or 100");
15.}
16.return 0;
17.}
```

Output:

```
enter a number:10
number is equal to 10
number is equal to 50
number is equal to 100
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
number is equal to 100
number is not equal to 10, 50 or 100
```

C Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1) It provides code reusability.

Using loops, we do not need to write the same code again and again.

3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in C language that is given below:

1. while
2. do-while
3. for

while loop in C

While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

Properties of while loop

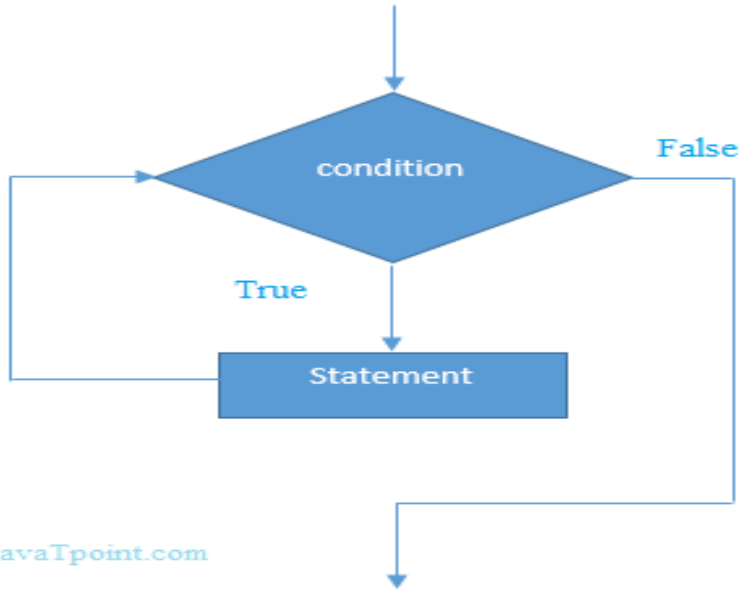
- A conditional expression is used to check the condition. The statements defined inside the while loop will repeatedly execute until the given condition fails.
- The condition will be true if it returns 0. The condition will be false if it returns any non-zero number.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

Syntax of while loop

The syntax of while loop in c language is given below:

```
while(condition){
//code to be executed
}
```

Flowchart of while loop in C



Example of the while loop in C language

```
#include<stdio.h>
int main(){
int i=1;
while(i<=10){
printf("%d \n",i);
i++;
}
return 0;
}
```

Output

```
1
2
3
4
5
6
```

```
7
8
9
10
```

Infinitive while loop in C

If the expression passed in while loop results in any non-zero value then the loop will run the infinite number of times.

```
1. while(1){
2. //statement
3. }
```

Example of the while loop in C language

```
#include<stdio.h>
int main()
{
while(1)
{
printf("Infinite while loop");
}
return 0;
}
```

do-while loop in C

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

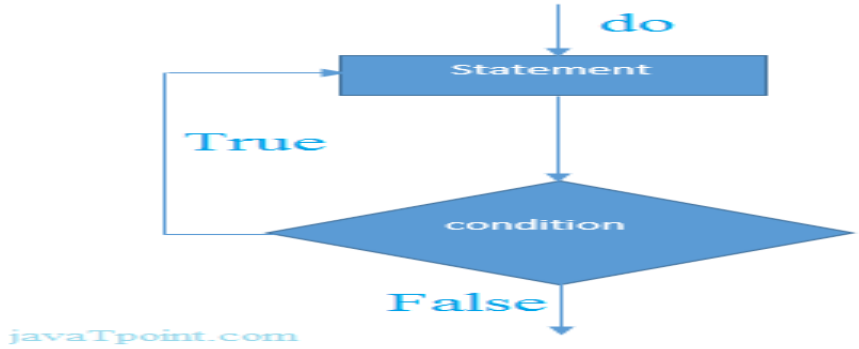
The syntax of do-while loop

```
do{

//code to be executed
```

}while(condition);

Flowchart of do while loop



do while example

```
#include<stdio.h>
int main(){
int i=1;
do{
printf("%d \n",i);
i++;
}while(i>=10);
return 0;
 }
```

Output

```
1
```

Infinitive do while loop

The do-while loop will run infinite times if we pass any non-zero value as the conditional expression.

```
1. do{
2. //statement
3. }while(1);
```

Example of infinite do-while loop:

```
#include<stdio.h>
```

```
int main()
{
do
{
printf("Infinite do-while loop");
}while(1);
return 0;
 }
```

for loop in C

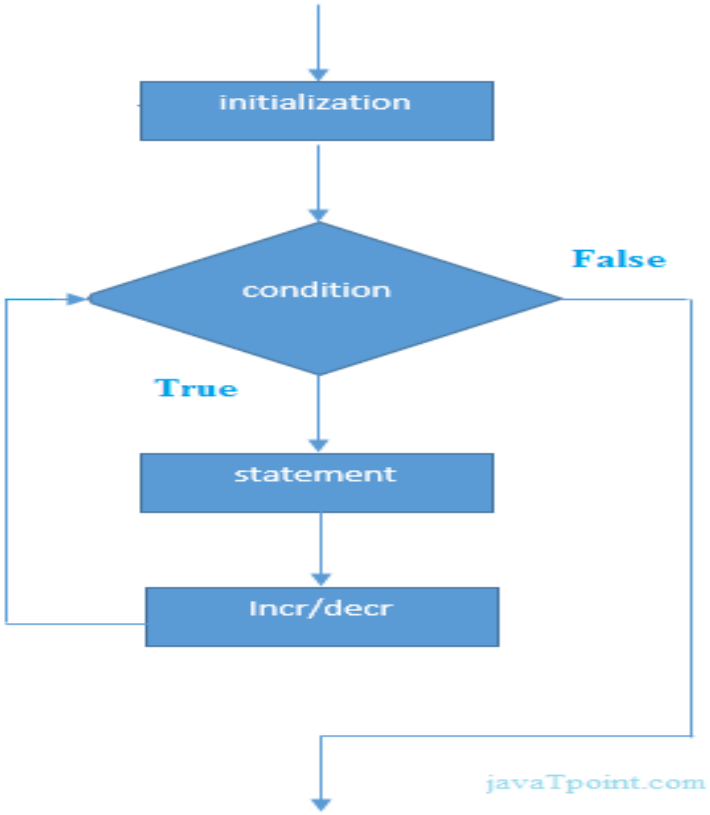
The for loop in C language is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

Syntax of for loop in C

The syntax of for loop in c language is given below:

```
for(Initialization;Condition;incr/decr){
//code to be executed
}
```

Flowchart of for loop in C



C for loop Examples

```
#include<stdio.h>
int main(){
int i=0;
for(i=1;i<=10;i++){
printf("%d \n",i);
}
return 0;
}
```

Output

```
1
2
3
4
5
6
```

```
7
8
9
10
```

Properties of Initialization

- The expression represents the initialization of the loop variable.
- We can initialize more than one variable in Expression 1.
- Expression 1 is optional.
- In C, we cannot declare the variables in Expression 1. However, It can be an exception in some compilers.

Properties of condition

- Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.
- Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.
- Expression 2 is optional.
- Expression 2 can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.
- We can pass zero or non-zero value in expression 2. However, in C, any non-zero value is true, and zero is false by default.

Properties of Increment

- Expression 3 is used to update the loop variable.
- We can update more than one variable at the same time.
- Expression 3 is optional.

Infinitive for loop in C

To make a for loop infinite, we need not give any expression in the syntax. Instead of that, we need to provide two semicolons to validate the syntax of the for loop. This will work as an infinite for loop.

Example of infinitive for loop

```
#include<stdio.h>
void main ()
{
   for(;;)
   {
      printf("welcome to javatpoint");
   }
}
```

If you run this program, you will see above statement infinite times.

Nested Loops in C

C supports nesting of loops in C. Nesting of loops is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define 'while' loop inside a 'for' loop.

Syntax of Nested loop

```
1. Outer_loop
2. {
3.     Inner_loop
4.    {
5.          // inner loop statements.
6.    }
7.        // outer loop statements.
8. }
```

Outer_loop and Inner_loop are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

Nested for loop

The nested for loop means any type of loop which is defined inside the 'for' loop.

```
1. for (initialization; condition; update)
2. {
3.     for(initialization; condition; update)
4.     {
5.           // inner loop statements.
```

```
6.      }
7.      // outer loop statements.
8.  }
```

Example of nested for loop

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.     int i;// variable declaration
5.     // Displaying the n tables.
6.     for(i=1;i<=3;i++)  // outer loop
7.     {
              int j;
8.          for( j=1;j<=3;j++)  // inner loop
9.          {
10.             printf("%d\t",i,j); // printing the value.
11.         }
12.         printf("\n");
13.   }
```

Output

```
11
12
13
21
22
23
31
32
33
```

Explanation of the above code

- First, the 'i' variable is initialized to 1 and then program control passes to the i<=n.
- The program control checks whether the condition 'i<=n' is true or not.
- If the condition is true, then the program control passes to the inner loop.
- The inner loop will get executed until the condition is true.
- After the execution of the inner loop, the control moves back to the update of the outer loop, i.e., i++.
- After incrementing the value of the loop counter, the condition is checked again, i.e., i<=n.
- If the condition is true, then the inner loop will be executed again.

- This process will continue until the condition of the outer loop is true

Nested while loop

The nested while loop means any type of loop which is defined inside the 'while' loop.

```
1. while(condition)
2. {
3.     while(condition)
4.     {
5.         // inner loop statements.
6.     }
7. // outer loop statements.
8. }
```

Nested do..while loop

The nested do..while loop means any type of loop which is defined inside the 'do..while' loop.

```
1. do
2. {
3.    do
4.   {
5.       // inner loop statements.
6.    }while(condition);
7. // outer loop statements.
8. }while(condition);
```

C break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With loop
2. With loop

Syntax:

1. //loop
2. break;

Flowchart of break in c

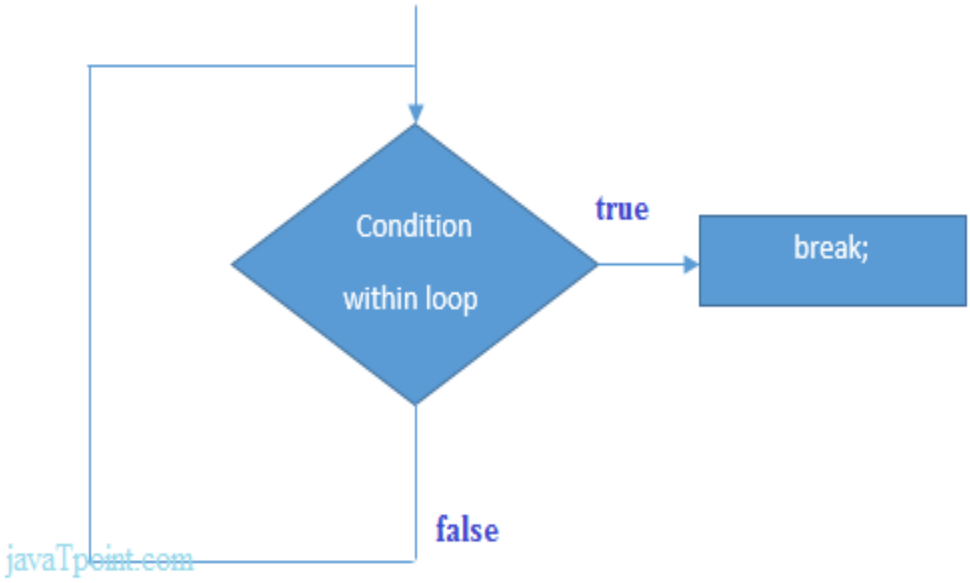


Figure: Flowchart of break statement

Example

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
   int i;
   for(i = 0; i<10; i++)
   {
      printf("%d ",i);
      if(i == 5)
          break;
```

```
}
printf("came outside of loop i = %d",i);
}
```

Output

```
0 1 2 3 4 5 came outside of loop i = 5
```

break statement with the nested loop

In such case, it breaks only the inner loop, but not outer loop.

```
#include<stdio.h>
int main(){
int i=1,j=1;//initializing a local variable
for(i=1;i<=3;i++){
for(j=1;j<=3;j++){
printf("%d &d\n",i,j);
if(i==2 && j==2){
break;//will break loop of j only
}
}//end of for loop
return 0;
}
```

Output

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

As you can see the output on the console, 2 3 is not printed because there is a break statement after printing i==2 and j==2. But 3 1, 3 2 and 3 3 are printed because the break statement is used to break the inner loop only.

C continue statement

The continue statement in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

```
//loop statements
continue;
//some lines of the code which is to be skipped
```

Continue statement example

```
#include<stdio.h>
int main(){
int i=1;//initializing a local variable
//starting a loop from 1 to 10
for(i=1;i<=10;i++){
if(i==5){//if value of i is equal to 5, it will continue the loop
continue;
}
printf("%d \n",i);
}//end of for loop
return 0;
}
```

Output

```
1
2
3
4
6
7
8
9
```

10

As you can see, 5 is not printed on the console because loop is continued at i==5.

C continue statement with inner loop

In such case, C continue statement continues only inner loop, but not outer loop.

```
#include<stdio.h>
int main(){
int i=1,j=1;//initializing a local variable
for(i=1;i<=3;i++){
for(j=1;j<=3;j++){
if(i==2 && j==2){
continue;//will continue loop of j only
}
printf("%d %d\n",i,j);
}
}//end of for loop
return 0;
}
```

Output

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

C goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement.

However, using goto is avoided these days since it makes the program less readable and complicated.

Syntax:
label:
//some part of the code;
goto label;

goto example

```
#include<stdio.h>

int main()

{

int age;

ineligible:

printf("You are not eligible to vote.\n");

printf("Enter your age:\n");

scanf("%d",&age);

if(age<18)

{

goto ineligible;

}

else

{

printf("You are eligible to vote.");

}
```

```
return 0;
}
```

Output

```
You are not eligible to vote.
Enter your age:
12
You are not eligible to vote.
Enter your age:
25
You are eligible to vote.
```