Unit-3

Control Structures / Statements In C Language

A program is nothing but the execution of sequence of one or more instructions.

Quite often, it is desirable to alter the sequence of the statements in the program depending upon certain circumstances. (i.e., we have a number of situations where we may have to change the order of execution of statements based on certain conditions)

(Or)

Repeat a group of statements until certain specified conditions are met.

This involves a kind of decision making to see whether a particular condition has occurred or not and direct the computer to execute certain statements accordingly.

Based on application, it is necessary / essential

(i) To alter the flow of a program

(ii) Test the logical conditions

(iii) Control the flow of execution as per the selection these conditions can be placed in the program using decision-making statements.

"C" SUPPORTS MAINLY THREE TYPES OF CONTROL STATEMENTS.

1. Decision making statements

a) Simple if Statement

b) if – else Statement

c) Nested if-else statement

d) else – if Ladder

e) switch statement

2. Loop control statements

a) for Loop

b) while Loop

c) do-while Loop

3. Unconditional control statements

a) goto Statement

b) break Statement

c) continue Statement

d) return statement

Decision Making Statements In C Language

These statements change the flow of execution depending on a given logical condition. Used to execute or skip some set of instructions based on given condition.

1) Simple "if" statement :

"if" statement is a powerful decision making statement, and is used to control the flow of execution of statements.

Syntax :

```
if (Condition or test expression)

Statement;

Rest of the program

(OR)

if (Condition or test expression)

{

Statement;

}

Rest of the program;
```

* It is basically a "Two-way" decision statement (one for TRUE and other for FALSE)

* It has only one option.

* The statement as executed only when the condition is true.

* In case the condition is false the compiler skips the lines within the "if Block".

* The condition is always enclosed within a pair of parenthesis i.e. ( ) ,

* The conditional statement should not terminated with Semi-colons (ie ;)

* The Statements following the "if"-statement are normally enclosed in Curly Braces {}.

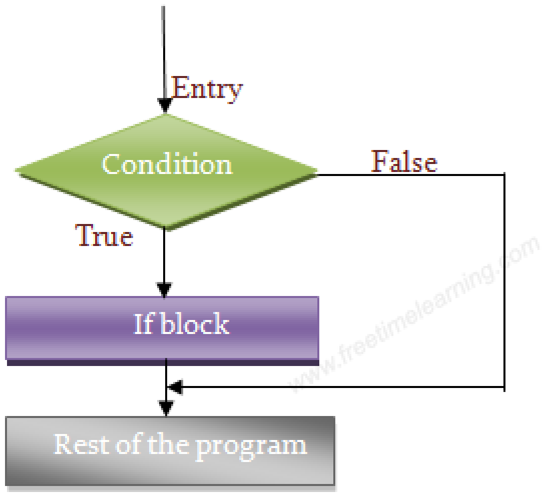
* The Curly Braces indicates the scope of "if" statement. The default scope is one statement. But it is good practice to use curly braces even with a single statement.

* The statement block may be a single statement or a group of statements.

* If the Test Expression / Conditions are TRUE, the Statement Block will be executed and executes rest of the program.

* If the Test Expression / Condition are FALSE, the Statement Block will be skipped and rest of the program executes next.

Flow chart for "if" statement :



Program: Write a program to check equivalence of two numbers. Use "if" statement.

```
# include<stdio.h>
 # include<conio.h>
void main( )
 {
    int  m,n;
    clrscr( );
    printf("\n Enter two numbers:");
    scanf("%d %d", &m, &n);
          if((m-n)==0)
          printf("\n two numbers are equal");
    getch();
}
```

Output :

```
Enter two numbers: 5 5
Two numbers are equal.
```

<u>(2) "if-else" Statement :</u>

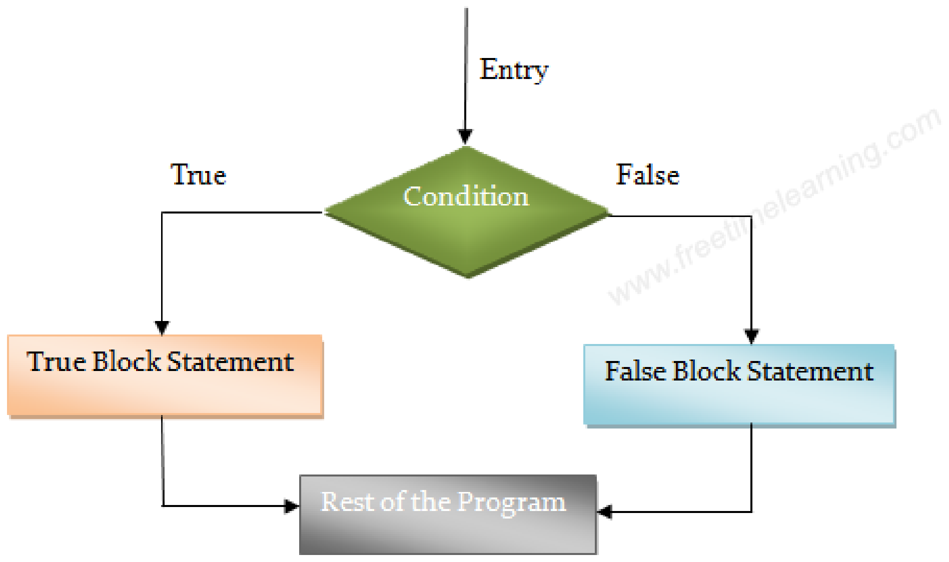
It is observed that the "if" statement executes only when the condition following if is true. It does nothing when the condition is false.

In if-else either True-Block or False – Block will be executed and not both. The "else" Statement cannot be used without "if".

Syntax :

```
if ( Test Expression or Condition )
     {
        Statements;  /*true block (or) if block */
      }
     else
      {
        Statements;   /* false block (or) else block */
      }
```

Flow chart :



Program: Write a program to print the given number is even or odd.

```
# include<stdio.h>

# include<conio.h>

main( )

  {

          int n;

clrscr( );

printf("Enter a number :");

scanf("%d", &n);

 if( (n%2)==0 )

        printf("\n The given number is EVEN ");

          else

        printf("\n The given number is ODD ");

          getch( );

  }
```

Output :

```
Run 1 :
Enter a number : 24
The given number is EVEN
Run 2: /* that means one more time we run the program */
Enter a number : 17
The given number is ODD
```

* Program: Write a program to print the given number is even or odd.

```
# include<stdio.h>
# include<conio.h>
void main( )
  {
        int a,b;
clrscr( );
printf("Enter Two numbers:");
scanf("%d%d", &a,&b);
 if( a>b )
      printf("\n %d is largest number",a);
   else
      printf("\n %d is largest number",b);
    getch( );
  }
```

Output :

```
Enter Two numbers: 13 30
30 is largest number
```

<u>(3) Nested "if-else" Statement :</u>

Using of one if-else statement in another if-else statement is called as nested if-else control statement.

When a series of decisions are involved, we may have to use more than one if-else statement in nested form.

Syntax :

```
if ( Test Condition1)
 {
if ( Test Condition2)
   {
     Statement -1;
   }
else
  {
     Statement -2;
  }
}
else
{
 if ( Test Condition3)
  {
   Statement -3;
  }
else
{
Statement-4;
 }
} /* end of outer if-else *
```

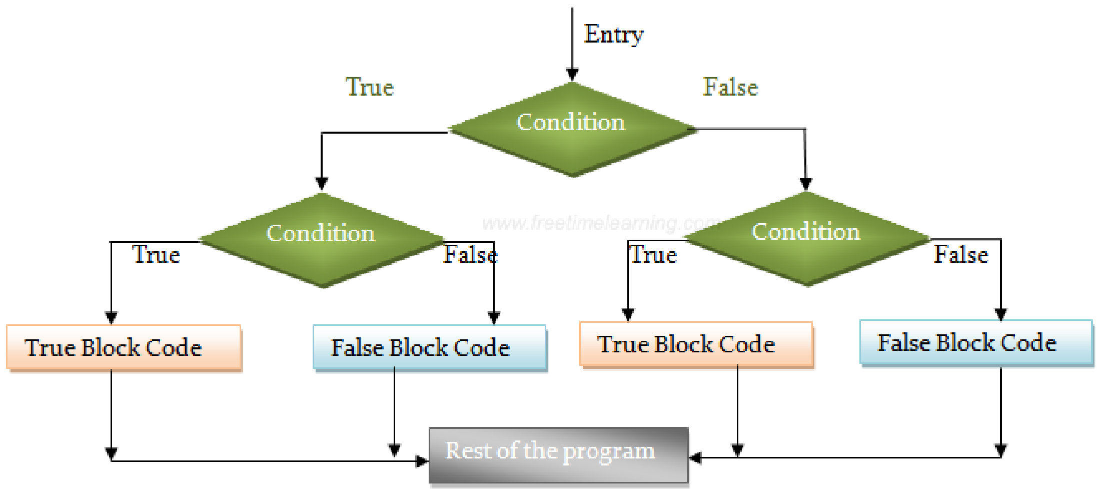
Syntax :

```
# include<stdio.h>

# include<conio.h>

 main( )  {

      float  a,b,c;

      printf("Enter Three Values:");

      scanf("%f%f%f", &a, &b, &c);

      printf("\n Largest Value is:") ;


    if(a>b) {

                   if(a>c)

              printf(" %f ", a);

           else

              printf(" %f ", c);

          }

     else {

          if (b>c)

                printf(" %f ", b);

          else

              printf(" %f ", c);

        }

getch();

     }
```

If Test Condition-1 is true then enter into outer if block, and it checks Test Condition2,if it is true then Statement-1 executed if it is false then else block executed i.eStatement-2.

If Test Condition -1 is false then it skips the outer if block and it goes to else block and Test Condition-3 checks if it is true then Statement-3 executed, else Statement-4 executed.

Flow chart :



* Program: Program to select and print the largest of the three float numbers using nested "if else" statements.

```
if ( Test Condition1)
 {
if ( Test Condition2)
  {
    Statement -1;
  }
else
 {
    Statement -2;
 }
}
else
{
 if ( Test Condition3)
 {
  Statement -3;
  }
else
{
Statement-4;
 }
} /* end of outer if-else *
```

Output :

```
Run 1: Enter three values: 9.12 5.34 3.87
   Largest Value is: 9.12
Run 2: Enter three values: 45.781 78.34 145.86
   Largest Value is: 145.86
```

 * Program: Program to select and print the largest of the three float numbers using nested "if else" statements.

```
# include<stdio.h>
# include<conio.h>
 main( )  {
     float  a,b,c;
     printf("Enter Three Values:");
     scanf("%f%f%f", &a, &b, &c);
     printf("\n Largest Value is:") ;
            if(a>b) {
                  if(a>c)
              printf(" %f ", a);
            else
              printf(" %f ", c);
          }
    else {
        if (b>c)
             printf(" %f ", b);
        else
           printf(" %f ", c);
      }
getch();
   }
```

Output :

```
Run 1: Enter three values: 9.12 5.34 3.87
   Largest Value is: 9.12
Run 2: Enter three values: 45.781 78.34 145.86
   Largest Value is: 145.86
```

<u>(4) The "else – if" Ladder :</u>

This is another way of putting if's together when multiple decisions are involved. A multipath decision is a chain of if's in which the statement associated with each else is an if. Hence it forms a ladder called else–if ladder.

if else-if, statement is used to execute one code from multiple conditions.

Syntax :

```
if (Test Condition -1)
 {
Statement -1;
  }
   else if ( Test Condition -2)
{
Statement -2;
}
```

```
 else if ( Test Condition -3)
{
 Statement -3;
}
   :
   :
   :
   :
  else  if ( Test Condition –n)
{
Statement –n;
}
  else
{
default statement;
 }
Rest of the Program Statements-X;
```

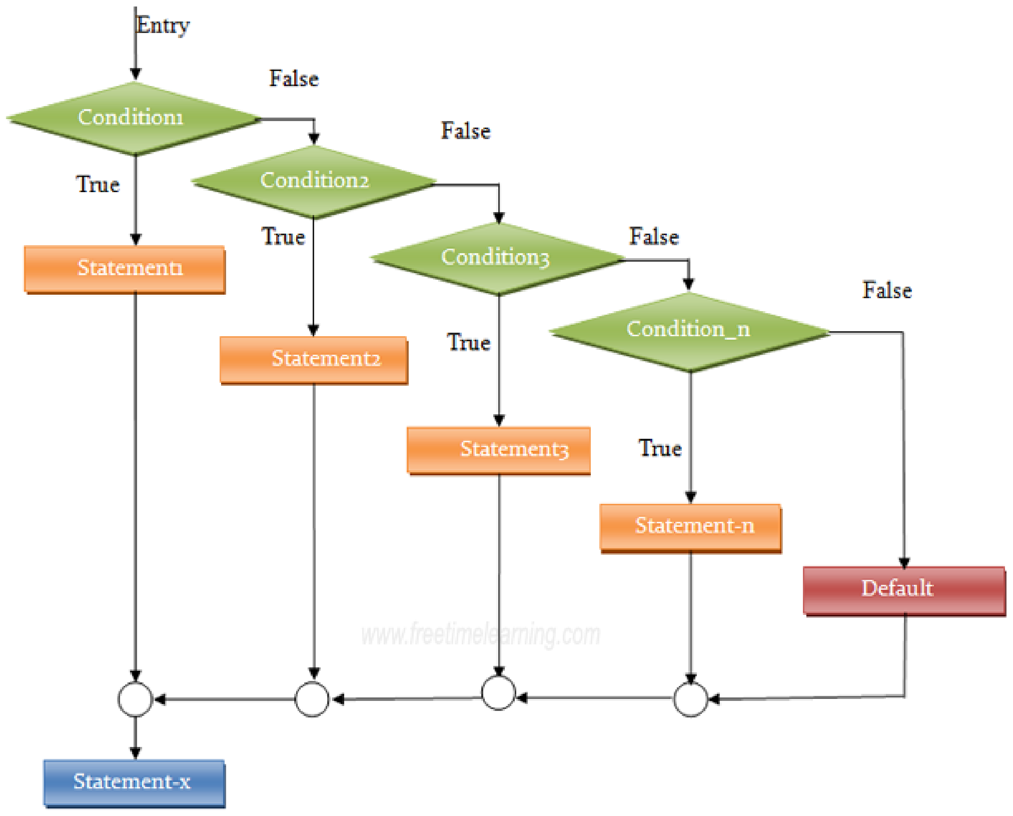
*The above construction is known as else if ladders.

*The conditions are evaluated from top to bottom.

*As soon as a true condition is found, the statement associated with it is executed and The control is transferred to the Rest of the Program Statement–X (skipping rest of the ladder).

*When all the "n" conditions become false, then the final else containing the default statement will be executed.

Flow chart :



*Program: Write a program to read three numbers and find the largest one by using "else-if" ladder.

```
# include<stdio.h>

# include<conio.h>

void main( )

{

  int  a, b, c

clrscr ( ) ;

printf("Enter 1st  number:");

scanf("%d", &a);

printf("Enter 2nd  number:");

scanf("%d", &b);

printf("Enter 3rd  number:");

scanf("%d", &c);
```

```
   if ((a>b) && (a>c))
          printf("Highest Number is: %d", a);
           else if ((b>a) && (b>c))
          printf("Highest Number is: %d", b);
            else
        printf("Highest Numbers is: %d", c);
getch( );
}
```

Output :

```
Enter 1st number: 52
Enter 2nd number: 90
Enter 3rd number: 74
Highest Numbers is: 90
```

5) The "switch-case" Statement :

Switch is another conditional control statement used to select one option from several options based on given expression value; this is an alternative to the if-else-if ladder.

The switch statement causes a particular group of statements to be chosen from several available groups.

The selection is based upon the current value of an expression which is included with in the switch statement.

The switch statement is a multi-way branch statement.

In a program if there is a possibility to make a choice from a number of options, this structured selected is useful.

The switch statement requires only one argument of int or char data type, which is checked with number of case options.

The switch statement evaluates expression and then looks for its value among the case constants.

If the value matches with case constant, then that particular case statement is executed.

If no one case constant not matched then default is executed.

Here switch, case and default are reserved words or keywords.

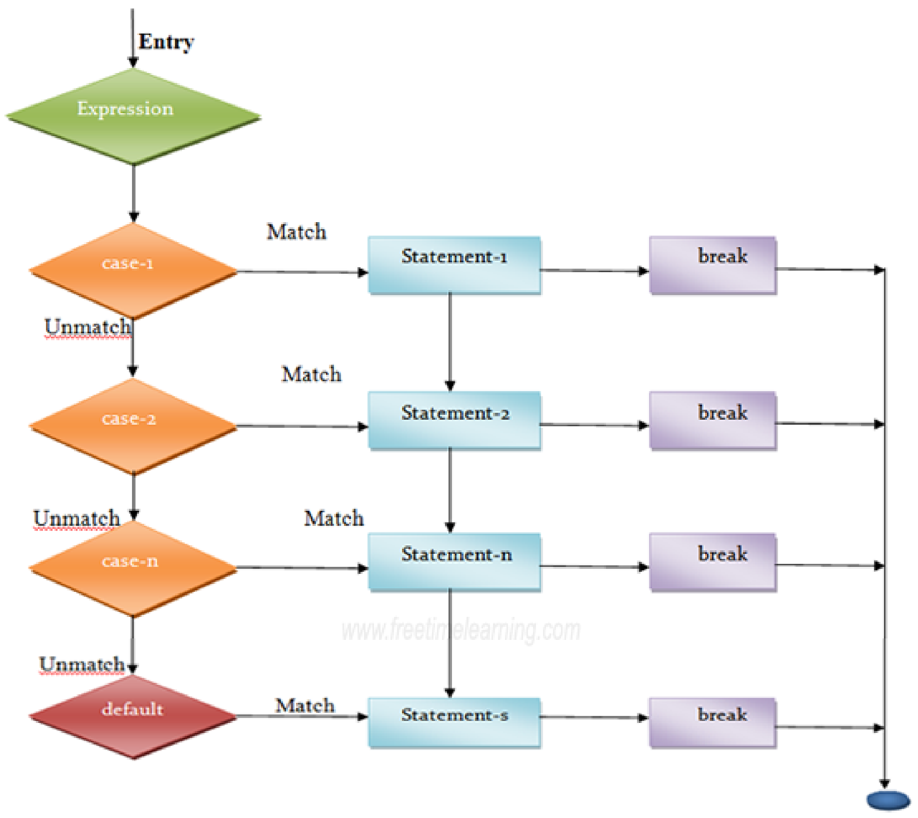
Every case statement terminates with colon ":".

In switch each case block should end with break statement, i.e.

Syntax :

```
Switch (variable or expression)
           {
             Case Constantvalue-1:Block -1;
                (Or)
                Statement-1;
                break;
             Case Constantvalue-2:Block -2;
                    (Or)
                    Statement-2;
               break;
          ________
       ________
         Case Constant value-n:          Block -n;
                    (Or)
          Statement-n;
                    break;

           default: default – block;
          }
```

Flow chart :



* Program: Write a program to provide multiple functions such as 1.Addition, 2.Subtraction, 3.Multiplication, 4. Division, 5.Remainder, 6.Larger out of two, 7. Exit, using "switch" statement.

```
# include<stdio.h>

# include<conio.h>

main( )

{

    int a, b, c, ch;

    clrscr ( ) ;

    printf("\t = = = = = = = = = = = = = =");

    printf ("n\t MENU");

    printf("\n\t=  = = = = = = = = = =");

    printf("\n \t [1] ADDITION" );

    printf("\n \t [2] SUBTRACTION" );
    printf("\n \t [3] MULTIPLICATION" );
    printf("\n \t [4] DIVISION" );
    printf("\n \t [5] REMAINDER" );
    printf("\n \t [6] LARGER OUT OF TWO" );
    printf("\n \t [7] EXIT" );
    printf("\n \t = = = = = = = = = =");
    printf(" \n\n\t ENTER YOUR CHOICE:");
      scanf("%d", &ch);
      if(ch <= 6 && ch >=1)
        {
         printf("ENTER TWO NUMBERS:");
          scanf("%d %d", &a, &b);
         }
        switch(ch)
         {
             case 1: c = a+b ;
         printf(" \n Addition: %d", c);
        break;
         case 2: c=a-b;
        printf("\n Subtraction: %d", c);
        break;
         case 3: c = a* b ;
    printf("\n Multiplication: %d", c);
        break;
         case 4: c = a / b;
    printf("\n Division: %d", c);
       break;
```

```
        case 5: c = a % b;
    printf(" \n Remainder: %d", c);
    break;
       case 6: if (a > b)
    printf("\n \t %d is larger than %d", a, b);
     else if (b > a)
    printf(" \n \t %d is larger than %d ", b, a);
      else
    printf("\n \t %d and %d are same", a, b);
    break;
    case 7:  printf( " \ n Terminated by choice");
    exit( );
    break;
    default: printf(" \ n invalid choice");
     }
    getch ( );
}
```

Output :

```
= = = = = = = = =
     MENU
= = = = = = = = =
[1] ADDITION
[2] SUBTRACTION
[3] MULTIPLICATION
[4] DIVISION
```

```
[5] REMAINDER

[6] LARGER OUT OF TWO

[7] EXIT

= = = = = = = = = = = = = = = =

Enter your choice: 6

Enter two numbers: 8 9

9 is larger than 8
```

Loop Control Statements In C Language

Loop : A loop is defined as a block of statements which are repeatedly executed for certain number of times.

In other words it iterates a code or group of code many times.

Why use loops in c language : Suppose you want to execute some code/s 10 times. You can perform it by writing that code/s only one time and repeat the execution 10 times using loop. For example: Suppose that you have to print table of 2 then you need to write 10 lines of code, by using loop statements you can do it by 2 or 3 lines of code only.

(1) for Loop :

The for loop works well where the number of iterations of the loop is known before the loop is entered. The head of the loop consists of three parts separated by semicolons.The first is run before the loop is entered. This is usually the initialization of the loop variable.

The second is a test, the loop is exits when this returns false.

The third is a statement to be run every time the loop body is completed. This is usually an increment of the loop counter.

The 3 actions are

“Initialize expression”, “Test Condition expression” and “updation expression”

The expressions are separated by Semi-Colons (;).

Syntax :

```
for (initialize expression; test condition; updation )

 {
   Statement-1;
   Statement-2;
 }
```

(i)The initialization sets a loop to an initial value. This statement is executed only once.

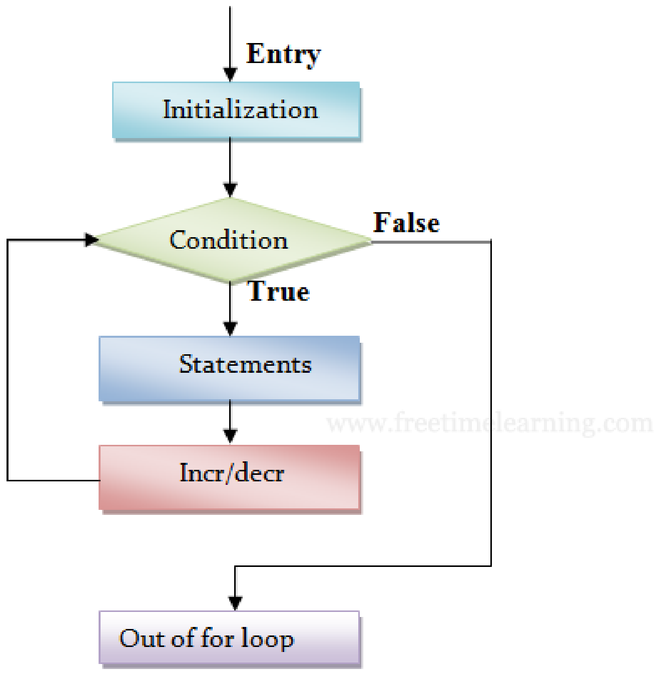
(ii) The test condition is a relational expression that determines the number of iterations desired or it determines when to exit from the loop.

For loop continues to execute as long as conditional test is satisfied.

When the condition becomes false the control of the program exits from the body of for loop and executes next statements after the body of the loop.

(iii) The updation (increment or decrement operations) decides how to make changes in the loop.

The body of the loop may contain either a single statement or multiple statements.



* Program : Print the first five numbers starting from one together with their squares.

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

```
#include<conio.h>
main( )
 {
int  i;
clrscr( ) ;
for(i = 1; i <=5; i++)
printf("\n Number: %d it's Square: %d", i, i*i);
getch( );
}
```

Output :

```
Number: 1 it's Square: 1
Number: 2 it's Square: 4
Number: 3 it's Square: 9
Number: 4 it's Square: 16
Number: 5 it's Square: 25
```

<u>Nested "for" loop :</u>

We can also use loop within loops. i.e. one for statement within another for statement is allowed in C. In nested for loops one or more for statements are included in the body of the loop. * ANSI C allows up to 15 levels of nesting. Some compilers permit even more.

Syntax :

```
for( initialize ;  test condition ;  updation)   /* outer loop */
{
 for(initialize ;  test condition ;  updation) /* inner loop */
   {
    Body of loop;
   }
}
```

* Program : The following program is an example of nested for loop.

```
# include<stdio.h>
# include<conio.h>
 main ( )
 {
  int  x, i, j ;
    printf("How many lines stars (*) should be print f? :");
    scanf("%d", &x);
    for(i=1; i<=x; i++)
     {
       for (j=1; j < =i;  j++)
      {
        printf( "*");
      }
        printf( " \n");
     }
   getch( );
  }
```

Output :

```
How many lines stars (*) should be print d ? : 5
*
* *
* * *
* * * *
* * * * *
```

* Program : The following program is an example of nested for loop.

```
#include<stdio.h>
#include<conio.h>
main( )
{
   int  i, j, x;
   printf("\n Enter Value of x :");
   scanf("%d", &x);
   for(j=1; j<=x; j++)
   {
     for(i=1; i<=j; i++)
        printf("%3d", i);
        printf("\n");
   }
    printf("\n");
    for(j=x; j>=1; j--)
      {
      for(i=j; i>=1; i--)
        printf("%3d", i);
        printf("\n");
      }
  getch( );
 }
```

Output :

```
Enter value of x : 5
1
1   2
1    2    3
```

```
1      2      3      4

1        2        3        4        5

5        4        3        2        1

4      3      2      1

3    2    1

2   1

1
```

<u>(2) The “while” loop :</u>

The while is an entry-controlled loop statement.

The test condition is evaluated and if the condition is true, then the body of the loop is executed.

The execution process is repeated until the test condition becomes false and the control is transferred out of the loop.

On exit, the program continues with the statement immediately after the body of the loop.

The body of the loop may have one or more statements.

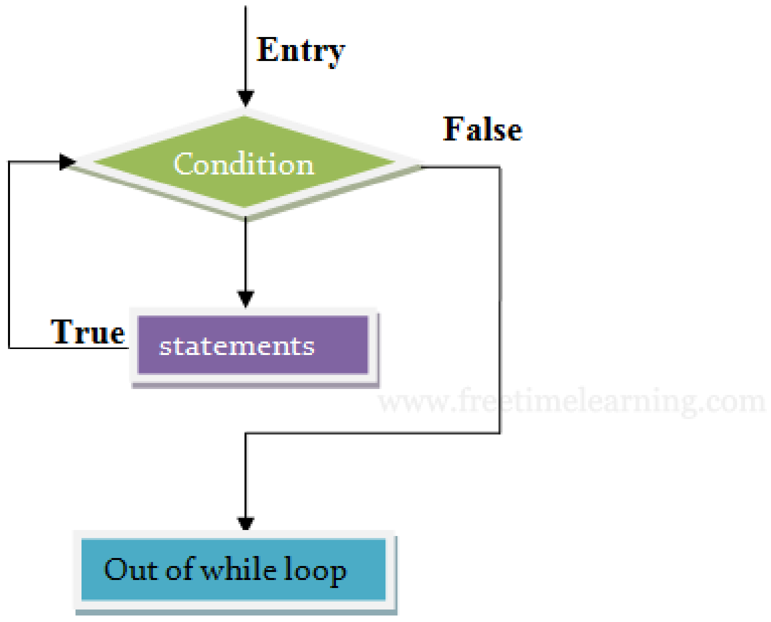
The braces are needed only if the body contains two or more statements.

It’s a good practice to use braces even if the body has only one statement.

The simplest of all the looping structures in C is.

Syntax :

```
Initialization Expression;
  while ( Test Condition)
{
  Body of the loop
  Updation Expression
}
```



* Program : To add 10 consecutive numbers starting from 1. Use the while loop.

```
# include<stdio.h>
# include<conio.h>
main( )
{
  int   a=1, Sum=0;
  while(a<=10)
{
   Sum = Sum + a;
   a++;
}
  printf("Sum of 1 to 10 numbers is: %d", sum);
  getch( );
}
```

Output :

```
Sum of 10 numbers is: 55
```

* Program : To calculate factorial of a given number use while loop.

```
# include<stdio.h>

# include<conio.h>

main ( )

{

    long int  n, fact =1;

    clrscr( ) ;

    printf( "\n Enter the Number:");

    scanf("%ld", &n);

    while(n>=1)

     {

       fact = fact*n;

       n - - ;

      }

  printf(" \n factorial of given number is %d", fact);

  getch( );

}
```

Output :

```
Enter the Number: 5 /* logic 5 * 4 * 3 * 2 * 1 = 120 */

Factorial of given number is 120
```

<u>(3) The "do-while" loop :</u>

To execute a part of program or code several times, we can use do-while loop of C language. The code given between the do and while block will be executed until condition is true.

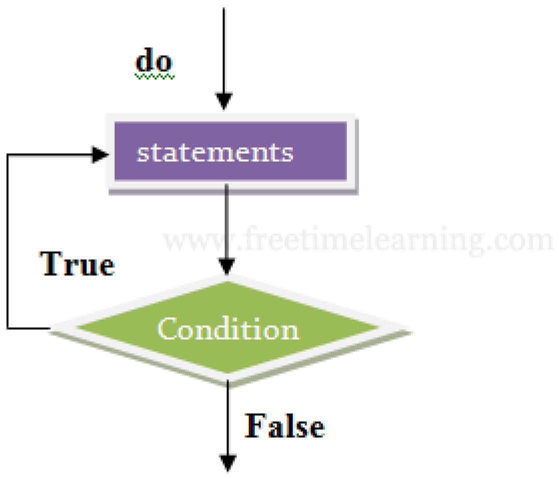
The do-while is an exit-controlled loop statement. Because, in do-while, the condition is checked at the end of the loop.

The do-while loop will execute at least one time even if the condition is false initially.

The do-while loop executes until the condition becomes false.

Syntax :

```
Initialization Expression;
  do
    {
  Body of the loop
  Updation Expression;
  } while ( Test Condition);
```



* Program : To check whether the given number is prime or not.

```
# include<stdio.h>
# include<conio.h>
main( )
 {
  int  n, x=2;
  clrscr( );
  printf( "Enter the number for testing (prime or not");
  scanf("%d", &n);
  do
```

```
    {
     if(n%x == 0)
     {
     printf(" \n the number %d is not prime", n);
     exit(0);
     }
     x++;
    } while ( x < n);
  printf(" \n the number %d is prime", n);
  getch( ) ;
}
```

Output :

```
Enter the number for testing (Prime or not): 5

The number 5 is prime.
```

Unconditional Control Statements In C Language

(1) The "break" Statement :

A break statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop. i.e., the break statement is used to terminate loops or to exit from a switch.

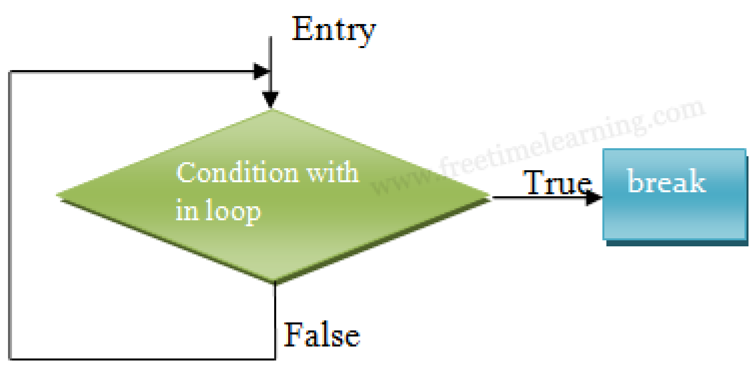
- It can be used within for, while, do-while, or switch statement.
- The break statement is written simply as break;
- In case of inner loops, it terminates the control of inner loop only.

Syntax :

```
Jump-statement;
    break;
```

The jump statement in c break syntax can be while loop, do while loop, for loop or switch case.

Flowchart :



* Program : The following program is an example of break statement

```
#include <stdio.h>

#include <conio.h>

void main(){

int i=1;//initializing a local variable

clrscr();

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

printf("%d ",i);

if(i==5)      //if value of i is equal to 5, it will break the loop

{

break;

}

}//end of for loop

  getch();

}
```

Output :

```
1 2 3 4 5
```

<u>(2) The “continue” Statement :</u>

The continue statement is used to bypass the remainder of the current pass through a loop.

The loop does not terminate when a continue statement is encountered. Instead, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.

The continue statement can be included within a while, do-while, for statement.

It is simply written as "continue".

The continue statement tells the compiler "Skip the following Statements and continue with the next Iteration".

In "while" and "do" loops continue causes the control to go directly to the test –condition and then to continue the iteration process.

In the case of "for" loop, the updation section of the loop is executed before test-condition, is evaluated.

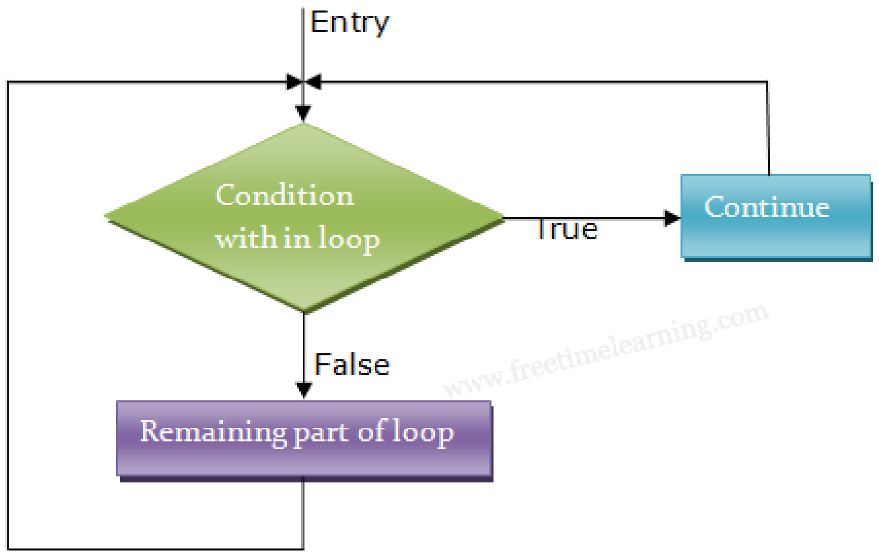
Syntax :

Jump-statement;

    Continue;

The jump statement can be while, do while and for loop.

Flowchart :



* Program : The following program is an example of break statement.

# include<stdio.h>

```
void main( )
 {
   int  i=1, num, sum =0;
   for(i=0; i < 5; i ++)
    {
     printf(" Enter an integer:");
     scanf( "%d", &num);
     if(num < 0)
    {
       printf("\nyou have entered a negative number");
       continue ;     /* skip the remaining part of loop */
  }
   sum += num;
 }
printf("The sum of the Positive Integers Entered = % d \ n", sum);
}
```

Output :

```
Enter an integer: 7
Enter an integer: 3
Enter an integer: 10
Enter an integer: 15
Enter an integer: 30
The sum of the positive integers entered = 65
```

<u>(3) The "goto" Statement :</u>

C supports the "goto" statement to branch unconditionally from one point to another in the program.

Although it may not be essential to use the "goto" statement in a highly structured language like "C", there may be occasions when the use of goto is necessary.

The goto requires a label in order to identify the place where the branch is to be made.

A label is any valid variable name and must be followed by a colon (:).

The label is placed immediately before the statement where the control is to be transferred.

The label can be anywhere in the program either before or after the goto label statement.

Syntax :

```
goto label;
............
............
............
label:
statement;
```

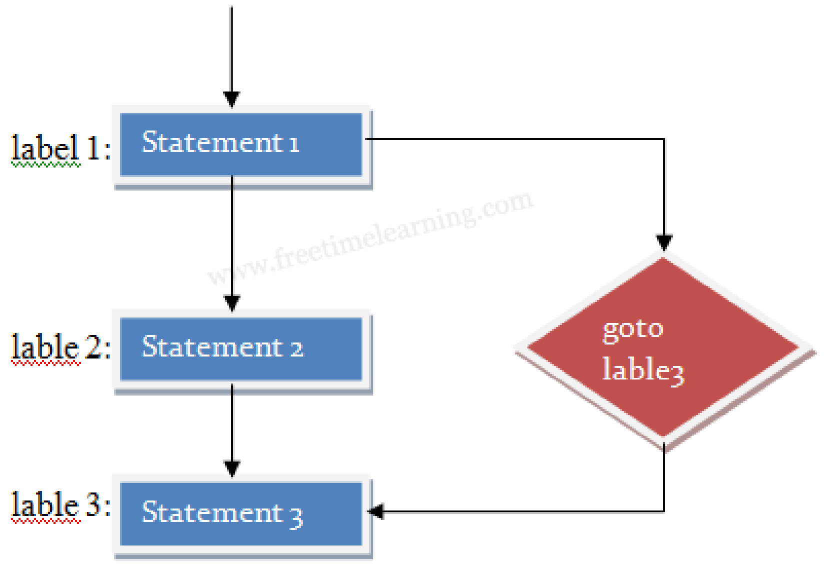
In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.

```
goto label;
............
............
............
label:
statement;
```

Forward Jump

```
label:
statement;
................
................
goto label;
```

backword jump



During running of a program, when a statement likes "goto begin;" is met, the flow of control will jump to the statement immediately following the label "begin:" this happens unconditionally.

"goto" breaks the normal sequential execution of the program.

If the "label:" is before the statement "goto label;" a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a "backward jump".

If the "label:" is placed after the "goto label;" some statements will be skipped and the jump is known as a "forward jump".

* Program : Write a program to detect the entered number as to whether it is even or odd. Use goto statement.

```
# include<stdio.h>

void main( )

 {

   int  i=1, num, sum =0;

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

    {

     printf(" Enter an integer:");

     scanf( "%d", &num);
     if(num < 0)
    {
       printf("\nyou have entered a negative number");
       continue ;     /* skip the remaining part of loop */
  }
   sum += num;
 }
printf("The sum of the Positive Integers Entered = % d \ n", sum);
}
```

Output :

```
Enter a Number: 5
5 is Odd Number.
```