UNIT – 2

Control Structures

BCA SEM – 1

PROBLEM SOLVING METHODOLOGIS AND PROGRAMMING IN C

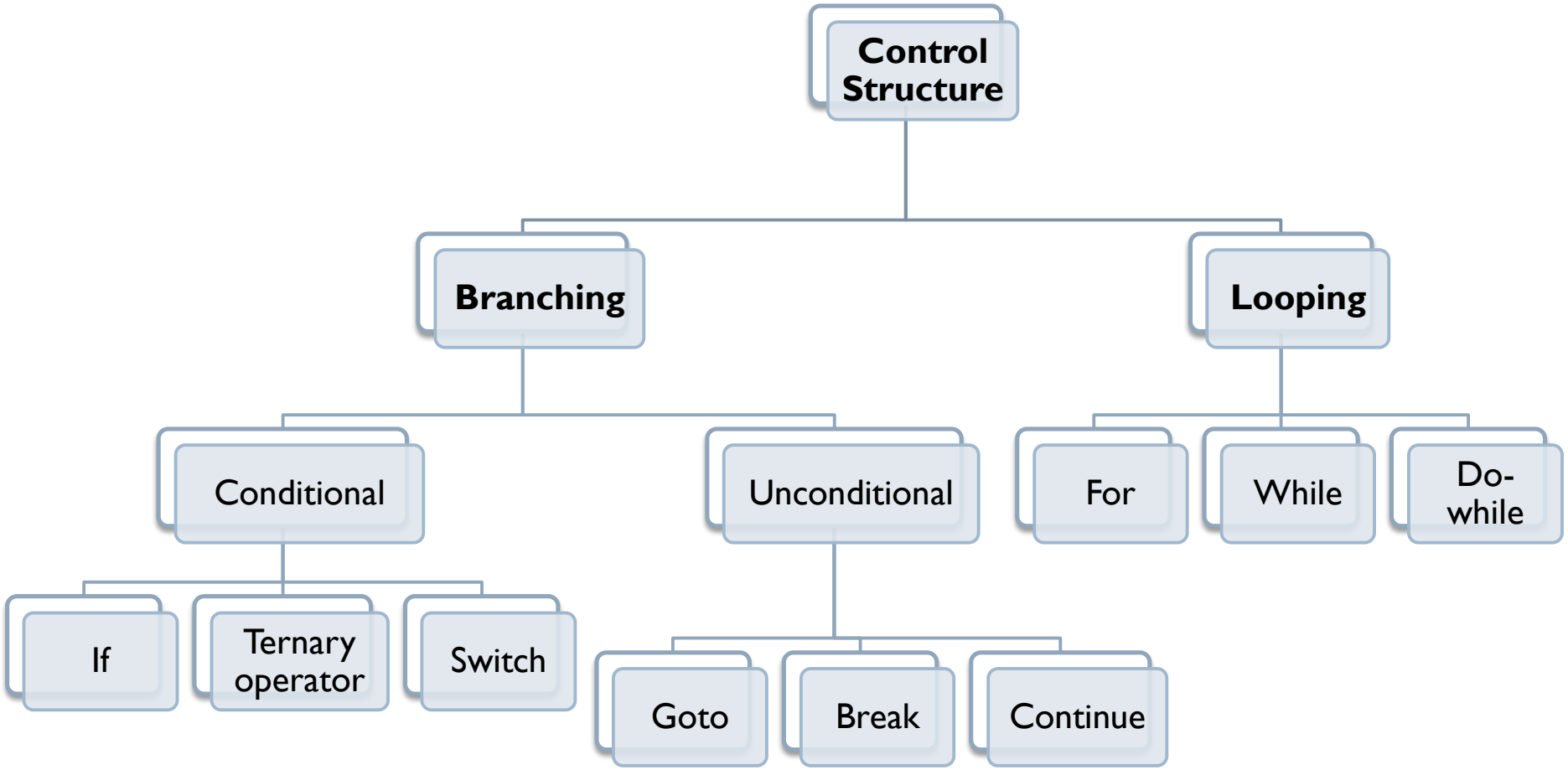
Code : CS-01

Presented By : Dhruvita Savaliya

Topics :

- Selective Control Structure
  - if Statements
  - switch Statements
- Conditional Ternary Operator
- Iterative (Looping) Control Statements
  - for Loop
  - do….while Loop
  - while Loop
- Nesting of Loop
- Jumping Statements
  - break
  - continue
  - goto statements

❖ Selective Control Structure :



❖ Branching Statements :

1. Conditional Statement
2. Unconditional Statement

- The **conditional statements** (also known as decision control structures) such as if, if else, switch, etc. are used for decision-making purposes in C programs.
- They are also known as Decision-Making Statements and are used to evaluate one or more conditions and make the decision whether to execute a set of statements or not.
- These decision-making statements in programming languages decide the direction of the flow of program execution.

1. **if statements**
2. **Ternary Operators**
3. **Switch Statements**

1. if Statements :

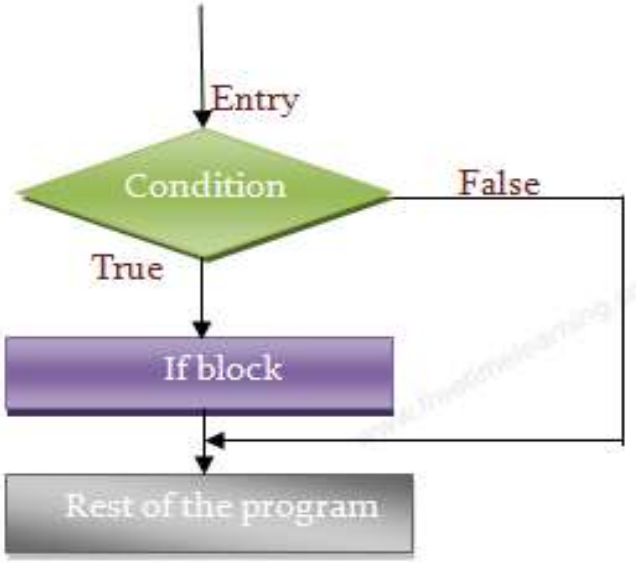
- There come situations in real life when we need to make some decisions and based on these decisions, **we decide what should we do next**.
- Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.
- The C programming language provides a general decision-making capability in the from of language construct known as the **if statements**.
- The if statements are powerful decision-making statement and it used to control the flow.
- The if statement can be used in following different forms :

1. Simple if statement
2. if-else statement
3. Nested if
4. else-if ladder

1. Simple if Statements :

- 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 :**

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



- 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 (**;**)
- The Statements following the "**if**"-statement (body) 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 if Statement Block will be executed and executes rest of the program.
- If the Test Expression / Condition are FALSE, the if Statement Block will be skipped and rest of the program executes next.

Example :

```
# include<stdio.h>
# include<conio.h>
void  main( )
{
      int  m,n;
      clrscr( );
      printf("\n Enter two no:");
      scanf("%d %d", &m, &n);
      if(m>n)
      {
          printf("\nM is Greater");
      }
      Printf("\n End of If statement");
        getch();
}
```

Output :

Run 1st time :

Enter two no: 10 5
M is Greater
End of If statement

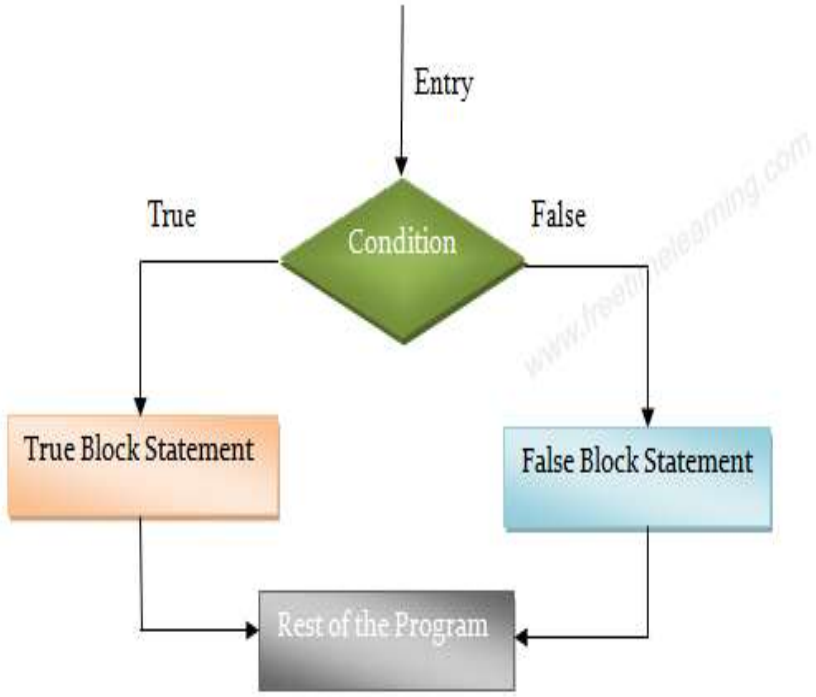
Run 2nd time :

Enter two no: 5 10
End of If statement

2. if-else Statement :

- 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 */
}
//Rest of the program;
```



Example :

```
# 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);
    }
    printf("after condition");
   getch();
}
```

Output :

Enter Two numbers: 13 30

30 is largest number

after condition

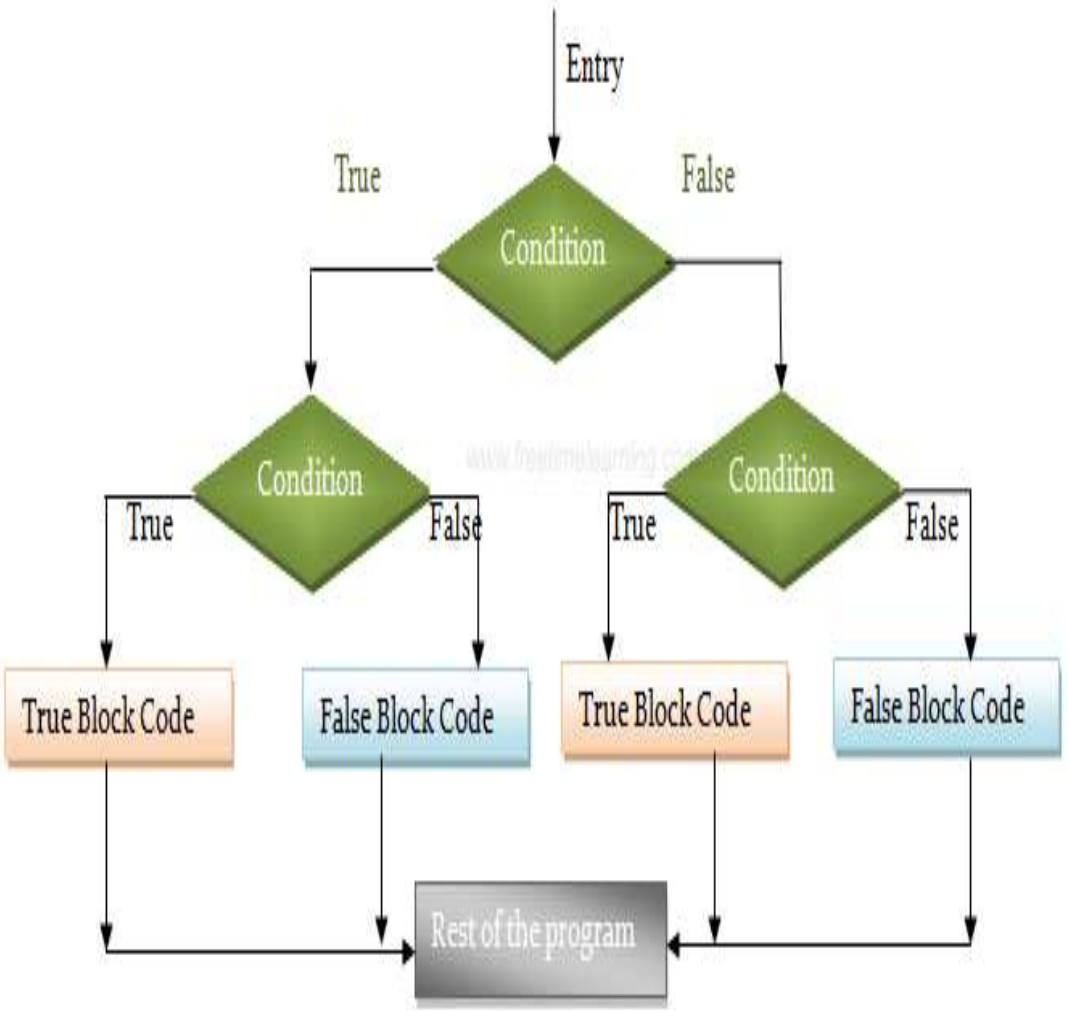
3. Nested if-else Statement :

- 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)
    {
            Statements -1;
    }
    else
    {
            Statements-2;
    }  /*end of inner if-else*/
} /*end of outer if*/
else
{
    if (Test Condition3)
    {
            Statements -3;
    }
    else
    {
            Statements-4;
    } /*end of inner if-else*/

} /* end of outer else */
```



- 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.

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int  a,b,c;
    clrscr();
    printf("Enter Three Values:");
    scanf("%d%d%d", &a, &b, &c);

    printf("\n Largest Value is:") ;
   if(a>b)
   {
          if(a>c)
              printf(" %d ", a);
          else
              printf(" %d ", c);
    }
    else
    {
          if (b>c)
              printf(" %d ", b);
          else
              printf(" %d ", c);
    }
    getch();
}
```

Output :

Run 1:

Enter three values: 10 50 30

Largest Value is: 50

Run 2:

Enter three values: 80 66 45

Largest Value is: 80

4. else-if Ladder :

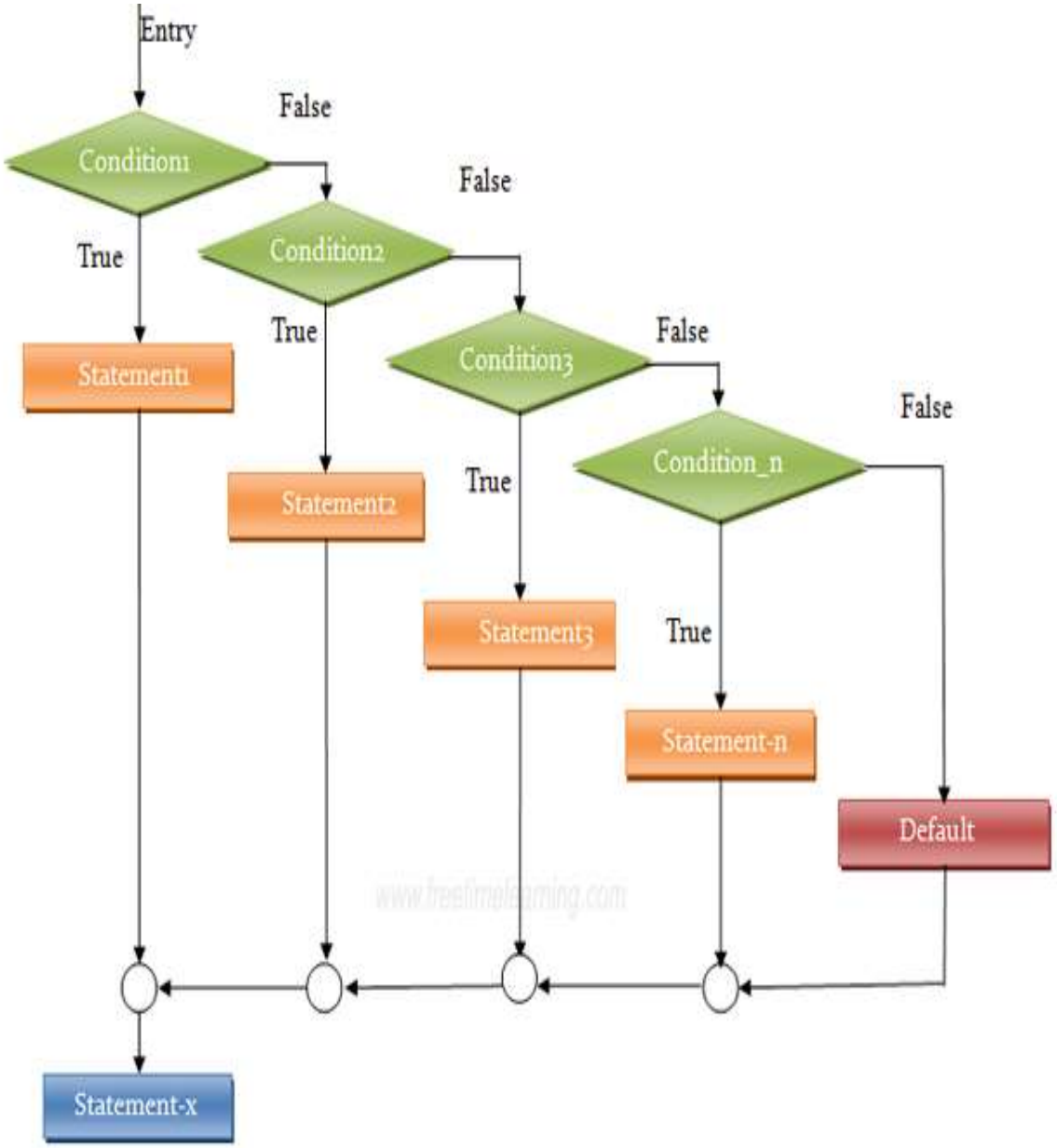
- This is another way of putting if's together when multiple decisions are involved.
- A multipath decision is a chain of if's in which the statement associated with each else is an if.
- Hence it forms a ladder called else–if ladder.
- if else-if, statement is used to execute one code from multiple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

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int  a, b, c;
        clrscr () ;
        printf("Enter Three Values:");
        scanf("%d%d%d", &a, &b, &c);

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

Output :

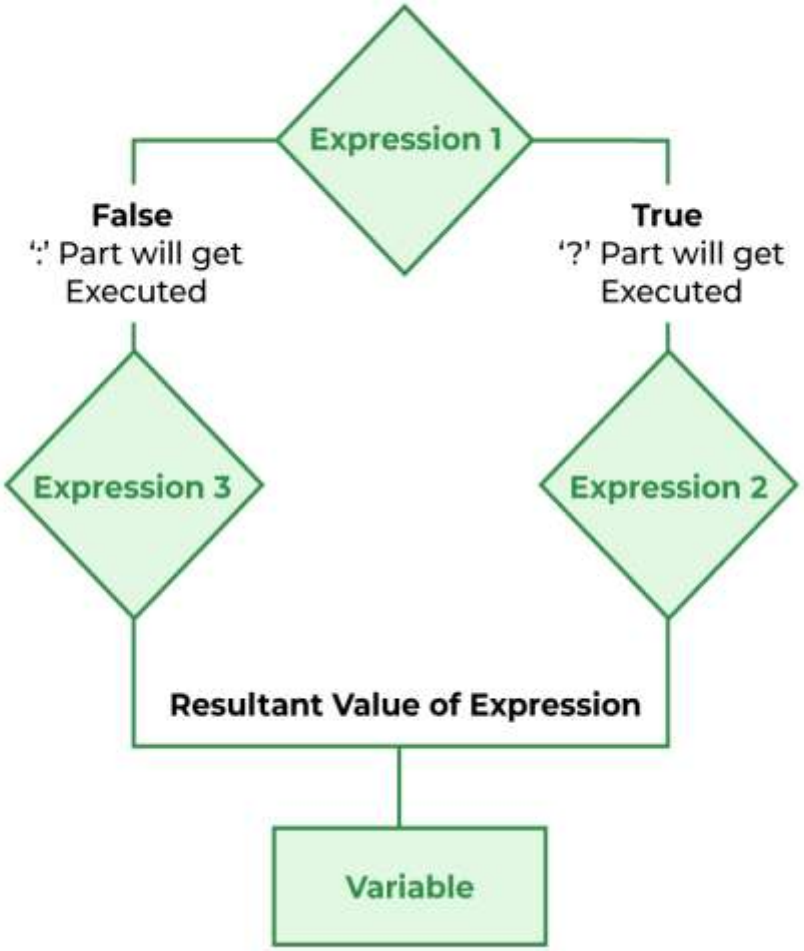
Enter Three Values: 52 90 74

Highest Numbers is: 90

❖ 2. Ternary Operators/Conditional Operator :

- The Conditional Operator is used to add conditional code in our program.
- It is similar to the if-else statement.
- It is also known as the ternary operator as it works on three operands.
- 1st operand as a Condition.
- 2nd operand as True Part.
- 3rd operand as False Part.
- **Syntax of Conditional Operator**

(*condition*) **?** *[true statements]* **:** *[false statements]*;



Example :

```
#include <stdio.h>
#include<conio.h>
void main()
{
   int num;
   printf("Enter an integer: ");
   scanf("%d", &num);
   num % 2 == 0 ? printf("%d is even", num) : printf("%d is odd.", num);
   getch();
}
```

Output :

```
Enter an integer : 44
44 is even
```

❖ 3. Switch Statements / Selection Statements :

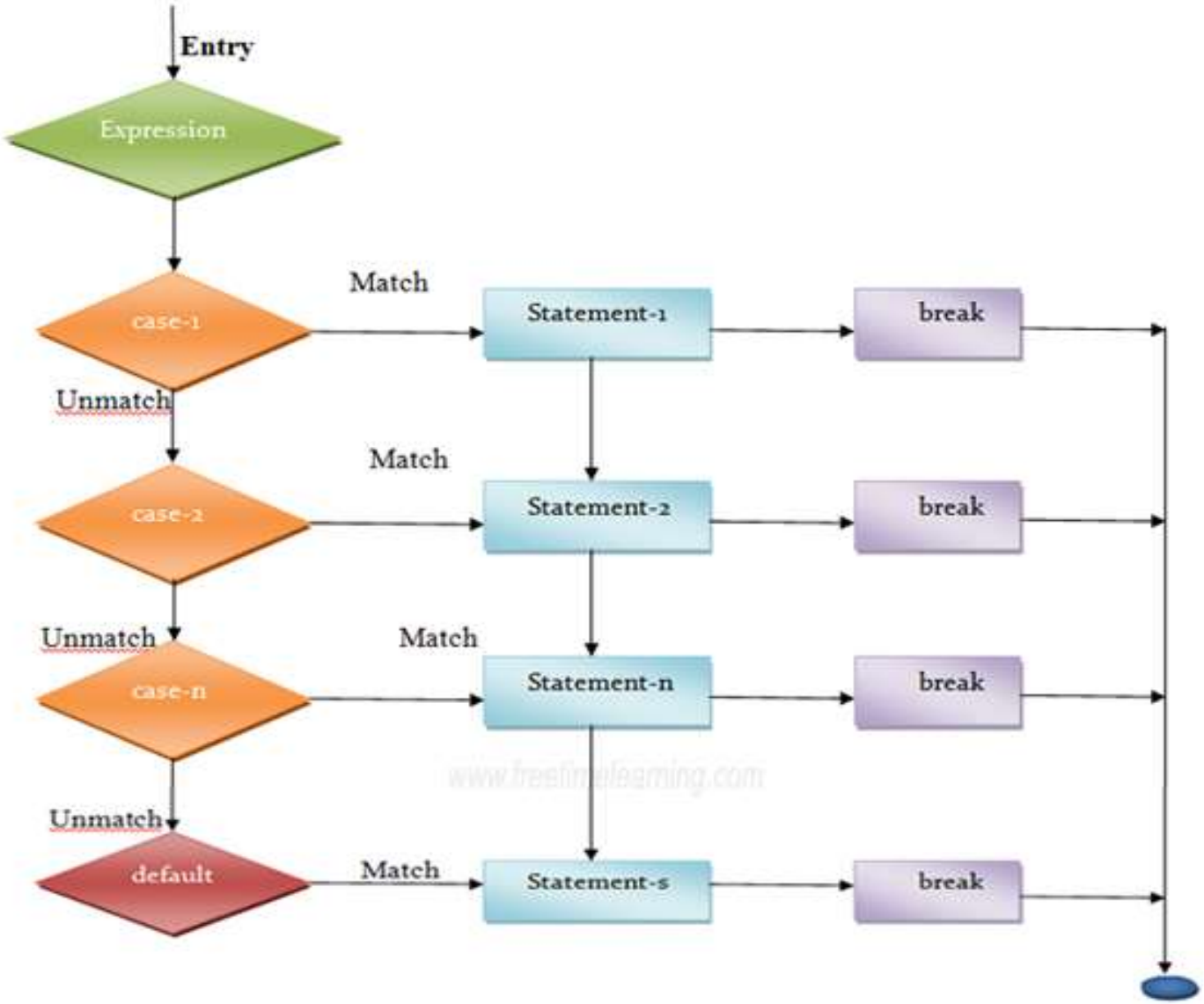
- Switch is another conditional control statement used to select one option from several options based on given expression value; this is an alternative to the if-else-if ladder.
- The switch statement causes a particular group of statements to be chosen from several available groups.
- The selection 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.

Syntax :

```
switch (variable or expression)
{
   case value-1:
          Block -1;
             (Or)
          Statement-1;
             break;

   case value-2 :
          Block -2;
             (Or)
          Statement-2;
             break;

          _ _ _ _ _ _ _ _ _
          _ _ _ _ _ _ _ _ _
   case Constant value-n :
                Block -n;
                   (Or)
                Statement-n;
                break;
   default :
                default – block;
                   (Or)
                Statement-n;
}
//Rest of Program;
```


Entry
Expression
case-1
Match
Statement-1
break
Unmatch
case-2
Match
Statement-2
break
Unmatch
case-n
Match
Statement-n
break
Unmatch
default
Match
Statement-s
break

Example :

```
# include<stdio.h>
# include<conio.h>
void main( )
{
    int day;
   clrscr( );
   printf(" \n Enter the number of day:");
   scanf("%d", &day);
   switch(day)     {
          case 1:
                    printf("hello");
                    break;
          case 2:
                    printf("how are you?");
                              break;
          case 3:
                    printf("hii");
                    break;
          default:
                    printf("Invalid choice");
   }
   getch();
}
```

Output :

Enter the number of day: 3

hii

❖ Loop Control Statements / Iteration Statements :

- **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.

- There are 2 Type of Looping Statements :

1. Entry Control Loop

Entry controlled loop is a loop in which the test condition is checked first, and then the loop body will be executed. If the test condition is false, the loop body will not be executed, For Loop and While Loop is the example of Entry controlled loop.

1. for Loop
2. while Loop

2. Exit Control Loop

An exit-controlled loop is a type of Loop in computer programming that tests the loop condition at the end of the Loop after executing the body of the Loop at least once.

1. do-while Loop

❖ for Loop :

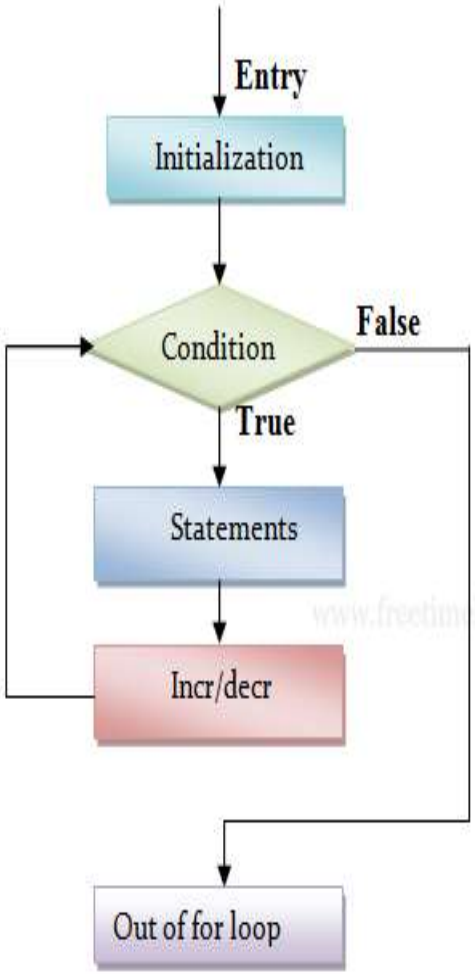
- The for loop works well where the number of iterations of the loop is known before the loop is entered. The head of the loop consists of three parts separated by semicolons.
- The first is run before the loop is entered. This is usually the initialization of the loop variable.
- The second is a test, 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 ; increment/decrement )
{
    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.

Flowchart :



Example :

```
#include<stdio.h>
#include<conio.h>
Void main( )  {
    int  i;
    clrscr( ) ;
    for(i = 0; i <5; i++)
            printf("%d ", i);
    getch( );
}
```

Output :

0 1 2 3 4

- for loop can be specified by different ways as shown

Syntax	Output	Remarks
for (; ;)	Infinite to loop	No arguments
for (a=0; a< =20;)	Infinite loop	“a” is neither increased nor decreased.
for (a=0; a<=10; a++) printf(“%d”, a)	Displays value from 1 to 10	“a” is increased from 0 to 10 curly braces are not necessary default scope of for loop is one statement after loop.
for (a=10; a>=0; a--) printf(“%d”, a);	Displays value from 10 to 0	“a” is decreased from 10 to 0.

❖ Nested for Loop :

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

Example :

```
# include<stdio.h>
# include<conio.h>
void main ( )
{
    int i,j ;
    for(i=1; i<=5; i++)
    {
            for (j=1; j < =i;  j++)
            {
                    printf( "*");
            }
            printf( " \n");
    }
   getch( );
}
```

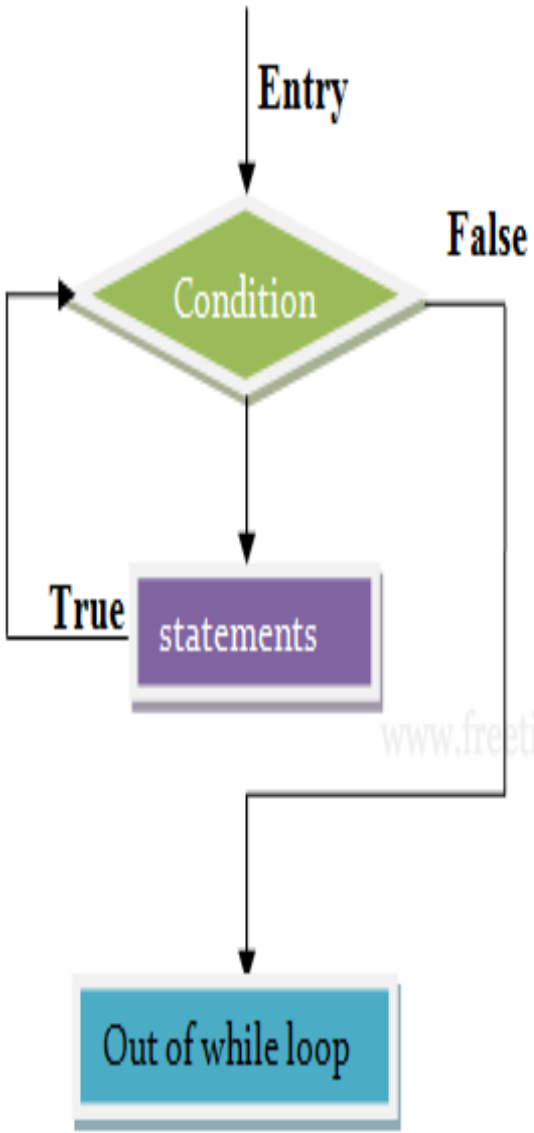
Output :

```
*
* *
* * *
* * * *
* * * * *
```

❖ while Loop

- 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.
- Initialization Expression;
- **Syntax :**

```
while ( Test Condition)
{
        Body of the loop
        Updation Expression (Increment/Decrement)
}
```



Example :

```
# include<stdio.h>
# include<conio.h>
void main( )
{
        int   a=1;
        while(a<=5)
        {
                printf("\n %d",a);
                a++;
        }
    getch( );
}
```

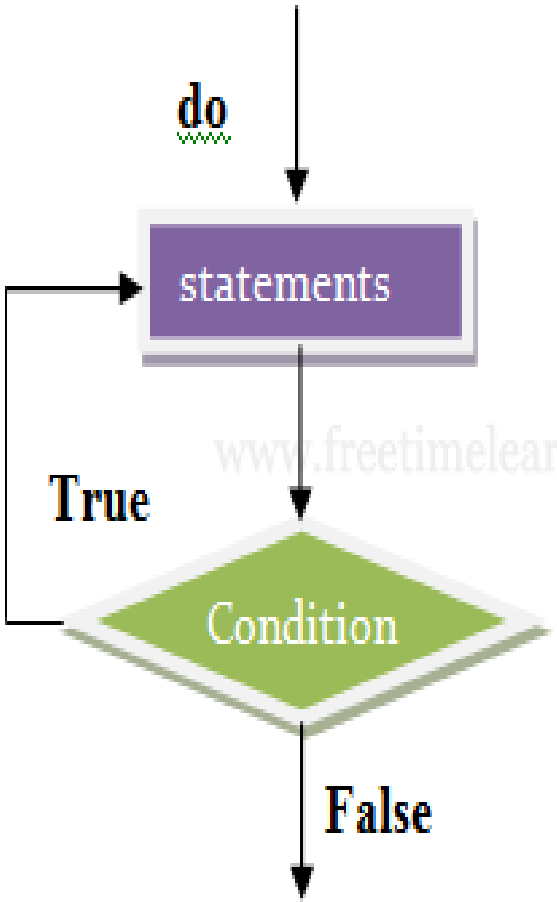
Output : 1
2
3
4
5

❖ do-while Loop

- 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);
Rest of program;
```

Flowchart :



Example :

```
# include<stdio.h>
# include<conio.h>
void main( )
{
        int  i=1;
        clrscr( );
        do
        {
                printf(" \n%d",i);
                i++;
        } while (i <= 5);
        getch( ) ;
}
```

Output :

```
1
2
3
4
5
```

❖ Unconditional control :

- These statements are used in C or C++ for the unconditional flow of control throughout the functions in a program. They support four types of jump statements:

1. Break Statement
2. Continue Statement
3. Goto Statement
4. Return Statement

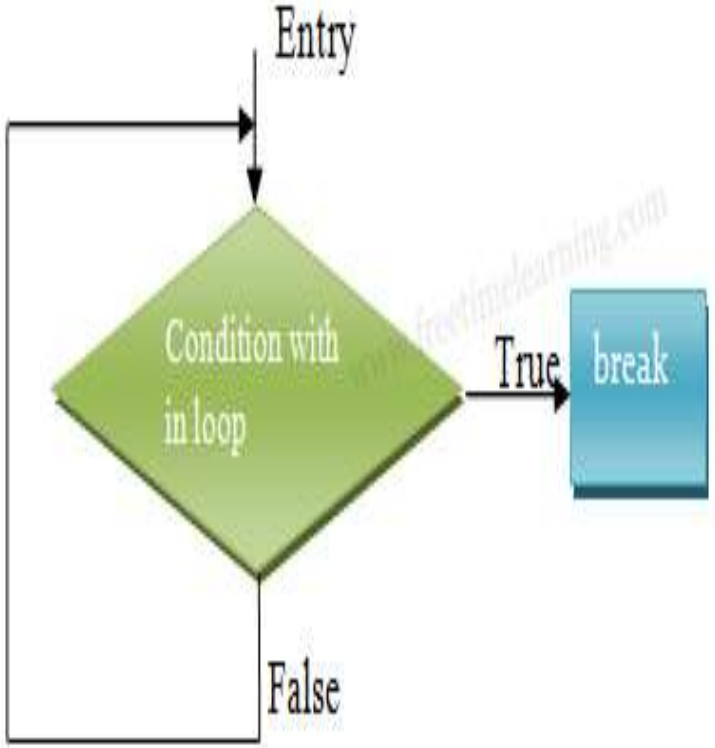
1. Break Statement :

- 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 :**

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

Example :

```
#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 :

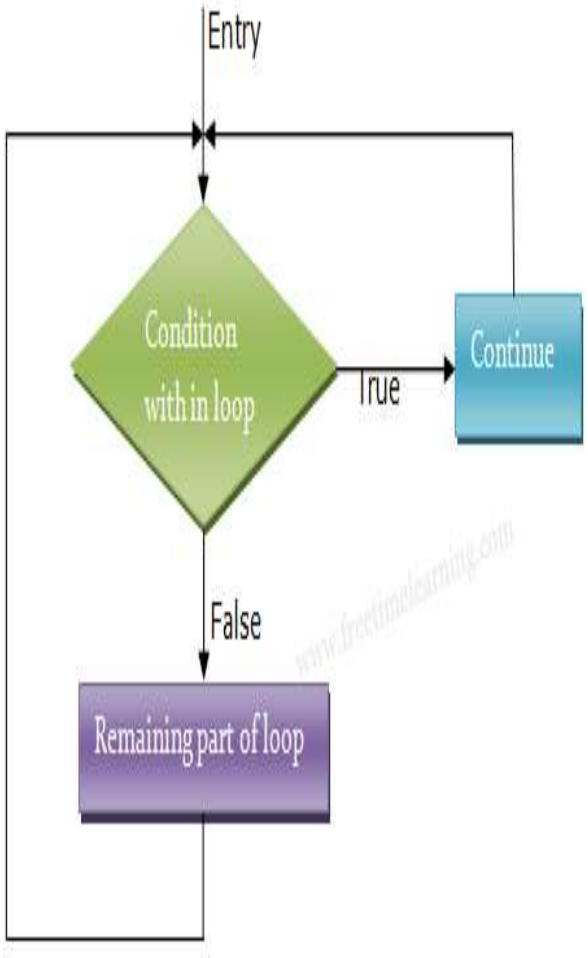
1 2 3 4 5

2. Continue Statement :

- 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 :**

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

Example :




```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i=1;//initializing a local variable
    clrscr();
    for(i=1;i<=10;i++)
    {
            if(i==5)  /* skip the remaining part of loop */
                    continue;
            else
                    printf("%d ",i);
    }//end of for loop
    getch();
}
```

Output :
1 2 3 4 6 7 8 9 10

3. goto Statement :

- 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;

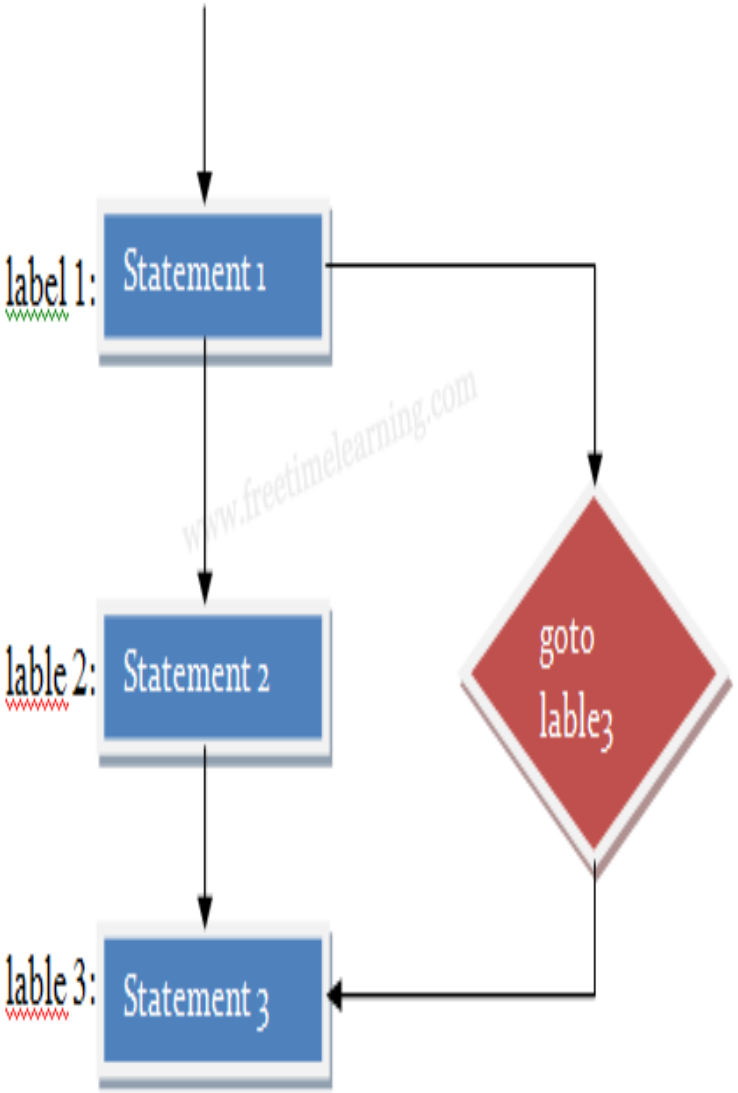
- In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.

- goto label;
  .....
  .....
  .....
  label:
  statement;

  Forward Jump

label:
Statement;
.....
.....
.....
goto label;

Backward



- 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".

Example :

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

void main()
{
   int i;
   int number;
   clrscr()

   for(i=1;i<=5;i++)
   {
        printf("Enter Number %d = ",i);
        scanf("%d", &number);
          // go to jump if the user enters
          //a negative number
          if(number < 0) {
               goto jump;
          }
   }
   Jump:
      printf("You Entered Negative
   Number");
      getch();
}
```

Output :

```
Enter Number 1 = 6
Enter Number 2 = 90
Enter Number 3 = -10
You Entered Negative Number
```