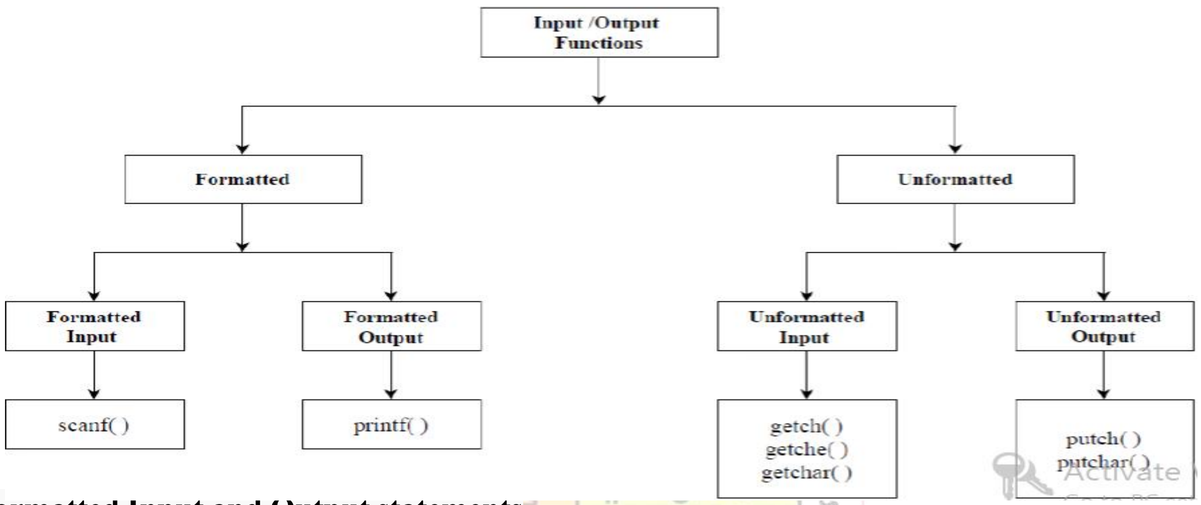
MODULE-2
MANAGING INPUT AND OUTPUT, BRANCHING AND LOOPING

In programming input mean reading data from the input device or a file and Output means displaying the results on the screen. C provides a number of input and output functions. These functions are predefined in the respective header files. The input and output functions are used in the program whose functionality are predefined in the header file **"#include<stdio.h>"**

Input and output functions are broadly classified into as



Formatted Input and Output statements

scanf( ): scanf() function reads all type of data value from input device or from a file. the address operator "&" is used to indicate the memory location of the variable. This memory location is used to store the data which is read through the keyboard.

Syntax:

scanf("format specifier",addresslist);

where:format specifier indicates the type of data to be stored in the variable.

address list indicates the location of the variable where the value of the data is to be stored. the address list is usually prefixed with an "&"(ampersand) operator for each variable.

Example: if we want to store the values 50 and 31from the keyboard in variables num1 and num2 then the input function is read as

```
scanf("%d%d",&num1,&num2);
```

the value 50 will be assigned to num1 and value 31 will be assigned to num2

printf( ): In C programming language, printf() function is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen. The features of printf() can be effectively exploited to control the alignment and spacing of printouts on terminals.

Syntax:

printf("Text Message");
OR
printf("format specifier",variablelist);
where:
format specifier indicates the type of data to be displayed
variable list indicates the value present in the variable.
the number of format specifier must match the number of variables in the variablelist.

Example: if we want to display the values stored in variables num1 and num2 then the printf statement can be written as

```
printf("The Value of num1 = %d and The value of num2 = %d\n",num1,num2);
```

This statement will display the values stored in the respective variables. The output will be of the form:

```
The Value of num1 = 50 and The value of num2 = 31
```

Example: /* C program to demonstrate Formatted Input and Output Statements */

```
#include<stdio.h>
void main()
{
        int a,b,sum;
        printf("Enter two numbers\n");
        scanf("%d%d",&a,&b);
        sum=a+b;
        printf(" Addition of two  Numbers=%d\n",sum);
}
```

Unformatted Input and Output statements

getch(): is used to read a character from the keyboard, the character entered is not displayed or echoed on the screen the functions don't need a return key pressed to terminate the reading of a character. A character entered will itself terminates reading

Example:

```
#include<stdio.h>
void main()
{
        char ch;
        printf("Enter a character\n");
        ch=getch();
        printf("The entered character is %c\n",ch);
}
```

getche(): is used to read a character from the keyboard, the character entered is echoed or displayed on the screen. the functions don't need a return key pressed to terminate the reading of a character. A character entered will itself terminates reading.

Example:

```
#include<stdio.h>
void main()
{
        char ch;
        printf("Enter a character\n");
        ch=getche();
        printf("The entered character is %c\n",ch);
}
```

getchar(): will reads a character from the keyboard and copy it into memory area which is identified by the variable ch. No arguments are required for this macro. Once the character is entered from the keyboard, the user has to press Enter key.

Example:

```
#include<stdio.h>
void main()
{
        char ch;
        printf("Enter a character\n");
        ch=getchar();
        printf("The entered character is %c\n",ch);
}
```

putch() and putchar(): This function outputs a character stored in the memory, on the standard output device.. The variable should be passed as parameter to the functions

Example:

```
#include<stdio.h>
void main()
{
        char ch;
        printf("Enter a character\n");
        ch=getchar();
        printf("The entered character is \n");
        putchar(ch);
}
```

CONDITIONAL BRANCHING AND LOOPING

C program is a set of statements which are normally executed sequentially in the order which they appear. If there occours a situation where we have to change the order of execution of the statements we make use of conditional statements.

C provides 5 types of conditional branching statements.

(i) if statement

(ii) if – else statement

(iii) Nested if statement

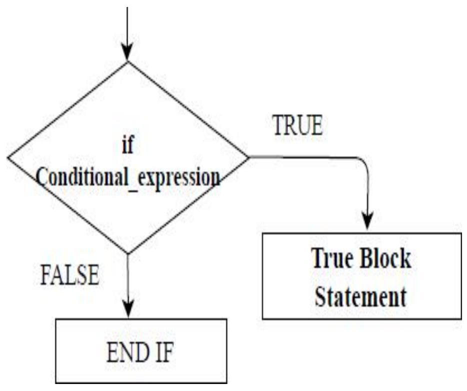
(iv) Switch Statement

(i) if: This is a one way selection statement which helps the programmer to execute orskip certain block of statements based on the particular condition.

Syntax:

```
if(conditional_expression)
{
        True block statements;
}
```

Flowchart:



Example: /* C programto check the voting eligibility of the person*/

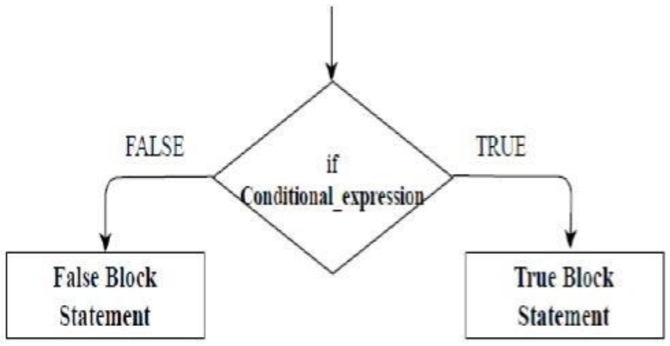
```
#include<stdio.h>
void main()
{
        int age;
        printf(" Enter the age of the person\n");
        scanf("%d",&age);
        if(age>=18)
        {
                printf("The person is eligible to vote\n");
        }
        if(age<18)
        {
                printf("The person is not eligible to vote\n");
        }
}
```

(ii) if-else statement: This is a two way selection statement which executes true block or false block of statements based on the given condition. The keyword "else" is used to shift the control when the condition is evaluated to false.

Syntax:

```
if(conditional_expression)
{
        True block statements;
}
else
{
        False block statements;
}
```

Flow chart:



Example: /* C program to check the entered num is even or odd*/

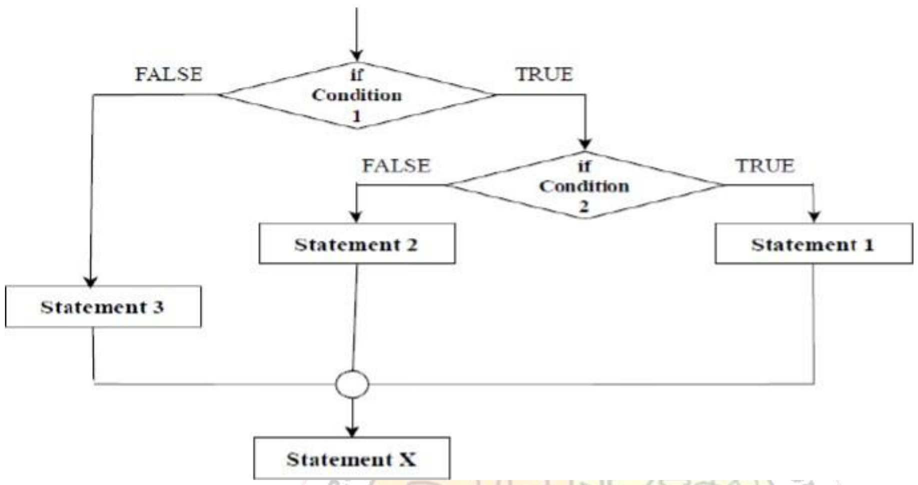
```
#include<stdio.h>
void main()
{
        int num;
        printf("Enter a number\n");
        scanf("%d",&num);
        if(num%2==0)
        {
                printf("%d is a even number\n",num);
        }
        else
        {
                printf("%d is a odd number\n",num);
        }
}
```

(iii) Nested if Statement: An if statement within another if statement is called as a nested if statement. This helps the programmer to select one among many alternatives based on a given condition.

Syntax:

```
if(conditional_expression1)
{
        if(conditional_expression2)
        {
                statement1;
        }
        else
        {
                statement2;
        }
}
else
{
        statement 3;
}
statement X;
```

Flow chart:



Example: /* C program to find smallest of three numbers*/

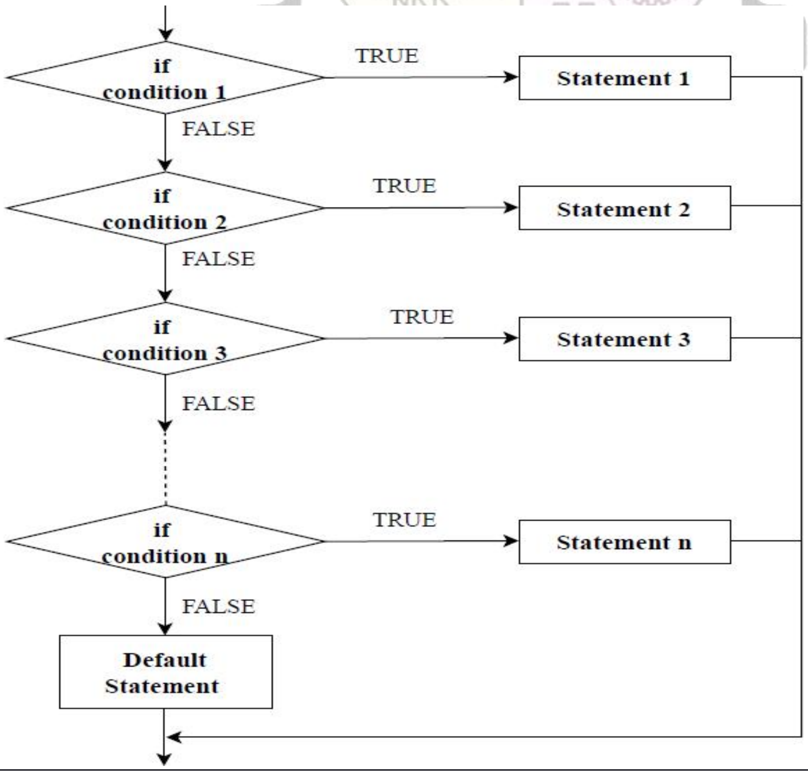
```
#include<stdio.h>
void main()
{
        int a,b,c,small;
        printf("Enter three numbers\n");
        scanf("%d%d%d",&a,&b,&c);
        if(a<b)
        {
                if(a<c)
                        small=a;
                else
                        small=c;
        }
        else
        {
                if(b<c)
                        small=b;
                else
                        small=c;
        }
        printf("Smallest among three numbers=%d",small);
}
```

(iv) Cascaded if-else or else if ladder: This is another way of putting all if's togather when multipath decision are involved The multipath decision is a chain of if statement in which the statement associated with each else is a if statement. Here the conditions are evaluated from top to bottom. As soon as the true condition is found the statement associated with it is executed and the control is transferred to statement X, skipping rest of the ladder.

Syntax:

```
if(condition 1)
        statement 1;
else if(condition 2)
        statement 2;
else if(condition 3)
        statement 3;
______________
______________
______________


else if(condition n)
        statement n;
else
        default statement;
statement X;
```

Flowchart:



Example: /* C program to display the grade of the student based on the average marks obtained */

```
#include<stdio.h>
void main()
{
        float avg;
        printf("Enter the Average marks\n");
        scanf("%f",&avg);
        if(avg>=80)
                printf("Distinction\n");
        else if(avg>=60)
                printf("First Division\n");
        else if(avg>=50)
                printf("Second Division\n");
        else if(avg>=40)
                printf("Third Division\n");
        else
                printf("Fail\n");
}
```

(v) Switch Statement: A switch statement tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed. Each case in a block of a switch has a different name/number which is referred to as an identifier. The value provided by the user is compared with all the cases inside the switch block until the match is found. If a case match is not found, then the default statement is executed, and the control goes out of the switch block. The break statement is used at the end of each case to come out of the switch block.

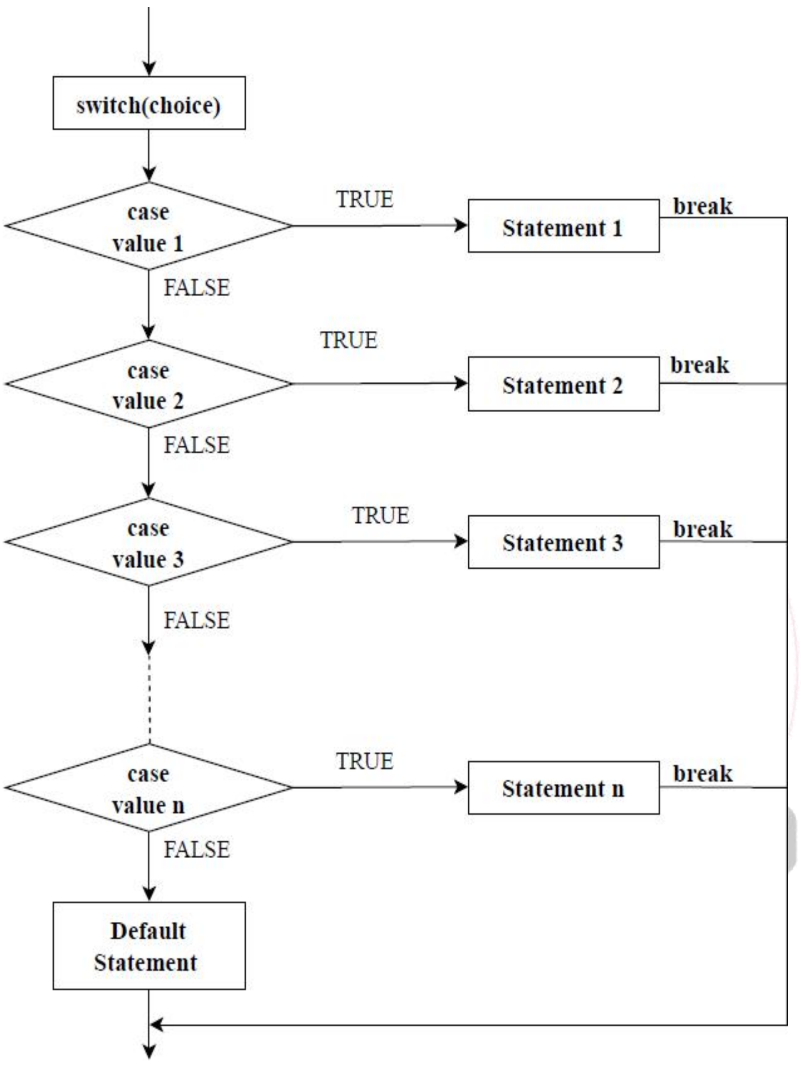
Syntax:

```
switch( expression )
{
        case value-1:  Statement-1;
                       break;
        case value-2:  Statement-2;
                       break;
        case value-3:  Statement-3;
                       break;
        ________________
        ________________
        ________________



        case value-n:  Statement-n;
                       break;
```

```
default:  default Statement
                break;
```

}
Statement-x;

Flowchart:



Example: /* C program to find Area of various geometric figures*/

```
#include<stdio.h>
void main()
{
        float a,b,area;
        int choice;
        printf("\n MENU\n")
        printf("1. Square\t 2. Circle\t 3. Rectangle\t 4. Triangle\n");
        printf("Enter your Choice as 1 OR 2 OR 3 OR 4\n");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1: printf("\nSQUARE\n");
                        printf("Enter the Side\n");
                        scanf("%d",&a);
                        area=a*a;
                        break;
                case 2: printf("\nCIRCLE\n");
                        printf("Enter the Radius\n");
                        scanf("%d",&a);
                        area=3.142*a*a;
                        break;
                case 3: printf("\nRECTANGLE\n");
                        printf("Enter the length and breadth\n");
                        scanf("%d%d",&a,&b);
                        area=a*b;
                        break;
                case 4: printf("\nTRIANGLE\n");
                        printf("Enter the base and height\n");
                        scanf("%d%d",&a,&b);
                        area=0.5*a*b;
                        break;
                default: printf("you have entered a wrong choice\n");
                        exit(0);
        }
        Printf("Area=%f\n",area);
}
```

Introduction to Conditional looping statements.

A set of statements have to be repeatedly executed for a specified number of times until a condition is satisfied. The statements that help us to execute the set of statements repeatedly are called as looping constructs or loop control statements.

The various looping constructs in C are:

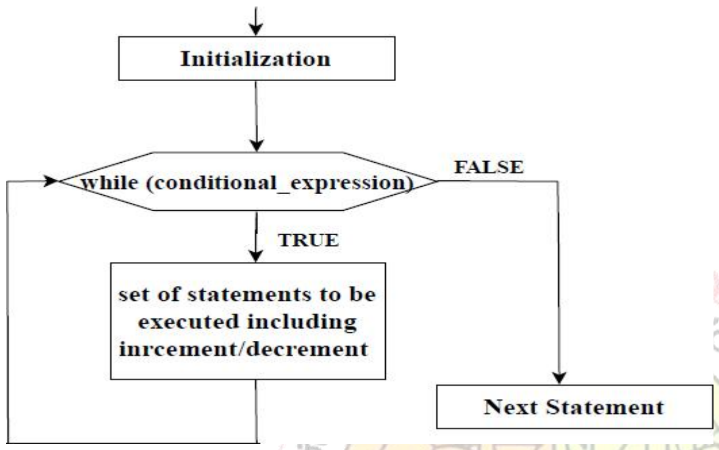
(i) while Loop **(ii)** do-while Loop **(iii)** for Loop

(i) while Loop: It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop. After exiting the loop, the control goes to the statements which are immediately after the loop.

Syntax:

```
initialization;
while(test condition)
{
        set of statements to  be executed
        including increment/decrement opetator
}
```

Flow chart:



Initialization
while (conditional_expression)
FALSE
TRUE
set of statements to be executed including inrcement/decrement
Next Statement

Example: /* C program to print Numbers from 1 to 5 using while loop*/

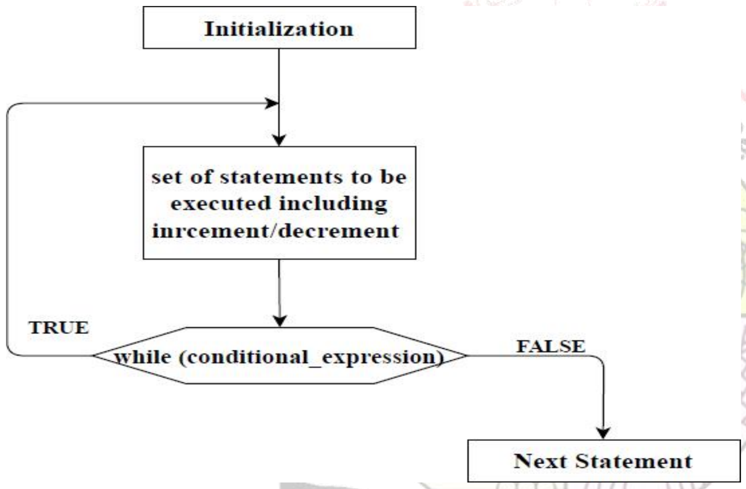
```
#include<stdio.h>
void main()
{
        int i;
        i=1;
        while(i<=5)
        {
                printf("%d\t ",i);
                i++;
        }
}
```

(ii) do-while loop : A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop. The body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop. In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop. Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed.

Syntax:

```
initialization;
do
{
        set of statements to  be executed
        including increment/decrement opetator
}while(test condition);
```

Flowchart:



Initialization
set of statements to be executed including inrcement/decrement
TRUE
while (conditional_expression)
FALSE
Next Statement

Example: /* C program to print Numbers from 1 to 5 using do- while loop*/

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

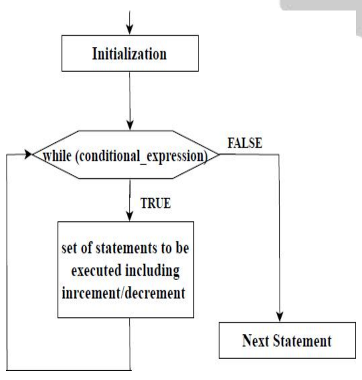
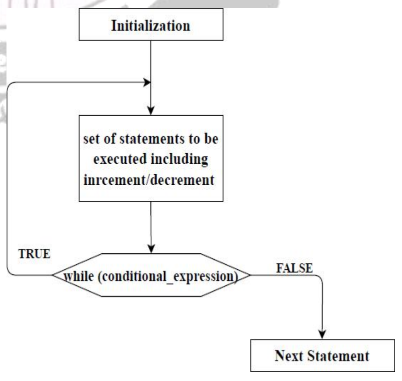
Difference between while loop and do-while loop

While loop	**Do while loop**

Syntax
initialization;
while(test condition)
{
set of statements to be executed including increment/decrement opetator
}
	Syntax:
	initialization;
	do
	{
	set of statements to be executed including increment/decrement opetator
	}while(test condition);

Condition is checked first.	Condition is checked later.
Since condition is checked first, statements may or may not get executed.	Since condition is checked later, the body statements will execute at least once.
The main feature of the while loop is,its an entry controlled loop.	The main feature of the do while loops is it is an exit controlled loop

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

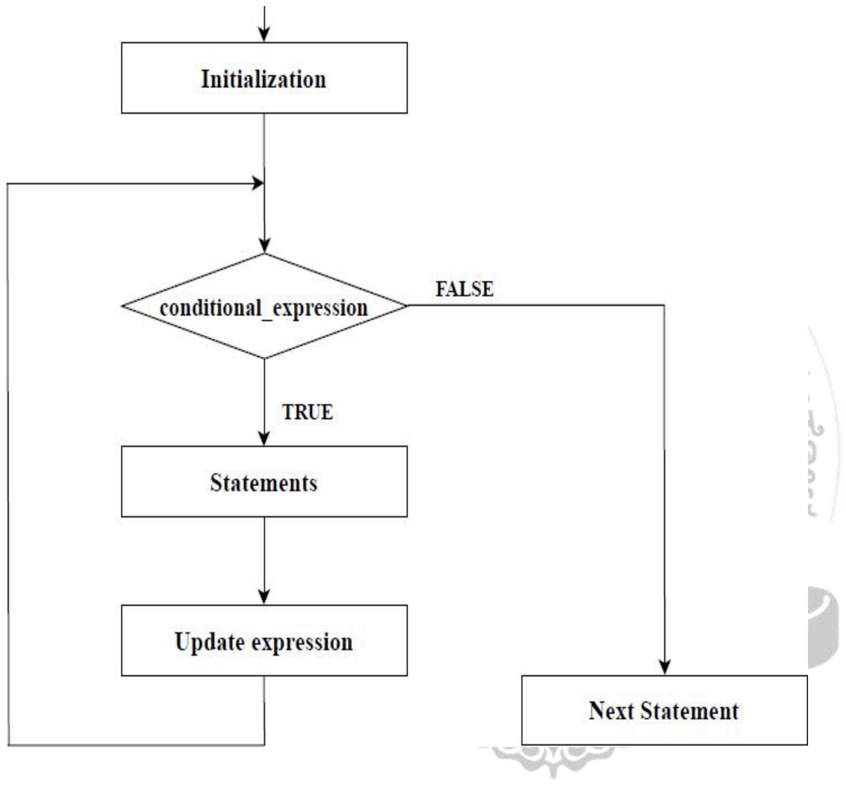
While loop Flowchart  Initialization while (conditional_expression) FALSE TRUE set of statements to be executed including inrcement/decrement Next Statement	Do while loop Flowchart  Initialization set of statements to be executed including inrcement/decrement TRUE while (conditional_expression) FALSE Next Statement

(iii) for loop : A for loop is a more efficient loop structure in 'C' programming which is used when the loop has to be traversed for a fixed number of times. The for loop basically works on three major aspects **(i)** The initial value of the for loop is performed only once. **(ii)** The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned. **(iii)** The incrementation /decrementation increases (or decreases) the counter by a set value.

Syntax:

```
for (initial value; condition; incrementation or decrementation )
{
        statements;
}
```

Flowchart:



Example: **/* C program to print Numbers from 1 to 5 using for loop*/**

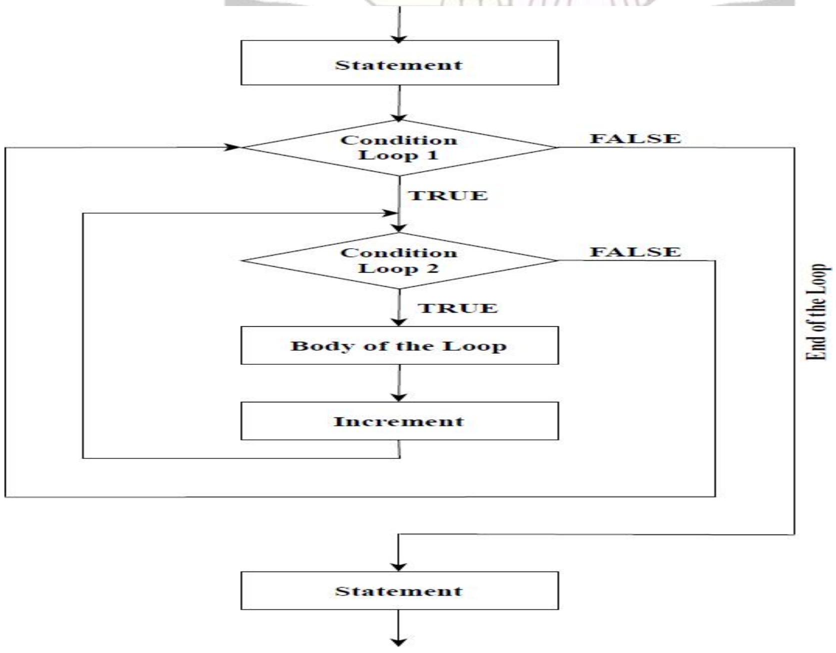
```
#include<stdio.h>
void main()
{
        int i;
        for(i=1;i<=5;i++)
           {
                printf("%d\t",i);
           }
}
```

Nested for loop : Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as **"**loop inside loop**".**In nested for loop one or more statements can be included in the body of the loop. In nested for loop, The number of iterations will be equal to the number of iterations in the outer loop multiplies by the number of iterations in the inner loop. When the control moves from outer loop to inner loop the control remains in the inner loop until the inner loop condition fails, once the condition fails the control continues with the outer loop condition Again when the control comes to inner loop the inner loop is resetto the initial value. The Nested for loop stops execution when the outer for loop condition fails.

Syntax:

```
for ( initialization; condition; increment )
{
        for ( initialization; condition; increment )
        {
                statement of inner loop
        }
        statement of outer loop
}
```

Flowchart:



Example: **C program to print the following pattern**

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

```
#include <stdio.h>
void main()
{
   int i,j;
            for(i=1;i<=4;i++)
            {
               for(j=1;j<=i;j++)
               {
                        printf(“ * ”);
               }
               printf("\n");
            }
}
```

Example: **C program to print the following pattern**

```
1
2    3
4    5    6
7    8    9    10
```

```
#include <stdio.h>
void main()
{
   int i, j, n=1;
            for(i=1;i<=4;i++)
            {
               for(j=1;j<=i;j++)
               {
                        printf("%d\t",n);
                           n++;
               }
               printf("\n");
            }
}
```

Programming examples on Looping constructs:

1. Write a C program to find factorial of a given number using while loop.

```
#include<stdio.h>
void main()
{
        int n,i,fact=1;
        printf("Enter a Number\n");
        scanf("%d",&n);
        i=1;
        while(i<=n)
        {
                fact=fact*i;
                i=i+1;
        }
        printf("Factorial  of a given number = %d\n",fact);
}
```

2. Write a C program to print even numbers in the range of 1 to10 using while loop.

```
#include<stdio.h>
void main()
{
        int i=1;
        while(i<=10)
        {
                if(i%2==0)
                        printf("%d\t",i);
                i=i+1;
        }
}
```

3. Write a C program to print sum of first n natural numbers using do-while loop

```
#include<stdio.h>
void main()
{
        int n,i sum;
        printf("Enter the number of elements\n");
        scanf("%d",&n);
        sum=0;
        do
        {
                sum=sum+i;
                i++;
        }while(i<=n);
        printf("Sum of natural numbers=%d\n",sum);
}
```

4. Write a C program to print multiplication table of a given number using do-while loop

```
#include<stdio.h>
void main()
{
        int n,i,p;
        printf("Enter a number\n");
        scanf("%d",&n);
        i=1;
        do
        {
                p=n*i;
                printf("%d X %d = %d\n",n,i,p);
                i=i+1;
        }while(i<=10);
}
```

5. Write a C program to print sum of first n natural numbers using for loop

```
#include<stdio.h>
void main()
{
        int n,i sum=0;
        printf("Enter the value of n\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                sum=sum+i;
        }
        printf("Sum of  natural numbers=%d\n",sum);
}
```

6. Write a C program to print sum of all odd numbers and even numbers up to a given range n using for loop

```
#include<stdio.h>
void main()
{
        int n,i,osum=0,esum=0;
        printf("Enter the value of n\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                if(i%2==0)
                        esum=esum+i;
                else
                        osum=osum+i;
        }
        printf("The sum of even numbers=%d\n",esum);
        printf("The sum of odd numbers=%d\n",osum);
}
```

7. Write a C program to print fibonacci series up to n numbers using for loop

```
#include<stdio.h>
void main()
{
        int n,i,fib1,fib2,fib3=0;
        printf("Enter the number of series to to be genetared:");
        scanf("%d",&n);
        fib1=0;
        fib2=1;
        if(n==1)
                printf("%d\n",fib1);
        else if(n==2)
                printf("%d\n%d\n",fib1,fib2);
        else
                printf("%d\n%d\n",fib1,fib2);
        for(i=3;i<=n;i++)
        {
                fib3=fib1+fib2;
                printf("%d\n",fib3);
                fib1=fib2;
                fib2=fib3;
        }
}
```

7. Write a C program to print the following pattern

```
1
1    2
1    2    3
1    2    3    4
```

```
#include <stdio.h>
void main()
{
   int i,j;
           for(i=1;i<=4;i++)
           {
                for(j=1;j<=i;j++)
                {
                        printf(" %d ",j);
                }
                printf("\n");
            }
}
```

Unconditional Looping Statements

An unconditional statements are the statements which transfer the control or flow of execution unconditionally to another block of statements. They are also called jump statements.

There are four types of unconditional control transfer statements.

(i) break **(ii)**continue **(iii)** goto **(iv)**return

(i) 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.

Syntax :

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

Example:

```
#include<stdio.h>
void main()
{
        int i=0;
        while(i<=5)
        {
                i++;
                if(i==3)
                        break;
                printf("%d\t",i);
        }
}
```

OUTPUT:

```
1       2
```

(ii) 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. It is simply written as “continue”. The continue statement tells the compiler “Skip the following Statements and continue with the next Iteration”.

Syntax :

```
Jump-statement;
    Continue;
```

Example:

```
#include<stdio.h>
void main()
{
        int i=0;
        while(i<=5)
        {
                i++;
                if(i==3)
                        continue;
                printf(“%d\t”,i);
        }
}
```

OUTPUT:

```
1       2       4       5
```

(iii) 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 :	**Forward jump**	**Backward jump**
goto label;	*goto label;*	*label:*
....	....	*statement;*
....	....	....
....	....	....
label:	*label:*	....
statement;	*statement;*	*goto label;*

If the label statement is below the goto statement then it is called **forward jump**. if the label statement is above the goto statement then it is called **backward jump**

Example:

Program without using goto

```
#include<stdio.h>
void main()
{
printf("MITE \t");
printf("is \t in\t");
printf("Moodbidri\n");
}
```

OUTPUT

```
MITE    is    in    Moodbidri
```

Program using goto

```
#include<stdio.h>
void main()
{
printf("MITE \t");
goto label1;
printf("is \t in\t");
label1: printf("Moodbidri\n");
}
```

OUTPUT

```
MITE      Moodbidri
```

Write a C Program to check if the entered number is positive Negative or Zero using goto statement.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int num;
        printf("Enter the number\n");
        scanf("%d",&num);
        if(num==0)
                goto zero;
        else if(num>0)
                goto pos;
        else
                goto neg;
        zero: printf("The entered number is Zero\n");
                exit(0);
        pos: printf("The entered number is Positive\n");
                exit(0);
        neg: printf("The entered number is Negative\n");
                exit(0);
}
```

return statement: The **return** statement terminates the execution of a function and returns control to the calling function. Execution resumes in the calling function at the point immediately following the call. A **return** statement can also return a value to the calling function.

Syntax :

Jump-statement:
return expression;

Finding Roots of a Quadratic Equation:

A quadratic equation, or a quadratic in short, is an equation in the form of $ax^2 + bx + c = 0$, where a is not equal to zero. The "roots" of the quadratic are the numbers that satisfy the quadratic equation. There are always two roots for any quadratic equation, although sometimes they may coincide.

The possible roots of the quadratic equation are:

(i) Roots are Real and Equal

(ii) Roots are real and distinct

(iii) Roots are imaginary

How to decide on calculation of roots?

Given the equation $ax^2 + bx + c = 0$, substitute the values of the coefficients a,b,c in the discriminant b^2-4ac

Outcome1: if the value of b^2-4ac is equal to zero then we say the "Roots are Real and Equal" The formula to calculate the real and equal root is $�� = \frac{-��}{2��}$

Outcome2: if the value of b^2-4ac is grater than zero i.e if the discriminant value is positive we say the "Roots are real and distinct" the formula to calculate real and distinct roots are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2}$$

Outcome3: if the value of b^2-4ac is lesser than zero i.e if the discriminant value is negative we say that the " Roots are imaginary " the formula to calculate imaginary roots are

$$x = \frac{-b \pm i\sqrt{b^{2}-4ac}}{2}$$

Develop a program to compute the roots of a quadratic equation by accepting the coefficients. Print appropriate messages

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
void main()
{
        float a,b,c,x1,x2,disc;
        printf("Enter the values of a,b,c\n");s
        canf("%f%f%f",&a,&b,&c);
        disc=b*b-4*a*c;
        if(disc>0)
        {
                printf("Roots are Real and Distinct\n");
                x1=((-b)+sqrt(disc))/(2*a);
                x2=((-b)-sqrt(disc))/(2*a);
                printf("Root1= %f\n Root2= %f\n",x1,x2);
        }
        else if(disc==0)
        {
                printf("Roots are Real and Equal\n ");
                x1=(-b)/(2*a);
                printf("Root1=Root2=%f\n",x1);
        }
        else
        {
                printf("Roots are Imaginary\n"); ;

        }
getch();
}
```