Unit-3

Looping

Iteration:- Repetitions

Iteration is nothing but repetition of one or more statements. in c language looping constructs are used to repeat one or more statement again and again until some condition is met.

C language provides following three types of loop

(I) while
(II) do-while
} Non-deterministic loops / Indeterminate loops

(III) for } Deterministic

→ Entry Controlled loops

→ Exit-controlled loops.

Entry Controlled loops:-

In this loops the condition is check before entering the loops.

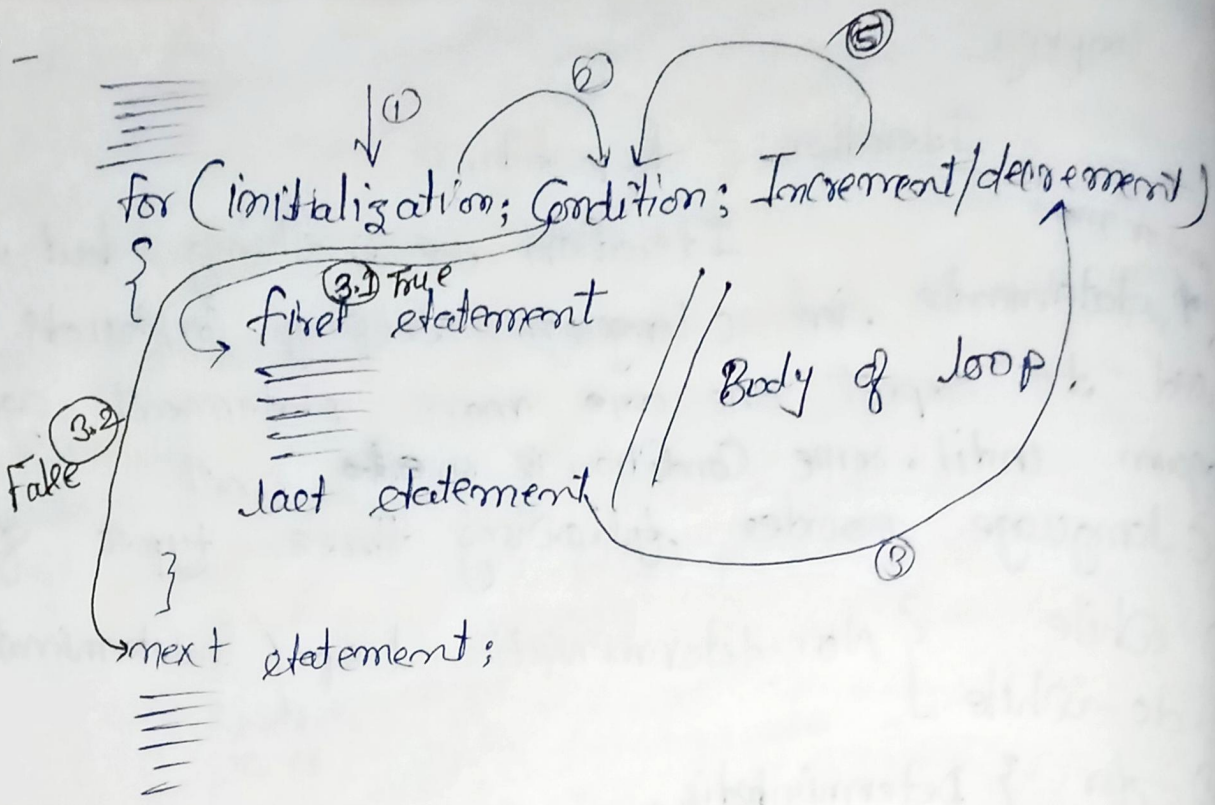
ex. while and for loops.

Exit Controlled loops:-

In exit control loop the loop condition is check at the exit of loop.

ex. do-while.

① For :-



Ex.

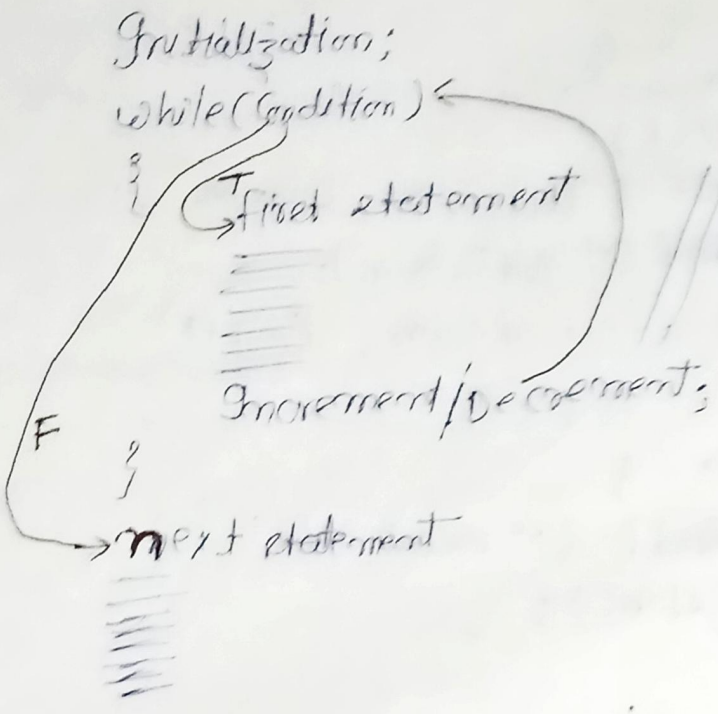
```
Void main
{
    int
    for (j=1; j<=10; j=j+1)
    {
        printf ("Impatlu");
    }
    getch();
}
```

or for (i=1; i<=10; i=i+2)

For decrement

for (i=10; i>=1; i=i-1)

While loop :-

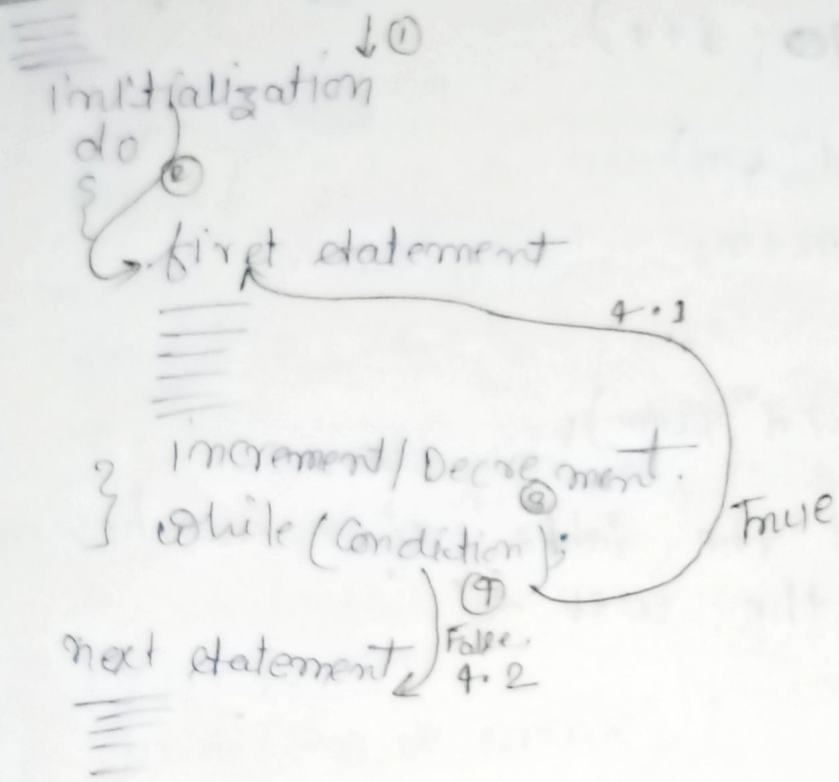


2. /* WAP to print name 5 times using while loop */

```
main()
{  int i;  // Initialization
   i=0;
   while (i<=4) //Condition
   {  printf("Rajnish Tripathi \n");// Body of the loop
      i=i+1; // Increment/decrement
   }
   getch();
}
```

4. Do while:-

Do-while loop



Ex.

```
Void main()
{  int i;
   j=1;
   do
   {   printf("Rajnish \n");
       j= j+1
   } while (i<=5);
   Correct.
   getch();
}
```

while and for - Entre control
do while - exit Control

/* Program to check whether an input no. n is prime or not */

```
main()
{
    int j,n;
    printf("Enter a number");
    scanf("%d", &n);
    for(j=2; j<=(n-1); j++)
    {
        if(n%j==0)
            printf("n is not prime");
            break;
    }
    if(j==n)
    printf("n is prime");
    getch();
}
```

or

```
{ if(n%j==0)
    break;
}
if(j==n)
printf("n is prime");
else
printf("n is not prime");
```

/* Program to sum the digite of an input number */

```
Void main()
{
    int n,r,sum=0;
    clrscr();
    printf("Enter a number");
    scanf("%d", &n);
// logic to calculate sum
while(n!=0)
{
    r=n%10;
    sum=sum+r;
    n=n/10;
}
printf("sum of digit=%d", sum);
```

n = 123

cycle 1: $r_1 = n\%10$ // r = 3; $n = n/10$ // n = 12

2: $r_2 = n\%10$ // $r_2 = 2$; $n = n/10$ // n = 1

3: $r_3 = n\%10$ // $r_3 = 1$; $n = n/10$ // n = 0

Nested loop :-

Whenever a looping construct is written inside the body of another looping construct then this is called nesting of loop. Here, the loop which is written inside is called the inner loop and the loop in which it is written is called outer loop. In other words we can say that inner loop is Nested inside outer loop.

ex.

```
for ( j=1 ; j<=2 ; j++)
{
    for(j=1 ; j<=3 ; j++)
    {
    }
}
```

outer loop

Inner loop

```
int j,j;
for (j=1 ; j<=2 ; j++)
{
    printf("outer loop\n");
    for (j=1 ; j<=3 ; j++)
    {
        printf("Inner loop\n");
    }
    printf ("Aagain outer loop\n");
}
```

T

F

F

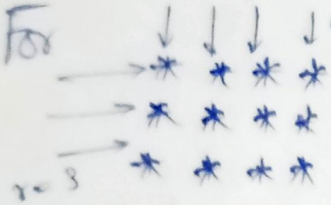
output

```
outpul loop
Inner loop
Inner loop
Inner loop
Again outer loop
outer loop
Inner loop
Inner loop
Inner loop
Again nested loop
```

```
int i=8, u=8, j, j;
for ( j=1; j<=u; j++)
{
    for (i=1; i<=10; i++)
    { printf (" %d * %d = %d", j * i, j * i);
    }
}
```

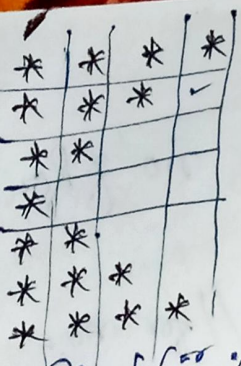
Pattern Printing :-

For r=3, c=4



```
    for (i=1, i<= 3; i++)
    {
        for (j=1; j<=4; j++)
        { printf("*");
        }
        printf ("\n");
    }
getch();
```

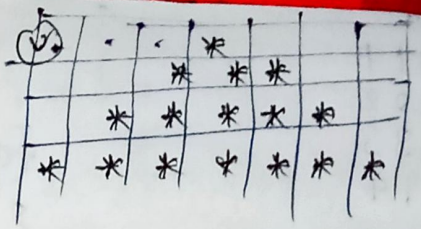
(iii)



```
Scanf("%d", &r);  no prob.
for (i=4; i>=1; i--)// for general i=r ;
{
    for (j=1; j<=i; j++)
    { printf("*");
    }
    printf("\n");
}
for (i=2; i<=4; i++)// i<=r
{
    for (j=1; j=i; j++)
    { printf("*");
    }
    printf("\n");
}
getch();
```

(iv)

(v)



```
for (i=1; i<=4; i++)
{
    for (j=1; j<=4-i; j++)
        printf(" ");
    for(j=1; j<=2i-1; j++)
    { printf("*");
    }
    printf("\n");
}
```

Series:-

① $1+2+3+4+\cdots\cdots+n$

```
void main()
{
    int i, sum=0, n;
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
        sum = sum+i;
    }
    printf("sum of the series = %d", sum);
    getch();
}
```

② $1^2+2^2+3^2+4^2+\cdots+n^2$

```
for(i=1; i<=n; i++)
{ sum = sum+i*i;
}
```

③ $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots\frac{1}{n}$

```
float i, sum=0, n;
for(i=1; i<=n; i++)
{
    sum = sum+(1/i);
}
```

—: Functions :—

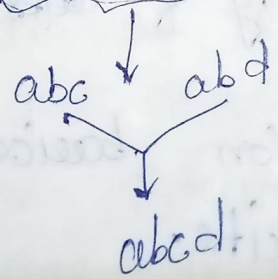
Function is a group of statement or instructions that are written with a purpose to perform a certain task whenever it is required. There are two types of functions use in C-programming —

① Built-in functions
② User defined functions

① Built-in functions :—

Built-in functions in C are provided with C-package and the definition of these functions are already define and stored in C-library. The declaration of these building functions are given in header (.h) files.

ex. printf( ) — To print something on standard I/O.
scanf( ) — To read something from standard I/O.
pow( ) — To calculate the power of a number.
strcat( ) — To Concatenate two strings



etc.

User defined functions:-

These functions are defined by the programmer for its specific purpose to be used in his program.

Advantages of using functions:-

(1) Support of modularity

or Modular programming :-

it is a programming Paradigm that breaks down the bigger problem into smaller subproblems. these subproblems are solved (programmatically) separately and their solutions are combined to solve the bigger problem

2. ~~Easier~~ and Understable

(2) Easier to write and understand the program:-

(3) Fault detection and isolation is easy.

(4) Debugging and maintenance is easier

(5) Facilitates Reusability of Codes.

A user defined function basically involve falling three aspicite.

① Function declaration / Prototyping :-

A function declaration tells a compiler following three things -

ⓐ Name of function :- It is an identifier (user defined name) given to the function.

ⓑ Return Type :- It is specify the data type of the value returned by the function.

ⓒ Arguments / Parameter list :- It is specifies the Number of arguments along with their datatype

return type function name (Argument-list);

↓ syntax :- int sum (int x, int y);

2 Argument

or int sum (int, int)

For ex. Consider a user defined function Name 'sum' which takes as input two Arguments of integer types and calculates the sum input values and then Returned. then the declarations of these functions is as following :-

② Function Definition:-

A function definition is used to specify the sequence of statements that are to be written in the body of the function

Syntax:-

```
return_type function_name (Argument_list)
{
    // Body of the function
} return Value;
```

Ex.
```
int sum (int x, int y)
{
    int z;
    z = x+y;
    return z;
}
```

Formal Parameters.

Value Compulsory.

Note:- in declaration name of Argument is not Compulsory but in Definition it is compulsory.

③ Function Calling :- Whenever there is requirement of performing the task written in the function, the function is called during the execution of the program. The syntax of calling any function is as following.

syntax :- func_name (Actual_Argument_List);

For ex. to call the function name 'sum' which takes two input integer values. —

sum(10,5) two input integer values 10 and 5.

Actual Arguments.

```
/* Function Definition of 'sum' */
int sum(int x, int y)
{
    int z;
    z = x+y;
    return z;
}
```

/* Program using function named 'sum' that adds two input integers and returns their sum */

```
#include<stdio.h>
#include<conio.h>
int sum(int, int); // Global declaration of sum
void main()
{
    int sum(int, int); // function prototyping & it should be a local declaration
    int a, b, c;
    printf("Enter two values to be added");
    scanf("%d %d", &a, &b);
    c = sum(a,b); // Function Calling
    printf("sum = %d", c);
    getch();
} // End of main function

/* Function Definition of 'sum' */
int sum(int x, int y);
{
    int z;
    z = x+y;
    return z;
}
```

formal Argument

/* Program using a function that compares two input values */

```
Void main()
{   void Compare(int, int); // Function declaration
    int a, b;
    printf("Enter two values to be added");
    scanf("%d %d", &a, &b);
    Compare(a, b);
    getch();
} // End of main function

/* Function Definition 'Compare' */

Void Compare(int x, int y)
{
    int i;
    if(x<y)
        printf("Val 1 is smaller");
    if(x==y)
        printf("Val 1 and Val 2 are equa");
    if(x>y)
        printf("Val 1 is greater");
}
```

Void

int i; is not used here because there is no value to return to program so we use void because void is not returned any value

Just return & control

* With Argument / Parameter - with Return.
* Without Argument - with Return
* With Argument - without Return
* Without Argument - without Return

*By without Argument - with Return

```
Void main()
{
    int sum(void);
    int sel;
    sel = sum();
    printf("sum = %d", sel);
}getch();

/* sum function */
int sum(void)
{
    int a,b,c;
    printf("Enter two values");
    scanf("%d %d", &a, &b);
    c = a+b;
}return c;
```

without Argument

with Return.

Break:- A break statement is used to exit from enclosing loop (or switch) block. During the execution of a program, whenever a break statement is encountered inside a loop (or switch) block, it takes the control of execution out from the loop (or switch) block and jumps to the imidiate next statement after the loop (or switch block).

ex.

```
void main
{   int j;
    for ( j=1; j<=10; j++)
    {   if (j==6)
        { break;
        }
        printf ("%d \n", j);
    }
    getch();
}
```

out =

1
2
3
4
5

Continue: during the execution of program, whenever Continue statement is encountered it causes to skip the execution of subsequent statement in the loop and begins the next iteration (cycle) of the loop.

Ex.

```
Void main()
{
    int j;
    for (j=1; j<=10; j++)
    {
        if (j==6)
        { Continue; }
        printf ("%d\n", j);
    }
    getch();
}
```

When i=6 statement true so skip printf & begin next iteration.

output:

1
2
3
4
5
7
8
9
10

Call by value ~~-~~ Completed ✓ still today

Call by reference :- remain

without Argument with Return

#

#

```
Void main ( )
{
        int sum(void);
        int c;
        c = sum( );
        printf("sum of value= %d", c); } output
        getch( );
}
/* Function Definition */

int sum(void)
{ int a, b, s;
Input { Printf ("Enter two values");
        Scanf ("%d %d", &a, &b);
logic-Processing - s = a+b;
        return s;
}
```

(18) with Argument without return

```
void main( )
{
    void sum(int, int); // Function Declaration
    int a, b;
    printf (" Enter two values ");
    scanf ("%d %d", &a, &b);
    sum(a,b); // Function Calling
    getch();
}
/* Function definition 'sum' */
void sum (int x, int y)
{ int s;
  s = x+y;
  printf (" sum = %d ", s);
}
```

(19) without Argument - without Return

```
void main()
{
    void sum(void);
    int c;
    sum( ); // Function Calling
    getch();
}
/* Function definition 'sum' */
```

Recursion :-

In programming language whenever any function calls itself then such calls are called Recursive calls and this phenomina is called Recursion.

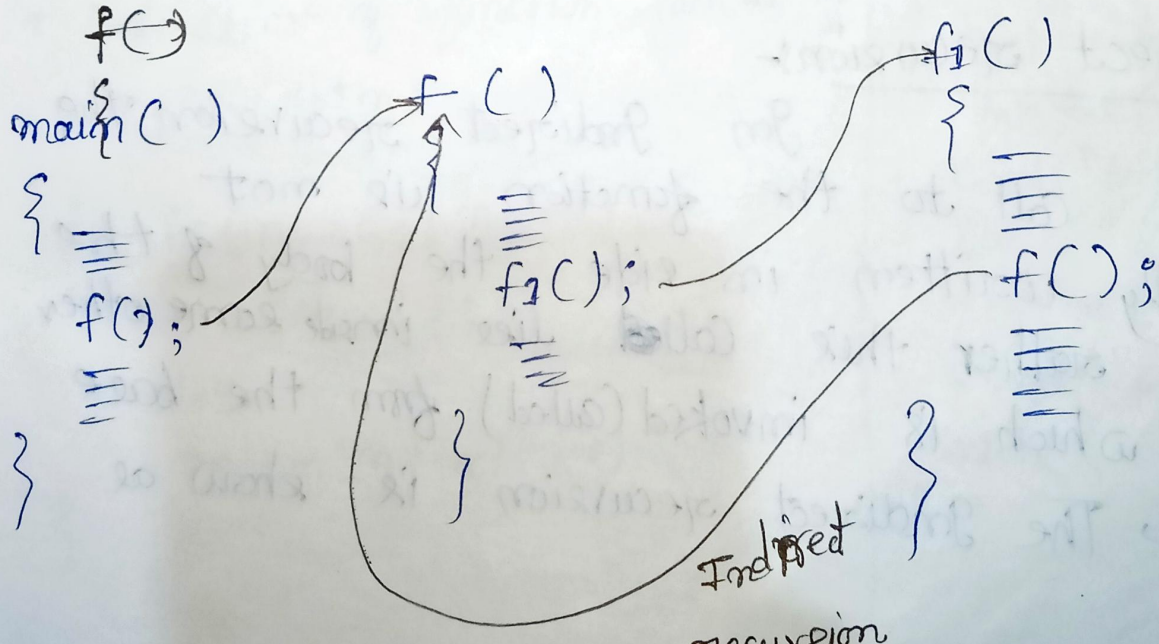
Ex

```
main()
{
    f();
}
```

```
f()
{
    ≡≡≡
    f();
}
```

recursive call (direct recursion)

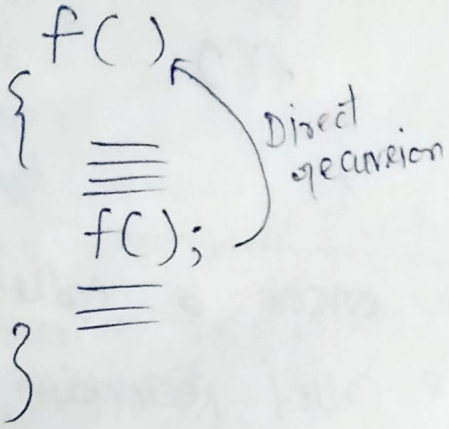
↳ This process of calling itself is called Recursion



There are two types of recursion based on whether function directally calls it'self or it is called via some other function.

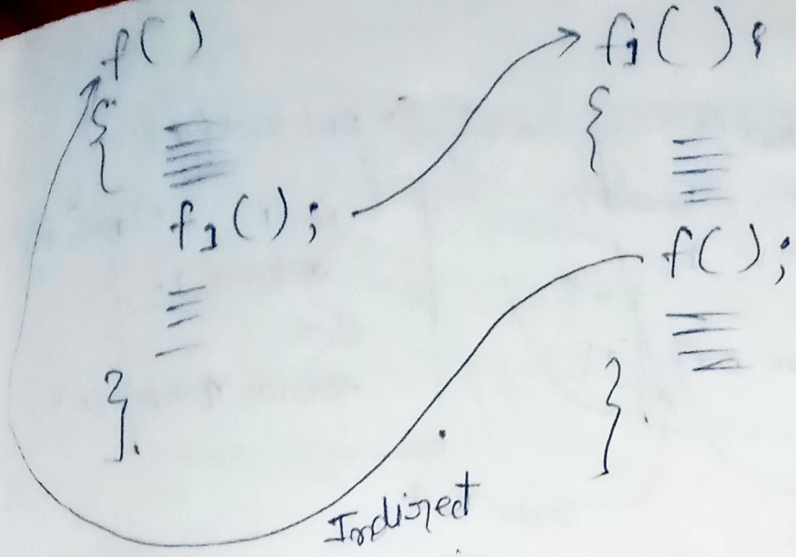
① Direct recursion:-

In direct recursion the recursive call to the function is directally written inside the body of the function.



② Indirect recursion:-

In Indirect recursion the recursive call to the function is not directally written in side the body of the function rather this call lies inside some other function which is invoked (called) from the base function. The Indirect recursion is show as below.



Here f1 function is called by base function f and the recursive call to function f lies in the body of function f1.

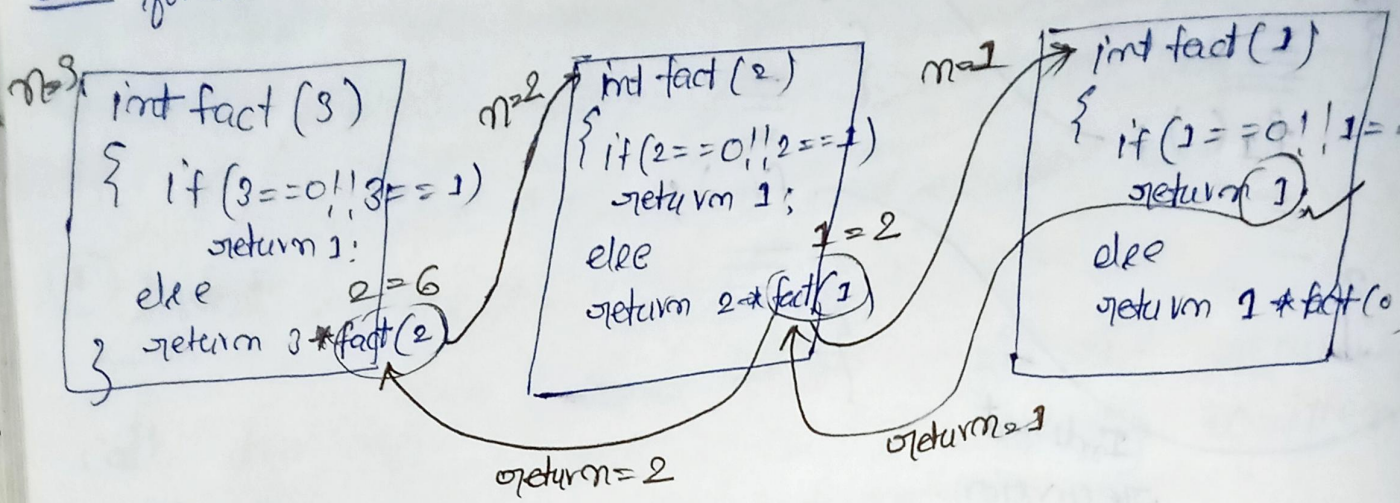
Ex① Base case.

$$F(n) = \begin{cases} 1 & \text{if } n=0 \text{ or } 1 \\ n * F(n-1) & \text{if } n>1 \end{cases}$$

```
/* Definition of function factorial */
int fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return n * fact(n-1);
}
```

```
void main()
{
    int n, fa;
    int fact(int); // Declaration
    Print("Enter the value");
    Scanf("%d", &n);
    fa = fact(n); // Function Calling.
    printf("Factorial = %d", fa);
    getch();
}
```

Ex. for n = 3



Rules of Recursion :- (Property)

Any recursive function must have following properties:

1. should have a base case.
* A base case is the case where the recursive function gives the concrete (exact) solution to the input. And therefore it helps in terminating the recursive.
2. The recursive process must proceed towards the base case during the execution.
3. There should be a recursive call.

(11) $fib(n) = \begin{cases} n & , \text{if } n=0 \text{ or } n=1 \\ fib(n-1)+fib(n-2) & , \text{if } n>1 \end{cases}$

```
/* Definition to Calculate fibbonacci value */
int fib(int n)
{
  if (n==0 || n==1)
    return n;
  else
    return fib(n-1)+fib(n-2);
}
```

Linear Recursion
ex. Factorial

```
factorial (3)
   ↓
3 * factorial(2)
      ↓
   2 * factorial(1)
```

Binary Recursion
ex. Fibbonacci

```
fib(4)
  ↓
fib(3) + fib(2)
```

```
void main()
{
  int n, i, f;
  printf("Enter a value");
  scanf("%d", &n);
  for (i=0; i<=n, i++) // Printing the fibbonacci series.
  {
    f = fib(i);
    printf("%d ", f);
  }
  getch();
}
```

* One more categorization of recursion is as follows.—

① Linear recursion:- A linear recursion includes only one recursive call inside the body of the function.

② Binary recursion:- In this type of recursion -A Binary recursion include the body of the function has two recursive call.

]

Advantages of recursion:- following are the advantages of using recursive called processes.

1) recursive processes are easier to write.
2) recursive processes are easier to understand and comprehend.

(3) It is easier to give recursive solutions to several real life complex problem rather iterative solutions are tougher or complex.

for ex.

(i) Tower of Hanoi.

(ii) Tree Traversal etc.

Disadvantage:- Following are the disadvantages of recursion over iteration —

(1) recursive procedure solutions are relatively slower than it's equivalent iterative solution.

(2) recursive solution occupy more memory as compare to equivalent iterative solution.

```
/* program to find binary of a given decimal no.
   by recursion */

int binary(int x)
{
    int r;
    r = n%.2;
    n = n/2;
    if(n==0)
      printf("%d", r);
    else
    {
      binary(n);
      printf("%d", r);
    }
}
main()
  {
    int a;
    clrscr();
    printf("Enter a number");
    scanf("%d", &a);
    binary(a);
    getch();
  }
```