RECURSION

Definition:

A function containing either a call statement to itself or a call statement to another function that may result in a call statement back to the original function, that function is known as recursive function (the procedure is termed as recursion.

Well defined Recursive function:

In order that a program should not indefinitely a recursive function must have the following properties.

1) There must be certain criteria, known as Base criteria, for which the function does not call itself.
2) Each time the function does call itself, either directly or indirectly, it must be closer to the Base criteria.

A recursive function ~~does call itself~~ with these two properties is said to be Well Defined.

Principles of Recursion

For implementing and designing the good recursive program we must make certain assumptions like:

a) Base case: It is the terminating condition for the problem. While designing any recursive algorithm, we must choose a proper terminating condition for the problem.

b) If condition: It provides terminating criteria.

c) Every time a new recursive call is made, a new memory space is allocated to each automatic variable used by recursive routine.

d) Each time recursive call is there, the duplicate values of the local variables of the recursive call are pushed onto the stack, within the respective call & all these values are available to the respective function, call. when it is popped of from the stack.

e) Recursive case: Generally, else part of the recursive definition calls the functions recursively.

Disadvantages of Recursion

1) It consumes more storage space because the recursive calls along with automatic variables are stored on the stack.

2) The computer may run out memory if the recursive calls are not checked.

3) It is not more efficient in terms of speed & execution time.

4) If proper precautions are not taken, recursion may result in nonterminating functions.

5) In recursion we cannot implement the concept of looping.

e.g.

```
#include <stdio.h>
#include <conio.h>
int sum(int y);
void main()
   {
      int x,y;
      printf("\n Enter any positive integer");
      scanf("%d", &x);

      y = sum(x);
      printf("\n The sum of first %d number is
                                        %d", x, y);
      getch();
   }
int sum(int y)
   {
      if (y==0)
            return 0;
      else
            return (y + sum(y-1));
   }
```

Difference between Recursion and Iteration

Iteration	Recursion
1) It is a process of executing a statement or a set of statements repeatedly, until some specific condition is specified.	1) It is a technique of defining anything in terms of itself.
2) Iteration involves four clear cut steps, initialization condition, execution & updation.	2) There must be an exclusive if statement inside the recursive function specifying the terminating condition.
3) Any recursive problem can be solved iteratively.	3) Not all problems of iteration have recursive solution.
4) Iteration is more efficient in terms of memory utilization & speed.	4) Recursion does not offer higher memory utilization or even greater speed.

ex:

```
int fact (int n)
{
    int i;
    int result;
    result = 1;
    for (i=1; i<=n; i++)
        result = result * i;
    return result;
}
```

```
int fact (int n)
{
    if (n == 0)
        return (1);
    else
        return n * fact(n-1);
}
```

Types of Recursion

The characterizations of types of recursion can be done as following -

1) Whether the function calls itself or not (Direct or Indirect Recursion).
2) Whether there are pending operations at each recursive call (Tail Recursion)
3) The shape of calling pattern - whether pending operations are also recursive (linear or tree recursive)

1> Direct Recursion

A C function is a direct recursive one if it contains an explicit call to itself.

e.g.

```
int direct (int x)
{
    if (x <= 0)
        return x;
    else
        return (direct (x-1));
}
```

2> Indirect Recursion

A C function is said to be an indirect recursive one if it contains a call to another function which ultimately calls the former one.

e.g.

```
int indirect (int x)
{
    if (x <= 0)
        return x;
    else
        return temp(x);
}
int temp(int y)
{
    return indirect (y-1);
}
```

3> Tail Recursion

A recursive function is said to be tail recursive if there are no pending operations to be performed on return from a recursive call.

- Tail recursive functions are often said to ~~be~~ "Return the value of the last recursive call as the value of the function".
- Tail recursion is very desirable because the amount of information which must be stored during the computation is independent of the number of recursive calls.

eg-

```
int fact(int n)
{
    if(n==0)
        return 1;
    else
        return (n*fact(n-1));
}
```

The infamous factorial functions fact is generally written in a nontail recursive manner.
It should be noted that, there is a pending operation namely "multiplication" to be performed on return from each recursive call. Whenever, there is a pending operation, the function is nontail recursive information about each pending operation must be stored, so the amount of information is not indep. of the number of calls.

The above function can be written in a Tail-Recursive way :-

```
int fact_aux(int n, int result)
{
    if (n==0)
        return result;
    else
        return fact_aux(n-1, n*result);
}
int fact(int n)
{
    return fact_aux(n,1);
}
```

4) Linear Recursion

There exists another way to characterize recursive function is by the way in which the recursion grows. the Two basic ways are :

- Linear
- Tree.

a) Linear : A recursive function is said to be linearly recursive when no pending operation involves another recursive call to the function.

b) Tree : A recursive function is said to be tree recursive or non linearly recursive when the pending operation involves another recursive call to function.

eg:

```
int fib(int n)
{
    if(n==0)
        return 0;
    if (n==1)
        return 1;
    return fib(n-1) + fib(n-2);
}
```