RECURSION

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

What is base condition in recursion?

In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

```
int fact(int n)

{

   if (n < = 1) // base case

      return 1;

   else

      return n*fact(n-1);

}
```

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

How a particular problem is solved using recursion?

The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop the recursion. For example, we compute factorial n if we know factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

Why Stack Overflow error occurs in recursion?

If the base case is not reached or not defined, then the stack overflow problem may arise. Let us take an example to understand this.

```
int fact(int n)

{

   // wrong base case (it may cause
   // stack overflow).

   if (n == 100)

       return 1;

   else

       return n*fact(n-1);

}
```

If fact(10) is called, it will call fact(9), fact(8), fact(7) and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error.

What is the difference between direct and indirect recursion?

A function fun is called direct recursive if it calls the same function fun. A function fun is called indirect recursive if it calls another function say fun_new and fun_new calls fun directly or indirectly. Difference between direct and indirect recursion has been illustrated in Table 1.

```
// An example of direct recursion

void directRecFun()

{

    // Some code....

    directRecFun();

    // Some code...

}

// An example of indirect recursion

void indirectRecFun1()

{

    // Some code...

    indirectRecFun2();

    // Some code...

}
```

```
void indirectRecFun2()

{

   // Some code...

   indirectRecFun1();

   // Some code...

}
```

What is difference between tailed and non-tailed recursion?

A recursive function is tail recursive when recursive call is the last thing executed by the function.

example :

```
// An example of tail recursive function

void print(int n)

{

   if (n < 0)  return;

   cout << " " << n;

   // The last executed statement is recursive call

   print(n-1);

}

int main()

{

   cout << fact(5);

   return 0;

}
```

How memory is allocated to different function calls in recursion?

When any function is called from main (), the memory is allocated to it on the stack. A recursive function calls itself, the memory for a called function is allocated on top of memory allocated to calling function and different copy of local variables is created for each function call. When the base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues.

- **Is the factorial method a tail recursive method?**

```
int fact(int x){

  if (x==0)

    return 1;

  else

    return x*fact(x-1);

}
```

- When returning back from a recursive call, there is still one pending operation, multiplication.
- **Therefore, factorial is a non-tail recursive method.**