Recursion

Tail Recursion

As we read before, that recursion is defined when a function invokes/calls itself.

Tail Recursion: A recursive function is said to be following Tail Recursion if it invokes itself at the end of the function. That is, if all of the statements are executed before the function invokes itself then it is said to be following Tail Recursion.

For Example:

```
1
2  // This is a Tail Recursion
3
4  void printN(int N)
5  {
6      if(N==0)
7          return;
8      else
9          cout<<N<<" ";
10
11     printN(N-1);
12 }
13
```

The above function call for **N = 5** will print:

```
5 4 3 2 1
```

Which one is Better-Tail Recursive or Non Tail-Recursive?

The tail-recursive functions are considered better than non-tail recursive functions as tail-recursion can be optimized by compiler. The idea used by compilers to optimize tail-recursive functions is simple, since the recursive call is the last statement, there is nothing left to do in the current function, so saving the current function's stack frame is of no use.

Can a non-tail recursive function be written as tail-recursive to optimize it?

Consider the following function to calculate factorial of N. It is a non-tail-recursive function. Although it looks like a tail recursive at first look. If we take a closer look, we can see that the value returned by **fact(N-1)** is used in **fact(N)**, so the call to fact(N-1) is not the last thing done by fact(N).

```
int fact(int N)
{
    if (N == 0)
        return 1;

    return N*fact(N-1);
}
```

The above function can be written as a tail recursive function. The idea is to use one more argument and accumulate the factorial value in second argument. When N reaches 0, return the accumulated value.

```
int factTR(int N, int a)
{
    if (N == 0)
        return a;

    return factTR(N-1, N*a);
}
```