Tail Recursion

What is tail recursion?

A recursive function is tail recursive when recursive call is the last thing executed by the function. For example the following C++ function print() is tail recursive.

```
// 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);
}
```

Why do we care?

The tail recursive functions 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 (See this for more details).

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)

C++

```
#include<iostream>
using namespace std;

// A NON-tail-recursive function.  The function is not tail
// recursive because the value returned by fact(n-1) is used in
// fact(n) and call to fact(n-1) is not the last thing done by fact(n)
unsigned int fact(unsigned int n)
{
    if (n == 0) return 1;

    return n*fact(n-1);
}

// Driver program to test above function
int main()
{
    cout << fact(5);
    return 0;
}
```

Java

```
class GFG {

    // A NON-tail-recursive function.
    // The function is not tail
```

```
    // recursive because the value
    // returned by fact(n-1) is used
    // in fact(n) and call to fact(n-1)
    // is not the last thing done by
    // fact(n)
    static int fact(int n)
    {
        if (n == 0) return 1;

        return n*fact(n-1);
    }

    // Driver program
    public static void main(String[] args)
    {
        System.out.println(fact(5));
    }
}

// This code is contributed by Smitha.
```

Python 3

```
# A NON-tail-recursive function.
# The function is not tail
# recursive because the value
# returned by fact(n-1) is used
# in fact(n) and call to fact(n-1)
# is not the last thing done by
# fact(n)
def fact(n):

    if (n == 0):
        return 1

    return n * fact(n-1)

# Driver program to test
# above function
print(fact(5))
# This code is contributed by Smitha.
```

C#

```
using System;

class GFG {

    // A NON-tail-recursive function.
    // The function is not tail
    // recursive because the value
    // returned by fact(n-1) is used
    // in fact(n) and call to fact(n-1)
    // is not the last thing done by
    // fact(n)
    static int fact(int n)
    {
        if (n == 0)
            return 1;

        return n * fact(n-1);
    }

    // Driver program to test
    // above function
    public static void Main()
    {
        Console.Write(fact(5));
    }
}

// This code is contributed by Smitha
```

PHP

```php
<?php
// A NON-tail-recursive function.
// The function is not tail
// recursive because the value
// returned by fact(n-1) is used in
// fact(n) and call to fact(n-1) is
// not the last thing done by fact(n)

function fact( $n)
{
    if ($n == 0) return 1;

    return $n * fact($n - 1);
}

    // Driver Code
    echo fact(5);

// This code is contributed by Ajit
?>
```

Output :

```
120
```

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.

C++

```cpp
#include<iostream>
using namespace std;

// A tail recursive function to calculate factorial
unsigned factTR(unsigned int n, unsigned int a)
{
    if (n == 0)  return a;

    return factTR(n-1, n*a);
}

// A wrapper over factTR
unsigned int fact(unsigned int n)
{
   return factTR(n, 1);
}

// Driver program to test above function
int main()
{
    cout << fact(5);
    return 0;
}
```

Java

```java
// Java Code for Tail Recursion

class GFG {

    // A tail recursive function
    // to calculate factorial
    static int factTR(int n, int a)
    {
        if (n == 0)
            return a;

        return factTR(n - 1, n * a);
    }

    // A wrapper over factTR
    static int fact(int n)
    {
```

```
        return factTR(n, 1);
    }

    // Driver code
    static public void main (String[] args)
    {
        System.out.println(fact(5));
    }
}

// This code is contributed by Smitha.
```

Python 3

```
# A tail recursive function
# to calculate factorial
def factTR(n, a):

    if (n == 0):
        return a

    return factTR(n - 1, n * a)

# A wrapper over factTR
def fact(n):
    return factTR(n, 1)

# Driver program to test
#  above function
print(fact(5))

# This code is contributed
# by Smitha
```

C#

```
// C# Code for Tail Recursion
using System;

class GFG {

    // A tail recursive function
    // to calculate factorial
    static int factTR(int n, int a)
    {
        if (n == 0)
            return a;

        return factTR(n - 1, n * a);
    }

    // A wrapper over factTR
    static int fact(int n)
    {
        return factTR(n, 1);
    }

    // Driver code
    static public void Main ()
    {
        Console.WriteLine(fact(5));
    }
}

// This code is contributed by Ajit.
```

PHP

```
<?php
// A tail recursive function
// to calculate factorial
function factTR($n, $a)
{
    if ($n == 0) return $a;
```

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

// A wrapper over factTR
function fact($n)
{
    return factTR($n, 1);
}

// Driver program to test
// above function
echo fact(5);

// This code is contributed
// by Smitha
?>
```

Output :

```
120
```

Next articles on this topic:

Tail Call Elimination

QuickSort Tail Call Optimization (Reducing worst case space to Log n)

References:

http://en.wikipedia.org/wiki/Tail_call

http://c2.com/cgi/wiki?TailRecursion

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

Tail Recursion for Fibonacci

Tail recursion to calculate sum of array elements.

Tail Call Elimination

QuickSort Tail Call Optimization (Reducing worst case space to Log n)

Recursion

Sum of the series 1^1 + 2^2 + 3^3 + ... + n^n using recursion

Find the value of ln(N!) using Recursion

Product of 2 Numbers using Recursion

Product of 2 numbers using recursion | Set 2

Generating subarrays using recursion

Generating all possible Subsequences using Recursion

Reversing a queue using recursion

Difference between Recursion and Iteration

Program to Calculate e^x by Recursion

Sort a stack using recursion

Improved By : Smitha Dinesh Semwal, jit_t

Article Tags : Analysis Recursion Quick Sort tail-recursion

Practice Tags : Recursion

13

To-do Done

2.5

Based on **54** vote(s)

Feedback/ Suggest Improvement | Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.