Session 8

Week -4

Tail vs. Non-Tail Recursion

What is tail recursion?

The tail recursion is basically using the recursive function as the last statement of the function. So when nothing is left to do after coming back from the recursive call, that is called tail recursion. We will see one example of tail recursion

Test case -1

```
#include <iostream>
using namespace std;
void printN(int n) {
    if (n < 0) {
        return;
    }
    cout << n << " ";
    printN(n - 1);
}
int main() {
    printN(10)
```

The tail recursion is better than non-tail recursion. As there is no task left after the recursive call, it will be easier for the compiler to optimize the code. When one function is called, its address is stored inside the stack. So if it is tail recursion, then storing addresses into stack is not needed.

We can use factorial using recursion, but the function is not tail recursive. The value of fact(n-1) is used inside the fact(n).

Test Case-2

```
#include <iostream>
using namespace std;
long fact(int n) {
    if (n <= 1)
        return 1;
    cout << n << " ";
    n* fact(n - 1);
}
int main() {
    fact(6);
}
```

Example-1

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

void Triangle(int x) {
    if (x <= 0) return;

    Triangle(x - 1);

    for (int i = 1; i <= x; i++)
        cout << "*";
    cout << endl;
}
int main()
{
    Triangle(7);


}
```

Example-2

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

void TriangleRev(int x) {
       if (x <= 0) return;

       for (int i = 1; i <= x; i++)
              cout << "*";

       cout << endl;

       TriangleRev(x - 1);
}

int main()
{
       TriangleRev(7);


}
```

Class task	22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Lab Question

Sort The Unsorted Numbers with both tail recursive and Normal recursive approach

Sample Input and Output

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

Algorithm Discussion {Bubble Sort}

```
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    // Base case
    if (n == 1)
        return;

    // One pass of bubble sort. After
    // this pass, the largest element
    // is moved (or bubbled) to end.
    for (int i = 0; i < n - 1; i++)
        if (arr[i] > arr[i + 1])
            swap(arr[i], arr[i + 1]);

    // Largest element is fixed,
    // recur for remaining array
    bubbleSort(arr, n - 1);
}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
    int n = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array : \n");
    printArray(arr, n);
    return 0;
                                                }
```

Direct vs. Indirect Recursion

Direct recursion: When function calls itself, it is called direct recursion,

```
#include <iostream>
using namespace std;
//Factorial function
int f(int n) {
    /* This is called the base condition, it is
     * very important to specify the base condition
     * in recursion, otherwise your program will throw
     * stack overflow error.
     */
    if (n <= 1)
        return 1;
    else
        return n * f(n - 1);
}
int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    cout << "Factorial of entered number: " << f(num);
    return 0;
}
```

Indirect recursion: When function calls another function and that function calls the calling function, then this is called indirect recursion. For example: function A calls function B and Function B calls function A.

```
#include <iostream>
using namespace std;
int fa(int);
int fb(int);
int fa(int n) {
    if (n <= 1)
        return 1;
    else
        return n * fb(n - 1);
}
int fb(int n) {
    if (n <= 1)
        return 1;
    else
        return n * fa(n - 1);
}
int main() {
    int num = 5;
    cout << fa(num);
    return 0;
}
```

Nested Recursion

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

int fun(int n)
{
    if (n > 100)
    {
        return n - 10;
    }
    else
    {
        return fun(fun(n + 11));
    }
}
int main()
{
    int a = 95;
    cout << fun(a) << endl;
}
```

Excessive Recursion

Excessive Recursion: Example

- To show how much this formula is inefficient, let us try to see how Fib(6) is evaluated.

```
int fib(int n) {
  if (n<2)
    return n;
  else
    return fib(n-2)+Fib(n-1);
}
```