Data Structures Lab 05(a)

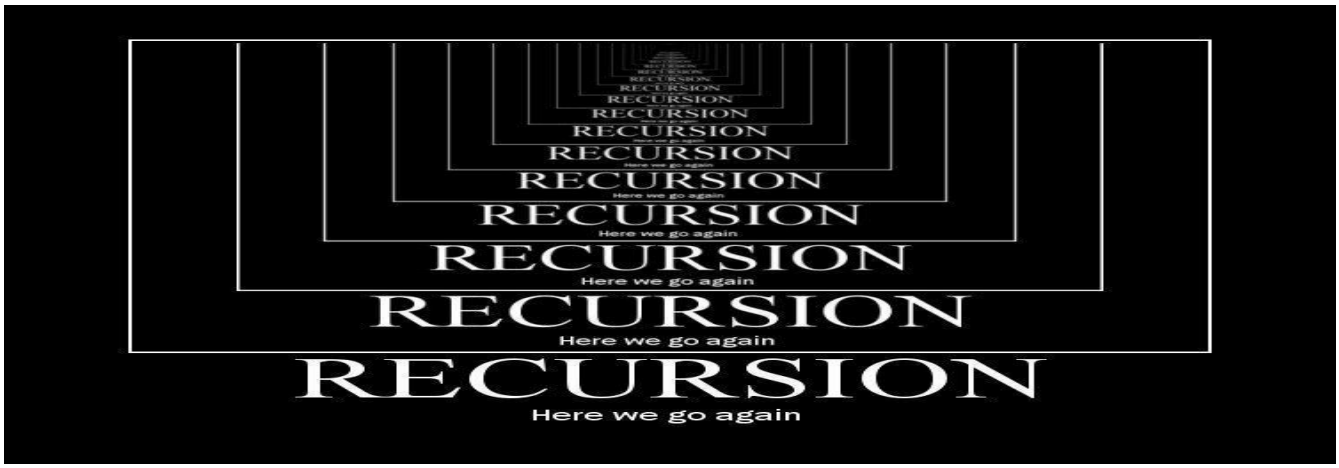
Course: Data Structures (CL2001) **Semester:** Fall 2024
Instructor: Sameer Faisal **T.A:** N/A

Note:

- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

RECURSION

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using a recursive algorithm, certain problems can be solved quite easily. Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write.



Base Condition in Recursion

```
int Funct(int n)
{    if (n < = 1) // base case       return 1;
   else
      return Funct (n-1);
}
void printAge(int n) {
   if (n < 18) {
      cout << "Age under 18: " << n << endl;
      n += 5; // Increment the age for the next iteration
      printAge(n); // Recursive call
   }
```

Key Points: 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.

Class Example 1: Generate the following sequence with recursive approach

1 , 3 , 6 , 10 , 15 , 21 , 28 . . . .

Class Example 2: Generate the following sequence with recursive approach

0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 . . .

Direct & Indirect Recursion

Direct Recursion:

```
void X()
{    // Some code....
   X();
   // Some code...
}
```

Indirect Recursion:

```
void printAge(int n); // Forward declaration
void incrementAndPrintAge(int n) {
   if (n < 18) {
      cout << "Age under 18: " << n << endl;
      n += 5; // Increment the age for the next iteration
      printAge(n); // Indirect recursive call
   }
}
void printAge(int n) {
   if (n < 18) {
      cout << "Age under 18: " << n << endl;
      n += 5; // Increment the age for the next iteration
      incrementAndPrintAge(n); // Indirect recursive call
  }
}
```

Class Example 3: Write an indirect recursive code for the above class example 2 with same approach as defined in the above sample code of In-Direct Recursion.

Tailed & Non Tailed Recursion

Tailed Recursion:

```
void Funct (int a)
{
   if (a < 1)  return;// base case

cout<<a;
   // recursive call
   funct (a/2);
}
```

Non Tailed Recursion:

```
void Funct (int a)
{
   if (a < 1)  return;// base case

   // recursive call
   return funct (a/2);
}
```

Nested Recursion

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

int fun(int n)
{
   if (n > 100)
      return n - 10;

   // A recursive function passing parameter
   // as a recursive call or recursion inside
   // the recursion
   return fun(fun(n + 11));
}
int main()
{
   int r;
   r = fun(95);

   cout << " " << r;

   return 0;
}
```

ISSUES IN RECURSION

Complexity of Recursive Logic: Problem: Designing and understanding recursive solutions can be complex, especially for problems with multiple recursive calls or complex base cases.

Stack Overflow: Recursion heavily relies on the call stack. If the recursion goes too deep, it can cause a stack overflow, leading to a program crash.

Example:

```
void infiniteRecursion() {
        infiniteRecursion(); // Calls itself indefinitely
}
```

1. **Infinite Recursion:** If the base condition is not defined or incorrect, the recursion may continue indefinitely, causing infinite recursion.

```
Example: void badRecursion(int n) {
   if (n > 0) {
      badRecursion(n); // Missing decrement of n
   }
}
```

Performance: Recursion can be inefficient for large inputs due to repeated calculations. Memoization or iteration may be better for performance in such cases.

Example: Calculating Fibonacci numbers using simple recursion leads to exponential time complexity.

```
int fibonacci(int n) {
   if (n <= 1) return n; // Base condition
   return fibonacci(n - 1) + fibonacci(n - 2); // Recursive calls
}
```