Instructor signature: ____

Data Structures Lab
Session 3

Course: Data Structures (CS2001)

Note:
- Lab manual cover following below recursion topics **{Base Condition, Direct and Indirect Recursion, Tailed Recursion, Nested Recursion, Backtracking}**
- Maintain discipline during the lab.
- 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.

Base Condition in Recursion

Sample Code

```
int Funct(int n)
{    if (n < = 1) // base case       return 1;
    else
        return Funct (n-1);
}
```

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.

Task-1:

a. Generate the following sequence with recursive approach

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

b. Generate the following sequence with recursive approach

1 , 1 , 2 , 4 , 7 , 11 , 16 , 22 . . . .

Direct and Indirect Recursion

Sample Code (Direct Recursion)

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

Sample Code (In-Direct Recursion)

```
void indirectRecFun1()
{   // Some code...
    indirectRecFun2();
    // Some code...
}
void indirectRecFun2()
{    // Some code...
    indirectRecFun1();
    // Some code...
}
```

Task-2:

a. **Write a indirect recursive code for the above task-1 (a,b) part with same approach as defined in the above sample code of In-Direct Recursion**

Tailed and Non Tailed Recursion

Sample Code (Non tailed Recursion)

```
void Funct (int a)
{
    if (a < 1)  return;
    cout << "The current Output is  " << n;

    // recursive call
    Funct (n-1);
}
```

Sample Code (Tailed Recursion)

```
unsigned Funct 1(int n, int a)
{
    if (n == 1)  return a;

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

```
 int Funct 2(unsigned int n)
 {
  return Funct 1(n, 1);
 }

int main()
{
   cout << Funct 2(5);
   return 0;
}
```

Task 3:
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
```

<u>Nested Recursion</u>

Sample Code

```
#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;
}
```

Task 4:

Dry run the outputs of the upper code in order to find out how the recursive calls are made

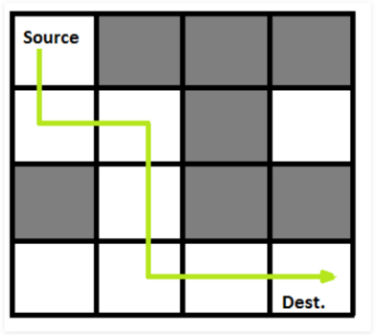
Backtracking

Sample Pseudocode

```
void findSolutions(n, other params) :
   if (found a solution) :
      solutionsFound = solutionsFound + 1;
      displaySolution();
      if (solutionsFound >= solutionTarget) :
         System.exit(0);
      return

   for (val = first to last) :
      if (isValid(val, n)) :
         applyValue(val, n);
         findSolutions(n+1, other params);
         removeValue(val, n);
```

A Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. A rat starts from source and has to reach the destination. The rat can move only in two directions: forward and down.



In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination. Note that this is a simple version of the typical Maze problem. For example, a more complex version can be that the rat can move in 4 directions and a more complex version can be with a limited number of moves.

Task-5

A. Design the function with recursive approach to find the number of existing destination path in the above provided sample code link
B. Change the Maze with following configuration .Find the optimal path to reach the destination with recursive approach

int maze[N][N] = { { 0, 0, 0, 1 },
{ 0, 1, 1, 1 },
{ 0, 1, 1, 0 },

Sample Code
https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-2/

Reference:

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

All recursive algorithms must have the following:

1. Base Case (i.e., when to stop)
2. Work toward Base Case
3. Recursive Call (i.e., call ourselves)

The "work toward base case" is where we make the problem simpler (e.g., divide list into two parts, each smaller than the original). The recursive call, is where we use the same algorithm to solve a simpler version of the problem. The base case is the solution to the "simplest" possible problem (For example, the base case in the problem 'find the largest number in a list' would be if the list had only one number... and by definition if there is only one number, it is the largest).

Direct and Indirect Recursion

A function X is called direct recursive if it calls the same function X. A function X is called indirect recursive if it calls another function say X_new and X_new calls fun directly or indirectly.

Tailed and Non Tailed Recursion

In Tail Recursion you complete your calculations first, then call the recursive function, passing the results of your current step to the next recursive step. As a result, the last sentence is written as (return (recursive-function prams)). The return value of any recursive action is essentially the same as the return value of the next recursive call. It's worth noting that tail recursion is essentially the same as looping. It's not merely a question of compiler optimization.

Nested Recursion:

In this recursion, a recursive function will pass the parameter as a recursive call. That means **"recursion inside recursion".**

In this recursion, there may be more than one functions and they are calling one another in a circular manner

Bactracking

Backtracking is an improvement of the brute force approach. It systematically searches for a solution to a problem among all available options. In backtracking, we start with one possible option out of many available options and try to solve the problem if we are able to solve the problem with the selected move then we will print the solution else we will backtrack and select some other option and try to solve it. If none if the options work out we will claim that there is no solution for the problem

Backtracking is a form of recursion. The usual scenario is that you are faced with a number of options, and you must choose one of these. After you make your choice you will get a new set of options; just what set of options you get depends on what choice you made. This procedure is repeated over and over until you reach a final state. If you made a good sequence of choices, your final state is a goal state; if you didn't, it isn't

Three Types of Backtracking solution which are **Decision Problem** – In this, we search for a feasible solution.**Optimization Problem** – In this, we search for the best solution. **Enumeration Problem** – In this, we find all feasible solutions.

Lab3: Implementing Recursion, solving a simple maze		
Std Name:		**Std_ID:**
Lab1-Tasks	**Completed**	**Checked**
Task #1		
Task #2		
Task #3		
Task# 4		
Task# 5		