**Data Structures Lab**

***Session 5***

**Course:** Data Structures (CL1002) **Semester:** Fall 2024

**Instructor:** Shafique Rehman  **T.A:**

**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.
* Recursion Visualization Tool (https://visualgo.net/en/recursion)

**Base Condition in Recursion**

**Sample Code**

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.

**Direct and Indirect Recursion**

**Sample Code (Direct Recursion)**

void X()

{ // Some code....

X();

// Some code...

}

**Sample Code (In-Direct 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

}

}

**Tailed and Non Tailed Recursion**

**Sample Code (Non tailed Recursion)**

void Funct (int a)

{

if (a < 1) return;// base case

// recursive call

return funct (a/2);

}

**Sample Code (Tailed Recursion)**

void Funct (int a)

{

if (a < 1) return;// base case

cout<<a;

// recursive call

funct (a/2);

}

**Nested Recursion**

**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;

}

|  |
| --- |
| **Issues in Recursion** |

1. **Complexity of Recursive Logic:** Problem: Designing and understanding recursive solutions can be complex, especially for problems with multiple recursive calls or complex base cases.
2. **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**

**}**

**}**

1. **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**

**}**

**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) :

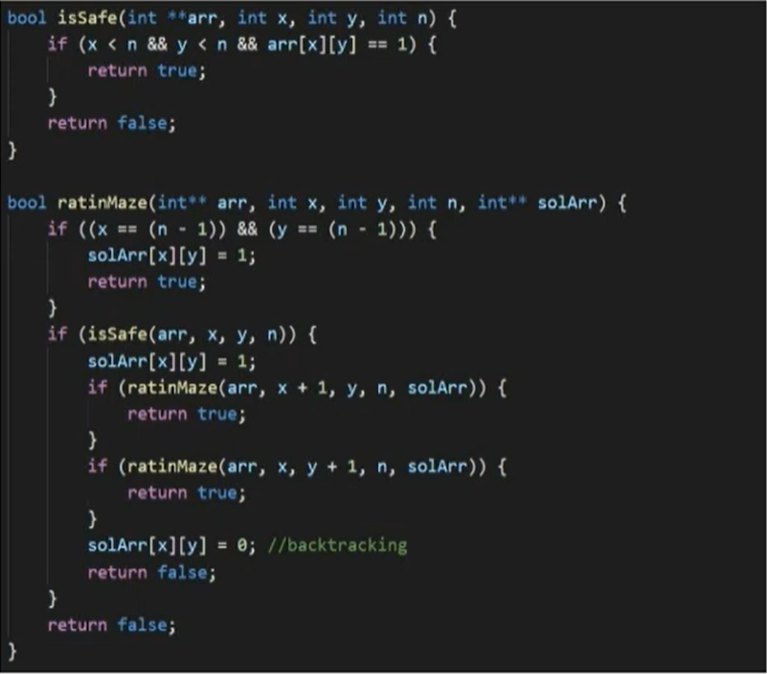
if (isValid(val, n)) :

applyValue(val, n);

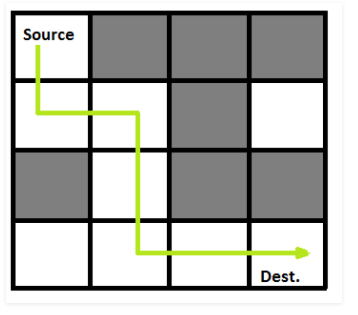
findSolutions(n+1, other params);

removeValue(val, n);

**Code Example : (Maze in Rate)**

****

**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**.

**N-Queen Problem :** The N-Queens problem is a classic problem in computer science and combinatorial optimization. The goal is to place N queens on an N×N chessboard in such a way that no two queens threaten each other. In other words, no two queens can share the same row, column, or diagonal.

bool isSafe(int board[], int row, int col) {

for (int i = 0; i < row; i++) {

// Check if there's a queen in the same column or diagonals

if (board[i] == col || abs(board[i] - col) == abs(i - row)) {

return false;

}

}

return true;

}

bool solveNQueens(int board[], int n, int row = 0) {

if (row == n) {

// All queens are placed successfully

return true;

}

for (int col = 0; col < n; col++) {

if (isSafe(board, row, col)) {

board[row] = col; // Place the queen in this column

// Recursively place queens in the next row

if (solveNQueens(board, n, row + 1)) {

return true; // If a solution is found, return true

}

// If placing the queen in this column doesn't lead to a solution, backtrack

board[row] = -1;

}

}

// If no safe position is found, return false (backtrack)

return false;

}