**UNIVERSITY INSTITUTE OF COMPUTING**

**PROJECT REPORT ON**

**MAZE BUILDER AND SOLVER**

Program Name: BCA(DATA SCIENCE)

Subject Name/Code: Data Structures(24CAT-152)

**Submitted by: Submitted to:**

**Name: Praveen Kumar Name: Ms. Shilpi Garg**

**UID:** 24BCD10004 **Designation: Assistant Prof.**

**Section: 24BCD-1/A**

# ABSTRACT

## Introduction:

**This C-based Maze Solver application enables users to input a custom 5x5 maze and find a path from a start point ('S') to an endpoint ('E') using a stack data structure. The maze is represented with '0' for paths, '1' for walls, 'S' for the start, and 'E' for the end. The program employs a stack to backtrack and explore possible paths, marking the solution with '\*'. Users can define their own maze, view it, and initiate the solving process. The application focuses on dynamic memory allocation and stack operations, providing an interactive and educational experience for learning data structures in C.**

## Technique:

**The application is developed using C with a focus on procedural programming and data structure implementation. It employs the following techniques:**

**1. Stack Data Structure: A stack is implemented using a linked list to store coordinates of visited positions, enabling backtracking when a dead end is reached (LIFO principle).**

**2. Dynamic Memory Allocation: The stack uses malloc() and free() for efficient memory management, avoiding fixed-size arrays.**

**3. User Input Handling: The program accepts a 5x5 maze as input from the user, validating the presence of 'S' and 'E' before solving.**

## System Configuration:

* + **OS: Windows 10, Linux, Macbook(intel or m series)**
  + **Processor: Intel Core i3 (minimum); Core i5 or higher recommended; M1 chip(minimum) or higher**
  + **RAM: 4 GB (minimum); 8 GB recommended**
  + **Development Environment: Any C IDE (e.g., Visual Studio, Code::Blocks) or Visual Studio Code with a C compiler (GCC or Microsoft C Compiler)**

# **SUMMARY**

**Input:**

**Main Menu:**

* **The user is prompted to choose between:**
* **1: Input and Solve Maze**
* **2: Display Current Maze**
* **0: Exit**

**For Input and Solve Maze:**

* **User inputs a 5x5 maze row by row:**
* **Enter 5 characters per row (e.g., "11011" for row 1).**
* **Valid characters: '0' (path), '1' (wall), 'S' (start), 'E' (end).**
* **Exactly one 'S' and one 'E' must be present.**

**Example Input:**

**- Row 1: 11111**

**- Row 2: 1S011**

**- Row 3: 10001**

**- Row 4: 11E11**

**- Row 5: 11111**

**- The program validates the input and proceeds to solve the maze.**

**For Display Current Maze: - The program prints the current maze as entered by the user or modified with the solution path.**

**1. Prompt the user to input a 5x5 maze, storing it in a 2D character array.**

**2. Validate the input to ensure exactly one 'S' and one 'E' are present.**

**3. Initialize a stack to store coordinates (row, col) of the current path.**

**4. Start at the 'S' position and push it onto the stack.**

**5. While the stack is not empty:**

* **Pop the top position from the stack.**
* **Check if it’s the endpoint ('E'). If yes, mark the path and stop.**
* **Explore four directions (up, right, down, left).**
* **If a direction is valid (within bounds, not a wall, not visited), push it onto the stack and mark it as visited.**

**6. If the stack empties without finding 'E', declare no solution exists.**

**7. Display the maze with the solution path marked by '\*'.**

**Main Menu:**

**1: Input and Solve Maze**

**2: Display Current Maze**

**0: Exit. Input and Solve Maze:**

**- Prompt: "Enter row 1 (5 chars: 0,1,S,E):**

**Example Input:**

**- Row 1: 11111**

**- Row 2: 1S011**

**- Row 3: 10001**

**- Row 4: 11E11**

**- Row 5: 11111**

**- Example Output (with solution path marked by '\*'):**

* **1 1 1 1 1**
* **1 S \* 1 1**
* **1 \* \* \* 1**
* **1 1 E 1 1**
* **1 1 1 1 1**

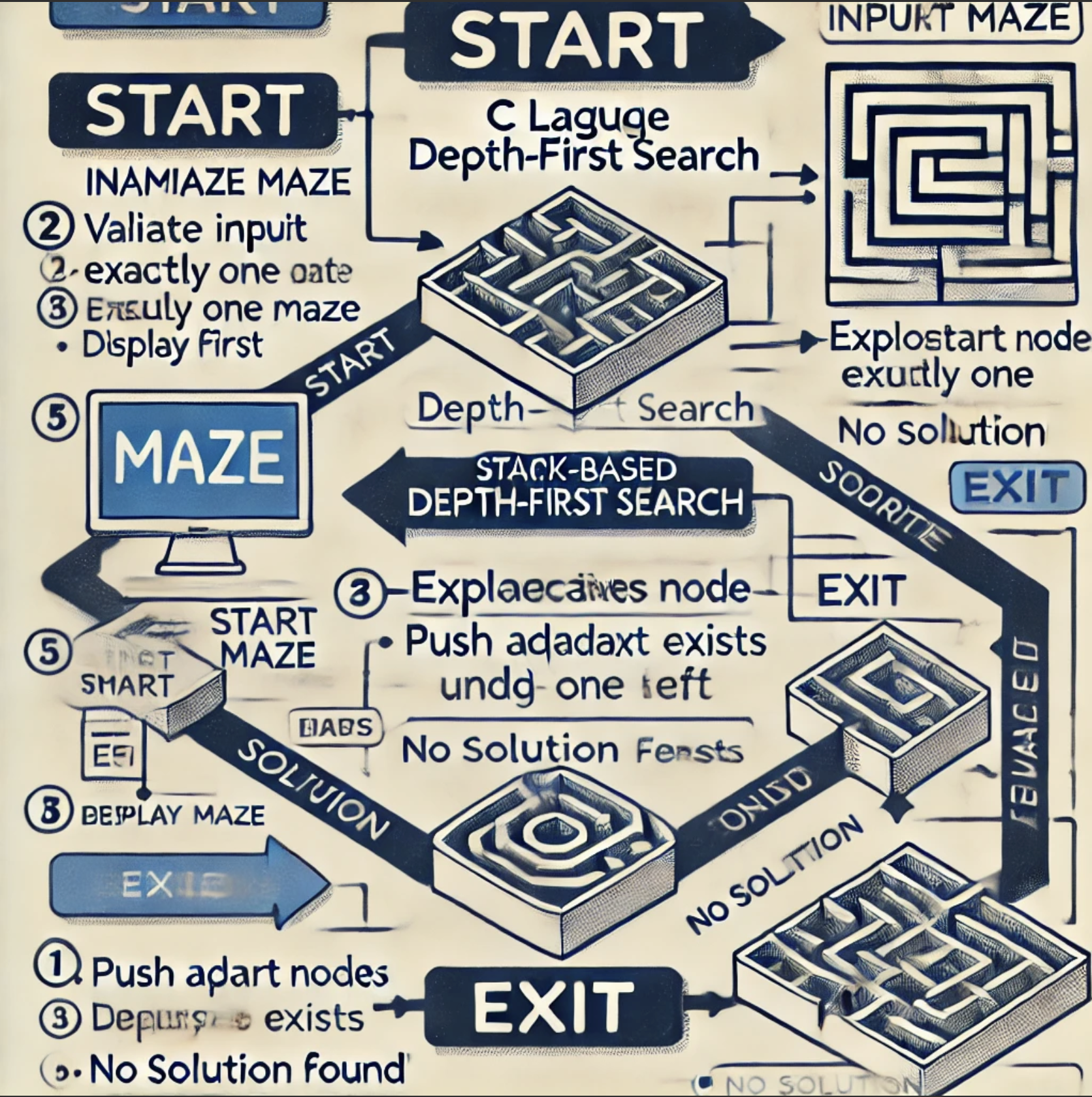
**- If no solution: "No solution found."**

**Display Current Maze:**

**- Example Output (before solving):**

* **1 1 1 1 1**
* **1 S 0 1 1**
* **1 0 0 0 1**
* **1 1 E 1 1**
* **1 1 1 1 1**

## Process:



Output:

