Data Structures and Algorithms Lab
SE-F22
LAB-05

Issue Date: March 8, 2024

Start Time: 10:00AM
Submission Time: 1:00PM

Total Marks: 70

The objective of this lab is to:
Implement Stack based algorithms.

Instructions:
1) Follow the question instructions very carefully, no changes in function prototypes are allowed.
2) First understand the problem, find the solution then code.
3) You may only use your own Stack class (implemented as home task), cannot use built in STL Stack.

Task 01(Calculator) [50 Marks]

You are tasked with building a calculator program that can evaluate complex mathematical expressions efficiently. The calculator should support addition, subtraction, multiplication, division, and parentheses for grouping operations.

Tasks to Perform:

- Implement Infix to Postfix Conversion. **[10 Marks]**
- Implement Infix to Prefix Conversion. **[10 Marks]**
- Evaluate Postfix Expression. **[10 Marks]**
- Evaluate Prefix Expression. **[10 Marks]**
- Support User Input **[10 Marks]**
    - Develop a user interface where users can input arithmetic expressions in infix notation.
    - Provide options for users to convert infix expressions to postfix or prefix and evaluate them.

Basic Structure of Class:

```
class Calculator
{
private:
    srting infix;
    char delimeter;
public:
    string infixToPostfix();
    string infixToPrefix();
    int evaluatePostfix();
    int evaluatePrefix();
};
```

Task 02 (Treasure Hunter) [20 Marks]

You are a treasure hunter exploring an ancient labyrinth filled with traps and secret chambers. Your goal is to find the hidden treasure located at the heart of the maze. To do so, you must navigate through the maze, avoiding traps and dead ends, and find a reachable path to the treasure.

You are given a maze of dimensions 5x7, you have to find a path to the treasure marked with letter 'E' in the maze. Your starting position is marked as 'S' in the maze at any valid index. An obstacle in the path is represented by '#' which means you cannot move at that index.

Given the maze and starting index, you have to print a valid path from start to treasure, in case there is no valid path then print a message displaying 'No Valid Path Found!' message.

Example:

```
{'.', '.', '.', '#', '#', '#', '#'},
{'#', '#', '.', '#', '#', '#', '#'},
{'#', '.', '.', '.', '.', '.', '#'},
{'#', 'S', '#', '#', '#', 'E', '#'},
{'#', '#', '#', '#', '#', '.', '#'},
```

Output:

```
Found the treasure! Path:
(3, 5) (2, 5) (2, 4) (2, 3) (2, 2) (2, 1) (3, 1)
```

Hints: At a given position you can move in four possible directions; UP, DOWN, RIGHT, LEFT & for each move check whether it is possible to move to that point or whether that point is previously visited or not. Store row and column information of a point using a Struct Point.

```
struct Point {
    int row, col;

    Point(int r, int c) : row(r), col(c) {}
};
```

Function Prototype:

```
void findTreaseure(const char maze[5][7], Point start);
```

No Cheating 🙂