**EXPERIMENT NO. 1**

| **Objective(s):**  Implement a stack and perform operations like creation, push, pop, traverse, peek, and search using a linear data structure efficiently |
| --- |
| **Outcome:**  Efficient data organization with the ability to quickly add, remove, find, and iterate elements in both forward and backward directions using stack operations. |
| **Problem Statement:**  Implement Stack and its operations like (creation push pop traverse peek search) using linear data structure |
| **Background Study:**  A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. This means that the last element added to the stack will be the first one to be removed. Stacks are used in various applications such as expression evaluation, backtracking algorithms, function call management in recursion, and undo mechanisms in text editors. Understanding stacks and their operations is fundamental in computer science and programming.  **Basic Operations on Stacks**   1. **Creation**: Initializing an empty stack. This can be done using an array or a list in most programming languages. 2. **Push**: Adding an element to the top of the stack. This operation increases the stack's size by one. 3. **Pop**: Removing the top element from the stack and returning it. This operation decreases the stack's size by one. 4. **Traverse**: Iterating through the elements of the stack. This can be done in either forward (from bottom to top) or backward (from top to bottom) direction. 5. **Peek**: Viewing the top element of the stack without removing it. 6. **Search**: Finding the position of a specific element in the stack from the top. If the element is not found, it returns -1.   **Applications of Stacks**   * **Expression Evaluation**: Stacks are used to evaluate arithmetic expressions, particularly those written in postfix (Reverse Polish Notation) form. * **Backtracking**: In algorithms like depth-first search, stacks are used to remember the paths that need to be explored. * **Function Call Management**: Stacks are used to manage function calls and local variables in programming languages that support recursion. * **Undo Mechanism**: Text editors and other applications use stacks to implement undo functionality, where the most recent operations can be reversed. |

| **Algorithm (Student Work Area):**   1. **Creation**   Step 1: Initialize an list `stack`empty  Step 2: Return `stack`   1. **Push**   Step 1: Append `element` to `stack`  Step 2: Return `stack`   1. **Pop**   Step 1: If `stack` is not empty:  a. Remove and return the last element of `stack`  Step 2: Else:  a. Return an error message "Stack is empty"   1. **Traverse**   Step 1: For each element `e` in `stack`:  a. Print `e`   1. **Peek**   Step 1: If `stack` is not empty:   1. Return the last element of `stack   Step 2: Else:   1. Return an error message "Stack is empty" 2. **Search**   Step 1: If `element` is in `stack`:  a. Return `len(stack) - stack.index(element)`  Step 2: Else:  a. Return -1 |
| --- |

| **Code:** |
| --- |
| **OUTPUT :** |