LAB # 7

Follow the steps to implement Stack data-structure to perform basic operations and show the results using C++ language

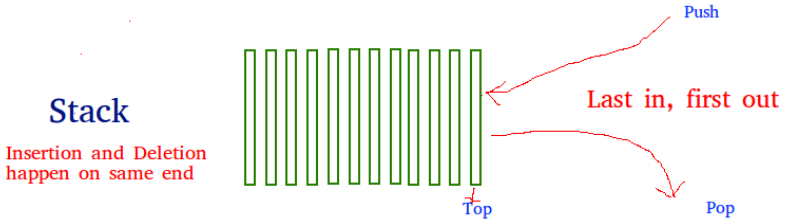
Objective

- To understand about Stack implementation in C++
- To implement different stack-based algorithms

Pre-Lab

Stacks

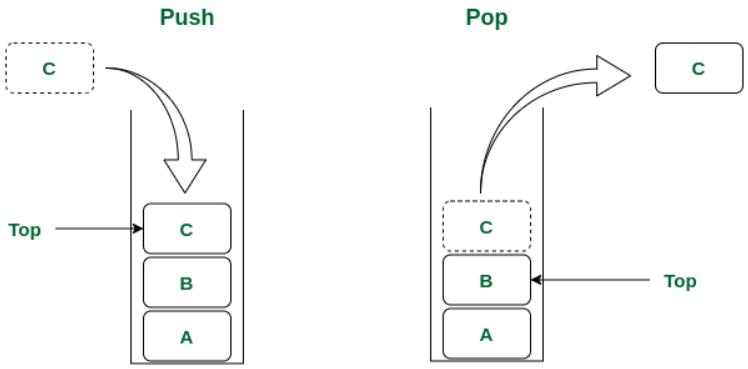
Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).



Stack Data Structure

The reason why Stack is considered a complex data structure is that it uses other data structures for implementation, such as Arrays, Linked lists, etc. based on the characteristics and features of Stack data structure

- **Description of elements:** A stack is defined to hold one type of data element. Only the element indicated by the top can be accessed. The elements are related to each other by the order in which they are put on the stack.
- **Description of operations:** Among the standard operations for a stack are:
  - insert an element on top of the stack (push),
  - remove the top element from the stack (pop),
  - determine if the stack is empty.



An example of a stack is the pop-up mechanism that holds trays or plates in a cafeteria. The last plate placed on the stack (insertion) is the first plate off the stack (deletion). A stack is sometimes called a Last-In, First-Out or LIFO data structure. Stacks have many uses in

computing. They are used in solving such diverse problems as "evaluating an expression" to "traversing a maze."

Basic Operations on Stack

To make manipulations in a stack, there are certain operations provided to us.

- **push()** to insert an element into the stack
  Adds an item to the stack. If the stack is full, then it is said to be an Overflow condition.
- **pop()** to remove an element from the stack
  Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- **top()** Returns the top element of the stack.
  Returns the top element of the stack.
- **isEmpty()** returns true if stack is empty else false
  Returns true if the stack is empty, else false.
- **size()** returns the size of stack.

Types of Stacks:

Fixed Size Stack: As the name suggests, a fixed size stack has a fixed size and cannot grow or shrink dynamically. If the stack is full and an attempt is made to add an element to it, an overflow error occurs. If the stack is empty and an attempt is made to remove an element from it, an underflow error occurs.

Dynamic Size Stack: A dynamic size stack can grow or shrink dynamically. When the stack is full, it automatically increases its size to accommodate the new element, and when the stack is empty, it decreases its size. This type of stack is implemented using a linked list, as it allows for easy resizing of the stack.

In addition to these two main types, there are several other variations of Stacks, including:

- **Infix to Postfix Stack:** This type of stack is used to convert infix expressions to postfix expressions.
- **Expression Evaluation Stack:** This type of stack is used to evaluate postfix expressions.
- **Recursion Stack:** This type of stack is used to keep track of function calls in a computer program and to return control to the correct function when a function returns.
- **Memory Management Stack:** This type of stack is used to store the values of the program counter and the values of the registers in a computer program, allowing the program to return to the previous state when a function returns.
- **Balanced Parenthesis Stack:** This type of stack is used to check the balance of parentheses in an expression.
- Undo-Redo Stack: This type of stack is used in computer programs to allow users to undo and redo actions.

Pre-Lab Task

Task 1: Writing to a File

For the following menu write a C++ program that will ask user for a choice and perform the specific functions for a constant valued stack.

```
                        M     E     N     U
Press 1: PUSH Operation
Press 2: POP Operation
Press 3: IsEmpty() function call
Press 4: IsFull() function call
Press 5: For Display (As our stack is implemented using Arrays, so we
can simply display the contents of the stack using the TOP pointer)
Press 0: Exit
```

In-Lab Tasks

For all in-lab tasks you are required to

1) Create function of all the algorithms and place them in single header file
2) Use proper "prompts" to tell the user what the program is doing, like printing the elements of array before asking for the value to be searched.
3) Assume the arrays are sorted, especially required for binary and interpolation search.
4) Try to keep your code neat and clean, don't use too many variables when work can be done with lesser number of variables.

Lab Task 1: Write a C++ program that check for balanced parentheses in an expression

Consider an input expression [{ ((( ))) }], using stack programming, write a code to find out whether the input expression is balanced or not.

Lab Task 2: Write a C++ program that converts an infix notation to postfix notation using stack programming

Proposed input: "a+b*(c^d-e)^(f+g*h)-i"

Output: abcd^e-fgh*+^*+i-

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date:_____