Data Structures and Algorithms — Lab 4

## **Topic**

To Understand the concept of Stack (Last In, First Out) through some basic applications.

## **Objectives**

* Understand the purpose of Stack and how it follows the Last in, First Out Approach.
* Explore the use of Push and Pop Operations.
* Practice the implementation of expression evaluations.
* Develop and manipulate Stack data structures with advanced functionalities.
* Gain familiarity with practical applications of Stack.

## **Outcomes**

1. Develop a comprehensive understanding of creating and using Stack to enforce specific behaviors.
2. Gain experience implementing Push and Pop Operations to define the essential working of the stack with the help of LIFO.
3. Build practical skills in implementing a Stack data structure with advanced functionality, such as custom insertion and inserting elements at the Top.
4. Enhance problem-solving skills by designing functions that meet specific operational requirements, such as Expressions Evaluation and Stack Reverse.

## **Content**

The following sections will be covered during this lab session:

1. **Implementation of Basic Stack Operations**
   * **Push**: Add an element to the top of the stack.
   * **Pop**: Remove the element from the top of the stack.
   * **Top**: Retrieve the top element without removing it.
   * **Empty**: Check if the stack is empty.
   * **Full: Check if the stack is full or not.**
2. **Advanced Stack Manipulation**
   * Extend Stack operations with specialized scenarios:
     + **Stack reverse**
     + **Brackets matching**

## **Task Instructions**

Students are required to complete the following tasks during lab time. Create a private repository on your GitHub accounts. The name of the repository should be **Lab-4-DSA**. All files should be uploaded to this repository, including the header and cpp files. Please note that the name of class-related files should be the same as the class name. If there is only 1 task, we can name the main file Task\_1.cpp or Task.cpp. We recommend you work in .h files for classes only since we must work on templates.

## **Task 1**

Create a generic class AbstractStack with the following attributes:

public:

virtual void push(T value) = 0;

virtual T pop() = 0;

virtual T top() const = 0;

virtual bool isEmpty() const = 0;

virtual bool isFull() const = 0;

virtual ~AbstractStack() {} // Virtual destructor

Create a child class myStack which implements the following functions of the parent class:

* + **push(T value)**: Adds an element to the top of the stack.
  + **T pop()**: Removes the top element and returns it.
  + **T top()**: Returns the top element.
  + **isEmpty()**: Checks if the stack is empty.
  + **isFull()**: Check if the stack is full.

More functions to implement:

* + **display():** To display the values present in the stack – this is an exclusive function of the class myStack.
  + Main Function: Provides a menu for users to perform stack operations interactively.

## **Task 2**

You are developing a simple library management system. In this system, you maintain a stack of book titles that represent the order in which they are stacked on a shelf. You want to implement a feature to reverse the order of the books in the stack. For example, if the stack has the books "Book A", "Book B", and "Book C" (with "Book C" on top), after reversing, the order should be "Book C", "Book B", and "Book A".

* Use the class AbstractStack for stack operations and the stack class myStack that performs basic operations. You do not need to re-implement these functions.
* **Implement the Reverse Function:** Write a function that reverses the contents of the stack using another stack.
  + This function must be created in your main file (above the main function). It must take a stack as input and return another stack.
  + Create another function with return type void, which takes two parameters stacks as input. One of the parameters must be the original stack; the second parameter must store the reversed stack (use pass-by-reference to ensure that the values are updated)

**User Interaction:** Provide a menu to allow users to perform operations such as pushing books onto the stack, reversing the stack, and displaying the stack contents.

## **Task 3**

You are developing a simple text editor feature that checks for balanced brackets in user-input expressions. The expression can contain various types of brackets: parentheses (), square brackets [], and curly braces {}. The goal is to determine if every opening bracket has a corresponding closing bracket in the correct order.

**Detailed Steps to Implement the Solution**

1. Use the class AbstractStack for stack operations and the stack class myStack, which performs basic operations. You do not need to re-implement these functions.
2. **Implement the Bracket Matching Function:** Write a function that checks if the brackets in the expression are balanced. This function must also be created in the main file (and not in AbstractStack or myStack)
3. **User Interaction:** Provide a menu to allow users to input expressions and check for balanced brackets.