Lab 5: Stack Using Link List and Array

CLO:
01,02

Stack:
Suppose that you have a program with several functions. To be specific, suppose that you have the functions A, B, C, and D in your program. Now suppose that function A calls function B, function B calls function C, and function C calls function D. When function D terminates, control goes back to function C; when function C terminates, control goes back to function B; and when function B terminates, control goes back to function A. During program execution, how do you think the computer keeps track of the function calls? What about recursive functions? How does the computer keep track of the recursive calls? This section discusses the data structure called the stack, which the computer uses to implement function calls. You can also use stacks to convert recursive algorithms into non recursive algorithms, especially recursive algorithms that are not tail recursive. Stacks have numerous other applications in computer science. After developing the tools necessary to implement a stack, we will examine some applications of stacks.

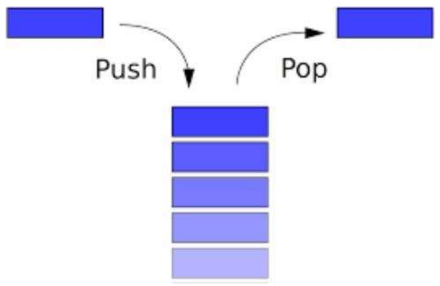
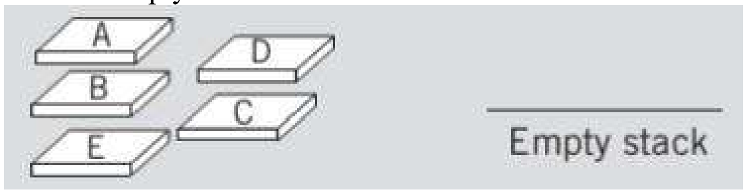


Figure 1. Stack data structure(LIFO)

A stack is a list of homogenous elements in which the addition and deletion of elements occurs only at one end, called the top of the stack. The elements at the bottom of the stack have been in the stack the longest. The top element of the stack is the last element added to the stack. Because the elements are added and removed from one end (that is, the top), it follows that the item that is added last will be removed first. For this reason, a stack is also called a **Last In First Out (LIFO) data structure.**
Push () and Pop() working: The push, top, and pop operations work as follows: Suppose there are boxes lying on the floor that need to be stacked on a table. Initially, all of the boxes are on the floor and the stack is empty.



Different push and pop will make this stack looks as:

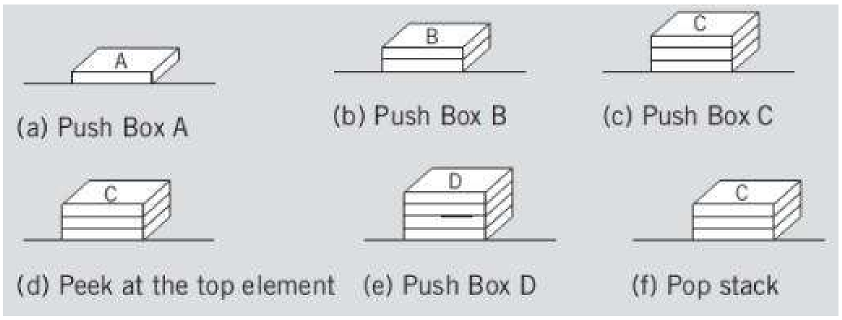


Figure 2. Push() and Pop() working

LAB TASKS

1. Implement following functions for **array**-based stack:

    a. Function **Push** to add element to stack
    b. Function **Pop** to delete element from stack
    c. Function **isEmpty** returns true if stack is empty otherwise return false.
    d. Function **isFull** returns true if stack is empty otherwise return false.
    e. Function **Size** returns current size of stack
    f. Function **Display** to print all data of stack

2. Implement following functions for **Link list**-based stack:

    g. Function **Push** to add element to stack
    h. Function **Pop** to delete element from stack
    i. Function **isEmpty** returns true if stack is empty otherwise return false.
    j. Function **isFull** returns true if stack is empty otherwise return false.
    k. Function **Size** returns current size of stack
    l. Function **Display** to print all data of stack

3. Take a sequence of braces from user entered through the keyboard and tell the user whether he has entered the correct sequence or not. Write following Function bool **IsBalanced**(Stack st, string str)

Sample Input	Sample Output
{}{}{{})	**Error:** Wrong Sequence
{{{(})}}	**Error:** Wrong Sequence
{{{()}}}{}(()){}	**Accepted:** Sequence is right