Program 1.1: Implementation of Stack using Array

Theory:

• A stack is a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle.
• Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed.
• Inserting an item is known as “pushing” onto the stack. “Popping” off the stack is synonymous with removing an item

A stack is an abstract data type(ADT) that supports
two main methods:-

- push(o)

Inserts object o onto top of stack

- pop()

Removes the top object of stack and returns it; if stack is empty an error occurs

• The following support methods should also be defined:

- size()

Returns the number of objects in stack

- isEmpty()

Return a boolean indicating if stack is empty.

- isFull()

Return a Boolean indicating if stack is full.

- peek():

return the top object of the stack, without removing it; if the stack is empty an error occurs.

- display()

display the elements of stack from top of the stack.

Program 1.2: Implementation of two stacks in an array

Theory:

Implementation of *two Stacks* should use only one array, i.e., both stacks should use the same array for storing elements. Following functions must be supported.

push1(int x) –> pushes x to first stack
push2(int x) –> pushes x to second stack

pop1() –> pops an element from first stack and return the popped element
pop2() –> pops an element from second stack and return the popped element

Method 1: Divide the space into two equal halves

A simple way to implement two stacks is to divide the array in two halves and assign the half half space to two stacks, i.e., use arr[0] to arr[n/2] for stack 1, and arr[n/2+1] to arr[n-1] for stack 2 where arr[] is the array to be used to implement two stacks and size of array be n.

The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the array size is 6 and we push 3 elements to stack1 and do not push anything to second stack2. When we push 4th element to stack1, there will be overflow even if we have space for 3 more elements in array.

Method 2: (A space efficient implementation)

This method efficiently utilizes the available space. It doesn't cause an overflow if there is space available in arr[]. The idea is to start two stacks from two extreme corners of arr[]. stack 1 starts from the leftmost element, the first element in stack1 is pushed at index 0. The stack 2 starts from the rightmost corner, the first element in stack 2 is pushed at index (n-1). Both stacks grow (or shrink) in opposite direction.

Post Lab Questions:

1) What do you mean by a pointer pointing to structure?
2) Write syntax for creating an array of 20 float values in a static and dynamic manner