**Question**

A stack is a linear data structure which enables the user to add and remove integers from one end only, using the concept of LIFO(Last In First Out). An array containing the marks of 50 students in ascending order is to be pushed into the stack.  
Define a class Array\_to\_Stack with the following details:   
Class name: Array to Stack  
Data members/instance variables:  
m[]: to store the marks  
st[ ]: to store the stack elements  
cap: maximum capacity of the array and stack  
top: to point the index of the topmost element of the stack  
Methods/Member functions:  
Array\_to\_Stack(int n): parameterized constructor to initialize cap = n and top = -1  
void input\_marks(): to input the marks from the user and store it in the array m[ ] in ascending order and simultaneously push the marks into the stack st[ ] by invoking the function pushmarks()  
void pushmarks(int v): to push the marks into the stack at top location if possible, otherwise, display “not possible”  
intpopmarks(): to return marks from the stack if possible, otherwise, return-999  
void display(): To display the stack elements

Specify the class Array\_to\_Stack, giving the details of the constructor(int), void input\_marks(), void pushmarks(int), int popmarks() and void display().

**Algorithm**

1. Start

2. Define a class `Array\_to\_stack` with the following instance variables:

- `m` of type `int[]` to store the marks.

- `st` of type `int[]` to store the stack elements.

- `cap` of type `int` to store the maximum capacity of the array and stack.

- `top` of type `int` to point to the index of the top-most element of the stack.

3. Define a parameterized constructor for the class `Array\_to\_stack`:

- Accept an integer `n` as a parameter.

- Initialize `cap` with `n`.

- Initialize `top` with `-1`.

- Initialize `m` with a new array of size `cap`.

- Initialize `st` with a new array of size `cap`.

4. Define a method `input\_marks()` to read marks from the user and push them onto the stack:

- Create a `Scanner` object to read input from the user.

- Prompt the user to enter the marks.

- For each mark:

- Read the mark and store it in the array `m`.

- Call the `push\_marks()` method with the read mark as the argument.

- Close the `Scanner` object.

5. Define a method `push\_marks(int v)` to push a mark onto the stack:

- If `top` is equal to `cap - 1`, print "Not Possible" indicating the stack is full.

- Otherwise:

- Increment `top` by 1.

- Assign the value `v` to `st[top]`.

6. Define a method `pop\_marks()` to pop a mark from the stack:

- If `top` is equal to `0`, print "-999" indicating the stack is empty.

- Otherwise:

- Store the value at `st[top]` in a variable `b`.

- Decrement `top` by 1.

- Print the popped value `b`.

7. Define a method `display()` to display the stack elements:

- Print the stack elements from index `0` to `top`.

8. In the main method:

- Create a `Scanner` object to read input from the user.

- Prompt the user to enter the length of the array.

- Read the length of the array into an integer variable `array\_length`.

- Create an object of the class `Array\_to\_stack` with `array\_length`.

- Call the `input\_marks()` method on the object to read and push the marks onto the stack.

- Call the `pop\_marks()` method on the object to pop and display the top mark from the stack.

- Call the `display()` method on the object to display the remaining stack elements.

- Close the `Scanner` object.

9. End

**Variable Description**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Purpose** |
| m | int[] | To store marks entered by user |
| st | int[] | To store marks using stack |
| cap | int | Maximum capacity of stack and array |
| top | int | Index of topmost element |
| n | int | Initialize capacity of array and stack |
| v | int | Marks to be pushed into stack |
| i | int | Loop counter for the array |
| b | int | To store value popped from the stack |
| array\_length | int | To store length of the array |