Offline 2: Extended Stack

Introduction

Stack is a linear data structure that maintains LIFO (Last In First Out) ordering. So, while removing an element the one inserted last is removed at first. A stack can be implemented using an array or a linked list. However, for this problem, you have to follow an object oriented approach for its implementation.

A stack can have different functionalities like inserting an element, removing the most recently inserted element, reporting number of elements in the stack and so on. But in the object orient approach, you have to implement all these traditional and some special functionalities (details in next section) by calling appropriate functions of your defined class.

Problem Specification & Instructions

You have to design & implement a class named stack having the following operations. Specification of these operations & what you need to do are also described in brief.

- **Initialization:** Write suitable constructor(s) so that your stack can be initialized & memory can be dynamically allocated. How much memory you should allocate may or may not be specified in terms of number of elements. Besides, you may need to write a special constructor so that the memory which is dynamically allocated in an instance of your class (i.e. object) does not get accidentally erased in another resulting in runtime error/garbage value. Print appropriate messages in each of these constructors so that they can be easily distinguished (in case you need multiple ones).

- **Termination:** Write suitable destructor(s) in which you should free dynamically allocated memory. Print appropriate messages in each of these destructors so that they can be easily distinguished (in case you need multiple ones).

- **Push:** You should be able to insert an integer or an array of integers or another stack object into your stack. The example below illustrate each of these.

Current elements in Stack	Push	Elements in Stack after Push
2, 4	10	2, 4, 10
2, 4	Array, {8, 10}	2, 4, 8, 10
2, 4	Stack with elements: 5, 6	2, 4, 6, 5

  Write suitable functions to achieve this. Note that, name of the functions must be the same for all these push operations.

- **Pop:** Removes the most recently inserted element from stack. Note that, this function will always return an integer.
- **Top:** Reports the most recently inserted element without removing it.

- **Size:** Reports number of elements currently in the stack.

- **Resize:** Reallocate memory dynamically for the storage of elements in the stack. Note that, the stack must never have memory allocated for 10 more elements than it actually has. So, if there are *x* elements in the stack, already allocated memory cannot exceed memory required by *x*+10 elements. For example, say your stack has 9 integers and memory allocated for 10. Now if you push two integers, during the second push, you have to expand available memory so that a total of 20 integers can be inserted. But the already inserted elements must be retained. Similarly, if you pop twice next, there will be additionally allocated memory for 11 more integers. So, during the second pop, you have to reallocate memory again shrinking the available space for your stack. You can use two different definitions (not mandatory though) of this function for expanding and shrinking available memory.

- **Similarity:** Write a function to check similarity between two stacks. You can calculate similarity by checking how many elements of a stack matches with corresponding ones of the other. A similarity score can be obtained by dividing it by the average size of the two stacks. Please see the following example for clarification.

Stack - 1 Elements	Stack - 2 Elements	# of Elements Matched	Average Size of Two Stacks	Similarity Score
2,4,6 [push 2, push 4, push 6]	3,5,6	1	3	1/3 = 0.3333
2,4,6	6	1	2	1/2 = 0.5
2,4,6	4,2	1	2.5	1/2.5 = 0.4
2,4,6	4,6,2	0	3	0/3 = 0

* You can use as many member variables and additional member functions as required for your implementation. But all your member variables must be private.

* There must be no method to access the elements of stack randomly from outside. You must use push & pop/top for this purpose.

* All your functions (except main function) must be member functions of the stack class.

In the main function, you will create an instance of stack and push some integers into it, let us call it mainStack. You will also maintain a pointer of stack type, let us call the pointer tempStack (how you have to use it is mentioned in the next section).

Input Output Specification

You have to prompt for input in a loop in the main function. Show a menu like the one shown below. Note that all operations will be done on the mainStack.

1: Push an element
2: Push an array
5: Top
6: Size

3: Push a stack
4: Pop
7: Similarity
8: Exit

Based on the response, take necessary inputs next. For example, if the user chooses 2, take n+1 integers as input where n denotes the size of array followed by n elements of the array. Similarly if the user chooses 3 or 7, take n+1 integers as input where n denotes the size of stack followed by n elements of the stack. For this, allocate memory dynamically for tempStack using suitable constructor and **push** the elements sequentially into it. When you are done with the operation, free this allocated memory.

[Note that, **Resize** was not mentioned in the menu. So you need not call it from outside. Try to set its access specifier accordingly.]

Finally, when the user chooses 8, pop all the elements of the mainStack one by one & print them.

Marks Distribution:

Task	Initializa-tion	Termination	Push	Pop	Top	Size	Re-size	Similarity	Total
Marks	5	2	5	1	1	1	2	3	20

Special Instructions:

- ***You must not copy anyone's code. If copy is found, you will be penalized severely resulting in a high risk of failing in the course. In this case, both students will face the same penalty, no matter who actually did it & who copied.***

- ***You must not copy from any source from internet. If you do so, you will face similar penalty as the one mentioned previously.***

- You can check the " Practice_A2(Array_of_Object) File" program on moodle. It may help with your implementation.

- A thread titled "Offline 2 Queries" will be opened in the news forum of CSE108 on moodle. Please post your queries there.

Submission Guidelines:

You will submit a single cpp file. Name it as follows: 1705xyz.cpp. Here, xyz are the last three digits of your student ID.

Deadline:

Friday, 16 November, 11:55PM.