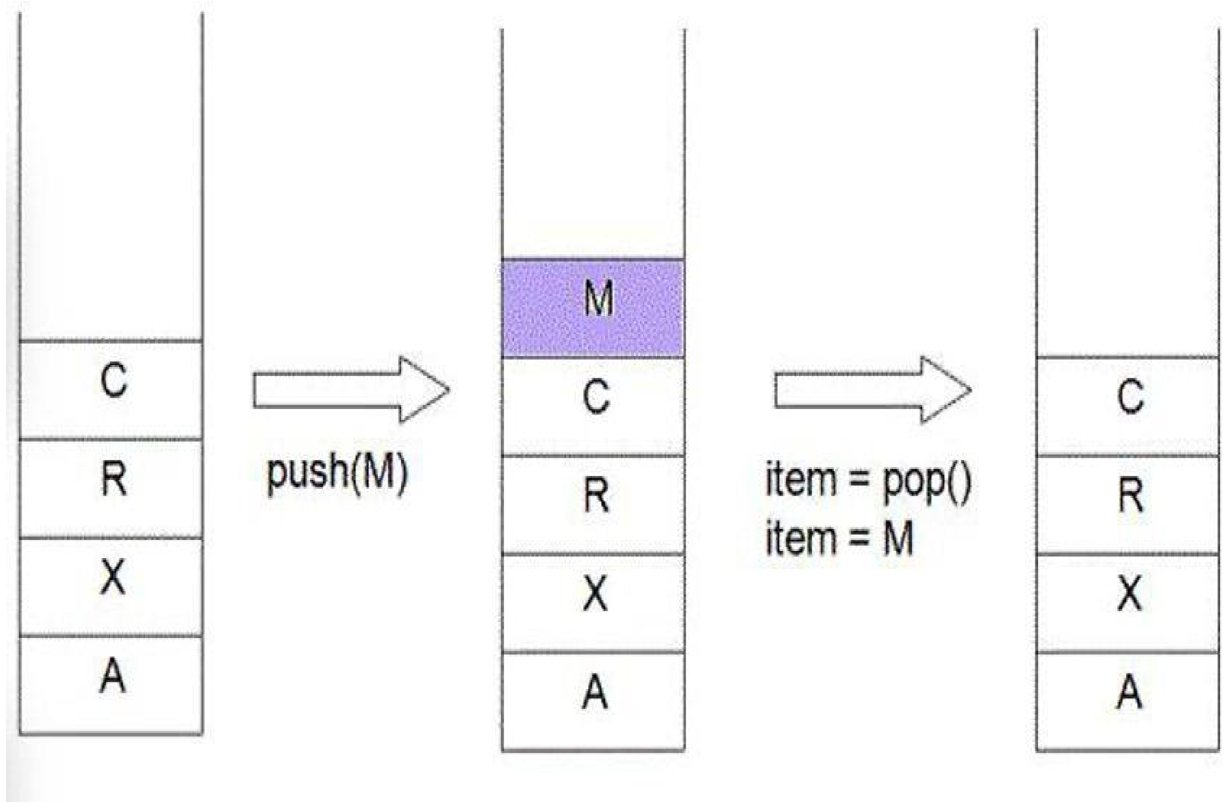
Assignment 2: Stack

Task 1: Stack Data structure

A stack is a container associated with objects which are inserted or removed based on the last-in first-out (LIFO) principle.



In this assignment, you need to implement a Stack ADT (abstract data type) as defined in Table 1. You need to implement it using the C or C++ programming languages.

Table 1: Stack ADT definition.

Fn #	Function	Param.	Ret.	After functions execution	Comment
	[before each function execution.]			<20, 23 , 12, 15>	The values were pushed in the following order: 20, 23, 12, 15. So, TOP points to the value 15. Note that this is a logical definition without any implementation related specification.
1	clear()	-		<>	Reinitialize the stack, i.e., make it (logically) empty stack. <> means an empty stack.
2	push(item)	19		<20, 23 , 12, 15, 19>	Pushes an element.
3	pop()		15	<20, 23, 12>	Pop the top element.
4	length()	-	4	<20, 23, 12, 15>	Return the number of elements in the list.
5	topValue()	-	15	<20, 23, 12, 15>	Return the top element.
6	isEmpty()	-	False	<20, 23, 12, 15>	If we have an empty stack, then this function will return true; otherwise, it will return false.

The implementation must support at least **two data types(int/char)** of elements. You can implement the int and char data types separately or use templates in C++. If you implement using **templates** in C++, you will get **bonus marks**.

You need to provide **Array Based (Arr) implementation**. The size of the stack is only limited by the memory of the computing system, there must be a way to **dynamically grow the stack size as follows**: the stack should double its current size by **allocating memory dynamically**, i.e., initially the list should be able to hold X elements; as soon as the attempt is made to insert the X+1th element, memory should be allocated such that it can hold 2X elements, and so on. **Static implementation would result in the deduction of marks.**

You may implement extra helper functions, but those will not be available for programmers to use. So, while using the stack implementations, one can only use the methods listed in Table 1. Please follow the following input/output format.

Input format (for checking the implementation):
First, you will ask the user what type of stack(int/char/..) he would like to construct.
Then, one line containing the memory chunk size and the initial length of the list, space separated: X K (K < X).
One line containing K elements, space separated, to be initialized the stack with K elements.
Several lines (indicating the tasks to perform on the stack), each containing one or two integers, Q (Fn #), 0<=Q<=6, and P (parameter) (Only for push function). For each line, the program will execute Fn # Q. If Q is 0, the program will exit.

Output Format:
At first, the system will output in one line the items of the list within the angle bracket, space separated, as shown in Table 1. Then, for each task, it will output two lines as follows:

The first line will output the items of the list, space separated as shown in Table 1, and the second line will output the return value (if available).

- Please output the necessary messages in the case of any corner scenario.

Task 2: Using the Stack (Expression Evaluation)

You are given an input arithmetic expression that contains the following operators "+, -, *, /" and operands. It has the parenthesis '()' as well. Please note that "– " can be used as either a unary or binary operator. For example, both (-3), and (4-1) are valid. **Check whether the input expression is valid**. **If it is valid, evaluate the expression also**. **If it is not valid, print "Not valid'.** (For this assignment, you can **consider that only integer operands** will be given as inputs. But output can be integer or float.). You have to consider the **precedence of arithmetic operators**. See this link

You need to implement the above-described task using your stack implementation in task 1. You need to write a program that takes input (see sample I/O below) and is able to check whether the input expression is valid and, if it is valid, evaluate the expression.

Sample Input	Sample Output	Comment
(9*3-(7*8+((-4)/2)))	Valid expression. Computed value: -27	
(9*3-(7*8+((4/2)))	Not valid	Because there is a missing parenthesis. // **You don't need to output the reason.**
6/3*2	Valid expression. Computed value: 4	
2+4/5*10/3	Valid expression. Computed value: 4.67	
2++4	Not valid	

Special Instructions:

Write **readable, re-usable, well-structured, quality** code. This includes but is not limited to writing appropriate functions for implementation of the required algorithms, meaningful naming of the variables, suitable comments where required, **proper indentation**, etc.

Please **DO NOT COPY** solutions from anywhere (your friends, seniors, the internet, etc.). Any form of plagiarism (irrespective of source or destination) will result in getting **-100% marks** in offline.

Submission Guideline:

1. Create a directory with your 7-digit student id as its name.
2. Put the source files only into the directory created in step 1.
3. Zip the directory (compress in **.zip** format; .rar, .7z or any other format is not acceptable)
4. Upload the .zip file on Moodle.

For example, if your student id is 2105xxx, create a directory named 2105xxx. Put only your source files (.c, .cpp, .h, etc.) into 2105xxx. Compress 2105xxx into 2105xxx.zip and upload the 2105xxx.zip on Moodle. Failure to follow the above-mentioned submission guidelines may result in upto 10% penalty.

Submission Deadline: June 24, 2023, 11:55 PM.

Evaluation Policy:

Task	Implementation	
1	int data type	25%
	char data type	25%
2	Checking the validity of the expression	25%
	Evaluate the expression if it is valid	25%
Bonus	Use templates to implement Stack in task 1	10%