CSE 4304-Data Structures Lab. Winter 23-24
Batch: CSE 22
Date: Sept 11, 2024
Target Group: All
Topic: Stacks

Instructions:

- Regardless of how you finish the lab tasks, you must submit the solutions in Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab took place.
- Task naming format: fullID_T01L01_2A.c/cpp
- If you find any issues in the problem description/test cases, comment in the Google Classroom.
- If you find any tricky test cases that I didn't include but that others might forget to handle, please comment! I'll be happy to add them.
- Use appropriate comments in your code. This will help you to recall the solution in the future easily.
- Obtained marks will vary based on the efficiency of the solution.
- Do not use <bits/stdc++.h> library.
- Modified sections will be marked with BLUE color.
- You are allowed to use the STL stack unless it's specifically mentioned to use manual functions.

Group	**Tasks**
2A	1 2 3 7 [solve Task4 for bonus mark]
1B	
1A	
2B	
Assignments (all)	All tasks except the ones you did in your respective labs. Plagiarism will be checked. Pls be careful. Deadline: Sunday 11.59 PM of that week

- Two ways to take inputs if the number of input elements is not specified:
  - Taking inputs until End-of-File (EOF) is reached
    (Press **Ctrl+Z** after typing in your inputs. You should see something like **^Z** for Windows.)

```
int n;
vector<int>a;
while(cin >> n)
{
    a.push_back(n);
}
```

- Taking inputs until “Enter” button is pressed

```
char c;
int n;
vector<int>a;
while (true) {
    c = cin.peek(); // Peek at the next character

    if (c == '\n') {
        // Exit loop when Enter is pressed
        break;
    }
    cin >> n;
    cin.get();  // Consume the character from the stream
    a.push_back(n);
}
```

Task 1: Implementing the basic operations of Stack

Stacks is a linear data structure that follows the Last In First Out (LIFO) principle. The last item to be inserted is the first one to be deleted. For example, you have a stack of trays on a table. The tray at the top of the stack is the first item to be moved if you require a tray from that stack.

The Insertion and Deletion of an element from the stack slightly differ from the traditional operation. We define the corresponding operations as Push() and Pop() from the stack.

The first line contains *N* representing the size of the stack. The lines contain the 'function IDs' and the required parameter (if applicable). Function ID 1, 2, 3, 4, 5, and 6 corresponds to push(), pop,() isEmpty(), isFull() (assume the max size of Stack=5), size(), and top(). The return type of isEmpty() and isFull() is Boolean. Stop taking input once given -1.

After each push/pop operation, print the status of the stack.

Input	**Output**	**Explanation**
5		
3	True	Stack is empty
2	Underflow	Empty stack, cant pop
1 10	10	Pushed 10
1 20	10 20	Pushed 20
5	2	Stack has 2 items
1 30	10 20 30	Pushed 30
6	30	Top of stack has 30
2	10 20	Popped item
1 40	10 20 40	Pushed 40
1 50	10 20 40 50	Pushed 50
4	False	The stack isn't full yet
1 60	10 20 40 50 60	Pushed 60
4	True	Stack full
5	5	Stach has 5 items
1 60	Overflow	Stack full, can't push
5	5	Stach has 5 items
2	10 20 40 50	Item popped
6	50	Top of stack has 50
-1		Break

Note:
You have to **implement** the stack operation functions **by yourself** for this task. Do **not** use the STL stack here.

Task 1.1 (assignment): C++ offers a header file called **<stack>** which has different stack operations implemented as library functions. Follow this (https://www.geeksforgeeks.org/stack-in-cpp-stl/) and try to understand the usage of each function. Finally, implement **Task-1** using those STL library functions (assignment).

Task 2: Parsing HTML code with Stacks

HTML stands for Hyper Text Markup Language. HTML is the standard markup language for creating Web pages. It describes the structure of a Web page using different **‘tags’**. Each tag has two portions - ‘opening’ and ‘closing’ with the notations‘ <tagName>’ and ‘</tagName>, respectively.

Given a snippet of HTML code, your task is to check whether the tags are properly nested or not.
Cases:

- ‘No error’: If all the tags are properly given.
- ‘Error at line ##’: If any error is detected.

Input	**Output**	**Comment**
8 <html> <head> <title> title of webpage </title> </head> <body> This is a paragraph </body> </html> -1	No error	After code traversal, the stack was empty
8 <html> <head> <title> title of webpage </head> </title> <body> This is a paragraph </body> </html> -1	Error at Line 3	The closing tag </head> was found, but the top of the stack doesn't contain the corresponding opening tag
7 <html> <head> <title> </title> </head> <body> An unordered HTML list This is a paragraph -1	Error at line 7	Code traversal is finished, but the stack isn’t empty. Interesting !!
9 <html> <head> <title> title of webpage </head> </title> <body>	Error at line 9	Closing tag </html> found, but the stack is empty

This is a paragraph </body> </html> </html> -1		
9 <html> <head> <title> </title> </head> <body> An unordered HTML list This is a paragraph </body> </html> -1	Error at line 6	The closing tag </h3> was found, but the top of the stack doesn't contain the corresponding opening tag
9 <html> <head> <title> </title> </head> <body> An unordered HTML list This is a paragraph </body> </html> -1	Error at line 7	The closing tag was found, but the top of the stack doesn't contain the corresponding opening tag
8 <html> <head> <title> </title> </head> An unordered HTML list This is a paragraph </body> </html> -1	Error at line 7	The closing tag </body> was found, but the top of the stack doesn’t match

Note:

- You just have to return the occurrence of the first error detected by your solution.
- For the last test case, there are some opening tags, for which no closing tags were found, although the entire code was checked. So we assume that there is an error in the last line.
- For simplicity, we’ve omitted the tags like
 for which no closing tags are defined.
- **You can use STL <stack> for this task.**

Task 3: Build an Array With Stack Operations

Suppose you are given a target array and an integer n. In each iteration, assume that you have n numbers available in a list (1,2,3,... n).

Your task is to build the target array using the following operations:

- "Push": Reads a new element from the beginning list and pushes it into the array.
- "Pop": Deletes the last element of the array.
- If the target array is already built, stop reading more elements.

Return a list of the operations needed to build the target array. If there are multiple valid answers, return any of them.

Input	**Output**	**Explanation**
1 3 3	Push Push Pop Push	Read number 1 and automatically push in the array -> [1] Read number 2 and automatically push in the array, then Pop it since 2 is not part of the target array -> [1] Read number 3 and automatically push in the array -> [1,3]
1 2 3 3	Push Push Push	Push 1, Push 2, Push 3
1 2 2	Push Push	Push 1, Push 2. No further operation is required.
2 3 4 4	Push Pop Push Push Push	Push 1, Pop 1, Push 2, Push 3, Push 4
1 4 4	Push Push Push Pop Pop Push **Alternative**: Push Push Pop Push Pop Push	Push 1, Push 2, Push 3, Pop 2, Pop 3, Push 4 Push 1, Push 2, Pop, Push 3, Pop, Push 4
1 3 4 6 6	Push Push Pop Push Push Push Pop Push	Push 1, Push 2, Pop, Push 3, Push 4, Push 5, Pop, Push 6

Note: You can use STL <stack> for this task.

Task-4: Next Greater Element
Given a set of numbers, your task is to print the ***Next Greater Element (NGE)*** for every element. The NGE for an element ***x*** is the first greater element on the right side of ***x*** in the array. Element for which no NGE exists, consider the NGE-value as -1.

Examples:
- For an array, the last element always has NGE as -1.
- For an array sorted in descending order, every element has NGE as -1
- For the input array (4, 5, 2, 25) the NGE for each element is (5, 25, 25, -1).
- For the input array (13, 7, 6, 12) the NGE for each element is (-1, 12, 12, -1}

We can solve this using two simple nested loops! The outer loop picks the elements one by one and the inner loop looks for the first greater element for element picked by the outer loop. If a greater element is found then that element is printed as the NGE, otherwise, -1 is printed.
But this approach has the worst-case Time complexity is $O(n^2)$. (worst case occurs when the elements are in descending order. It will lead us to look for all the elements.)

We can improve the time complexity to **O(n)** using stacks! Your task is to propose an algorithm to find the NGE of every element using stack in O(n) time.

Input:
Each test case can consist of a different number of integers. Every test case will end with a -1 indicating the end of a particular test case (-1 will not be considered as part of the input.)

Output:
Find the NGE of every element using a stack with O(n) worst-case time complexity.

Input	**Output**
4 5 2 25 -1	5 25 25 -1
13 7 6 12 -1	-1 12 12 -1
11 13 21 3 20 -1	13 21 -1 20 -1
12 17 1 5 0 2 2 7 18 25 20 12 5 1 2 -1	17 18 5 7 2 7 7 18 25 -1 -1 -1 -1 2 -1
10 20 30 40 50 -1	20 30 40 50 -1
50 40 30 20 10 -1	-1 -1 -1 -1 -1

Task 07: Evaluating postfix notation

Write a program to take an expression represented using **postfix notation** as input and evaluate the expression using stack.

Input expression may contain addition (+), subtraction (-), multiplication (*), and division (/) operators and numbers between (0~9).

Sample Input	**Sample Output**
2 345*+6- 225+*1+5-52**	 17 100

Note: You can use STL <stack> for this task.