Assignment 2: Implementation of Stack Data Structure

January 2024, CSE 106

September 15, 2024

Task 1: Introduction to Stack

A **stack** is a linear data structure that follows the *Last In, First Out (LIFO)* principle. In other words, the last element added to the stack will be the first one to be removed. You can think of a stack like a pile of books: when you add a new book, it is placed on top, and when you remove a book, you remove the one at the top.

The fundamental operations on a stack include:

- **Push:** Adds an element to the top of the stack.
- **Pop:** Removes the top element from the stack.
- **Top:** Returns the top element without removing it.
- **Length:** Returns the length of the stack.
- **IsEmpty:** Checks if the stack is empty.
- **Clear:** Removes all elements from the stack.

How Stack Works

Here's a basic example of how a stack works:

Example

Assume we start with an empty stack and perform the following operations:

- **Push 5:** The stack now contains [5].
- **Push 10:** The stack now contains [5, 10].
- **Pop:** Removes the top element (10), leaving [5].
- **Push 3:** The stack now contains [5, 3].
- **Length:** Returns (2) because the stack contains 2 elements.
- **Top:** Returns the top element (3) without removing it.
- **IsEmpty:** Returns false because the stack is not empty.
- **Clear:** Empties the stack.

Visual Representation

Operation	Stack Contents	Returns
Push 5	[5]	
Push 10	[5, 10]	
Pop	[5]	10
Push 3	[5, 3]	
Length	[5, 3]	2
Top	[5, 3]	3
IsEmpty	[5,3]	False
Clear	[]	

Your Task

You are required to implement stack data type for integers in C++. You have to provide two implementations. One is using Array List and another is using Linked List. Your implementation should provide the following functions:

1. **push(x):** Push the element `x` onto the stack.
2. **pop():** Remove and return the top element of the stack.
3. **top():** Return the top element without removing it.
4. **length():** Returns the number of elements in the stack.
5. **isEmpty():** Returns `True` if the stack is empty; otherwise, returns `False`.
6. **clear():** Remove all elements from the stack.

Input Format

The input will be taken from a file, where each line contains an operation followed by a value (if applicable).

- **1 x**: Push the element x onto the stack.
- **2**: Pop the top element from the stack.
- **3**: Return the top element without removing it.
- **4**: Return the number of elements in the stack.
- **5**: Return `True` if the stack is empty; otherwise, return `False`.
- **6**: Clear all elements from the stack.

Sample Input

```
1 10
1 20
1 30
1 40
2
2
3
4
5
6
```

Example Output

```
Pushed 10 onto the stack.
Current stack: 10
-------------------
Pushed 20 onto the stack.
Current stack: 10 20
-------------------
Pushed 30 onto the stack.
Current stack: 10 20 30
-------------------
Pushed 40 onto the stack.
Current stack: 10 20 30 40
-------------------
Popped value: 40
Current stack: 10 20 30
-------------------
Popped value: 30
Current stack: 10 20
-------------------
Top value: 20
Current stack: 10 20
-------------------
Current stack length: 2
Current stack: 10 20
-------------------
Is stack empty? No
Current stack: 10 20
-------------------
Cleared the stack.
Current stack:
-------------------
```

Hints

- You have been given two main functions: (i) **main_array_stack.cpp** for checking array based implementations and (ii) **main_list_stack.cpp** for checking linked-list based implementations. You do not have to edit a single line of code in those files. If you want to give your own input, just change in **input.txt** and you will get the output with proper messages in **output.txt**. Initially a sample input and output are given in those files.
- You have been given two header files: (i) **StackArray.h** for array based implementations and (ii) **StackLinkedList.h** for linked-list based implementations. You have to edit only in those files at the mentioned places "`//write your codes here`".
- Please use the best practices from the 1st Offline like when to make the array capacity double or half. Most of the functionalities are same as the first offline. You have to apply the same concept except in C++ now. You are not allowed to use vector, list or any of the STL functionalities.
- When doing linked-list based implementation, first understand the theory behind it e.g how to add or remove a node from linked list etc.
- Carefully handle the pointers !!!

Submission Guidelines

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 2205xxx, create a directory named 2205xxx. Put only your source files (.cpp, .h, etc.) into 2205xxx. Compress 2205xxx into 2205xxx.zip and upload the 2205xxx.zip on Moodle. Failure to follow the above-mentioned submission guidelines may result in some penalty.
5. **Please DO NOT COPY solutions from anywhere (your friends, seniors, the internet, etc.). Any form of plagiarism (irrespective of source or destination) will be penalized severely.**

Submission Deadline

September 22, 2024, 11:55 PM

Tentative Marks Distribution

ArrayList-based Implementation	**40%**
Push	12%
Pop	12%
Top	4%
Length	4%
IsEmpty	4%
Clear	4%
LinkedList-based Implementation	**60%**
Push	15%
Pop	15%
Top	10%
Length	10%
IsEmpty	5%
Clear	5%

Version History

The document may be updated periodically to address any ambiguities, clarify instructions, or incorporate feedback from students to ensure the assignment is fully understandable and free of confusion. All changes in the document will be marked as red text with underline, along with a version change. Please keep yourself updated on the changes in the document.
Version 1.0