stack – using a singly-linked list

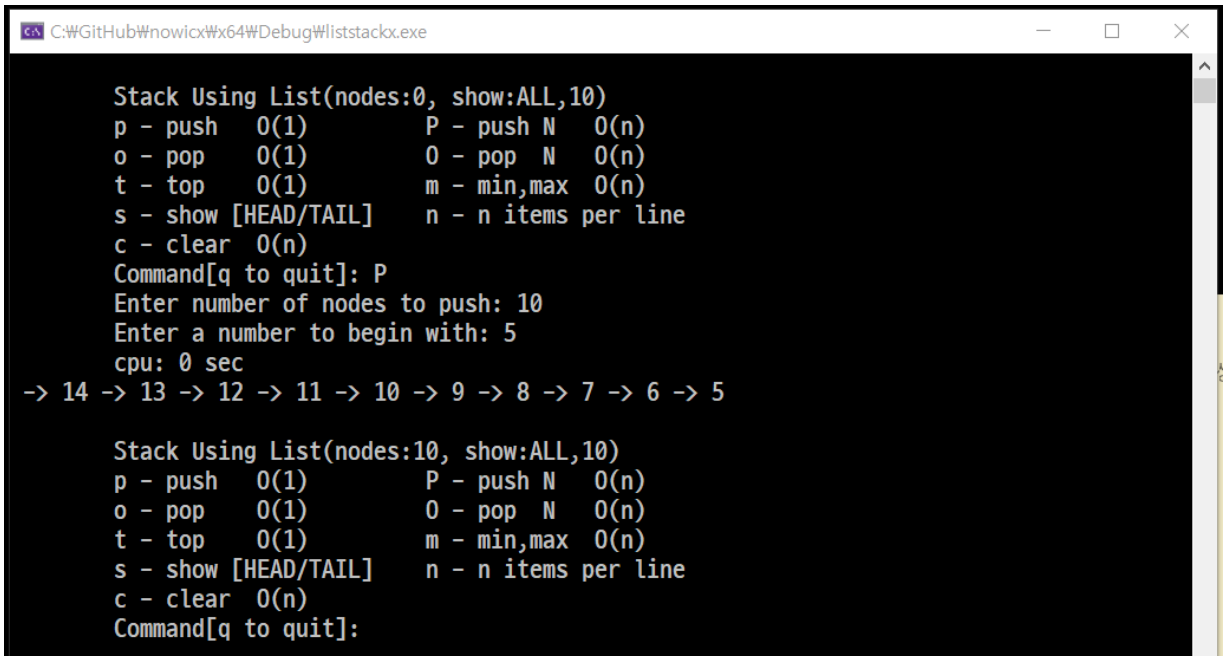
Table of Contents

Getting Started

This problem set consists of implementing a simple stack by using singly-linked list of nodes. Your job is to complete **liststack.cpp** program that implements a singly linked list that do not have a header structure nor sentinel nodes. It simply links node to node and the first node always becomes a head node which we consider as the top of the stack.

- liststack.cpp – a skeleton code, **your implementation goes here**.
  liststack.h – an interface file, you are **not** supposed to modify this file.
- liststackDriver.cpp – a driver code to test your implementation.
- liststackx.exe – a sample solution for Windows PC
  Mac Users: Use **Wine or WineBottler** to run Windows executable on your Mac.

Sample run.



Step 1: push(), pop(), and top()

First of all, you must read **liststack.h** file so that you may see the whole picture of the task.
The **push()** is supposed to insert a node or N nodes in front of the list.

```
// pushes a new node with val at the beginning of the list or
// onto the top of the stack and returns the new first node.
pNode push(pNode s, int val, int N = 1);
```

The skeleton code inserts one node (N = 1 by default). If N is not 1, N nodes are added from the starting value of `val` which are passed from the driver. For instance, if val=10 and N=5, the five nodes added must be 10, 11, 12, 13, and 14.

Since we don't have any extra header structure to maintain, the first node always acts like the head node which we consider as the top of the stack. Since the new node is inserted at the front or the top of the stack, the function must return the new pointer to the head node to the caller. The caller must reset the first node (or the top of the stack) information. This is O(1) operation.

This **pop()** is supposed to remove the top (or head) node or N nodes from the stack.

```
// removes N nodes in the list and returns the new first node.
// This deallocates the removed node, effectively reduces its size by N.
pNode pop(pNode p, int N = 1)
```

The function removes one or N nodes (N = 1 by default). After it removes one or N nodes from the list, it returns the new pointer of the list if there is at least one node. It is now the top of stack if it is not an empty list. If empty, returns nullptr, This is O(1) operation.

The **top()** function returns the top elements of the stack or head (first) node in the linked list unless the stack is empty. If empty, it returns nullptr. This is O(1) operation.

Step 2: size(), empty(), and minmax()

The **size()** function returns the number elements (or nodes) in the stack (or linked list).
The **empty()** function returns true, if the stack is empty, false otherwise.
The **minmax()** function finds the minimum and maximum values in the list. In fact, there is no necessity to have the min or max value in stack ADT. Nevertheless, we have this operation to check out the integrity of the code on purpose.

```
// sets the min and max values which are passed as references in the list
void minmax(pNode p, int& min, int& max);
```

Step 3: clear()

The clear() function deallocates all the nodes in the list. Make sure that you call "delete" N times where N is the number of nodes. One tricky part is that you must save the next pointer of the node that you are about to delete.
This function is implemented for you. It is a grace of ~ ~.

Step 4: show()

The **show()** displays the stack, or the linked list of nodes from top to bottom. The function has an optional argument called **bool show_all = true**. Through the menu option in the driver, the user can toggle between "show [ALL]" and "show [HEAD/TAIL]". This functionality is useful when you debug your code.

It has another optional argument i**nt show_n = 12**. If the size of the list is less than or equal to show_n * 2, then it displays all the items in the list. If "show [ALL]" option is chosen, it toggles to "show [HEAD/TAIL]" option and vice versa. The [HEAD/TAIL] option displays the show_n items at most from the beginning and the end of the list.

Step 5: Stress test – "P" and "O"

Do not start this step unless you finish Step 1 through Step 3, completely.

We want to implement a piece of code that tests your implementation of functions you have done so far.

- These two options ask the user to specify the number of nodes to add or remove. Then perform the operation N times repeatedly.
- If the user specifies a number out of the range in O option, it simply removes all the nodes.

Submitting your solution

- Include the following line at the top of your every source file with your name signed.
  On my honor, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.
  Signed: _____ Section: _____ Student Number: _____
- Make sure your code **compiles** and **runs** right before you submit it. Every semester, we get dozens of submissions that don't even compile. Don't make "a tiny last-minute change" and assume your code still compiles. You will not get sympathy for code that "almost" works.
- If you only manage to work out the problem sets partially before the deadline, you still need to turn it in. However, don't turn it in if it does not compile and run.
- Place your source files in the folder you and I are sharing.
- After submitting, if you realize one of your programs is flawed, you may fix it and submit again as long as it is **before the deadline**. You will have to resubmit any related files together, even if you only change one. You may submit as often as you like. **Only the last version** you submit before the deadline will be graded.

Files to submit

Submit **one** file listed below in **pset7-1** folder in Piazza.

- liststack.cpp

Due and Grade

- Due: 11:55 pm,
- Grade: 1 point per step.
- Finish Step 1 & 2 successfully, you get Step 3 as a bonus point + 1.