EECE.3220: Data Structures

Spring 2017

Programming Assignment #4: Stacks and strings

Due **Friday, 4/14/17**, 11:59:59 PM

1. Introduction

This assignment provides an introduction to working with stack objects. You will write a linked stack class, as well as a basic main program to interact with the stack.

2. Deliverables

You should submit three files for this assignment:

- ***prog4_main.cpp:*** Source file containing your main function.
- ***Stack.h:*** Header file containing the `Stack` class definition.
- ***Stack.cpp:*** Source file containing definitions of `Stack` member functions.

Submit all three files by uploading them to your Dropbox folder. Ensure your file names match the names specified above. Do not place these files in a subdirectory—place them directly in your shared folder. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

Class Design: Your `Stack` class should be implemented as a linked stack holding values of type *stackElement*. For the purposes of this assignment, you can simply replace *stackElement* with type `string`. Alternatively, you may earn extra credit by designing `Stack` as a class template (see page 2).

The class should contain, at a minimum, the following functions:

- `Stack()`: Default constructor
- `~Stack()`: Destructor
- `bool empty()`: Returns true if stack is empty, false otherwise
- *`stackElement`* `top()`: Return the top value on the stack
- `void push(const` *`stackElement`* `&val)`: Push val on top of the stack
- `void pop()`: Remove the top element from the stack
- `ostream & operator<<(ostream &out, const Stack &s)`: Output operator that prints the contents of `s` to the output stream `out`, starting at the top and printing one element per line.

3. Specifications (continued)

Command Line Input and Corresponding Output: Your main program should accept the commands below:

- **push:** Prompt the user to enter a word that should then be pushed on the top of the stack. Immediately print the state of the stack after the push operation is complete.
- **pop:** Remove the topmost word from the stack. Immediately print the state of the stack after the pop operation is complete.
- **match:** Prompt the user to enter a word and test if that word matches the topmost item on the stack. Print an appropriate message in either case; the message should contain both the user input and the word at the top of the stack.
- **exit:** End the program. *(original specification listed exit command as "quit," but test cases show the correct behavior)*

Error checking: Your program should print error messages in the following cases:

- The user enters a command other than the four valid ones listed
- The user attempts to pop a value from an empty stack
- The program runs out of memory when attempting to push a value on the stack

EXTRA CREDIT: You may earn up to **5 points** of extra credit for each of the following upgrades to your program (and may implement both upgrades for up to 10 points):

- Implement your `Stack` as a class template, not simply a stack that holds string objects. See Lecture 24 for details on class templates.
    - Note that if you write a `Stack` template, `Stack.cpp` should be empty. In this case, that file should only contain a comment explaining that your function definitions are in `Stack.h`.
- Write your own code to make all string input case-insensitive. In particular:
    - All commands should be case-insensitive (in other words, `push`, `PUSH`, and `pUsH` should all perform the same operation)
    - All words entered should be stored with lowercase letters only (`Geiger` would be pushed on the stack as `geiger`)
    - The match command should indicate two words match even if the cases of all letters do not match

  Note: Any comparisons formed in your solution should not simply pass the strings into a built-in function, such as (but not limited to) `stricmp()`, `strcasecmp()`, or `boost::iequals()`.

4. Test Cases

Your output should closely match these test cases exactly for the given input values, at least in terms of format and general functionality. I will use these test cases in grading of your assignments, but will also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.

In the test cases below, user input is underlined. *The program may behave differently if you add the case-insensitive upgrade.*

```
Enter command: pop
ERROR: Stack is empty

Enter command: push
Enter word: program
Stack: program

Enter command: push
Enter word: four
Stack: four
       program

Enter command: push
Enter word: EECE
Stack: EECE
       four
       program

Enter command: match
Enter word: EECE
User input EECE matches top of stack

Enter command: match
Enter word: four
User input four doesn't match top of stack (EECE)

Enter command: pop
Stack: four
       program

Enter command: pop
Stack: program

Enter command: pop
Stack is empty

Enter command: quit
ERROR: Invalid command quit

Enter command: exit
```