Homework Assignment #1 – Basic I/O and Reversible Stack Class

by Evan Olds

Cpt S 223

Submission Instructions:

Submit source code (zipped) to Angel BEFORE the due date/time. If the Angel submission is not working, then submit to TA via email BEFORE the due date/time.

Optional: Include a readme.txt file in the zip with any relevant information that you want the grader to be aware of.

Assignment Instructions:

Read all the instructions *carefully* before you write any code.

Part 1: Set up a main function to read commands from an input file:

In this assignment you will read commands from an input file. There is a file main.cpp on Angel that does most of the file I/O stuff for you, although you'll probably have to write it from scratch on your own in future assignments.

The input file name will be passed as a parameter to your program; it will be at index 1 in the argv array. Each line in the command file will contain a single command for you to execute. Most commands translate directly to calls of your stack class member functions. See the sample input/output files posted in Angel if you want specific examples.

Input files will look something like this:

```
header
push 10
push -6
push 5
display
reverse
display
pop
reverse
display
isempty
pop
pop
isempty
```

You must support the following commands:

- **header** : display (on screen, using printf or cout) header information that includes your name and ID number. **Print it all on ONE LINE**.
- **push** (followed by a space and then an integer value) : pushes an integer on to your stack (calls the push member function of your stack)
- **pop** : calls the Pop member function of your stack
- **reverse** : calls the Reverse member function of your stack
- **display** : Displays the contents of the stack, from top to bottom, on one line. Two important things to note here:
    1. This must NOT change the contents of the stack
    2. This is NOT a member function of the stack (declared in main.cpp)

  Separate each value with a space and put a newline/endline after all the contents have been displayed.
- **isempty** : Displays either "true" or "false" on a line based on whether the stack is empty or not. Put a newline/endline after displaying this value.

Part 2: Implement the ReversibleStack class:

Write an integer stack class in C++ that uses a singly linked list internally. It must have the following public methods:

- **void Push(int item)** : Pushes an item onto the top of the stack
- **int Pop()** : Pops an item off the top of the stack and returns it
- **bool IsEmpty()** : Returns true if the
- **void Reverse() :** Reverses the order of items on the stack by reversing the singly linked list

You must not only implement the stack with a singly linked list, but also you must have the push and pop methods run in constant time. You should not be traversing through nodes to the end of your linked list every time you do a push or pop.

The node structure used within the class must be declared privately and code outside of the ReversibleStack class implementation should NEVER have direct access to items in the linked list. Do NOT add functions that return any nodes from the list. You may add helper functions for debugging and/or testing purposes.

Part 3: Implement the display function in main.cpp:

Write a method that takes a ReversibleStack object as a parameter and displays its contents on one line. The contents must be displayed from top to bottom, so that the first value you see on the line is the value at the top of the stack and the last is the value at the bottom of the stack. When the method completes the stack must have the same contents as it did when the method was first called. Intermediate changes can happen to the stack within this function, but make sure when the function completes the stack is back to what it was originally.

This must be implemented correctly to get any credit. More efficient solutions will be worth more points. Think about an efficient way to display a stack using only the four member functions listed in part 2.

Part 4: Link up all the pieces:

Download main.cpp from the Angel web site and complete the implementation. It's set up to load the lines from the file for you. You just need to parse them and make appropriate calls dealing with your stack.

Final notes:

Only display information on the screen with printf and/or cout when the command says to. An automated grading application may be used to grade these assignments and the output must exactly match the assignment requirements for it to be properly recognized and graded. Take a good look at the sample input and output files posted on Angel to make sure your application produces the same results for each input.