Basic Command Processing and Reversible Stack Class
Cpt S 223 Homework Assignment
Washington State University

Submission Instructions: (see syllabus)

Assignment Instructions:

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

Part 1: Set up a main function to read commands from stdin:

In this assignment you will read simple commands from standard input. You will display your name and ID number on a single line to start with and then enter the command processing loop. There will be one command per line, so your main function (pseudo-code) will be something like:

```
cout << "Student Name, ID#\n";
while (true)
{
  string cmd;
  getline(cin, cmd);

  if (cmd.find("cmd1name(") == 0) { HandleCmd1(); }
  else if (cmd.find("cmd2name(") == 0) { HandleCmd2(); }
  else if (cmd.find("quit()") == 0) { break; }
}

cout << "Done\n";
```

Most command strings will be much like C function calls in terms of syntax. You must support the following commands:

push(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. See the description of the Display member function below in part 2 for more specific details.

isempty() : Displays either "true" or "false" on a line based on whether the stack is empty or not. Put a newline character ('\n') after displaying this value.

quit() : Breaks the command processing loop and terminates your program. Display the line "Done" as the very last thing before returning from main.

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() const :** Returns true if the
- **void Reverse() :** Reverses the order of items on the stack by reversing the singly linked list
- **void Display(std::ostream& writer) const :** Displays the contents of the stack from top to bottom on one line. Numbers must be separated by a single space and a single newline character should be at the end of the line. This operation must not change the contents of the stack.
    - The writer parameter is used to write the line. It works like std::cout, so instead of **std::cout << "Some string\n"** your code in this function will display lines with **writer << "Some string\n"**.
    - Following logically from part a, when you call this function from main pass std::cout as the parameter.

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: Link up all the pieces:

Process the commands in main.cpp, have a ReversibleStack object created there, and call the appropriate member functions on the stack while processing the commands.

Put all the .cpp and .h files from your project, and ONLY these files, into a .zip file and submit it online when you are done. Do not put the files in a .rar or some other format. Use .zip or you will not get credit for your submission.

Testing using I/O redirection: (this will be covered in class and you must attend lecture to get all the details)

Final notes:

Only display information on the screen with 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 included with the assignment to make sure your application produces the same results for each input.

Also, for commands that you cannot get working, you must still process the command so that the number of lines in your output matches up with the expected number of lines. Simply process the command and display something like "not implemented" if you did not implement the command fully. This will ensure that the grading app can grade your output line-by-line with no problems.