Stack ADT, Templates, Reverse Polish Notation (Postfix) Evaluation

Summary: In this lab you will implement an array-based Stack class. Initially you should implement a Stack class of type integer and thoroughly test that your implementation has correct functionality. Then you will generalize (templatize) your Stack class so that it can have an arbitrary underlying data type. For example: primitive data types such as int and double, and more complex types (classes) such as STL string. Last you will employ your user-defined Stack class in an implementation of a Reverse Polish Notation (Postfix) Calculator.

Stack Interface

- **void push(const int& x)** - Adds a new element x at the top of the stack.
- **int& top()** - Returns a reference to the next(top) element in the stack, if the stack is not already empty.
- **void pop()** - Removes the element on top of the stack, reducing the stack's size by 1 if the stack is not already empty.
- **bool is_empty() const** - Tests whether the stack size is zero. Primarily used to ensure that a pop() or top() operation is not performed on an empty instance of a Stack. Returns true if stack is empty, false otherwise.

Stack (Template) - the functionality of the following Stack member functions are semantically the same as for a Stack of type int, except now we generalize the type of the elements in the Stack to be of type T, to denote any arbitrary type we can store in a Stack object, i.e. int, double, string etc.

- **void push(const T& x)**
- **T& top()**
- **void pop()**
- **bool is_empty() const**

File I/O - To automate testing you will read input from a file to test insertion(push), peek at top(top), and removal(pop) from your Stack class. You can echo the results of a running of your program to the console (cout). Your tests should address the following.

- Use input files that contain a number of elements equal to and greater than the statically allocated size of the Stack.
- Test your Stack class on at least two data types, a primitive and a complex data type, i.e. int/string, double/string, char/string. You do NOT need to output the results of your test harness to a file, simply output your results to the console.
- Your test harness exhaustively and automatically tests both types in a single execution of program.

Reverse Polish Notation Evaluation - Below is the pseudo-code for RPN (Postfix) evaluation. Implement the algorithm as a function in C++ using your Stack. Next, read in several Postfix expressions(each on a separate line) and evaluate and write their result to a file.

```
let P be the given postfix expression;
create empty stack S;
for each symbol s in P do
if is_number(s)
then S.push(s)
else // s is an operator, + or *
if S.empty()
then report_error()
else
b <- S.pop();
if S.empty()
then report_error()
else
a <- S.pop();
compute c <- a s b;
S.push(c);
v = S.pop();
if not S.empty()
then report_error()
else print(v);
```

You are welcome to use random.org to generate input files of various types for testing your Stack Implementation.

Demo: Demo your working code for your TA.
Submission: Submit your work as lab3.tgz via iLearn.

Rubric -
20 pts Attendance (On-time)
80 pts TBD