COEN 70: Formal Specification and Advanced Data Structures

Winter 2015

Lab 4: Stack ADT – Postfix evaluation

We are going to use the stack ADT to help in evaluating a postfix arithmetic expression (See Section 7.4 of textbook). In the postfix form, each operator is placed immediately after its operands. For example, an arithmetic expression (written in the infix form) of

```
(-3.0 + 5.1) * (4.2 - 2.1)
```

is written in postfix as

```
-3.0 5.1 + 4.2 2.1 - *
```

In this form, it is much easier to evaluate the expression from left to right. The stack is used to store the intermediate results.

For this lab, we will use spaces as separators between the operands/operators. We will assume the input consists of valid real numbers and only the four basic arithmetic operators (addition, subtraction, multiplication, and division) are used.

We will first need to implement the stack ADT using either the dynamic array or single linked-list. We will then write a program to evaluate a postfix expression given through the standard input. The expression can be evaluated using the following algorithm:

1) If the input is a real number, then push onto the stack
2) If the input is an operator, then top/pop twice and store into `operand1` and `operand2`
3) Combine the operands using the operator as follows: `results = operand2 operator operand1`
4) Push the result onto the stack
5) When the expression is completely evaluated, the result is at the top of the stack

Your program should allow the user to evaluate additional expressions until the user wants to end the program. You might also enhance your program so that the expression need not be well formed; if it is not well formed, then the user must reenter the expression.

You need to properly test this program with at least 10 different expressions.