Computer Science I – Fall 2011
Lab #6: Stack Applications – Infix/Postfix Expressions

Part 1: Convert the following INFIX expressions into POSTFIX expressions using a stack. In order to demonstrate that you know how to use the stack correctly, you must show the contents of the stack at the indicated points (A, B, C) in the infix expression.

The TA will demonstrate exactly what this means, but you should keep a “Working” stack that you use to solve the problem. Then just copy the contents of the “Working” stack over at to stacks A, B, and C when appropriate.

```
         A           B              C
1) 11 * ( 6 – 5 + 3 )  – ( 3 + 7 ) /  2


           A          B        C
2) 42 – 16   / (8 – 4   * 3) +  10


      A        B      C
3) 12  – ( 4 –  8 ) –  6


               A      B        C
4) ( 2 + 9 + 6  – 11)  / 2 + 6  + 3 – 7


                 A             B                       C
5) 13 – (18 + (10  – 7 + 3) * 2 – 10) / ( ( 10 – 6) * 3 – 2)
```

Part 2: Evaluate the POSTFIX expressions that you just developed (after the conversion). Use the stack method that we discussed in class.

Cheat Sheet:

Infix to Postfix	**Evaluating Postfix**
1) For all operands, automatically place them in the output expression. 2) For an operator (+, -, *, /, or a parenthesis) **IF the operator is an open parenthesis**, push it onto the stack. **ELSE IF the operator is an arithmetic one**, then do this: Continue popping off items off the stack and placing them in the output expression until you hit an operator with lower precedence than the current operator or until you hit an open parenthesis. At this point, push the current operator onto the stack. **ELSE** Pop off all operators off the stack one by one, placing them in the output expression until you hit the first(matching) open parenthesis. When this occurs, pop off the open parenthesis and discard both ()s.	1) Each number gets pushed onto the stack. 2) Whenever you get to an operator **OP**, you pop off the last two values off the stack, **s1** and **s2** respectively. Then you push the value **s2 OP s1** back onto the stack. (If there are not two values to pop off, the expression being evaluated is not in valid post-fix notation.) 3) When you are done, you should have a single value left on the stack that the expression evaluates to.