CS 111,

Lab Assignment #3,

Due 10/26/2007

Solving Prefix and Postfix expressions using Queue's and Stacks

This is a four part assignment, in Part A & C, you will use stacks to evaluate algebraic expressions in postfix notation. In Part B & D, you will write a method that uses queues to evaluate algebraic expressions in prefix notation.

Algebraic expressions can be written in 3 notations: infix, prefix, and postfix. Infix notation is the standard form we commonly use in mathematics, 2 * (3 + 4), where operators are written between the operands they act upon: op1 operator op2

In prefix notation, an operator is written in front of its two operands operator op1 op2, + 3 4 is the same as 3 + 4

In postfix notation, an operator is written after its two operands: op1 op2 operator, 3 4 +

in both prefix and postfix notation the order of operands is the same as the original infix expression.

2 * (3 + 4) 2 3 4 + * * 2 + 3 4

and operators can be scattered through out the expression.

For simplicity, we will only consider the operators + - / and *. Also, all operands will be a single digit integer 0 – 9.

This assignment requires 4 steps!!!

1) Write a linked implementation of a Stack ADT. The stack will contain integers.
2) Write a linked implementation of a Queue ADT. The queue will contain Strings.
3) Implement a program that reads an expression in postfix notation and solves it using your stack ADT.
4) Implement a method that reads an expression in prefix notation, and solves it using your queue ADT.

Part 1: the Stack

You will need to implement the body ofa linked Stack ADT that adheresto the following interface:

```
public interface IntStack {

public boolean isEmpty();
// determines whether the stack is empty.
// Precondition:  None.
// Postcondition:  returns true if the stack is empty, and
false otherwise.

public void push( int newValue);
// adds a new value to the top of the stack
// Preconditions:  none.
// Postconditions:  NewValue is inserted at the top of the
stack

public int pop();
// retrieves the value at the top of the stack
// Precondition:  The stack must not be empty.
// Postconditon:  If the stack is not empty, the value at
the top of the stack
//_____is returned.
//_Errors:  returns -999 if called with an empty stack.

public int count();
//  returns the number of elements in the stack.
// Preconditions:  NONE;
// Postconditions:  None;
}
```

Part 2: The Queue

You will need to implement the body ofa linked Queue ADT that adheres to the following interface:

```
public interface SQueue {

public boolean isEmpty();
// determines whether the queue is empty.
// Precondition:  None.
// Postcondition:  returns true if the queue is empty, and
false otherwise.

public void enqueue( String newValue);
// adds a new value to the end of the queue
// Preconditions:  none.
// Postconditions:  NewValue is inserted at the end of the
//                  queue

public String dequeue();
// retrieves the value at the front of the queue
// Precondition:  The queue must not be empty.
// Postconditon:  If the queue is not empty, the value at
//                  the front of the queue is removed and
//                  returned.
//_Errors:  returns -999 if called with an empty queue.

public String peek();
//  returns the value at the front of the queue without
//   removing it.
// Preconditions:  The queue must not be empty.
// Postconditon:  If the queue is not empty, the value at
//                  the front of the queue is returned.
//_Errors:  returns -999 if called with an empty queue.

public int count();
//  returns the number of elements in the queue.
// Preconditions:  NONE;
// Postconditions:  None;

}
```

Part 3: Evaluating Postfix Expressions.

In this part of the assignment, you will write a main method that will allows users to do the following:

1) Enter the name of a file that contains an expression in postfix notation. Each character in the initial expression will be separated by a blank space.

3 4 + or **2 3 4 + ***

2) To evaluate the expression, you will read characters from the input one at a time (use next().charA(0) again), until the end of the input is reached (use hasNext() is false).

- As a character is read, use the isDigit method of class Character to determine it is a "digit". If it is, use a method from class Character to convert it into an integer, and push it ONTO your stack. THESE ARE YOUR OPERANDS!!!!!!
- If an operator is read, you must check the number of operands, on the stack there MUST be at least 2. If there are less than 2 you have an illegal expression, you must print an error, and STOP evaluating this expression. Also an error should be printed if the operator isn't one of the 4 listed.
- If an operator is read, and there are at least 2 operands on the stack, POP the top two operands from the stack. The first value popped will be operand 2, and the second will be operand 1. You will then evaluate the sub-expression **operand1 operator operand2** and push the result back ONTO the stack.
- When the input ends, there should be ONE value on your stack, this is the result of the expression, it should be popped and displayed to the user. If there is more than one value on the stack an ERROR has occurred.

3) You may find it useful to implement a support function with a heading such as:

public static int evaluate(int op1, char operator int op2)

The body of this method would evaluate the sub-expression and return the result.

Part 4: Evaluating Prefix Expressions.

This is much more difficult than the previous part!!!!

In this part of the assignment, you will write a main method that will allows users to do the following:

1) Enter a filename of a file containing an expression in prefix notation, evaluate the expression and show the result. These expressions will adhere to the following format. Each component of the expression will be separated by a blank, the only operators you will need to evaluate is *, +, /, and -. Each operand will be a single digit in the range of 0-9. ie:

+ 3 4 or *** 2 + 3 4**

2) To evaluate the expression, you will read characters from the input one at a time (use next(), and enqueue them as "strings" onto your queue, until the end of the input is reached (use hasNext() is false). **HINT: you can verify that you are reading the expression correctly by dequeueing it and printing each value.**

3) Once the entire expression is enqueued, you will then evaluate the expression doing the following:

- You will write a method with the following format:

public static int calcPre(Queue aQueue)

- If there are < 3 elements in the queue, or the 1st element is not one of the 4 operators, then the expression is invalid.
- Dequeue the 1st 3 values. Into 3 String variables ie: a, b, & c
- While not done
  - If a is not an operator, and B & C are not numbers, then we do not have a subexpression. So;
    - Enqueue A back onto the queue.
    - A = B
    - B = C
    - Dequeue a new value into C
  - Else is we have a subexpression
    - Use the pareInt() method of class Integer to convert B & C to integers.
    - Use charAt(0) to extract the operator from A
    - Evaluate the subexpression
    - Enqueue the result back onto the queue.

        - Next, if the number of elements == 1, we are “done”.
        - Else, dequeued 3 new values into a , b, & c and continue processing.
    - When done, return the result to the calling program.

Hint: you may want to search class Integer for methods that would help you test if a string contains an integer, or you may want to write a helper method to test if the first character in the string is a digit, or an operator.

You will receive more hints in class as the project progresses.