California State University, Sacramento
College of Engineering and Computer Science

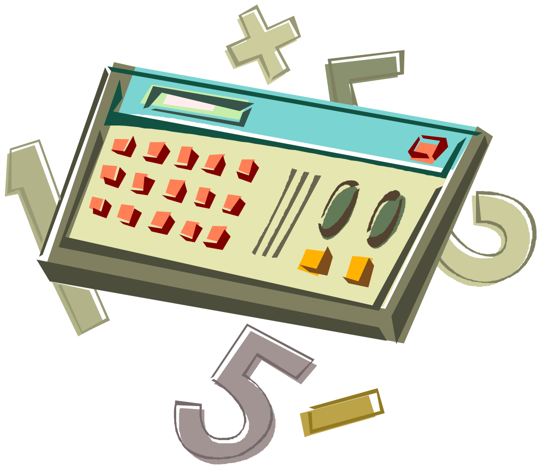
Computer Science 130: Data Structures and Algorithm Analysis

Assignment #1 – Postfix Evaluator

Overview

For this assignment, you are going to create a postfix expression evaluator.

Postfix notation, also known as Reverse Polish Notation, is a stack-friendly format. We aren't implementing Dijkstra's Shunting Algorithm quite yet but be patient.



Your Task

Part 1: Interface

Manipulating strings is not as easy in all programming languages. Java, Visual Basic, and C# all have a 'split' function, but these can a tad troublesome when parsing a string without whitespace.

For example, splitting on spaces is easy for "`3 * 4`", but not for "`3*4`". I don't want this assignment to be more difficult for one language than another. As a result, you won't have to manually parse an expression and turn it into a collection of strings. Instead, to initialize the input queue, just Enqueue the items:

```
input.enqueue("9");
input.enqueue("7");
input.enqueue("-");
input.enqueue("21");
input.enqueue("*");
```

The Evaluator Class must have the following interface. You can add some additional methods if you want.

public class Evaluator		
`String`	`about()`	Returns text about you – the author of this class.
`void`	`enqueue(String item)`	Enqueue the item onto the postfix input queue.
`double`	`evaluate()`	Evaluates the values and returns a double result.

Your postfix input queue should store strings. Do not use an ArrayList. For this assignment, we will rely on our own skills to create an efficient queue and stack.

Part 2: Evaluator

The code that evaluates the input queue must use the stack-based approach we went over in class. Naturally, your solution will contain both a private queue and a private stack. How you implement these is completely up to you – but I would strongly recommend using a linked-list.

Your stack should store double floating-point numbers. Your evaluator method must return a double containing the result of the calculation. You must support the following operators. Don't worry about unary minus or function calls (like Sin and Cos).

Operator	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponent

Part 3: Testing

Once you have finished your code, you need to test it using some real postfix expressions. Make sure to make come up some good test data. Have at least 5 different ones. Hard-code these tests into your program. Don't just write an input loop.

I will also test your program using my own data.

Due Date

You have one week to finish this assignment. Given you already have developed excellent programming skills in CSc 20, this shouldn't be a difficult assignment.

E-Mail the following to dcook@csus.edu:

- The source code.
- The main program that runs the tests.
- Output generated by your tests