CSS 342 Assignment #4 (Due: Tuesday, Nov 15, 2016) Fall 2016

PART 1 (programming, due Tue, Nov 15 11:45pm):

Write a program to evaluate a prefix expression and to convert a prefix expression to postfix using recursion. You may assume each expression is a valid prefix expression where the operands consist of just one digit and the operators are either + ,- , *, / . For simplicity, we use integer arithmetic (answer is an int). E.g., (3+4)/2 which is /+342 in prefix notation results in the answer of 3 and is 34+2/ in postfix notation.

By definition, an infix expression is leftOperand operator rightOperand
e.g., a+b (leftOp operator rightOp). In a prefix expression, the operator is first. In postfix, the operator is last. E.g., a+b becomes +ab (operator leftOp rightOp) prefix and ab+ (leftOp rightOp operator) in postfix. In prefix and postfix expressions, there is no need for parentheses to change the order of operations. The precedence of the operators is built into the expression. Humans prefer infix expressions, but computer programs prefer prefix or postfix expressions (so the expression can be easily scanned to figure out which operation to do first).

In the data, there is one prefix expression per line. You can assume valid prefix expressions with single digit operand. For example,

sample input

```
+23
-+231
*+51-24
/*+91-42+53
```

sample output

```
Prefix Expression = Answer
--------------------------
+23 = 5       As postfix:  23+
-+231 = 4       As postfix:  23+1-
*+51-24 = -12       As postfix:  51+24-*
/*+91-42+53 = 2       As postfix:  91+42-*53+/
```

Notes

-- To evaluate a prefix expression, the key to the recursion is that every prefix expression is made up of an operator followed by 2 operands and each of the operands is itself a prefix expression. The simplest of prefix expressions is a single digit (number). The function isdigit() is useful for determining if a char is a digit or not. The recursive solution to convert a prefix expression to postfix is similar to evaluating a prefix expression.

-- While it is still object-oriented, the assignment focus is on recursion. The prefix expression is stored in an array. The function to evaluate is very short. If you are doing a lot of work, then you are doing it incorrectly. At the beginning of the recursive routine, the next symbol in the expression is considered. Ignore the array, i.e., think *symbol* in your problem solving.

-- The type of char can also be treated as an int. If you assign a char to an int, it will yield the ascii value. For example, `int i = 'a';` gives i the ascii value of 97. So, you can turn a char '2' into an int 2 using the code `i = ch - '0';` The difference in ascii values (which you don't even need to know) of character '2' and character '0' is integer 2.

-- So that you don't get hung up on the reading, all the reading and printing for the assignment is given. You can find main for lab4 and a partial Prefix class linked off the assignments page. You won't do any other reading than what is given. So that the given main works properly, be sure in your data files on a PC, the cursor is on the line following the last expression. When you're solving this problem, all that is left is the recursion so you can focus on the recursive problem solving. Think high-level to solve this program. Focus on prefix expressions, not the array.

The loop in main handles one file's worth of prefix expressions. One call to evaluatePrefix deals with one line of data, one prefix expression. The eof function is true when the EOF character is read, so we read beyond the last expression to know that it is end-of-file.

PART 2, written (To turn in at lecture, Thursday, Nov 17, 3:30pm) (on 8.5 by 11 inch paper)

Also hand in an execution of your program for evaluating a prefix expression. (Draw an execution tree, in a fashion similar to what was done in class; handwritten is best.) Show all calls starting with a call from main and keep track of relevant information at each level. Be clear about your calls and returns. Do this for just the following expression: `*+4*32++156`