

Computer Science 3310

Program 4

Your assignment is to write a program that evaluates postfix expressions. The form of a postfix expression and the algorithm for evaluating them are discussed in chapter 7 of your text on pages 349-351. In particular, the algorithm uses a stack for storing the operands until used in carrying out a calculation. For this program, the operands will be upper case letters and the operators will be from the set of characters $\{+, -, *, /\}$. The semantics for these operators conforms to their semantics in Java. Integer arithmetic will be used. In order to assign values to the upper case letters, the first portion of the input consists of assignments of integer values to letters. Any letter not assigned a value will have the default value of 0. The second portion of the input consists of postfix expressions to evaluate.

Input Format: The first portion of the input (one or more lines) consists of assignments of integer values to upper case letters. The letters are in no particular order, but no letter will be assigned two values. An example *assignment line* is

D = -45

The integer values assigned to letters will be in the range of a 32 bit integer. The assignment lines always begin with the letter in column 1 of the input line. You may assume that there will be one or more spaces between each token (i.e. the $\langle \text{letter} \rangle$, $\langle \text{asst_op} \rangle$, and $\langle \text{number} \rangle$). A sentinel line of the form $\$PART2$ indicates the end of the assignment lines. The second portion of the input (one or more lines) consists of postfix expressions. An example *expression line* is

CBCAE-+-X-

Each of the input expressions will consist of only upper case letters and the four operators. The expression lines begin in column 1 and contain no embedded spaces. Any letter appearing in an expression but not assigned a value is assumed to have the default value of 0. You may also assume that the values of any calculations involved in evaluating an expression will all be in the range of a 32 bit integer.

The input is terminated by the sentinel line $\$END$.

Input Processing:

- Read from standard input. Use the scanner class described on pages 44-45 of your text. Individually process each token of the assignment lines.
- Process the expressions as strings. The String functions $length()$ and $charAt()$ should be very helpful.

Error Handling: The following error checking should be performed.

- After the last character of an expression is processed, the stack should consist of one number, which is the result.
- When an operator is encountered, there should be at least two operands on the stack.

If any of these errors occur, a suitable error message should be printed, and the rest of the input expression ignored. You may use exception handling or traditional conditional logic for doing your error handling. *Processing should resume with the next line.* Error handling that causes termination of execution of the program will be severely penalized.

Output Format: See sample test case. The expression line should be echoed before the result is printed.

Data Structures: To keep up with the assigned values of the operands, use an array of 26 integers. A slight complication is that the array subscripts will run from 0 to 25, while the ASCII codes associated with the upper case letters range between 65 and 90. However, the following function will map an upper case letter into a number between 0 and 25 where 0 will correspond to 'A', and 25 correspond to 'Z'.

```
// int addr(char ch) returns the equivalent integer address for the letter
// given in ch, 'A' returns 0, 'Z' returns 25 and all other letters return
// their corresponding position as well.
```

```
public static int addr(char ch)
{
    return (int) ch - (int) 'A';
}
```

To hold the operands until they are used in calculations, a stack will be used. You can use the API Stack (URL provided on course WWW page). Note that it is a Java generic as discussed in class and Section 9.4 of the text. All this means is that you have to specify the type of the stack element when you declare and initialize your stack.

Input Redirection: I encourage you to use it. Refer to Program 2 description for guidance.

Program Development: I would encourage you to do this program in 3 phases. In phase 1 make sure you can read in the assignment lines, associate letters with values, and look up those values when reading expression lines. In phase 2, you add in the processing of legal expressions, and in phase 3 you include the error handling.

PROGRAM GRADING

This program will be worth **150** points. The grade weights are as follows.

Correctness - Basic Processing	50%
Correctness - Error Handling	20%
Coding Style	15%
Documentation	10%
Test Plan	5%

PROGRAM DEADLINES

April 14, 11:00 P.M.

Deadline for electronic submission of completed program, including all documentation. Submission should conform to the requirements discussed in the **Programming Standards** handout. You will only need to send your application program which should be named *PostEval.java*. Use the *submit* command with an assignment number of 4. Submissions that do not follow the naming conventions discussed in this assignment for file names are subject to a 25% penalty.

SAMPLE TEST CASE

Input:

```
C = -13
X = 5
H = 25
D = 4
$PART2
DC+HX*C--
AH-DB+/
AH+-
$END
```

The output would be

```
DC+HX*C-- = -147
AH-DB+/ = -6
AH+- = ERROR: Missing operands for -, expression ignored.
```