Postfix Revenge

Problem Description

We went over an Expression Tree in class which was able to:

- Parse a Postfix expression into an Expression Tree
- Evaluate the expression
- Generate code for the expression

All of the code for this example is available on Canvas. You will be making several enhancements to it. The required changes are straight-forward but there is an opportunity to earn extra credit in this assignment (see **Bonus Section** at the bottom).

1) Infix Notation String: The tree already has the ability to print itself out in a graphical tree type format: ExpressionTree.printTree(). Add the ability to convert an expression tree to an infix notation String. Do this by overriding the ExpressionTree.toString() method. You can use parentheses to make sure the expression would be interpreted correctly. In the Postfix.main() method, the tree is already printed this way:
   ```
   System.out.println("The Infix expression is: " + tree);
   ```
2) Parse and Evaluate Variables: Currently the expressions accepted include only integer constants and operators (+, -, *, /). Add variables. A variable is a single lowercase letter, a through z. When *evaluating* a tree with variable, ask the user for the value of each variable. Note that the user is asked when evaluating the tree, not when parsing the expression.
3) Generate Code for Variables: Modify the code generation part of the code to handle these new variables. Assembly language does not have variables, per se. However, it does have memory which can be labelled with a name. That's how we will code any variable that is seen in the expression. If a variable 'x' is used in an expression, produce the following two lines of code for it:
   ```
   la $at, x
   lw $t0, 0($at)
   ```
   As way of explanation, the *address* of the memory location labeled 'x' is loaded into a special register called **`$at`**. **`la`** stands for Load Address. We then load the contents of that address, offset by 0 bytes, into our working register for use later in the expression. **`lw`** stands for Load Word. Note that when you generate code, you will use whatever variable is next available, not necessarily **`$t0`**.

Solution Strategy

You should create your solution by following the steps outlined below:

☐ Copy the code from Canvas into your own workspace

- The files are not defined in a package. You will save yourself some issues if you keep this arrangement for your workspace.

☐ Make sure the code works as is. Try some expressions such as:

- `5`
- `1 2 +`
- `3 4 * 2 5 + - 4 2 / *`

☐ Change Postfix.main().

- Include your name in the following line:

```
System.out.println("Postfix: First name Last name\n");
```

☐ Change ExpressionTreeOp to be able to also be able represent a third type: variable.

- You probably want a method called isVariable(), similar to isOperator().
- Change ExpressionTreeOp.toString() to be able to handle the variable type.

☐ Define ExpressionTree.toString().

- Use a simple recursive solution to print the expression in infix notation.
- Use the pattern for other recursive walks of the tree (evaluate & evaluateNode, generateCode & generateCodeNode).
- Make sure you handle the base cases: constant or variable, as well as the recursive case: operators.
- Always printing parentheses for operators will work.

☐ Change PostfixEvaluator.parse()

- Recognize when you have a variable (letter a through z, lowercase) and create an ExpressionTreeOp for the variable. Add the new node to the tree appropriately.

☐ Change ExpressionTree.evaluateNode()

- When you are evaluating a variable, ask the use for the value to use.
- **Bonus:** remember that value for that variable and don't ask for the same one again.

☐ Change ExpressionTree.generateCodeNode()

- When you are generating code for a variable, create the two lines of code required (see explanation above).

Notes

- The main() must be in a class called Postfix.
- Turn in only your Java source files, including all of the code you downloaded, even if it remained untouched.
- Exception handling is not required.

Examples

Your output should look something like the following example. It should include *your* name. Input is shown in green.

```
Postfix: Prof. Eckert

Enter a valid post-fix expression one token at a time with a space between each token
(e.g. 5 4 + 3 2 1 - + *)
Each token must be an integer, operator (+,-,*,/), or variable
5

Code:
        li $t0, 5


Result in $t0
The Infix expression is: 5
Expression value is: 5

Evaluate another expression [Y/N]? Y

Enter a valid post-fix expression one token at a time with a space between each token
(e.g. 5 4 + 3 2 1 - + *)
Each token must be an integer, operator (+,-,*,/), or variable
1 2 +

Code:
        li $t0, 1
        li $t1, 2
        add $t0, $t0, $t1


Result in $t0
The Infix expression is: (1+2)
Expression value is: 3

Evaluate another expression [Y/N]? Y

Enter a valid post-fix expression one token at a time with a space between each token
(e.g. 5 4 + 3 2 1 - + *)
Each token must be an integer, operator (+,-,*,/), or variable
3 4 * 2 5 + - 4 2 / *

Code:
        li $t0, 3
        li $t1, 4
        mul $t0, $t0, $t1
        li $t1, 2
        li $t2, 5
        add $t1, $t1, $t2
        sub $t0, $t0, $t1
        li $t1, 4
        li $t2, 2
        div $t1, $t1, $t2
        mul $t0, $t0, $t1


Result in $t0
The Infix expression is: (((3*4)-(2+5))*(4/2))
Expression value is: 10

Evaluate another expression [Y/N]? Y

Enter a valid post-fix expression one token at a time with a space between each token
(e.g. 5 4 + 3 2 1 - + *)
Each token must be an integer, operator (+,-,*,/), or variable
a 2 b + +
What is the value of a? 12
What is the value of b? 20
```

```
Code:
        la $t0, a
        lw $t0, 0($t0)
        li $t1, 2
        la $t2, b
        lw $t2, 0($t2)
        add $t1, $t1, $t2
        add $t0, $t0, $t1


Result in $t0
The Infix expression is: (a+(2+b))
Expression value is: 34

Evaluate another expression [Y/N]? Y

Enter a valid post-fix expression one token at a time with a space between each token
(e.g. 5 4 + 3 2 1 - + *)
Each token must be an integer, operator (+,-,*,/), or variable
a a *
What is the value of a? 7
What is the value of a? 7

Code:
        la $t0, a
        lw $t0, 0($t0)
        la $t1, a
        lw $t1, 0($t1)
        mul $t0, $t0, $t1


Result in $t0
The Infix expression is: (a*a)
Expression value is: 49

Evaluate another expression [Y/N]? N
```

Bonus Section

There are a couple things that can be improved about our solution.

- [+10 points]: If a variable appears in the expression multiple times, only ask for its value only once. In the last example above, when evaluating `a*a`, the computer asks the user for the value of `a` twice. This is fixed in an example below.
- [+10 points]: Memory space should be set aside for all variables that appear in the expression. Since we are going to define memory space, we must now distinguish between our data segment (space for data) and our text segment (space for code). You will see how to use the `.text` and `.data` directives in the examples below.

Examples:

```
Postfix: Prof. Eckert

Enter a valid post-fix expression one token at a time with a space between each token
(e.g. 5 4 + 3 2 1 - + *)
Each token must be an integer, operator (+,-,*,/), or variable
a 2 b + +
What is the value of a? 12
What is the value of b? 20

Code:
.data
a:      .word
b:      .word

.text
        la $t0, a
        lw $t0, 0($t0)
        li $t1, 2
        la $t2, b
        lw $t2, 0($t2)
        add $t1, $t1, $t2
        add $t0, $t0, $t1


Result in $t0
The Infix expression is: (a+(2+b))
Expression value is: 34

Evaluate another expression [Y/N]? Y

Enter a valid post-fix expression one token at a time with a space between each token
(e.g. 5 4 + 3 2 1 - + *)
Each token must be an integer, operator (+,-,*,/), or variable
a a *
What is the value of a? 7

Code:
.data
a:      .word

.text
        la $t0, a
        lw $t0, 0($t0)
        la $t1, a
        lw $t1, 0($t1)
        mul $t0, $t0, $t1


Result in $t0
The Infix expression is: (a*a)
Expression value is: 49

Evaluate another expression [Y/N]? N
```