Programming Assignment 4

Assigned: Nov. 7
Due: Nov. 30

In this assignment, you will write an interpreter for a toy programming language.

Input

The input consists of a sequence of assignment statements, loops, functions definitions, and function calls.

An assignment statement has the form \texttt{ASSIGN \langle variable \rangle \langle expression \rangle}. A variable is an alphabetic string. An expression is a prefix expression whose operators are \texttt{+, \(-, \ast, \text{and } /}, and whose leaves are positive integers and variables. The symbols in the expression are separated by white space. An assignment statement is one line long. For example, the following are assignment statements:

\begin{verbatim}
ASSIGN X 1
ASSIGN X + X 2
ASSIGN Number * + X 2 - X 2
\end{verbatim}

A function call has the form \texttt{CALL \langle function \rangle}. A function name is an alphabetical symbol. A function call is one line long. For example,

\begin{verbatim}
CALL TryThis
\end{verbatim}

A loop has the form \texttt{FOR \langle expression \rangle \langle statement \rangle} where the statement is either an assignment statement, a for loop, or a function call. The expression is evaluated when the loop is entered; it is not reevaluated. For example, the following are loops:

\begin{verbatim}
FOR 10 ASSIGN N + N N
FOR / N 2 ASSIGN X + X 1
FOR 8 CALL Fib
FOR M FOR N ASSIGN X + X 1
\end{verbatim}

A function definition consists of multiple lines:

- The first line has the form \texttt{DEFINE \langle function \rangle}.
- The last line has the form \texttt{END}
- All the middle lines are either assignment statements, loops, or function calls.

For example, the following are two function definitions:

\begin{verbatim}
DEFINE IncrementX
ASSIGN X + X 1
END

DEFINE FF
CALL IncrementX
ASSIGN Y * X X
END
\end{verbatim}
The input may contain blank lines; these are ignored.

**Output**

The output of the interpreter is a trace of its actions.

- Every time a variable is assigned a value, the interpreter should print out “Assigning ⟨value⟩ to ⟨variable⟩.”
- Every time a function is defined, the interpreter should print out “Defining function ⟨function name⟩.”
- Every time a function is called, the interpreter should print out “Calling function ⟨function name⟩.”

For example, suppose the input has the following form:

```
ASSIGN X 1
ASSIGN Y 1

DEFINE Fib
ASSIGN TMP Y
ASSIGN Y + X Y
ASSIGN X TMP
END Fib

FOR 3 CALL Fib
```

Then the output will be

```
Assigning 1 to X
Assigning 1 to Y
Defining Fib
Calling Fib
Assigning 1 to Tmp
Assigning 2 to Y
Assigning 1 to X
Assigning 1 to W
Calling Fib
Assigning 2 to Tmp
Assigning 3 to Y
Assigning 2 to X
Calling Fib
Assigning 3 to Tmp
Assigning 5 to Y
Assigning 3 to X
```
Assumptions

You may assume that:

- The input is correctly formatted.
- Every variable is assigned a value before being accessed.
- Each function is defined only once in the input
- Every function is defined before being called. As a consequence, no function is directly or indirectly recursive. If function F calls function G, then the definition of G must precede the definition of F in the input.

You may take the input either from terminal input or from a file “input.txt” (your choice).

Data Structures

Define a class Variable with two data fields: the name and the current value. Construct a hash table SymbolTable that uses the name as the key and the Variable as the value.

Define a class Function that represents a function. This should have two data fields: the name of the function, and a list of statements. Construct a hash table FunctionTable which uses the name of the function as the key and the Function as the value.

Define an enumerated class Operator with nine values: number, variable, plus, minus, times, divide, assign, call, and loop.

Define an abstract class ExpressionTree with three data fields:

- label is an Operator
- left and right are ExpressionTrees.

There should be an abstract method int evaluate().

Define the following subclasses of ExpressionTree:

NumberLeaf is a leaf of the tree whose Operator is number. It has an additional data field value which an Integer

VariableLeaf is a leaf of the tree whose Operator is variable. It has an additional data field myVariable which is a Variable.

Call is a leaf of the tree, whose operator is call. It has an additional data field myFunction which is a Function.

ArithExpression is an internal node whose Operator is one of the arithmetic operators, and whose left and right children are the trees for the arguments.

Assignment is an internal node whose operator is assign, whose left child is a VariableLeaf for the variable on the left side of the assignment and whose right child is the ExpressionTree for the expression on the right side.

Loop is an internal node whose operator is loop, whose left child is the ExpressionTree for the number of iterations, and whose right child is the ExpressionTree for the body of the loop.
Top level pseudo-code

main {
    loop until (end of input) {
        line = readLine();
        if (line[0].equals("DEFINE"))
            readFunctionDefinition(line[1]);
        else executeStatement(line);
    }
} // end main

readFunctionDefinition(name) {
    functionBody = new LinkedList<ExpressionTree>();
    loop {
        line = readLine();
        if (line[0].equals("END")) break;
        ExpressionTree tree = constructExpTree(line);
        functionBody.addLast(tree);
    }
    Function fn = new Function(name, functionBody);
    FunctionTable.put(name, fn);
}

executeStatement(line) {
    constructExpTree(line).evaluate();
}