

Programming Assignment 4

Assigned: Nov. 8

Due: Dec. 6.

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

Input

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

An assignment statement has the form `ASSIGN <variable> <expression>`. A variable is an alphabetic symbol. An expression is a prefix expression whose operators are `+`, `-`, `*`, 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:

```
ASSIGN X 1
ASSIGN X + X 2
ASSIGN Number * + X 2 - X 2
```

A function call has the form `CALL <function>`. A function name is an alphabetical symbol. A function call is one line long. For example,

```
CALL TryThis
```

A function definition consists of multiple lines:

- The first line has the form `DEFINE <function>`.
- The last line has the form `END`
- All the middle lines are either assignment statement or function calls.

For example, the following are two function definitions:

```
DEFINE IncrementX
ASSIGN X + X 1
END
```

```
DEFINE FF
CALL IncrementX
ASSIGN Y * X X
END
```

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

CALL Fib
ASSIGN W X
CALL Fib
ASSIGN Z - * W Y * X X
```

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
Assigning -1 to Z
```

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 **Statement** that represents input statements (assignment statements and call statements). You may use either a list or array of symbols or an expression tree. The node in the data structure that reference the variable should have a reference directly to the **Variable** object, not the name of the variable.

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

Honors assignment

Augment the above language with loops. A loop has the format “FOR \langle expression \rangle \langle statement \rangle ”, where the statement is either an assignment statement or a function call. The expression is evaluated when the loop is entered; it is not reevaluated. For example, the following are loops:

```
FOR 10  ASSIGN N + N N
FOR / N 2 ASSIGN X + X 1
FOR 8  CALL Fib
```

The output trace does not have to show the FOR loop explicitly, just the execution of the assignment statements or the call statements.