Assignment 4: Operational Semantics via Tree Interpretation

ISSUED: Wednesday, 27 April 2005
DUE: 5PM Friday 13 May, via handin on falcon/hornet
POINTS POSSIBLE: 100
WEIGHT: 13% of total class grade
READING: Lecture Notes Weeks 4 and 5, Textbook Chapters 3-5

Specification

The deliverable for this assignment is a parse-tree interpreter for EJay. The interpreter takes as input the tree and associated symbol table generated by the solution to Assignment 3. The interpreter runs the EJay program, producing a dumpable memory that can be inspected after execution to determine if the program ran correctly.

You should use the provided solution to Assignment 3 as the starting point for Assignment 4. This solution is in the ejay.cup file linked to from the Assignment 4 online directory. The provided solution includes additional features for memory management and data type handling that were not required for your solution to Assignment 3.

Helpful Examples

To help you get started, there is an example interpreter for a subset of the Pascal programming language. It, and its supporting files, are in the directory www.csc.calpoly.edu/~gfisher/classes/330/examples/interpreter. The example presents the general form recommended for this assignment, and illustrates some of the implementation details described below.

You may use any and all of the code in the Pascal example. However, you must be prepared to modify the code as necessary to deal with the differences between Pascal and EJay. You will also need to fill in missing details that are not fully implemented in the Pascal example. We will discuss the significant differences when we go over the Pascal example in class.

There are also examples in the reading materials cited above, i.e., Lecture Notes weeks 4 and 5, plus chapters 3 through 5 of the book. Specifically relevant sections of the book are the following: 3.1, 3.2, 3.6, all of chapter 4, 5.1, 5.2, 5.4, 5.5, and 5.6.

Implementation Details

Your interpreter must be defined in a file named "EJayInterpreter.java". It is recommended that your interpreter follow the structure illustrated in example PascalInterpreter.java. However for this assignment, you have more design latitude than in previous assignments, since the strict guidelines for JFlex and CUP are taken care of in the provided solution to Assignment 3. At this point, we are dealing with straight Java programming.

There is an updated set of support files in the online Assignment 4 directory. These include the same parse tree and symbol table classes from Assignment 3, with upgrades to support memory management and data type semantics.

The provided solution in ejay.cup is designed to integrate smoothly with the upgraded support. As with Assignment 3, if there is anything you feel you need to change or add to in the support files, you may do so. Please run it by me first, so we can discuss if your updates are really necessary.

There are two important implementation details in the areas of type equivalence and equality checking that we will discuss fully in lecture. For the purposes of this assignment, the following are the specific requirements:

a. the interpreter will use structural type equivalence when comparing two types for equality; this rule is fully implemented in a provided support method named "Types.equiv(TreeNode, TreeNode)"

b. the interpreter will implement shallow equality for array and struct types; i.e., the EQ and NOT_EQ operators compare the value of references to array and struct values, not the values of individual components within the array and struct values
Error Handling

The errors that the interpreter will detect are based on a limited amount of runtime type checking. This is necessary since we are not implementing a static type checking pass prior to interpretation.

The specific runtime type checking rules you must implement are the following:

a. arithmetic operators work only on numeric types (int and float), and the types of both operands must be the same (i.e., no mixed type int/float or float/int expressions)
b. the types of the LHS and RHS of an assignment must be the same
c. all relational operators except EQ and NOT_EQ work only on arithmetic types, and the types of both operands must be the same
d. EQ and NOT_EQ work on any types
e. the type of the expression in an IF and WHILE statement must be boolean
f. in a function call, the number and type of actual and formal parameters must match exactly

What it means for two types to be the same is defined by the provided `Types.equiv` method noted above. We’ll discuss details in lecture.

Recommended Testing Phases

There are two supplied test drivers for Assignment 4. These are used in the separate testing phases described below.

As with Assignment 3, there is a recommended progression of phases for implementing and testing your solution to Assignment 4. The steps are the following:

1. Modify the supplied `ejay.cup` parser to compute memory locations for variables and function parameters.
2. Implement interpretation for statements and expressions with simple types (no arrays or structs yet).
3. Add interpretation of functions, including recursive calls, but not yet call-by-ref.
4. Add interpretation for arrays and structs, including designator assignment.
5. Add interpretation of call-by-ref parameters.

For steps 1 and 2, use the simplified testing program in `EJayInterpreterNoFuncsTest.java`. For steps 3 through 5, use the full version of the tester in `EJayInterpreterTest.java`. There are some sample `ejay` test programs in the Assignment 4 directory online.