

## PROJECT ASSIGNMENT #3

Due Monday, March 14, 2005

1. Read Sections 7.1-7.6 in the text.
2. Design appropriate symbol table data structures for Two-Level Grammar. Note that function “identifiers” are actually a collection of identifiers. For example, in the Pam TLG, class `RelOp`, there is a function of `Integer1` and `Integer2`, where `of` and `and` define the function “name.”
3. Write procedures to locate and insert identifiers in the table. To locate an instance variable or method identifier for a class, keep in mind that inheritance in the class hierarchy corresponds to the “most closely nested” scope rule in block structured languages.
4. Integrate calls to your symbol table building methods in the grammar so the symbol-table is constructed as the TLG specification is parsed.

### Symbol Table Descriptions

The symbol table which you will need for TLG is described below. Procedures will have to be written to locate identifiers in the table and to insert new identifiers. Symbol tables define an environment for all identifiers in the program.

- name - string representation for identifier name
- category - variable, class, or function (method)
- type - integer, float, boolean, string, character, list, set, or map, class (with attributes for the components of the class and nesting level in the class hierarchy), or function (with attributes formal parameters, local identifier table, and object code representation)

The table should be structured as a nested collection of tables in the form of a tree structure. For example, all global identifiers (i.e. those identifiers declared at the outermost level) should be in the root table. All class and function identifiers declared at the root level should have a subtable containing their local identifiers. Note that classes may be nested to any level, thereby establishing the tree-like structure of this symbol table. This type of static tree structure can be conveniently manipulated using a compile-time display to keep track of which path in the tree is currently active when classes are being declared, so as to facilitate searching back through the class hierarchy for inherited identifiers.

It may also be convenient to represent some attributes as references to auxiliary objects. For example, a list type may be denoted by a `List` object, function attributes may be denoted by a `Function` object, etc. Finally, the entire symbol table may be most conveniently implemented in Java using `java.util.Map` objects.