UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS 264
Fall 1995

S. L. Graham
A. B. Downey

Assignment 3: Symbol Tables

Due: Friday, 13 October 1995

The purpose of this assignment is to strengthen your knowledge of symbol tables and related mechanisms. Please give a hardcopy of your solution to Allen.

1 Review Two Papers on Symbol Table Implementation

The papers by Graham/Joy/Roubine and by Cook/LeBlanc discuss some symbol table implementation techniques for languages with explicit scope control. The papers are now somewhat old and they reflect an active period for language design ideas, only some of which have survived. Read these papers in order to determine the key ideas and their application to contemporary implementation. Following are some issues to consider:

- What are the key ideas in each paper?
- Which of the key ideas are motivated by one-pass processing? What can be done instead if multiple passes are used?
- Which ideas are intended for "non-persistent" symbol tables? How are they changed if the symbol table structure is retained?
- Which ideas are motivated by saving space? How can the implementation be made easier at the price of more space?
- Which methods can be simplified if more time is allowed? Is the additional time a linear or nonlinear function of program size?
- Which methods are appropriate to object-oriented languages (relate to a particular OO language if you can)?
- In summary, what are the set of techniques that are important today, given what you regard as the typical language features and time/space/persistence requirements? Be sure to state those assumptions.

2 Evaluate Cadaver Symbol Table/Binding Implementation

Study the way name binding and the recording and access of attributes of named entities is done in your Cadaver compiler and relate it to the material discussed in class. Following are some examples of issues that would be interesting to discuss. You should add any others that you discover or that you think are relevant. The purpose of these questions is to motivate you to understand details about how the source language is handled, not just find out (superficially) what category of symbol table organization is used.

- What are the binding issues for the source language of your cadaver compiler? What kinds of bindings, if any, must be resolved at runtime? Can programs generate new named entities (as opposed to entities that are referenced indirectly)? Are there anonymous entities that must be recorded and accessed? Give an example of a source language situation that is challenging for the implementation. (Note: if some of the work is done at runtime, explain the runtime mechanisms in answering the subsequent questions.)
- Is there a symbol table *per se* at compile time? At runtime? If so, is the complete table built or only the portion visible within the current scope (i.e. can the table persist after the conclusion of compilation/execution?)
- What information from the symbol table is available to the debugger at runtime? How is this information passed to the debugger?
- What is the architecture of the symbol table or its equivalent? How are entries made? Deleted (if that happens)? How are lookups done? Give order of growth bounds for the time required for each operation (insertion, lookup, deletion, scope entry/exit, etc.)
- What kinds of attributes are recorded for the named entities? How are the attribute values determined (is the information explicit in the text of the program or inferred by analysis)? What complications, if any, are there in determining the values of those attributes?
- What appear to be the primary design criteria for this part of the compiler (e.g. speed, small space, only functionality)? Is the implementation economical in time? In space? Does the design scale to large programs?
- How robust is this part of the compiler (or runtime) in the face of errors? How good are the diagnostics? How did you determine that?
- What support is provided for separate compilation? What information from one module is available during the compilation of another module? How do external references get bound?