

Names, Scopes, and Bindings

3.11 Explorations

- 3.39 Learn about the `.stabs` directives used by Unix compilers and assemblers to include symbol table information in object and executable files, where it will be accessible to symbolic debuggers. Write a brief tutorial that might help a systems programmer understand the directives found in assembly language files.
- 3.40 Learn about the *reflection* mechanisms of Java, C#, Prolog, Perl, PHP, Tcl, Python, or Ruby, all of which allow a program to inspect and reason about its own symbol table at run time. How complete are these mechanisms? (For example, can a program inspect symbols that aren't currently in scope?) What is reflection good for? What uses should be considered good or bad programming practice? For more ideas, see Section 15.3.1.
- 3.41 Learn about the `typeglob` mechanism of Perl, which allows a program to modify its own symbol table at run time. What are `typeglobs` good for? (See the sidebar on page 707 for some initial pointers.)
- 3.42 Create a C program in which a variable is exported from one file and imported by another, but the declarations in the files disagree with respect to type. You should be able to arrange for the program to compile and link successfully, but behave incorrectly. Try the same thing in Ada or C++. What happens?
- 3.43 Investigate the use of module hierarchies in the standard libraries of C++, Java, and C#. How is each organized? How fine grain is the division into separate headers or packages? Can you suggest an explanation for any major differences you find?

