Joseph Schmidt

Lab 4: Symbol Tables and you (and your compiler): a recipe for health and (compiler) happiness.

Crafting a Compiler:

8.1

Advantages for using binary search trees or hash tables for a symbol table are the simplicity and commonly implemented nature of each, as well as the efficiency of each. Disadvantages are the average case for binary search, and collisions in hash tables (which can be mitigated by collision handling).

8.3

The two alternatives to handling multiple scopes in a symbol table are to make a table for each scope, or two have one symbol table handle the symbols. With the first method, all scopes are handled with a stack, so a new scope is pushed onto the stack, and popped off when it is closed. In the second, the open scope and closed scope depends on the scope name or depth, which is then used to identify scope uniquely.

The first method for the program in figure 8.1:

- openScope: f() and g() are pushed to the stack
- openScope: w, x, and g() are pushed to the stack
- openScope: x,z,f(),w,z are pushed to the stack
- closeScope: x,z,f(),w,z are popped off the stack
- closeScope: w, x, and g() are popped off the stack
- closeScope: f() and g() are popped off the stack

The second method for the program in figure 8.1:

- f() and g() are identified at depth 0 scope
- w, x, and g() are identified at depth 1 scope
- f(),x,z,and w are identified at depth 2 scope

(No exercises for the *Dragon Bible*)