Lab Five

Marcus A. Zimmermann

Marcus.Zimmermann1@Marist.edu

March 19, 2018

Crafting a Compiler

Exercise 8.1

The two data structures most commonly used to implement symbol tables in production compilers are binary search trees and hash tables. What are the advantages and disadvantages of using each of these data structures for symbol tables?

Binary search trees are advantageous for their combination of insertion and retrieval efficiency. We can expect both of these operations to complete in O($log\ n$) time. Binary search trees are also a highly documented and commonplace data structure, making their implementation as a symbol table relatively easy compared to some alternatives. The only notable disadvantage is that average-case performance does not necessarily hold for symbol tables. Since programmers do not usually choose identifier names at random, lookup could take O(n) time. However, worst-case scenarios can be avoided if the tree is maintained balanced form.

Hash tables are the most common way to manage a symbol table. If implemented properly, insertion and retrieval operations can be performed in constant time, making it a highly efficient alternative to binary search trees. However, this data structure tend to consume much more memory for sufficiently large tables, and potential for collisions must be accounted for.

Exercise 8.3

Describe two alternative approaches to handling multiple scopes in a symbol table, and list the actions required to open and close a scope for each alternative. Trace the sequence of actions that would be performed for each alternative during compilation of the program in Figure 8.1.

There are two common approaches to block-structured symbol tables. A symbol table can be created for each new scope, or a single, global table can be used to keep track of all symbols.

If a symbol table is created for each new scope, a stack can be used to keep track of the current and open scopes. If a scope is opened, push it onto the stack, and if a scope is closed, pop it off the stack. We can perform search and retrieval operations by simple traversing the stack (currently open, nested scopes).

If a single, global table is used, then a specially formatted symbol table entry and two index structures are required. A hash table allows for efficient insertion and retrieval of names, and a scope display keeps track of symbols declared at the same level. When a symbol is entered into the table, a level field links it with symbols of the same scope, and a var field maintains a stack of active scope declarations for the symbol and its associated variables. Lastly, a depth field records the nesting depth of a symbol.

8.1 Program

```
1  import f(float, float, float)
2  import g(int)
3  {
4      int w,x
5      {
6          float x,z
7          f(x,w,z)
8      }
9      g(x)
10 }
```