CMPT 432 - Fall 2019 - Dr. Labouseur

Lab Five

Timothy Polizzi
Timothy.Polizzi1@Marist.edu

November 12, 2019

1 *Crafting a Compiler*

1. 8.1 - There are various benefits and demerits to both binary search trees and hash tables when they are applied as symbol tables in production compilers. In regards to binary search trees, they are of average case to get or add an item in O(n log n) time and they are quite simple. This however is not always the case and due to symbol tables not being balanced, they can cause the tree to run in up to O(n) time. On the other hand we have hash tables which offer up to constant time adds or retrieval, however this also requires a good deal of space and can also run into time issues if the hash table algorithm uses chaining as it would allow the program to run in worst case O(n) time.

2. 8.3 - There are two ways of handling multiple scopes in regards to symbol tables, where they are quite straightforward: one table or one per scope. With a single table, it is quite complex to think about as the only way to distinguish levels of scope is to declare a scope name of the individual levels of scope, while in the case of individual scopes one needs to just make a symbol table for the current scope and push it to a scope stack. In order to append a scope to a single table method, one could add a individual item to the hash table however they would need to then specify what level of scope what value has. To remove a scope from a one table method, one would need to remove the values of the given level of scope from the table which would be slightly more costly then the alternative of a multiple table's method of just popping the current scope off the scope stack.

In the case of figure 8.1, a single table method would first generate a hash table that would store an item that stores the value of a variable, it's type, it's scope and it's hash value. The first two items f and g would then be added to the table. Once the program enters the next scope it would add w and x to the table and have them listed as the next scope. Finally in the last scope the table would add x and z to the table as the current scope which would not change the old value of x, but add a new one at the current scope. Once the method call on that scope was complete, it would remove all items at the lowest scope and then follow suit once the method call on the next line is complete.
Alternatively, in the case of a table per scope method, the first thing that would occur is a stack would be generated and then a first table would be pushed to it containing all of the information about the current scope, such as the values of the variables and what type they are. Once the scope has been pushed to the stack and the next scope is to be added, the same process would occur. This would happen until the innermost scope is reached, at which point it would pop the current scope off the stack once the scope is concluded. If an item needed to be searched for, it would first be searched for on the topmost level of scope and then would search the next layer of scope until it reached the bottom of the stack.