David Lane

004638553

Project 4 Report

For my data structures I used primarily a hashtable of linked lists of Nodes with stacks of integers representing lines and a string representing the identifier. I knew that searching a hashtable uses (near constant time so I figured it would be the most efficient way to search for an Id and what lines correspond with it. I chose the stack so that I could push on new declarations and then pop them once their scope was exited, then check the top item to find the most recent line. The linked list is to manage when multiple items are in the same bucket. I also have a stack to store pointers to newly allocated nodes to delete them at destruction and I kept the vector from the example code.

Enterscope is constant time (works same way as the example code). Exit scope is linear time for the number of items until it reaches the empty string in the m\_ids vector times the time complexity of the popline function which is linear time for the number of nodes in the linked list in that bucket of the hashtable. Declare is linear time for the number of items until it reaches the empty string in the m\_ids vector + linear time for the number of nodes in the linked list in that bucket of the hashtable (this is close to constant time). The find function is linear time for the number of nodes in the linked list in that bucket of the hashtable (this is close to constant time).

I will give an overview of each of the four functions and then pseudocode for the declare function.

Enterscope: adds an empty string to the ids vector, this marks the beginning of a scope

Exit scope: for each item on the id stack, that item was declare in the scope we are exiting, so it pops the stack of that id’s hashtable node so that the top of that stack is the most recent line that that identifier was declared at that is within the new scope. Then it pops that id from the id stack, and finally once it reaches the empty string, pops that as well.

Declare: checks the ids in the vector until it reaches the empty string, if these match the id being declare it returns false, otherwise it adds the id to the m\_ids vector, then goes into the proper bucket of the hashtable and looks for the string in that buckets linked list, if it finds it, it pushes the line number onto the stack of lines, if not it creates and appends a new node to the linked list and pushes the line number onto the new node’s stack. Any time a new node is created, its pointer is added to the to delete stack to avoid a memory leak at destruction.

Find: goes to the proper hashtable bucket than cycles through linked list until it finds a node with a matching string then returns the top of that node’s line stack or if it doesn’t find it returns -1.

Declare:

If id is empty, invalid return false

Until we find the empty string, cycle through the ids vector

If we find an id that matches the id passed into the function, return false because it was already declared in this scope.

Otherwise add the id to the m\_ids vector

Hash the id string

Go to its bucket,

If that bucket is empty, add a new node there with the corresponding string and push the line num onto its lines stack and return true

If the bucket is not empty, until you find the corresponding string, cycle through the bucket’s linked list

If you find this id string, push its line num onto the matching node’s line stack and return true

If you don’t find this id string,

Add a new node to the end of the linked list with the corresponding string and push the line num onto its lines stack then return true