Attiano purpura-pontoniere

504-847-318

Project 4 Report

a) I used a hash table in conjunction with a vector of iterators to keep track of the variables to delete when leaving a scope. I used a hash table because it has, on average, constant time lookups, insertions and deletions. The hash table was constructed from a fixed-size array of singly linked lists of "symbols" where symbols were structures I created. Each symbol just holds two integer variables (scope of the symbol, and line number that the symbol was define), and one string object (identifier name of the symbol). To keep track of scope, I had a private integer data member called current_scope that recorded the scope currently in, and I would just increment or decrement it when entering or exiting scope. Since each symbol object knows its own scope all the information I need is what scope I am currently in.

   To enter a scope I just increment scope, to exit a scope I first check that there is a scope to exit, and then I delete all items from the old scope. To declare an identifier I error check the input, make sure there isn't already an identifier with the same name in the scope, and then if that's not the case I just create a new symbol with all the needed information and hash its id and then add it to the hash table.

   To look up an identifier I just hash the input after error checking it, and then look in the corresponding bucket to see if its there, if I found it I return the line number of the matching symbol with matching scope, if its not there I return -1

b) Enterscope is constant time. Exitscope is O(n), if you double the number of items in the scope to be delete, you double the length of the while loop, and everything else is constant time. Declare is O(n), if you double the number of items in each bucket, you double the length of the for loop, and everything else is constant time. Find is O(n), if you double the number of items in each bucket, you double the length of the for loop, and everything else is constant time.

   However Exitscope(), Declare, and Find are all O(1) ON AVERAGE because its constant time to do lookups, insertions, and deletions with hash tables when there are essentially a constant amount of items in each bucket. The constant of proportionality on the O(n) is so small, it is growing at such a slow rate, that it is essentially constant time when the number of items in each bucket isn't extreme, e.g. N items in each bucket.

c) pseudocode for non-trivial algorithms. Enterscope is trivial

EXITSCOPE:

If there is no scope to exit

Return false

While there are still items to be deleted from the current scope

Hash the id of the item to be deleted

Using the hash as the bucket number find the item in the hash table

Delete the item from the hash table

Decrement the scope

Return true

DECLARE:

If the id input is empty, or the linenumber is negative

Return false

Hash the id input to find the bucket number to put the id

Check if the bucket in question already has an identifier declared in the same scope:

For each item in the bucket

If the item in the bucket has the same id and the same scope as the input id

Return false

Create a symbol object with the id of the identifier, the line Number of the identifier, and the scope of the identifier

Add the symbol object to the bucket corresponding to the hashed id

Add the symbol object to a vector to keep track of what variables to delete when exiting scope

FIND:

If the input id is empty

Return -1

Hash the input id

For each item in the bucket

if the item id matches the input id

return the line number of the first item matched (has the greatest scope)

if no matches were found in the bucket

return -1

d) No serious inefficiencies I know of. One notable problem I had was how to efficiently deal with the deletion of all objects in a scope when exiting scope. Also it was difficult to come up with a way of keeping track of scope intelligently.

There was one bug I mentioned to Prof.Smallberg and he told me not to worry about:

When variables are declared with line numbers ('a 1'), and exit scope ('{') is called without having entered a scope, the SymbolTable.Slow.cpp executes a while loop that deletes all the variables that were declared before exit scope was called. The exit scope call returns fine, but its when trying to access the variables (find 'a') that were declared in the outside scope that my implementation differs from SymbolTable.Slow.cpp. SymbolTable.Slow.cpp returns -1 since it cant find it since the vector of ids was emptied by calling exit scope. My implementation returns the line number of the variable, in this case '1'. My implementation returns false before deleting any variables from scope, whereas SymbolTable.Slow.cpp doesn't.