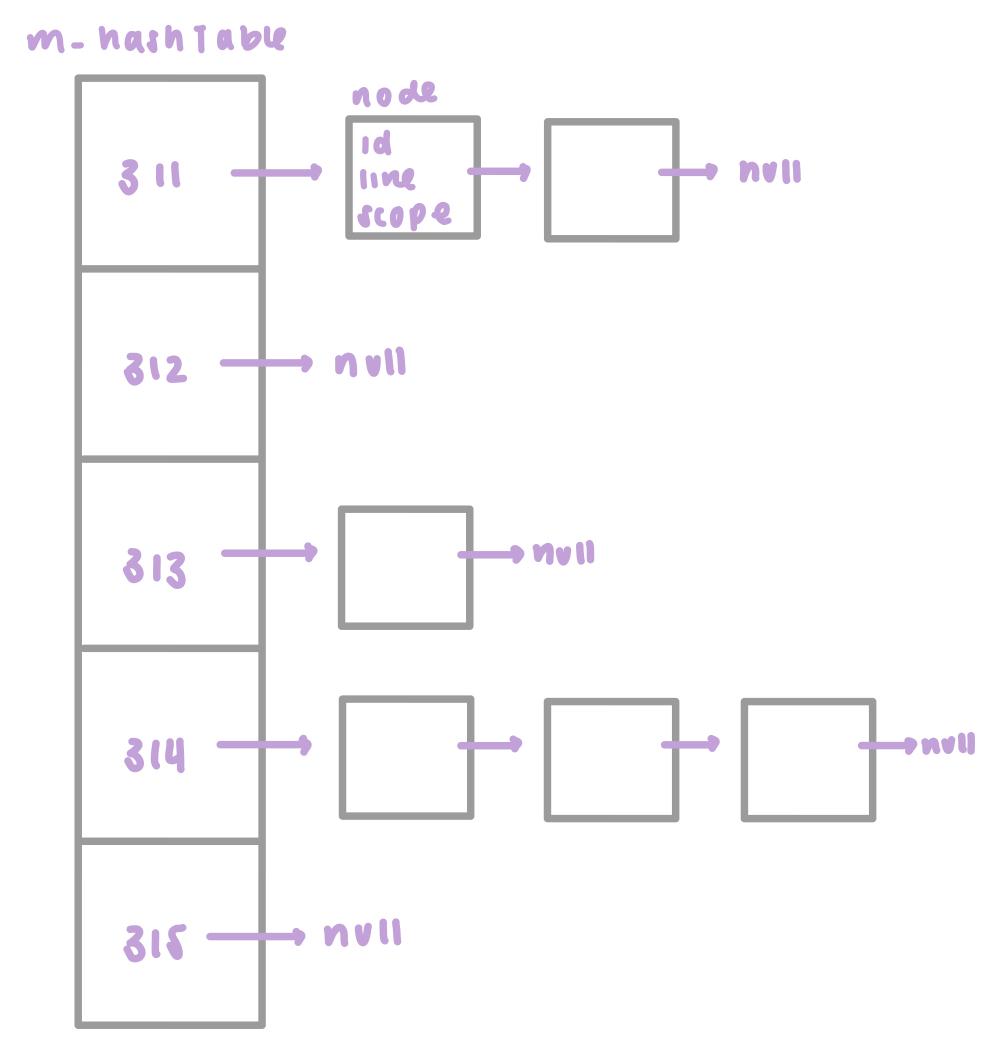
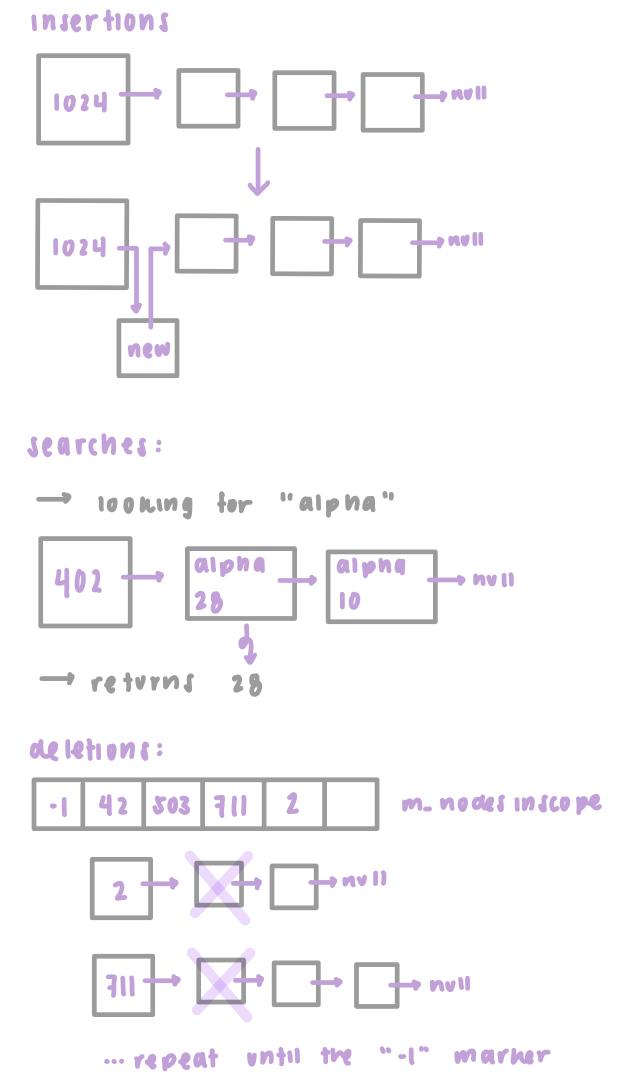
Ying Chou

CS 32: Project 4

**Description of algorithms + data structures:**

* For the name table, I chose to implement it with a hash table, as this was the most efficient choice, with constant time search, insertion, and deletion. The hash table itself I implemented as an array of node forward lists. 
* For insertions, the hashing function will derive the correct bucket number based on its string identifier and check the forward list for the same identifier declared in this scope. If not found, it will push the new node onto the front of the forward list.
* For searches, the hashing function will derive the correct bucket number based on its string identifier and look through that corresponding forward list. Since we want the most recently entered “version” of the identifier, it will return the line number of the first node that matches the identifier.
  + With the way my data structure is implemented, the identifier declared in the current scope will always be before identifiers declared in earlier scopes. This is because everything is inserted at the front of the list.
* For deletions, the bucket numbers of identifiers that need to be deleted are recorded in a stack. When it comes time to delete (exiting a scope), it will visit each bucket and pop off the first item (most recently declared).
  + Every time a new scope is entered, a “-1” is pushed onto the stack as a marker. So when exiting a scope, the identifiers recorded up to that marker will be deleted.



**Time complexities:**

* enterScope(): O(1)
* exitScope(): O(S), where S is the number of identifiers being deleted
* declare(string id, int line): O(B), where B is the average number of nodes per bucket
* find(string id): O(B), where B is the average number of nodes per bucket
  + essentially constant time since in many cases the function will just return the first node in the list.

**Pseudocode:**

exitScope()

until there are no more identifiers to be removed

delete the node from the hash table

pop the bucket number off the stack

if the stack is empty (tried to exit a scope we weren’t in)

return false

pop the stack to remove the scope entry marker

decrement the scope counter

return true

declare(string id, int line)

if id is “”

return false

derive a bucket number based on the id

for every node in the corresponding bucket number

if the same id has been used in this scope

return false

add the new identifier to the hash table

add the bucket number to the stack (to be deleted when exiting the scope)

find(string id)

derive a bucket number based on the id

if the forward list at that bucket number is empty

return false

for every node in the corresponding bucket number

if the id matches the id passed

return the line number stored in that node

if not found

return -1

**Bugs, inefficiencies, and notable problems:**

* No bugs or serious inefficiencies that I could find
* A notable problem that I had was a big discrepancy in time between testing on Xcode and on g32. When running my program in g32, it takes double the time it takes when running in Xcode (I used g32fast and the release build configuration in Xcode). For example, in Xcode it took around 9 ms to run 200,000 lines, but under g32 it took around 18 ms. I couldn’t figure out the reason it was doing this.