* a description of your algorithms and data structures (good diagrams may help reduce the amount you have to write), and why you made the choices you did. You can assume we know all the data structures and algorithms discussed in class and their names.

-I use open hash table to store all my data.

-I create a struct called Identifier to represent each Identifier appeared in the commands. Each identifier struct contains a vector of integer of line number of location, a vector integer of scope number and a vector of string of the original name.

Each bucket of my hash table hold one identifier, and the later declared identifier that have the same name as the existing one will be stored at the back of the vector. If there is no such identifier existing, the bucket will be set to nullptr.

The string vector can help recognize and identify the identifier we need in case that different identifier names may have same integer value after conversion.

The integer vector help store the line number of each identifier declared in the commands.

The integer of scope number helps us identify whether there is a previously existing identifier in the same scope. This is used mostly by exist function in find.

In the SymbolTableImpl class, other than the functions declared already in the original interface, I have an array of 449 pointers to buckets (identifier), an integer to keep track of the scope we are in when processing, a private member function exist to see whether an identifier already exists in the same scope, and an vector of integer to store the order of the converted integer of the original string of identifier name. I also add a constructor to initialize all the pointers of buckets to nullptr and a destructor to delete all the buckets added by new.

The integer of scope number helps us keep track of the scope number, and will be used when we declare an identifier.

The vector of integer stores every identifier’s integer conversion so that we have the order of declaration, which will be helpful when we exit a scope and try to delete the identifier in reverse order.

hashFunction:

I use the hashFunction provided by the library <functional> to convert the string into integer.

* a note about the time complexity of the SymbolTable functions. For example, in our inefficient implementation, enterScope is constant time, exitScope is linear in the number of identifiers going out of scope, declare is linear in the number of identifiers in the current scope (because of the check for duplicates), and find is linear in the number of identifiers currently accessible.

Exist is linear in the number of identifier name with the same conversion integer. (constant time in most cases)

Constructor/destructor is constant time with respect to total table number.

enterScope is constant time (linear in the number of identifier with the same conversion integer.)

exitScope is linear in the number of identifier going out of scope

declare is constant time (linear in the number of identifier with the same conversion integer.)

find is constant time (linear in the number of identifier with the same conversion integer.)

* [pseudocode](http://web.cs.ucla.edu/classes/spring17/cs32/pseudocode.html) for non-trivial algorithms.

hashFunction

*create a string hasher*

*hash out string to a integer*

*use modulo to get the bucket number that fits into the hash table*

exist

*set n to the index of the last position in the vector of the corresponding bucket*

*number*

*while n is still a valid index and our string name is not what we want yet…*

*decrement n*

*if n is less than zero (there is no such string name existing)*

*return false*

*else*

*if the identifier at position we find is in the same scope as the current*

*scope*

*return true*

*else*

*return false*

enterScope

*increment the scope number*

constructor

*for every spot in the buckets*

*set the pointer to nullptr*

destructor

*for every spot in the buckets*

*delete the pointer*

exitScope

*if we are not in any scope, return false*

*while we are still in the scope*

*remove the last identifier and its information with the corresponding*

*position*

*if the identifier does not exist in the whole program any more, set the*

*bucket pointer to nullptr*

*remove the identifier conversion integer from record*

*decrement the scope number*

*return true*

declare

*if the string id is empty*

*return false*

*if the bucket of the identifier’s conversion integer is not nullptr but the string*

*name does not exist in the buckets*

*return false*

*(if we are here, that means the declaration is valid)*

*if the bucket is a nullptr*

*create a new identifier bucket and add one identifier and its information*

*if the bucket is not nullptr*

*add one identifier and its information to the identifier’s bucket*

*record the identifier’s conversion integer in the vector*

*return true*

find

*if the string id is empty*

*return -1*

*if the identifier’s bucket is nullptr*

*return -1*

*else*

*set n to the index of the last position in the identifier’s bucket*

*while the index is still valid and the string id is still not in the identifier’s*

*bucket yet ….*

*Decrement n*

*If n is invalid*

*Return -1*

*Else*

*Return the line number of the position we find*

* a note about any known bugs, serious inefficiencies, or notable problems you had.

When we use the hashFunction, there may be more than one identifier names that has the same integer value after conversion, thus I introduce stringId vector in the identifier.

At first, I didn’t use an integer vector to represent the order all identifiers put in. I use find to search for every identifier with in the current scope, which makes the process extremely inefficient.