1. Generally, I designed a hash table (an array of 500 elements) with a hash function that allocates each certain node to its appropriate bucket. Each node has a string variable named id, an int variable named lineNum and an int variable named depth. The hash function only concerns about id. However, for exitscope(), declare() and find(), depth could help us check whether it’s valid to declare an identifier and lineNum could record where we have declared certain node and return its value when find() is called. The type of elements in that array is list<Node>; that is, each element is a list of nodes. We can directly access each node by pre-defined functions of list like back(), end() and so on.

I used an array as a hash table since array is a convenient data structure that could satisfy all requirements to classify nodes. And for elements of this array, I chose list rather than vector since in list library, pre-defined functions are various and have fewer limits to be used. For example, when I want to access each node, an iterator and its algorithm in the list is more simple than those in the vector.

1. The time complexity of enterscope() is constant time; the time complexity of exitscope() is linear in the number of identifiers in the same bucket with the same depth; the time complexity of declare() is linear in the number of identifiers in a certain bucket since we need to check duplicates; the time complexity of find() is linear in the number of identifiers currently accessible in a certain bucket.
2. **Exitscope():**

If there is no outer scope, we cannot exit a scope so that return false;

From the first bucket to the last one:

If this bucket is empty, continue;

If the bucket is not empty, check from the last node to the first one in the

list contained in the bucket, if the depth of the node is the same as the

currentDepth, pop that node from the list and continue to check next one

Decrement the currentDepth and return true

**Declare()**

If the string in argument is empty, return false;

For strings that are not empty, find the appropriate bucket for it by hash function

If the bucket is empty, push\_back the string with parameters to the list contained in that bucket

If not, create an iterator starting from the end() of the list contained in that bucket:

Repeatedly (if the iterator hits the begin() of the list, end the

while-loop):

Decrement the iterator

If the node’s depth that iterator is checking is different from

currentDepth, break;

If the node’s id is the same as the one that we want to declare, and their depths are same, we encounter a duplicate and return false;

Push\_back the node to that bucket and return true

**Find()**

If the string is empty, return -1

For strings that are not empty, find the appropriate bucket for it by hash function

Create an iterator starting from the end() of the list contained in that bucket:

Repeatedly (if the iterator hits the begin() of the list, end the

while-loop):

Decrement the iterator

If the node’s id is the same as the one that we want to find, return the

lineNum of that node;

Return -1 if not find anything

1. When I tried to design the data structure, I first used a list of vectors of nodes. For enterscope(), I would just create a new vector of Node and push\_front to the list. For exitscope(), if the size of the list is one, we could not execute exitscope() because there was no outer scope and if the size of the list is bigger than 1, just pop\_front the list. For declare(), the subtle part is that if the list was empty, I had to create a vector of nodes and push\_front to the list and create a node and push\_back to the vector. This inconvenience might waste plenty of time. For find(), the inconvenient part comes from a fact that I had to access each node in a certain vector and each vector in the list. The time complexity of this algorithm is O(N2). Therefore, my first method is not an appropriate one and wastes plenty of time.