```cpp
#ifndef SYMBOLTABLE_H
#define SYMBOLTABLE_H

#include<iostream>
#include<string>
#include<stack>
#include<fstream>

using namespace std;

//simple node class with variable for if a node is to be seen
//all seen variables are true, if false needed, a function will be added later
struct Data
{
    string type;
    string name;
    int address;
};

class Node
{
public:
    Node() : data({ "int", "", 5 }), scope(0), ptrToNext(NULL) {}
    Node(const Data& theData, int scopeVar, Node *newPtrToNext) : data(theData), 
      scope(scopeVar), ptrToNext(newPtrToNext) {}
    Node *getNext() const { return ptrToNext; }
    Data getData() { return data; }
    int getScope() { return scope; }
    void setData(const Data& theData) { data = theData; }
    void setNext(Node *newPtrToNext) { ptrToNext = newPtrToNext; }
    void setScope(int scopeVar) { scope = scopeVar; }
    ~Node() {}

private:
    Data data;
    int scope;
    Node *ptrToNext;
};


class LinkedList
{
public:
    //Default Constructor for a linked list
    LinkedList();

    //set modifier
    void insert(const Data& insertedData, int scope);

    //tests if the list is empty (used for the hashtable class)
    bool empty();

    //prints the list
    void print() const;

    //deletes all variables in the given scope from the hash table and puts them into a stack to be inputed into a larger stack
    void del(int scope);

    void remove() { tempStack.pop(); }

    //returns the layer to insert the new variable
    Node *search(const string& searchTerm);

    //outputs the stack
    stack<Data> getStack() { return tempStack; }


    //destructor
    ~LinkedList() {}
private:
    Node *ptrToFirst;
    stack<Data> tempStack;
};

class HashTable
{
public:
    //Default Constructor for a Hash Table
    HashTable();

    //Converts a String to a key for sorting
    int convertString(const string& strToConvert);

    //inserts the string in the position designated by the key
    //if a string is already in the same scope, the variable doesn't get inserted (error code needs to be written)
    void insert(const Data& data);

    //I'm not sure why you want two different functions for find local vs. find all
    //This 'find' just finds the first available instance of the variable and ignores the rest
    Data* find(const string& searchString);

    //initiates a new scope and indexes the localScope level
    void newScope();

    //deletes all of the variables in the closed scope from the hash table and puts them in a stack
    //closes the scope and decriments the scope counter
    //instructions: decriment the scope counter anytime you are closing a scope
    //does not need to be decrimented if a new scope is opened inside an existing scope
    void closeScope();

    //closes all scopes and removes all variables from the hash table
    //prints all variables in the stack
    void print();

    //destructor
    ~HashTable() {}

private:
    LinkedList *table[23];
    int counter;
    int localScope;
    stack<Data> obsoleteVariables;
};

#endif
```