Bangladesh University of Engineering and Technology Department of Computer Science and Engineering CSE 310 - Compiler Sessional

Session: July 2023

Assignment

Implementation of SymbolTable

---

1 Introduction

The purpose of this course is to construct a simple compiler. In the first step to do so, we are going to implement a symbol-table. A symbol-table is a data structure maintained by the compilers in order to store information about the occurrence of various entities such as identifiers, objects, function names etc. Information of different entities may include type, value, scope etc. At the starting phase of constructing a compiler, we will construct a symbol-table which maintains a list of hash tables where each hash table contains information of symbols encountered in a scope of the source program.

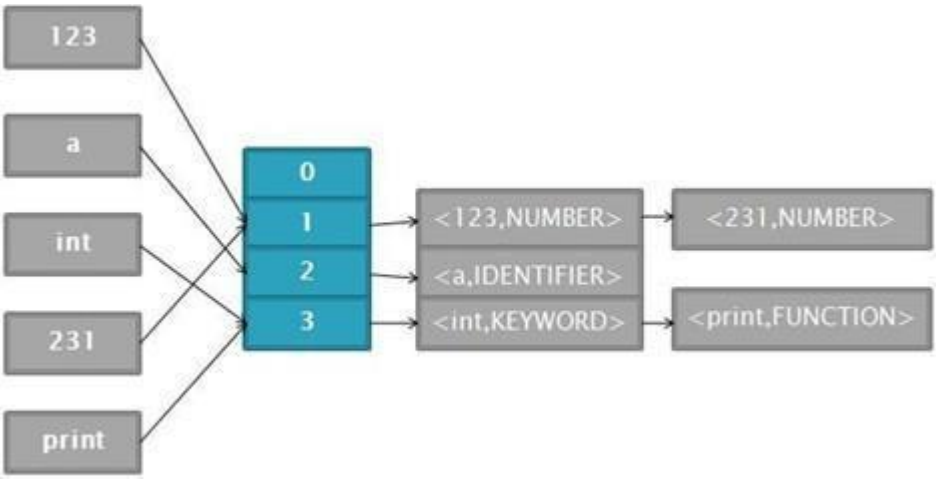

Figure 1: Symbol Table (of a single scope) Using Hashing.

2 Symbol Table

In this assignment we will construct a symbol table that can store type and scope information of a symbol found in the source program. If the source program consists of a single scope then we can use a hash table to store information of different symbols. Hashing is done using the symbol as a key. Figure 1 illustrates such a symbol table.

Now consider the following C code:

```
1. int a,b,c;
2. int func(int x) {
3.    int t=0;
4.    float c;
5.    if(x==1){
6.          int a=0;
7.          t=1;
7.    }
      if(){}
8.    return t;
9. }
10.   int main(){
11.   int x=2;
12.   func(x);
13.   return 0;
14.}
```

To successfully compile this program, we need to store the scope information. Suppose that we are currently dealing with line no 6. In that case, global variables `a`, `b` and `c`, function parameter `x`, function `func`'s local variable `t` and the variable `a` declared within the `if` block, are visible. Moreover the variable `a` declared within the if block hides the global variable `a`. How can we store symbols in a symbol-table which can help us to handle scope easily? One way is to maintain a separate hash

table for each scope. We call each hash table a *Scope-Table*. In our *Symbol-Table*, we will maintain a list of scope-tables. You can also think of the list of scope-tables as a stack of scope-tables. Whenever we enter a new block, we create a new scope-table and insert it at the top of the list. Whenever we find a new symbol within that block of the source code, we insert it in the newly created scope-table i.e. he scope-table at the top of the stack. When we need to get the information of a symbol, at first we will search at the topmost scope-table. If we could not find it there, we would search in its parent scope-table and so on. When a block ends, we simply pop the topmost scope table. Suppose, we give a unique number to each block, 1 to global scope, 2 to the function func, 3 to the if block and 4 to the main function. Figure 2 illustrates the state of the symbol table when we are in line no. 6 of the given code.

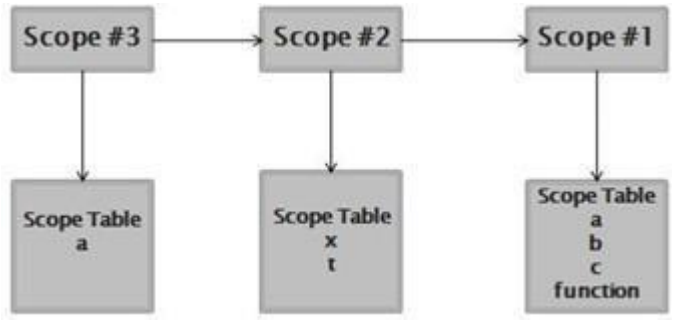

Figure 2: Symbol Table with Scope Handling.

3 Tasks

You have to implement the following three classes.

- **SymbolInfo:** This class contains the information regarding a symbol faced in the source program. In the first step, we will limit ourselves to only two members. One is for the **Name** of the symbol and other is the **Type** of the symbol. You can use C++ Standard Template Library string. These two member variables will be private and you should write any getter setter method as needed. This class will

also contain a pointer to a **SymbolInfo** object as you need to implement a *chaining mechanism* to resolve collisions in the hash table. Also keep in mind that you may have to extend this class as we progress to develop our compiler.

❑ **ScopeTable:** This class implements a hash table. You may need an array (array of pointers of SymbolInfo type) and a hash function. The hash function will determine the corresponding bucket no. the SymbolInfo object will go into. The hash function would be *sdbmhash(Name) % total_buckets* where sdbmhash is a standard string hash function used on the Name string (remember the **Name** is taken as a string), and *total_buckets* is the total no. of buckets in the hash table. You will also need a pointer of **ScopeTable** type object named *parentScope* as a member variable so that you can maintain a list of scope tables in the symbol table.

Also give each table a unique **id** which will be a string. The id will be determined as:

<parent_id> . <current_id>

Where *parent_id* is the **id** of the parent, and *current_id* is a serial no. relative to its parent. That is, after it's parent scope-table was created, if there were *p* scope-tables previously deleted at the current level, the *current_id* of the scope table will be *p+1.* For example, suppose the parent scope-table has id *1.3.2* and 8 scope-tables were deleted before at this level (notice the new scope table is at the 4th level, assuming level count started from 1) after *1.3.2* was created, then the scope-id of the current scope-table will be *1.3.2.9* , see the sample io files provided for further clarification.

You also need to add the following functionalities to your Scope Table:

- ❑ **Insert:** Insert into symbol table if already not inserted in this scope table. Return type of this function should be boolean indicating whether insertion is successful or not.
- ❑ **Look up:** Search the hash table for a particular symbol. Return a SymbolInfo pointer.

- **Delete:** Delete an entry from the symbol table. Return true in case of successful deletion and false otherwise. → boolean type
- **Print:** Print the scope table in the console.

You should also write a constructor that takes an integer ***n*** as a parameter and allocates ***n*** buckets for the hash table. You should also write a destructor to deallocate memory.

- **SymbolTable:** This class implements a list of scope tables. The class should have a pointer of *ScopeTable* type which indicates the current scopetable. This class should contain the following functionalities:
  - **Enter Scope:** Create a new ScopeTable and make it current one. Also make the previous current table as its parentScopeTable.
  - **Exit Scope:** Remove the current ScopeTable.
  - **Insert:** Insert a symbol in current ScopeTable. Return true for successful insertion and false otherwise. → boolean type
  - **Remove:** Remove a symbol from current ScopeTable. Return true for successful removal and false otherwise. → boolean type
  - **Look up:** Look up a symbol in the ScopeTable. At first search in the current ScopeTable, if not found then search in its parent ScopeTable and so on. Return a pointer to the SymbolInfo object representing the searched symbol.
  - **Print Current ScopeTable:** Print the current ScopeTable.
  - **Print All ScopeTable:** Print all the ScopeTables currently in the SymbolTable.

4 Input

The first line of the input is a number indicating the number of buckets in each hash table. Each of the following lines will start with a code letter indicating the operation you want to perform.

The letters will be among 'I', 'L', 'D', 'P', 'S' and 'E'.

- 'I' stands for insert which is followed by two space separated strings where the first one is symbol name and the second one is symbol type. As you already know, the symbol name will be the key of the record to be stored in the symbol-table.
- 'L' means lookup which is followed by a string containing the symbol to be looked up in the table.
- 'D' stands for delete which is also followed by a string to be deleted.
- 'P' stands for print the symbol table which will be followed by another code letter which is either 'A' or 'C'. If 'A' is followed by 'P' then you will need to print all the scope tables otherwise you will print only the current scope table.
- Finally, 'S' is for entering into a new scope and 'E' is for exiting the current scope.

You should also check the sample input file.

5 Output

All the output must be written to file. Check the sample output file. Note that the output of your code will be matched with the solution output with a standard diff tool. So you must match the output format exactly as in the sample.

6 Important Notes

Please try to follow the instructions listed below while implementing your assignment:

❏ Read the specification carefully by yourself before implementation. Don't get the specification from your friend's mouth.

❏ Implement using the C++ programming language.

❏ You are not allowed to use any C++ STL objects such as Vector, Linked List etc.

- ❏ Avoid hard coding.
- ❏ Use dynamic memory allocation.
- ❏ Take input from file. The name of the input file must be *input.txt*. You must write the output to a file. The name of the output file will be *output.txt*. However output to console is optional which you may do for your own debug purposes.
- ❏ Bear in mind that the root scope cannot be exited.
- ❏ Try to get accustomed to a Linux platform. If you are a Windows user, it is suggested that you use WSL (Windows Subsystem for Linux).

Also, here are some stuffs you should be careful about while implementing this offline, as you will use this symbol-table throughout the rest of the course and any error kept in this implementation will haunt you in the later assignments.

- ❏ Make sure you are accessing the exact position where your symbolInfo pointer is placed while you are doing lookup in the symbolTable.
- ❏ Make sure you do implement the destructor function for all three classes, and the elements are deleted *in proper order*.
- ❏ While iterating through the pointers, make sure all the relevant pointers are at the desired positions after the iterations are over.
- ❏ Besides the given sample IO, additionally test your code yourself, especially the potential corner cases.

7 Submission

All submissions will be taken via Moodle. Please follow the steps given below to submit your assignment.

1. In your local machine create a new folder whose name is your 7 digit Student Id.
2. Put the file(s) containing your solution in the abovementioned folder. Every file

should contain your 7 digit Student Id as prefix. **Do not put any object file or exe file in that folder**.

3. Compress the folder into a zip file which should be named as your 7 digit student Id. **Extension of the zip file should be '.zip'**.
4. Submit the zip file.

8 Rules

Any type of plagiarism is strongly forbidden. -100% marks will be assigned to a student who will be found to be involved in plagiarism. It does not matter who is the server and who is the client. You must take the caution seriously as in a 0.75 credit course even a single penalty makes a HUGE impact.

9 Deadline

Deadline is set on **Sunday, Dec 03, 2023 at 10:00 PM** for ALL lab groups.