Wei Fei (2538810)
EECS 665 Lab7 Report
Elluru, Bharath Chandra
11/05/15

Objective

In this lab, we need to learn the csym compiler framework and build and print a symbol table for the C-like language that the framework parses. To reach the correct result, we have to complete the rest of implementation of given codes. For the implementation part, we need to build an executable named "csym" that can be tested. In the file "sym.c", we should modify three functions: enterblock(), leaveblock(), and dump(); in the file "sem_sym.c", we should modify four functions: fname(), ftail(), blockdcl(), and btail(). Finally, we are supposed to build up the symbol table correctly and print out the symbol table at the correct time.

Implementation

sym.c

1. In the function "**dump()**", we need to dump identifiers with block level. The function will go through all of identifiers in the loop within the table size. If the block level greater and equal to blev, we should print all all identifiers.
2. In the function "**enterblock()**", we should call the function new_block() and increase the block level by one.
3. In the function "**leaveblock()**", we need to implement the strategy that how to leave the scope when the pointer find the closed bracket. As similar as dump() function, we need to create a loop to free the pointer one by one. At the end of the function, we should exit the block.

sem_sym.c

1. In the function "**blockdcl()**", we need to call the function enterblock() since we need to enter a new scope.
2. In the function "**btail()**", we need to call the function leaveblock() since we need to leave the scope.

3. In the function "**fname()**", we call the function enterblock() again since we need to know the current status of block level.
4. In the function "**ftail()**", we call the function leaveblock() again since the parser get to the last line of the function.

Discussion

- ## How the Program Works at a High Level

In order to build the symbol table, we need to keep tracking the pointer position of current block level by using loops. In sample C code, if we see some variables out of main function, we need to add block level by one. When we the the open bracket of the main function, we need to increase the block level by one again. Same as this strategy, if the track pointer seeks more open bracket, increase by one for each open bracket. On the other hand, if the track pointer finds any close brackets, the block level will be decreased by one. If blocks can be nested, several declarations of the same identifier can appear within a single block. Meanwhile, if the parser sees any variable declaration, the program will record the information of the variable by creating an id_entry with storing all information of the variables.

- ## Troubles

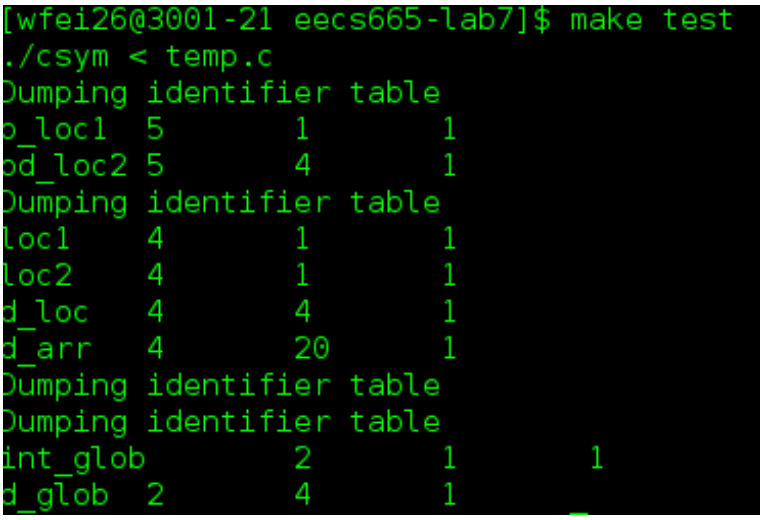
1. In the implementation part of this lab, it is hard to start since I need to understand some functions before writing the codes. Sometimes read other people's codes even harder than writing from scratch. Especially in the dump() and leaveblock() function in sym.c file, I did not know how to create a pointer to point the block level. After discuss with my classmates, I knew I could use struct id_entry to define the pointer. Overall, I need to match the logic of the given code, it is not a straight forward logic, which is harder.

2. When I run my program, the symbol table results are not correct. All level of variables is greater than the normal results by one. When I debug my program, I find the problem is because I called the enterblock() function again in fhead() function in sem_sym.c file. So it adds one level again. After I comment the function enterblock() in fhead() function, the results turns correct.

Conclusion

After we adding the codes in the required functions for each file, the program has been executed. I understand the csy, compiler framework. I know the symbol table is the data structures that are used by compilers to hold information about source program constructs. In the implementation part, we built and printed a symbol table for the C-like language that the framework parses. Eventually, we can see the correct symbol table in the output.

Sample Outputs



(Remark: the order of occurrence may random.)