Jay Offerdahl - 2760730
Lab Time: Friday, 4:00pm - 5:50pm

# Lab 7 Report

The objective of this lab was to expand on the given code in order to correctly build and print a symbol table for a parsing of a language similar to C. For this lab, we were concerned with where variables belonged in scope. Generally, compilers use a “most-closely nested” principle. This means that an identifier x is in scope of the most closely nested declaration of x. Since symbol tables are usually implemented with tree structures, you can simply traverse up the tree (going up via parents) in order to find the variable. This essentially means that blocks are examined inside out, rather than outside in.

From a high-level perspective, this program operates in similar fashion to some of our previous labs. It uses yacc and lex to scan and parse the input file (which is why we pipe the file to our program so it is fed in via standard input). The symbol table management module takes care of adding identifiers, their level of scope (which increases based on the current level), the type value, and the boolean signifying that the ID has been declared and is currently in scope. For this lab, I have only printed out the name of the variable and the block level that variable exists. Looking at the output, we can clearly see that each time the scope is left, we print “Dumping identifier table” and one row for each entry that was added to the symbol table for that row. The reason we do this is because the variable becomes out of scope (when we leave the block), and will never be accessed again, unless the same function is invoked again (which would have the same variables, but perhaps with different values).

Looking at the design of the project, I’ll start with the file *sem_sym.c*. For this lab, we were required to implement four functions: blockdcl(), btail(), fname(...), and ftail(). This implementation was easy to glean from the lab slides. When a block is declared (blockdcl()), we should enter a block. Also, when we detect a function declaration (fname(...)), we should enter a block as well. Next, when we detect the end of a block or the end of a function body (btail(), ftail()), we should leave the current block. Moving on to *sym.c*, the required function to implement included enterblock(), leaveblock(), and dump(). To begin, enterblock() has a very simple job: increase the current level variable (integer) and generate a new block with the new_block() function. This function new_block() is of type void, so we don't have to do anything other than call it because it takes care of recording the previous top of the stack while at the same time marking a new one. Next, leaveblock() requires that we destroy all data related to the block that we were just in (as it is now out of scope until the function is invoked again, or other reasons). With this requirement, this function simply frees all entries with the block level that is greater than or equal to the current level. Although, we really only need to check if the levels are equal, because you would never have a level greater than the current in your symbol table (deeper scopes would have been left and destroyed). We must also call the exit_block() function here to free up the semantic records, and decrement the level size. This function is also responsible for printing out the relevant information associated with the entries of the scope we're leaving, which is done by a call to dump(). The method dump() prints out the symbol table entries if they're level is again, equal to the current level. This function parses through the id_table data structure to check the entries' level to see if it matches.

As far as problems I had while completing this lab, I can say that they were minimal. I was able to break the problem down into pieces and start with the base functionality. However, I did run into a slight problem when writing the leaveblock() and enterblock() functions because I was not including a call to the exit_block() and new_block() methods, respectively. However, after reading through the code and getting a better understanding of it, I added these calls and was able to finally complete the assignment.