COVER PAGE
CS323 Programming Assignments

Fill out all entries 1 - 6. If not, there will be deductions!

Peer Review (Check one)

1. Names [1. Gregory Vasquez], (ThumbUP [**X**] or ThumbDown [])

   [2. Esteban Montelongo], (ThumbUP [] or ThumbDown [ ])

   [**if 3**.], (ThumbUP [] or ThumbDown [ ])

2. Assignment Number [Project 3]

3. Turn-In Dates: **Final Iteration with Documentation** [12/16/19]

4. Executable FileName [AssemblyCodeGenerator.out]
   (**A file that can be executed without compilation by the instructor**)

5. LabRoom [CS 202]
(Execute your program in a lab in the CS building before submission)

6. Operating System/Language [Linux / C++]

To be filled out by the Instructor:

GRADE:

COMMENTS:

1. Problem Statement

The goal of this project was to utilize our syntax analyzer from project 2. The program reads in a file containing the source code where the lexical analyzer then parses it into tokens and lexemes. Next the Syntax analyzer then verifies if source code is syntactically correct. Once both of those processes are completed the program puts every identifier declared in the program into a symbol table. The program then produces the assembly code instructions for arithmetic and assignment operations.

2. How to use your program

To compile the program just run the shell script (bash run.sh). Once that command is executed you will be prompted to type in the txt file. For this project we have it running for the “in.txt” Type in that file name and hit enter. After entering the file name and hitting enter, it will first remove all files with extention .out and .o ; it will compile the .cpp files, create a linux executable named AssemblyCodeGenerator.out and it will execute the program.
Note: The shell script was created to create and compile changes faster. You can manually compile the files with (g++ main.cpp lexer.cpp syntax.cpp)

3. Design of your program

For the symbol table we used a hashmap, which holds a key-value pair, with the key being the lexem and the value being a pair of (memory address, type). Our implementation holds 3 values in total key = (lexem), value = (pair(memory address, type)). A hashmap was used because it provides a lot of useful functions to help implement many of the procedures for part (b) and it provides O(1) access time. The hashmap also helps implement the assembly code generation by allowing us to view the memory and type of declared lexemes. To print the assembly code we created a vector of strings to save every valid assignment statement. Next we created a function that converts the vector into the assembly code, saving it onto a file. We converted only the valid statements, for invalid statements we displayed a proper error and continued the parsing. Finally all that was left to do was to display the assembly file and the symbol table to the user, so we created a functions to display them both. Our program displays errors for mismatching type assignments and arithmetic, displays errors for using undeclared identifiers, and it gives priority to expressions in parenthesis. If an error is displayed the program will exit and display what caused the error along with what needs to be fixed. Once corrected re-compile and the program will work as intended. Our program also creates files for the results of the lexical analyzer(lexer_result.txt), syntax analyzer(syntax_result.txt), assembly code(assembly_result.txt), and declaration symbol table(symbol_table_result.txt).

4. Limitation

The symbol table output is not displayed in increasing memory order as the sample output given but all memory addresses, identifiers, and types all have the proper values. This ordering is due to the nature of hashmaps as each element in the table is placed into the map based on the key(lexem). We could have fixed this by inserting each element into a list and sorting them by memory values, but this increases the amount of unnecessary code. The Symbol Table still

serves its purpose intended.

5. Shortcomings

None, that we know of.
Only completed assembly code instructions for arithmetic and assignment operations because our parser does not have the “if grammar” necessary to parse it.