**CS323 Assignment 3: Lexical Syntax (RAT20F)**

**1.** **Problem Statement**

Every identifier declared in the program should be placed in a symbol table and accessed by the symbol table handling procedures. Each entry in the symbol table should hold the lexeme, and a "memory address" where an identifier is placed within the symbol table. For example, define a global integer variable called "Memory\_address" and set initially 5000 and increment it by one when a new identifier is declared and placed into the table. We will need to write a procedure that will check to see if a particular identifier is already in the table, a procedure that will insert into the table and a procedure that will printout all identifiers in the table. If an identifier is used without declaring it, then the parser should provide an error message. Also, if an identifier is already in the table and wants to declare it for the second time, then the parser should provide an error message. Also, you should check the type match.

**2.** **How to use your program**

Step1) Click on Assignment3.exe

Step2) Enter the input file you want to open. (Ex. If you want to open “test.txt”, type in “test”. **No .txt is needed**) The input file has to be in the same directory with Assignment3.exe. Program would output an output file titled with output followed by the input file that was specified.

Step3) Either type in another file that you would want to open or type in “Quit” to exit the program.

**3.** **Design of your program**

The program is a modified version of the syntax analyzer for the programming language Rat20F. Certain productions now contain additional code that generate assembly instructions and places them in a standard vector called "Instr\_table". Modifying the parser according to the simplified Rat20F and added the code to our parser that will produce the assembly code instructions. The instructions are kept in an array and at the end, the content of the array is printed out to produce the listing of assembly code. Your array should hold at least 1000 assembly instructions. The instruction starts from 1. The listing should include an array index for each entry so that it serves as a label to jump to. The compiler should also produce a listing of all the identifiers.

**4.** **Any Limitation**

*Unfortunately, we encountered a difficulty in which the program once it encounters an error the output will stop printing to the output file. We were able to fix this error that was first encountered in Assignment 2 before continuing on and implementing the new functions and procedure into Assignment 3. Although this limitation was addressed and handled, it could still use a bit of clean up and have a better functionality.*

**5.** **Any shortcomings**

*Shortcomings included were the difficulties in allowing the output file to be printed out correctly as well as the time constrain of accomplishing this task in a timely manner. Although we were able to address the issue of the output file not printing out correctly. It does run but close once it reaches the end of the file but the output file should have all the correct information to display per text file. Figuring out how to get the program not to close only led to the program resulting in errors so we decided to just have the program as it to prevent the code from not being able to compile. The Assignment3.exe file was tested various times to ensure that the application file will run on first try and not result in the same error as Assignment2’s exe file.*