**CS323 Documentation**

# 1. Problem Statement

The assignment is to write a lexical analyzer with the function called (lexer). Input some code from a text document and organizing the characters into meaningful items such as, Token and Lexemes. We’ve constructed a FSM by building your own compiler for identifiers, keywords, integer and real numbers, comments, and separators.

**2. How to use your program**

**(Use XCode/ MAC)**

**Option 1.)** Download the folder to a MAC and unzip folder. Open the Compiler\_1.xcodeproj file with a machine that has XCode. Do not skip next step! Press Command+R to compile the code. Should see Build Succeeded!

Place the input files in the Debugger directory. In order to find this directory in XCode, on the left hand side under main.cpp file you will see a Products folder, expand that folder and Right click Compiler\_ 1 and select “Show in finder”.

Drag/move the **input1.txt, input2.txt, input3.txt** files to that directory.

If you still cannot find the Directory, it’s path is: Users/<username>/Library/Developer/Xcode/DerivedData/<projectname>Build/Products/Debug

Then press Command+R to run the program:

Please enter file name: input1.txt

Please enter output file name: output1.txt

**Option 2.)** Create a new project in XCode. macOS project > Command line tool > Product Name: Compiler > Organization Name: 335 > Organization Identifier: 335 > Language: C++. You can save this project on Desktop if convenient. Move your main.cpp file into the compiler folder, replacing the old main.cpp.

Press Command+B to create the debug directory. Place the input files in the Debug directory. In order to find this directory in XCode, on the left hand side under your main.cpp file you will see a Products folder, expand that folder. Right click the file in that folder and select “Show in finder”.

Drag/move the **input1.txt, input2.txt, input3.txt** files to that directory.

If you still cannot find the Directory, it’s path is: Users/<username>/Library/Developer/Xcode/DerivedData/<projectname>Build/Products/Debug

Command+R to run the program:

Please enter file name: input1.txt

Please enter output file name: output1.txt

# 3. Design of your program

This program is using C++ as the main language coded in a MAC environment with XCode. It uses case switching to go through each condition. At first it skips any blank spaces, tabs, or new lines. It uses a Struct to keep the Token and Lexeme organized within a Vector container. Thus using object oriented programming creating a Struct for this specific file, the main creates and calls the class compiler.lexer() function. The lexer() function reads in the file, points to each character in the file one by one and does switch casing for each character. If it matches one of the conditions, it stores it into the buffer. After each switch case, it checks to see if the State of the computer is in an END condition. Once it reaches an END condition, it will store the corresponding LEXEME and TOKEN into the vector container and proceed to the next character. This will continue to happen until it has reached the end of the file.

# 4. Any Limitation

The maximum number of size of the buffer is [100], no comment can reach past that buffer size. It is a static memory size which can be increased easily by allocating more memory and assigning the buffer a larger number. The program uses a container instead of outputting the file directly, so all of the lexemes and tokens are saved into an object. Since it uses a Vector container, to store all these lexemes and tokens, it’s using more memory. A less resourceful approach would have been to just run through the file and printing out each one when it comes to a END statement. The reason for using the vector seemed to be a more organized way of keeping this data, just incase you’d like to gather more information form it further by printing out all of one group, for example. Printing out only KEYWORDS, or printing out only OPERATORS, or counting how many there are in the file, etc.

# 5. Any shortcomings

There were some advanced operations such as .length() functions and ++ functions that were a bit too difficult to implement. Some of which seemed to be language proprietary or library intensive, hopefully this wasn’t a shortcoming and the expectations are based off a simpler compiler that we had to implement.