**Project 3 Documentation**
**By: Steven Pham & Samantha Ibasitas**
**Spring 2021, CPSC 323-02**

1. **Problem Statement**

   Create a program that reads in a file and identifies every identifier and places them into a symbol table while outputting assembly code instructions based on the grammar rules.

2. **How to use your program**

   The program requires an input of a .txt file of code to analyze. Once the file is ready to be read, run the program and it will take the file named “test.txt” and output a new file that displays the assembly code instructions and the symbol table.

3. **Design of your program**

   - Major components
     - void lexer();
       - Iterates through text file character by character and utilizes functions checkInput() & switchState() to determine which lexemes to label with which tokens

     - int checkInput();
       - Takes char input and filters them through a series of if statements to determine the token it is classified as. After the token is selected, the global variable input_category is set to the assigned int according to our FSM table and returns that value.

     - void switchState();
       - Takes global int variables input_category and current_state to determine the next state to traverse via the FSM 2D array
       - Uses switch cases to execute the next state for each input
       - Utilizes function saveUnit() to save which state goes with each input

     - void saveUnit();
       - Adds the labeled parameter token and lexeme to the vector “codewords” as a struct

     - string rules();
       - Based on the tokens and lexemes identified from the functions above, outputs the grammar rules associated with tokens and lexemes

     - void saveCMD();

        - Adds the labeled parameter token and lexeme to the vector “codewords” as a struct

- Data Structures Utilized:
    - Structs
    - Vectors
    - Arrays
        - 2D Array for the FSM table
        - char, string arrays for keywords, operators, characters, separators

4. **Any Limitations**
    - NEQ (which represents not equal → !=) could not be outputted if the .txt file contained “!=” due to our code dismissing code that contains ! since it represents the beginning of a comment.
    - PUSHM and POPM commands are not accurately implemented

5. **Any Shortcomings**
    - One significant shortcoming within our programming assignment is the missing implementation of for loop statements.

‘

**Software Requirements Document**

1. **Introduction**
    - The purpose of this program is to modify the parser from Assignment 2 that reads through a text file that will, according to the Grammar rules, produce assembly code instructions. A symbol table will also output with each entry holding the lexeme and memory address.

2. **Overall Description**
    - Analyzes the code character by character to store a symbol table of identifiers and stores a list of assembly commands in the order of execution. Once this is all stored, they are printed out in the output.txt

3. **External Interface Requirements**
    - Requires permission to put code into the test.txt
    - Other than the text files and the termination in the console, there is no graphic interface for the user to interact with.

4. **System Features**
    - The program was created with multiple methods in order to execute the proper output. We used structs, vectors, and arrays in order to program this assignment. Within this project, there were 5 major functions.
    - The lexer() iterates through the inputted text file by each character in order to assign the lexemes to the tokens.
    - The checkInput() takes in a character input and reads through multiple if statements to determine what that token should be classified as.
    - The switchState() uses switch cases to execute the next state for each input.
    - The saveUnit() adds the token and lexeme label to the vector “codewords” as a struct.
    - The rules() take in all of the information from the above functions and output the grammar rules associated with tokens and lexemes.
    - The saveCMD() adds the labeled parameter token and lexeme to the vector “codewords” as a struct.