COVER PAGE

CS323 Programming Assignments

Fill out all entries 1 - 7. If not, there will be deductions!

1. Names [1. Yesenia Huerta] Section [Tues/Thurs: 2:30 pm class]

   [2. Victoria Guzman] Section [Tues/Thurs: 2:30 pm class]

   [3. Luke Piña] Section [Tues/Thurs: 1:00 - 2:15 pm]
   [4. Ayman Sadek] Section [Tues/Thurs: 2:30 pm class]

2. Assignment Number [1]

3. Due Date [Saturday, March 2nd]

4. Submission Date [Saturday, March 2nd]

5. Executable File name [program]
(A file that can be executed without compilation by the instructor, such as .exe, .jar, etc - **NOT** a source file such as .cpp)

6. Names of the test case files - input test file output test file
   test 1. [T1.txt] [T1_output.txt]
   test 2. [T2.txt] [T2_output.txt]
   test 3. [T3.txt] [T3_output.txt]

7. Operating System [Linux]
(Window – preferred or Unix/Linux)

To be filled out by the Instructor:

Comments and Grade:

Documentation

Problem Statement:

For the first assignment, our group's task was to write a lexer() that returns a token when needed. When the program is run, our lexer returns a record – one field for the token and another field for the actual value of the lexeme. For instance, if the keyword "while" were to be read by our lexer(), the result would be: "Keyword: while." Furthermore, this assignment reads a file containing the source code of Rat24S to generate tokens and writes the results to an output file. Our group utilized three test cases – varying in length and complexity, to test our lexer function. The following document will lay out the following: the actual program itself, the test cases we utilized, detailed steps on how to execute the program, the overall design of the program detailing its major components, and finally, any limitations/shortcomings our group encountered.

Program Instructions

For our following program, depending on your chosen IDE, the compilation and execution of the program may vary. However, let's assume (Professor Choi) is running this program on Visual Studio on a <insert OS here>. From here, he will access his terminal and the directory where the folder can be found. <Insert directory path here> then navigate to chosen folder via cd <directory path> <filename>. From here, the professor will compile the C++ program and enter the given .txt file into the program's user prompt. However, the file the professor wants to run must be in the folder. It is also case-sensitive, so if it is misspelled or mistyped compared to the file name, it will not find it and will return an error message.

1. Enter into ComplierProject folder inside the terminal
2. Enter the compile command: g++ -std=c++20 -o program Lexical.cpp
3. After it compiles completely, run executable: ./program
4. Then, enter the text file in the terminal. T1.txt , T3.txt
5. The program will then output a txt file starting with the original txt file name in the folder.
    a. If you wish to run your Txt file, it must be in the folder to avoid an error.

Program Design

Tokenization: The program is designed to read input from a file, tokenize each line of the input, and output the tokens along with their types (e.g., KEYWORD, IDENTIFIER, INTEGER, REAL, OPERATOR, SEPARATOR, ILLEGAL, STRING) into an output file.
Lexer Function: The lexer function takes and tokenizes a string line as input. It iterates over each character in the line and identifies tokens based on specific criteria:

- It skips characters within comment blocks (enclosed within [and * or * and]).
- It handles string literals enclosed in double or single quotes.
- It identifies separators (e.g., (), ;, ,) and operators (e.g., +, -, *, /, <, >, =, !).
- It identifies keywords, identifiers, integers, and real numbers.
- It marks any unrecognized characters or combinations as ILLEGAL tokens.

Token Structure: Tokens are represented using a struct called Token, which contains two members:

- Type: An enumeration (TokenType) representing the type of token.
- Lexeme: A string representing the actual text of the token.

Token Type: The TokenType enumeration defines different types of tokens the lexer recognizes.
Keyword Detection: The program checks if a token lexeme matches any of the predefined keywords. If so, it marks it as a KEYWORD token.
Identifier Handling: If a token lexeme does not match any keyword and does not contain invalid characters, it is considered an IDENTIFIER. However, if it contains invalid characters or is enclosed in underscores, it is marked as ILLEGAL.
Integer and Real Number Detection: The lexer distinguishes between integers and real numbers based on the presence of a decimal point (.). A lexeme containing a decimal point is considered a REAL; otherwise, it is an INTEGER. A REAL token is marked as ILLEGAL if it contains more than one decimal point.
Output: The program writes each token's token type and lexeme to an output file.
File Input/Output: The program prompts the user to enter the input file name, reads input from the file, and creates an output file with the same name appended with "_output.txt."
Main Function: The main function orchestrates the process by reading input lines, tokenizing them, updating token types, and writing the results to the output file.

The program is designed to tokenize input text according to predefined rules, classify tokens into different types, and generate an output file containing token information for further processing or analysis.

Limitations

One of the limitations is that if the content in the text file is not spaced out, it will read as one. For example, if it is a +b, it will read it as a and +b with the plus and b together. If comments are on the same line, it will skip it, but if it follows into the next line, it will not skip over.

Shortcomings

None.