Lexical Analyzer : Lab 1

Corbin T. Rochelle

September 15, 2022

1 Question 1

Choose at least two regular expressions from your rules.l file. List each rule, and explain how that rule operates. You may cut and paste excerpts from your code, but you must also explain what is happening in prose.

The two REGEX I would like to example are: `[0-9]{4}\/[0-9]{2}\/[0-9]{2}` and `\"[Hh]ispanic\/[Ll]atino\"`.

For the first, we see the three sections, all of which are `[0-9]` followed by a number in braces. This part searches for a 4, a 2, and a 2 digit number to math a date (there is no checking to see if the date is valid.) Each section is deliniated by an escaped forward-slash to correctly search for a date.

The second selection shows how I selected both the upper and lowercase versions of the words while escaping the quotes and the forward-slash.

These two regular expressions showcase all of the odd cases of escaping special characters that went into creating these expressions.

2 Question 2

Describe, in a few paragraphs, how flex operates. Examine the lex.yy.c file, and give your impressions of that file. Discuss how flex fits into general development flow.

What flex does is take a .l file and use that to create a c file that defines a function called `yylex()`. This function is then used to take `stdin` and analyze it for tokens that defined in a table in the `yylex()` function. In more detail, the lexer.h file is defined that associates a token string with a token integer, which is used by the parser after flex has done its job. We then, in the exp-rules.h file, set rules using the regular expressions to match stdin of the function with token strings. These token strings are then converted to the integer tokens.

My impressions of the `lex.yy.c` file is that there is a lot going on under the hood, a lot more than I used in the project. It looks as though the general set up is a buffer is created to store characters, which is compared with rules drawn from the lexer.h file with a switch statement, and then a table is referenced to pull the token ID.

It is typically used as the first part of compilation of code (the parsing phase.) It scans the code to create block lexemes, which are much easier for a computer to parse (when compared to a stream of single characters. This is mainly setup for the logical analysis of the code.

3 Question 3

Give, in a few concluding sentences, your opinion of flex. How difficult was it to learn? Did you like it? Do you feel that, compared to writing lexical scanning code by hand, you saved time?

My opinion of flex is neutral. I think it is a valuable tool to use to parse code into chunks; however, the documentation was poorly implemented, which lead me to using other sites to figure out which symbols were special in this language, so I could escape them if needed. Ignoring that, most of my issues were created from the files that were already given to me! I find it much easier when I create everything myself, so I have a fundamental understanding of everything. This did save a lot of time when compared to scanning code by hand.