CS365 – Organization of Programming Languages
Program 4

Objective
Learn to implement an LL grammar

Due Date
3/25/19

Assignment
Finish the ad hoc scanner designed to handle the following language. The input for the scanner will be the output from the previous program. Each line of input will contain a token and characters that make up the token.

The input for an execution can be considered “a program.” State that the program is correct or halt the program with a simple error message if a lexical error is found or when an <error> token is read. If there is an error condition, with either an <error> token or a lexical error, be sure to state what invalid text was found.

Specifics

- This should be written in Python.
- You must use file input. Pick up the source code file name from the command line. Exit **gracefully** with an error message if the input file does not exist on the command line.
- If you create more than one Python file make sure the you upload all of your files.
- The input for this program will be the output from program 2. There will be one token/lexeme pair per line. There should be no concerns about illegal information in the file. There may be a <error> token in the file, but the file will not have any structure errors.
- If you have any problems or concerns about your program 2 talk to me about getting test files. Feel free to share the outputs from program 2 with other students, it’s just test data.
- $$$ is the end of the grammar marker. If you are out of tokens and there hasn’t been a problem yet, the input was lexically correct.

- Use the following LL grammar:

```
<program> → <stmt_list> $$$

<stmt_list> → <stmt> <stmt_list>
            | ε

<stmt> → <assign_stmt>
       | input id
       | print <value>
       | <if_stmt>
       | begin <stmt_list> end

<if_stmt> → if <boolean_expr> <stmt>
          | if <boolean_expr> <stmt> else <stmt>

<assign_stmt> → <id> = <value>

<boolean_expr> → <value>
               | <value> <rel_op> <value>

<value> → <id> | <number>

<rel_op> → < | <= | > | >= | == | !=
```

The sample output provided at the bottom of program 2 would be a good place to start testing. That program is not meant to provide a through test, only to provide a sample of an accepted input. If you delete literally any of the lines from the sample output from program 2 it should create a line with a lexical error. Adding an <error> token anywhere in the input file is also something to test.

And just as a reminder, a “real” compiler would implement program 2 and 4 into a single task, they were only broken up to simplify the processes. This is also the part of the process where many of the perceived errors in the input file will be identified. For example, in program 2, an input file of “a a a” would correctly find 3 identifiers. This is the program that will determine that 3 identifiers in a row makes no sense in the context of our language.