PPL Exercise 2: Lexer

Note 1: Please install ANTLR4 (version 4.9.2 since this version is used for the Assignment). Follow this link: https://www.antlr.org/download.html for installing ANTLR 4

For installing antlr4-python3-runtime, run the following command:

pip install antlr4-python3-runtime==4.9.2

Note 2: The template code for doing Lexer + Parser exercises is in the file MT232.g4 (go to MT232/src/main/MT232/parser/MT232.g4). Your code will start at line 10.

Before we move on to the exercise, let's talk a little bit about ANTLR 4 syntax:

In ANTLR4, you will define some rules for the language (in this case, MT232) using RegEx. Each ANTLR4 program must have 1 non-terminated rules (in the file MT232.g4, it's the "program" rule). Non-terminated rule starts with the lowercase letter while terminated rule starts with uppercase letter.

In this exercise, we will focus on terminated rule.

Some syntax in ANTLR4's RegEx:

- [abc]: match 1 letter: a or b or c

- [a-z]: match 1 letter from a to z (must have ascending order)

- [A-Za-z]: match 1 letter, could be an uppercase letter (A to Z) or a lowercase letter (from a to z)

- ~: not match a rules

- (): for grouping

- *: match 0 or more

- +: match 1 or more

- ?: match 0 or 1

- .: match all

- a | b: match rule a or rule b

We do not have ϵ for empty string, so in ANTLR4, instead of using a | ϵ , use a |

Each rule can have a {} block (called block action) to specify the action happens if that rule is match. Each rule ends with a semi-colon

Each rule will be counted as one token, except a fragment (define by keyword fragment). Fragment is also a terminated rule, but can only be called by another terminated rules.

ANTLR4 will match the input string to the first specified rule, so as you can see in MT232.g4, the ERROR_CHAR rule will be match first, no matter what the input string is, so focus on the order of rules while doing the exercise (also in Assignment).

Let's jump into the exercise:

Before doing the exercise, change the ERROR_CHAR rule to this one:

ERROR_CHAR: .{raised ErrorToken(self.text)};

1. Write the Lexer rule `ID` based on the Identifer's description in MT232 specification

2. Write the Lexer rule `INT_LIT` based on the Integer's description in MT232 specification (note that all underscores must be removed from the input string if the input string matches this rule)

For number with leading zero, please raise exception instead of splitting into 2 tokens (0234 will raise error at the token 0 instead of splitting into 0 and 234)

The testcase will make sure that all input strings will not have two consecutive underscores, underscore at the beginning or at the end of the string.

3. Write the Lexer rule `FLOAT_LIT` based on the Float's description in MT232 specification

4. Write the Lexer rule `STR_LIT` based on the String's description in MT232 specification (remove the double quotes at the beginning and at the end of the input string if that string matches the rule).

Also implement the `UNCLOSED_STRING` and `ILLEGAL_ESCAPE` rule (if needed) to catch the following error:

`UNCLOSED_STRING:` When there is a newline character ('\n') in string, or the string does not have a double quote at the end of the string. Raise the `UNCLOSED_STRING` exception (which is defined file lexererr.py) with the error string. The error string counts from the beginning (exclude double quote at the beginning) to the newline character (if exists), or to the end of the string.

`ILLEGAL_ESCAPE:` When there is an escape sequence that is not supported by MT232 appears in the string. Raise the `ILLEGAL_ESCAPE` exception (which is defined in file lexererr.py) with the error string. The error string counts from the beginning (exclude double quote at the beginning) to the error escape sequence.

5. Write the Lexer rule for the Comment's section in MT232 Specification (also implement the `UNTERMINATED_COMMENT` rule. The C-style comment with no */ at the end of the comment is an unterminated comment)

6. Define Lexer rule for keywords, operators, separators to finish off the first 4 sections in the MT232 Specification, make sure that you place these rules in the correct position.

There will be 25 testcases (5 for each exercise from 1 to 5) to test your implementation. The testcase is in the file LexerSuite.py (go to MT232/src/test/LexerSuite.py). Read the *printUsage* function in the file *run.py* (go to MT232/src/run.py) to know how to run your testcase.

Optional:

7. Write a Lexer rule IPv4 to match the valid IPv4 input string. IPv4 has a format: A.B.C.D, where A, B, C, D are integers from 0 to 255

8. Write a Lexer rule HEX to match the valid hexadecimal input string. The hexadecimal format starts with the 0 digit, then a letter 'x' or 'X'. It finalized with a sequence of hexa digits (from 0 to 9, A to F with case insensitive)

9. Write a Lexer rule BIN to match the valid hexadecimal input string. The hexadecimal format starts with the 0 digit, then a letter 'b' or 'B'. It finalized with a sequence of binary digits (0 or 1)