German University in Cairo
Department of Computer Science
Assoc. Prof. Haythem O. Ismail

CSEN1002 Compilers Lab, Spring Term 2023
Task 8: ANTLR Lexical Analysis

Due: Week starting 13.05.2023

1 Objective

For this task, you need to implement a lexical analyzer using ANTLR (`www.antlr.org`). Your tutor will introduce you to ANTLR during the session, but you are urged to prepare by taking a look at the ANTLR documentation:

`https://github.com/antlr/antlr4/blob/master/doc/index.md`

2 Requirements

- We would like to construct a tokenizer with the following specifications. Given an input string over a subset of a pseudo programming language, the tokenizer should split the input into lexemes and produce an output token for each lexeme. Each lexeme is one of the following types:
  - `IF`
    `if,If,iF,IF`
  - `ELSE`
    `else,...,ELSE`
  - `COMP`
    `>,<,>=,<=,==,!=`
  - `ID`
    any letter or '`_`', optionally followed by letters, digits or '`_`'
  - `NUM`
    a decimal numeral consisting of the following sequence
    a) a whole-number part consisting of at least one digit
    b) an optional decimal part consisting of '`.`' followed by at least one digit
    c) an optional exponent part consisting of a case insensitive '`e`' followed by an optional sign ('`+`' or '`-`') followed by at least one digit
  - `LIT`
    zero or more ASCII characters enclosed in double quotes, excluding '\' and '`"`', where each must be preceded by a '\'
  - `LP`
    `(`
  - `RP`
    `)`

- ***Note that, unenclosed spaces should be recognized and discarded***
- You should write a lexer rule for each of the token types described above. These rules should have the **exact** names as token names, and should recognize **all** strings matching the description and **only** these strings.
- The provided method `tokenStream` uses the ANTLR grammar to return a semicolon-separated sequence of tokens; where a token is in the form "*lex*`,`*q*", where *lex* is as indicated in Lecture 2 of CSEN1003, and *q* is the token type.
- For example, running the lexical analyzer implementing the above tokenizer on the string:

```
if (x > 10) printf("Yes") else printf("No")
```

  output the following (split for readability):

```
if,IF;(,LP;x,ID;>,COMP;10,NUM;),RP;printf,ID;(,LP;"Yes",LIT;),RP;
        else,ELSE;printf,ID;(,LP;"No",LIT;),RP
```

3 Important Details

- Your implementation should be done within the template file uploaded to the CMS.
- You are not allowed to change the grammar name.
- You are allowed to write as many helper fragment lexer rules within the same grammar file (if needed).
- Public test cases have been provided on the CMS for you to test your implementation.
- Please ensure that the public test cases run correctly without modification before coming to the lab to maintain a smooth evaluation process.
- A Java file is provided in order to easily test your grammar with custom strings in addition to the public test cases.
- Private test cases will be uploaded before your session and will have the same structure as the public test cases.

4 Evaluation

- Your implementation will be tested on ten input strings.
- You get one point for each correct output; hence, a maximum of ten points.

5 Online Submission

- You should submit your code at the following link.

  `https://forms.gle/uaA9TvEs7Rr7U6SZ9`

- Submit one file "`Task8.g4`" containing the grammar.
- **Online submission is due by the end of your lab session.**