Name: ______________________________

Grammars, Parsers and Lexers

For this assignment, you are allowed to get assistance from your classmates to help you with the setup and starting the project. However, your final submission should be your own work entirely.

Three main steps of a program compilation are: 1) scanning, 2) parsing and 3) code generation (into an object, .exe file, etc.). The first and second steps are done by a lexical analyser and a parser, respectively.

In a typical grammar file (in ANTLR 4 grammar files are specified by the ".g4" extension) there are two main categories of rules. Rules that are used by the parser and those that are used by the lexer. The lexer rules are generally written at the end of the grammar files. A rule that starts with a lower case letter is used by the parser and rules that start with a capital letter are used by the lexer.

For example, the rules

```
Integer : '0' | '1'..'9' '0'..'9'* ;
Star : '*' ;
WS : [ \t\r\n] -> skip ;
```

are all used by the lexer.

The *lexical analyser* converts the input (generally a program that is implemented by the programmer) into a stream of tokens. This conversion is done by the lexical analyser based on the lexer rules that are specified in the grammar. The conversion into tokens is done, iteratively, by converting the longest prefix of the input that fits the description of a token into that token.

For example, suppose that the rules above are the first lexer rules specified in a grammar file. In that case, when the lexer analyses an input and at some point sees "10938 * 345", it first converts the prefix substring "10938" into the "Integer" token and changes (this part of) the input into "Integer * 345". After that it sees a white space (a blank) that only fits the description of WS and *skips* it. Subsequently it sees a star (*), another white space to skip, and finally another Integer token. Therefore, this part of the input is converted (tokenized) into "IntegerStarInteger".

After the input is analysed and *tokenized* by the lexer, a parser, such as a recursive descent parser, or a bottom-up parser like the one we talked about in class, parses the stream of tokens and attempts to organize it into a tree, etc. If this process is successful, the input is accepted. If not, then the input is rejected.

This assignment is considered an extra credit assignment and its grade will be added to your assignments' grade at the end of classes.

In this assignment, you should perform the following steps:

a. Choose any grammar in the ANTLR 4 grammar repository (`https://github.com/antlr/grammars-v4`).

b. Use the grammar you selected in the previous step with ANTLR 4 to generate lexer/parser files.

c. Create two input files (called "input1.txt" and "input2.txt"). The first input file should be a valid text based on the grammar and the second one should be a slightly modified version of the first input so that it is not accepted by the grammar.

d. Implement a program in Java (and name it "Recognizer.java") that uses the generated lexer/parser files, reads both inputs, and specifies whether they are accepted or rejected (of course, it should say that it accepts the first input but rejects the second one).

Clearly, if your program is tested for any other valid/invalid input it should also correctly output that it accepts/rejects the input.

Your code should run on any system with ANTLR 4 properly installed. At the beginning of your code, write a comment that explain what grammar have you selected, why the second input should not be accepted by the grammar, and any other information needed to run your code.

ZIP the whole project into a single file and upload it on Canvas.