Project 1
Implementation of a Lexical Analyzer
Due April 7, Wednesday, 2021

Note: You need to COMPLETELY follow the instructions in this sheet.

1. **Problem:**

In this assignment you are requested to use the tool **ANTLR** to generate a lexical analyzer for a small language **Cactus**. **Cactus** contains the following six types of tokens:

1. This type of tokens includes all identifiers. An identifier is a sequence of underscores, letters, and digits; the first character must be an underscore or a letter. All identifiers are returned as the same token.
2. This type of tokens includes the following 18 keywords:
   And, Begin, Do, Else, End, EndIf, EndWhile, Exit, If, Set,
   Not, Or, Program, Read, Then, Var, While, Write.
   These keywords are identifiers reserved for special use, and may not be used as general identifiers. Each keyword is returned as a different token. Namely, there are 18 different tokens in this type.
3. This type of tokens includes all integer constants. An integer constant consists of a sequence of digits. All integer constants are returned as the same token.
4. This type of tokens includes the following 14 operators:
   + - * / % = == <> > >= < <= ( )
   Each operator is returned as a different token. Namely, there are 14 different tokens in this token type.
5. This type of tokens includes all white spaces. A white space may be a blank (' '), a tab ('\t'), or a newline ('\n'). These white spaces are mainly served to separate tokens. These tokens are ignored and are not returned.
6. This type of tokens includes all comments. A comment starts with the characters // and ends with a newline. These tokens are ignored and are not returned.

You can follow the following steps:

1. Edit a grammar Cactus.g4 that contains the regular expressions for each of the token types as follows.

```
// The grammar for Cactus language
grammar Cactus;

// Parser rules
token : (ELSE | … | ID | CONST | ADD | … )*

// lexer rules
ELSE : ‘else’
   …
ID : …
CONST : …
ADD : ‘+’
   …
WHITESPACE : …
COMMENT : …
```

2. Use the ANTLR tool to generate the lexical analyzer and parser java code.

```
$antlr4 Cactus.g4
```

3. Compile the generated java code.

```
$javac Cactus*.java
```

4. Use the ANTLR tool to execute the lexical analyzer and parser.

```
$grun Cactus token -tree
```

If the input is as follows:

```
// A program to sum 1 to n
Program sum Begin
        Var    n
        Var    s

        Read n
        If n < 0 Then
              Write -1
```

```
            Exit
        Else
            Set s = 0
        EndIf
        While n > 0 Do
            Set s = s + n
            Set n = n – 1
        EndWhile
        Write s
    End
```

The output should be

(token Program sum Begin Var n Var s Read n If n < 0 Then Write – 1 Exit Else Set s = 0 EndIf While n > 0 Do Set s = s + n Set n = n – 1 EndWhile Write s End)

2. Handing in your program:

To turn in the assignment, upload a compressed file proj1 containing Cactus.g4, Cactus.tokens, Cactus*.java, and Cactus*.class to eCourse2 site.

3. Grading

The grading is based on the correctness of your program. The correctness will be tested by a number of test cases designed by the instructor and teaching assistants.

It is best to incrementally generate your program so that you always have a partially-correct working program.