Project 2
Implementation of a Recursive Descent Parser
Due Wednesday, May 12, 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 recursive decent parser for the small language **Cactus**. The grammar for the small language **Cactus** is as follows:

program → **Program Identifier Begin** declarations statements **End**
declarations → declarations **Var Identifier** | ε
statements → statements statement | ε
statement → **Set Identifier =** arithmeticExpression
| **If** booleanExpression **Then** statements **EndIf**
| **If** booleanExpression **Then** statements **Else** statements **EndIf**
| **While** booleanExpression **Do** statements **EndWhile**
| **Read Identifier**
| **Write** arithmeticExpression
| **Exit**
booleanExpression → booleanExpression **Or** booleanTerm
| booleanTerm
booleanTerm → booleanTerm **And** booleanFactor
| booleanFactor
booleanFactor → **Not** booleanFactor | relationExpression
relationExpression → arithmeticExpression **==** arithmeticExpression
| arithmeticExpression **<>** arithmeticExpression
| arithmeticExpression **>** arithmeticExpression
| arithmeticExpression **>=** arithmeticExpression
| arithmeticExpression **<** arithmeticExpression
| arithmeticExpression **<=** arithmeticExpression
arithmeticExpression → arithmeticExpression **+** arithmeticTerm
| arithmeticExpression **–** arithmeticTerm
| arithmeticTerm
arithmeticTerm → arithmeticTerm ***** arithmeticFactor
| arithmeticTerm **/** arithmeticFactor
| arithmeticTerm **%** arithmeticFactor
| arithmeticFactor
arithmeticFactor → **-** arithmeticFactor | primaryExpression

primaryExpression → **IntConst** | **Identifier** | **(** arithmeticExpression **)**

You can follow the following steps:

1. Edit a grammar Cactus.g that contains a parser rule for each of the productions in the above context-free grammar. Because the above context-free grammar is not an LL(1) grammar, you need to perform the left recursion elimination transformation and the left factoring transformation to transform it into an LL(1) grammar.

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

// Parser rules
program : PROGRAM ID BEGIN declarations statements END
   ;
   …

// lexer rules
ELSE : 'Else'
   …
ID : …
CONST : …
ADD : '+'
   …
WHITESPACE : …
COMMENT : …
```

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

```
$antlr4 Cactus.g4
```

3. Compile the generated java code.

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

4. Use the ANTLR tool to execute the scanner and parser.

```
$grun Cactus program -tree
```

If the input is as follows:

Program main Begin End

The output should be

(program Program main Begin declarations statements 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.