Exercise 9: ANTLR LEXER

1 Objective

Learn about Type-3 or regular grammar or regular expressions. Learn how to write a recognizer for regular grammars.

Using ANTLR, you will create a Java based lexer that reads xml content and outputs the details of xml tags in the content.

Work with your group (or by yourself). Each group is to upload only one submission.

ANTLR reference can be found at http://www.antlr.org and

https://theantlrguy.atlassian.net/wiki/display/ANTLR4/ANTLR+4+Documentation

We have several EXAMPLES that you should first view and tryout and understand. That will give you a good idea of the format of the ".g4" files.

- **The structure of ".g4" files** can be found at https://theantlrguy.atlassian.net/wiki/display/ANTLR4/Grammar+Structure
- Allowable Regular expressions in lex are shown in the slides. There are a few more such as [ ] etc.
- Non-Greedy rules: See https://theantlrguy.atlassian.net/wiki/display/ANTLR4/Wildcard+Operator+and+Nongreedy+Subrules

Credits (for edits): Naresh Somisetty, Isaac Martin, and Zahra Hosseini.

2 Warm Up: Try Some Examples

1. First, open blackboard, go to Course Contents, and then download exercise09.zip file into your workspace (U:\workspace or something like that!). Then, unzip.
2. Browse through the the pdf file on antlr that has been provided. It will help you understand the background.

2.1 Setting up Antlr

- The antlr-4.4-complete.jar) file is located in your unzipped folder.
- Using System Properties dialog > Environment variables > Create or append to CLASSPATH variable with the path to the jar file you copied above.
- Naresh created antlr.bat & grun.bat (which are in the examples directory) so that you can start using the commands antlr and grun on windows.

2.2 Running the examples

You will be running the following commands on each of the example grammar files.

Note that antlr4 is an alias for "**java -jar ../antlr-4.4-complete.jar HelloWorld.g4"**
Note that grun is an alias for "java org.antlr.v4.runtime.misc.TestRig"

a) antlr4 <<Grammarfile.g4>> // this will create java files for lexer
b) javac *.java // this will compile and create appropriate class files
c) grun <<GrammarName>> tokens < <<input-file>>
d) Also, grun <<GrammarName>> tokens -tokens < <<input-file>>

Note that "tokens" is used as a start rule for grun when using lexer.
Note that –tokens is an option to grun to printout the tokens that it has recognized.

Here is an example of the steps you will need to take

```
Step 1 : antlr E1_Hello.g4  // this creates the java program
Step 2 : javac E1_Hello.java
Step 3 : grun E1_Hello tokens    < E1_Hello.in   // this runs the lexer
Step 4 : grun E1_Hello tokens   -tokens < E1_Hello.in
```

Repeat the steps for all the examples by replacing E1_Hello with other examples in the above sample.

3. There are many grammar files in the 02_examples directory. Corresponding to each grammar file, there will be an input file with extension ".in". For each grammar file (E1_Hello.g4 etc), run the commands a through d. Try and understand what is going on. Make minor changes and see what happens. Each grammar file has comments – please read the comments in each file.

3 Lexical Analysis of XML Content

Here is the assignment itself.

Write antlr lexer grammar rules (i.e. create a ".g4" file) that tokenizes xml content.

Your grammar should **tokenize** the xml content successfully.

Example of input xml content(".in") file
<email>smitra@iastate.edu</email> <date>20/01/2015</date> <phone>(800) 515-3463</phone> <creditcard>4321-1111-2222-3333</creditcard>

Here are the rules for the different Elements:

3.1 Element names (such as email, date etc)

- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, or periods
- Element names cannot contain spaces
- Names cannot contain spaces

3.2 Email element

May use

- localpart@domainpart (example: simanta.mitra@abc-def.org)
- local part rules (local part may consist of following characters)
    - Uppercase and lowercase Latin letters (a–z, A–Z)
    - Digits 0 to 9
    - These special characters: `- _ ~ ! $ & ' ( ) * + , ; = :`
    - Character `.` provided that it is not the first or last character, and provided also that it does not appear consecutively
- domain part rules
    - letters, digits, hyphens and dots.

3.3 Date element

- dd/mm/yyyy
- day must be number between 1 and 31
- month must be number between 1 and 12
- year must be number between 2000 and 2100

3.4 Phone element

- ###-###-####
- (###) ###-####
- ### ### ####
- ###.###.####

3.5 Creditcard element

Valid cards would be one of the below.

- Visa: All Visa card numbers start with a 4. New cards have 16 digits. Old cards have 13.
- MasterCard: All MasterCard numbers start with the numbers 51 through 55. All have 16 digits.
- American Express: American Express card numbers start with 34 or 37 and have 15 digits.
- Diners Club: Diners Club card numbers begin with 300 through 305, 36 or 38. All have 14 digits. There are Diners Club cards that begin with 5 and have 16 digits. These are a joint venture between Diners Club and MasterCard, and should be processed like a MasterCard.
- Discover: Discover card numbers begin with 6011 or 65. All have 16 digits.
- JCB: JCB cards beginning with 2131 or 1800 have 15 digits. JCB cards beginning with 35 have 16 digits.

4 Submission:

Zip all your files and submit on black board. Remember there is only one submission per group. Make sure to include all the files that are needed in order to run your program.