Exercise 7: 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 tutorials/examples can be found at

- http://supportweb.cs.bham.ac.uk/docs/tutorials/docsystem/build/tutorials/antlr/antlr.html
- https://github.com/antlr/grammars-v4
- http://www.theendian.com/blog/antlr-4-lexer-parser-and-listener-with-example-grammar/
- http://www.antlr2.org/doc/lexer.html
- http://web.mit.edu/dmaze/school/6.824/antlr-2.7.0/doc/lexer.html

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. Make sure to look at the cheatSheet and at the other pdfs in the zip file.

Credits (for edits): Naresh Somisetty, Isaac Martin, and Zahra Hosseini, Sai Nemalikanti, Zelong Li.

2. Warm Up: Try Some Examples

1. First, open blackboard, go to Course Contents, and then download exercise07.zip file into your workspace (U:\workspace or something like that!). Then, unzip.

2. Carefully read notes on grammar and on lexer. Those document give you good insight about Antler. Also, read the notes on Docker.

3. Note that you will need to use Docker for this lab. Please read the Dockerinstructions.pdf first. There are examples provided in the Docker folder. Try them out. https://docs.docker.com/engine/understanding-docker/#what-is-dockers-architecture

1.1. Setting up Antlr

THIS ENTIRE SECTION (Section 1.1 Setting up Antlr) IS NOT NEEDED ANYMORE AS WE WILL BE USING DOCKER. SO SKIP THIS SECTION.

- The antlr_4.5.3_complete.jar file is located in your unzipped folder.
- You will need to make sure you have a jdk (i.e. with you have a java compiler). The path to the jdk must be on your PATH environment variable.
- Both the current directory and also the antlr jar file must be on your CLASSPATH variable. Using System Properties dialog > Environment variables > Create or append to CLASSPATH variable with the path to the jar file you copied above.

For OS X make sure to do these steps to setup Antlr:

```
$ cd /usr/local/lib

$ sudo curl -O http://www.antlr.org/download/antlr-4.5.3-complete.jar

$ export CLASSPATH=".:/usr/local/lib/antlr-4.5.3-complete.jar:$CLASSPATH"
```

Note we have provided three .bat files: antlr4.bat, compile.bat, and grun.bat

You can use them OR just alias antl4 and grun as shown below.

```
$ alias antlr4='java -jar /usr/local/lib/antlr-4.5.3-complete.jar'

$ alias grun='java org.antlr.v4.gui.TestRig’
```

test the the installation of Antlr by typing in

```
$ java  org.antlr.v4.Tool
```

For Windows make sure to do these steps to setup Antlr:

1-Add antlr-4.5.3-complete.jar to CLASSPATH, either:

Permanently: Using System Properties dialog > Environment variables > Create or append to CLASSPATH variable

Temporarily, at command line:

```
SET CLASSPATH=.;C:\Javalib\antlr4-complete.jar; %CLASSPATH%
```

2- antlr.bat, grun.bat, compile.bat are already created (which are in the examples directory) so that you can start using the commands antlr4, grun, compile on windows.

1.1.1. Note that **antlr4** is an alias for "**java -jar ../antlr-4.5.3-complete.jar** "
Note that **grun** is an alias for **java -cp ".;../antlr-4.5.3-complete.jar"** **org.antlr.v4.runtime.misc.TestRig %***
Note that compile is an alias for **javac -cp ".;../antlr-4.5.3-complete.jar" %***

1.2. Running the examples

RUN THE EXAMPLES AS EXPLAINED IN THE DOCKER INSTRUCTIONS. SKIP THIS SECTION.

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

a) antlr4 <<Grammarfile.g4>> // this will create java files for lexer

a) **MAC users**: javac *.java

Windows Users: compile *.java // this will compile and create appropriate class files

b) grun <<GrammarName>> tokens < <<input-file>>

c) 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 : antlr4 E1_Hello.g4 // this creates the java program
Step 2 : javac *.java**(MAC users)** compile *.java **(Windows Users)**
Step 3 : grun E1_Hello tokens < E1_Hello.in // this runs the lexer
Step 4 : OR 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 the 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 FILES

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

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

Note that each line is one token. Thus, <email>smitra@iastate.edu</email> would be ONE token.

Example of input xml content(".in") file
<email>smitra@iastate.edu</email>
<date>20/01/2015</date>
<phone>(800) 515-3463</phone>
<creditcard>4321111122223333</creditcard>
<Address>226 Atanasoff Hall, Ames, IA 50011</Address>

Here are the rules for the different Elements:

3.1. Element names (such as email, date, phone, creditcard etc)

- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)

Hint: consider cases to include all the other element names other than element names starting with ('ab')

Consider various cases like

1. Include all the element names that don't start with 'a ' like [b-z]char*

2. Include all the element names not having 2nd character as b like char[ac-z]char*

3. Similarly cover all the remaining cases

- Element names can contain letters, digits, hyphens, underscores, or periods
- Element names cannot contain spaces
- Names cannot contain spaces (repeated from above ?)

1.2. Email element

- 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

    Hint: Can use operator ('a' 'b' ?) that accepts ab or a
- domain part rules
  - letters, digits, hyphens and dots.

1.3. 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
- NOTE that you can ignore issues such as leap years, feb, 30 day months, 31 day months etc. Thus, for example 2/30/2000 will be considered valid.

Note : For defining a range of a 2 digit number [r1][r2]

1.4. Phone element

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

  Hint: to check for an n-digit number take fragment in the form of DIGIT DIGIT.. ntimes where DIGIT is number between 0-9

1.5. 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.

  Hint: you can put "| mastercard fragment " for processing like 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.

Make sure to report all Error in XML file once.

Hint : to show error reporting all at once you can try non greedy approach discussed in class

3.6 Other elements (i.e. not one of email, date, phone, creditcard)

- The string inside the element consists of only a combination of the following.
  - Uppercase and lowercase Latin letters (a–z, A–Z)
  - Digits 0 to 9
  - These special characters: – _ ~ ! $ & ' ( ) * + , ; = :
  - And space

4. Lexical Analysis of JSON File

Here is the second part of the assignment.

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

Your grammar should **tokenize** the json content successfully. Each line is a separate token. For example, "EMAIL": smitra@iastate.edu is ONE TOKEN.

Example of input JSON content(".in") file

```
{
    "EMAIL": "smitra@iastate.edu",
    "DATE": "20-01-2015",
    "PHONE": "(800) 515-3463",
    "CREDITCARD": "4321111122223333"
    “Address”:”226 Atanasoff Hall, Ames, IA 50011”
}
```

Rules are same as XML part

You don’t need to tokenize parentheses ‘{ ‘,’ }’ and comma ‘,’. You can ignore them

Make sure to report all Error for every element in one execution. For example if there are errors in phone and credit card elements. Then, you should recognize phone and credit cards as two separate elements and report error for each of them.

Hint:

#1) The file "grammer2.pptx.pdf" has rules on lexer part of antlr.

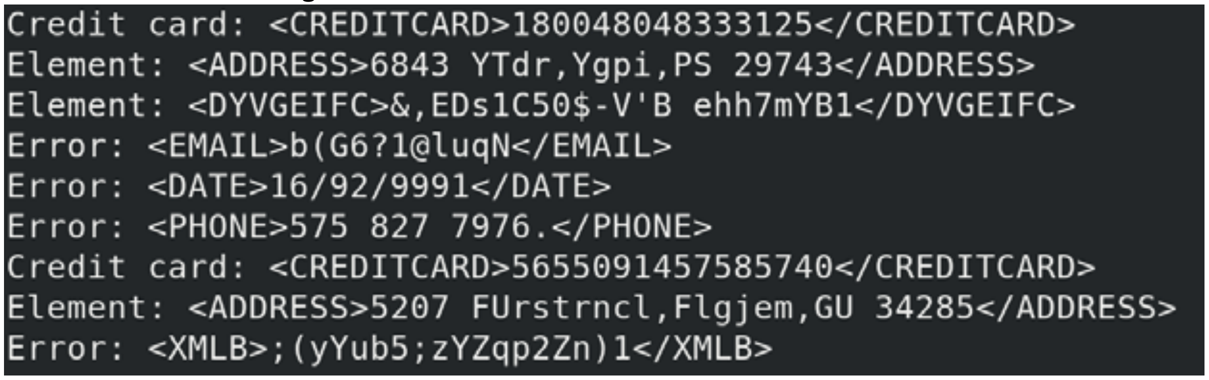
#2) Use ( ) to group sequence of characters and | to indicate OR

#3) do use fragments to simplify your rules (look up fragments in one of the examples in 02_examples directory).

Before submission make sure that :

- Use regex for defining lexer rules. For example the following code is NOT acceptable:

  ELEMENT: something {
  String temp = getText();
  if(cond) system.out.println("Invalid");
  }
  You cannot manipulate the string using java inside the code section like the above example. However , you can use simple command like System.out.println("example"+getText())
- For report error part, After you check for a correct element name, for each of the four hard-coded element names, write a rule that uses optional greedy match. Then, have it output the error message u want with system.out.println. The sample error report shows in following.



```
Credit card: <CREDITCARD>180048048333125</CREDITCARD>
Element: <ADDRESS>6843 YTdr,Ygpi,PS 29743</ADDRESS>
Element: <DYVGEIFC>&,EDs1C50$-V'B ehh7mYB1</DYVGEIFC>
Error: <EMAIL>b(G6?1@luqN</EMAIL>
Error: <DATE>16/92/9991</DATE>
Error: <PHONE>575 827 7976.</PHONE>
Credit card: <CREDITCARD>5655091457585740</CREDITCARD>
Element: <ADDRESS>5207 FUrstrncl,Flgjem,GU 34285</ADDRESS>
Error: <XMLB>;(yYub5;zYZqp2Zn)1</XMLB>
```

4. Submission:

Zip all your files and participation file (i.e. who worked on which part or if you worked together). Then, 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.

Participation file is a simple txt file, which clarifies the specific participation of two members.