Homework 01 (Due: Friday, February 16, 2018, 11 : 59 : 00PM Central Time)

CSCE 322

1 Instructions

In this assignment, you will be required to scan, parse, and check the semantics of a file that encodes the state of a variation of basic train Dominoes. The definition of a properly formatted input file is given in Section 1.1.

You will be submitting one `.java` file and two `.g4` (ANTLR) files via web hand-in.

1.1 File Specification

- The file contains two (2) labeled sections: **Trains** and **Hands** . Each section is enclosed by start and end tags (`<<` and `>>`, respectively).
- **Trains** is a pound-separated (`#`) list of lists (the dominoes that an individual has played) that appear between `{` and `}` tokens. Valid dominoes are of the form `[N,N]` where `N` is an integer.
- **Hands** is a pound-separated (`#`) list of lists (the dominoes that an individual holds) that appear between `{` and `}` tokens. Valid dominoes are of the form `[N,N]` where `N` is an integer.

An example of a properly formatted file is shown in Figure 1.

```
   <Trains <<
{ [6,9]       [9,9]         }   #
{    [0,1]      [1,1]      }   #
{  [2,11]      [11,11]      }    #
{   [10,10]    [10,12]       }   #
{  [5,5]         [5,12]      }   #
{    [4,4]      [4,11]     }  #
{ [1,8]          [8,8]    }
   >>
  <Hands   <<
{     [3,7]      [1,10]     [2,5]       [1,2]     [4,5]       [11,12]       [5,7]          [3,8]       [0,12]  [3,3]       [7,11]          }  #
{  [6,8]          [0,0]       [2,6]       [7,10]     [2,12]      [8,9]       [1,7]       [4,9]       [1,5]       [3,12]      [1,4]    }  #
{   [3,5]    [0,9]      [4,7]      [6,10]      [3,11]        [0,10]     [2,10]       [7,8]      [1,12]       [2,2]     [0,5]      }  #
{  [4,8]          [3,9]     [5,6]       [9,11]     [2,4]       [0,6]    [0,7]       [0,3]   [9,10]      [7,7]      [0,8]       }   #
{     [4,6]       [3,4]      [3,6]      [1,6]      [6,7]       [0,11]      [6,6]       [5,8]       [10,11]       [7,12]       [0,4]        }   #
{    [3,10]       [8,11]     [4,12]      [2,7]     [7,9]        [8,12]      [8,10]       [2,3]      [1,9]       [4,10]          [5,9]          }    #
{     [6,11]       [1,11]      [2,8]          [6,12]        [2,9]       [1,3]     [9,12]       [5,11]     [5,10]          [12,12]      [0,2]     }
 >>
```

Figure 1: A properly formatted DOmino Configuration (doc) encoding

The assignment is made up of two parts: scanning the text of the input file and parsing the information contained in the input file.

1.2 Scanning

Construct a combined grammar in a `.g4` file that ANTLR can use to scan a supplied Domino Configuration encoding. The logic in this file should be robust enough to identify tokens in the encoding and accurately process any correctly formatted encoding. The rules in your `.g4` file will be augmented with actions that display information about the input file. An example of that output is specified in Section 2.

The purpose of the scanner is to extract tokens from the input and pass those along to the parser. For the Domino Configuration encoding, the types of tokens that you will need to consider are given in Table 1.

Type	Form
Section Beginning	`<<`
Section Ending	`>>`
Section Title	`<Trains` and `<Hands`
Domino Symbol	One or more Numerical Symbols
Numerical Symbol	`0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, or `9`
Row Ending	`#`
List Beginning	`{`
List Ending	`}`
White Space (to be ignored)	spaces, tabs, newlines

Table 1: Tokens to Consider

1.2.1 Invalid Encodings

For invalid Domino Configuration encodings, the output `Notification: Problem on Line L` should display. `L` would be the line of input where the unknown symbol was read. Your scanner should stop scanning the file after an unrecognized token is found.

1.3 Parsing

Construct a combined grammar in a `.g4` file that ANTLR can use to parse a supplied Domino Configuration encoding. In addition to the rules for scanning, there are several parsing rules:

- There must be more than one (1) player (hand) in a game
- There must be more than one (1) player (train) in a game

The semantics of a properly formatted Domino Configuration encoding are:

1. There must be between 3 and 10 (inclusive) hands in a game
2. The number of played dominoes (dominoes in trains) must not exceed 20% of all dominoes in a game
3. **Extra Credit (10 points or Honors contract)**: No train may be more than one (1) domino longer than any other train in the game

2 Output

2.1 Scanner

Your `.g4` file should produce output for both correctly formatted files and incorrectly formatted files. For the correctly formatted file in Figure 1, the output would have the form of the output presented in Figure 2

```
Portion: Trains
Begin Section
Begin List
Begin Domino
Half: 6
Half: 9
Conclude Domino
Begin Domino
Half: 9
Half: 9
Conclude Domino
Conclude List
Conclude Row
Begin List
Begin Domino
Half: 0
...
Begin Domino
Half: 8
Half: 8
Conclude Domino
Conclude List
Conclude Section
Portion: Hands
Begin Section
Begin List
Begin Domino
Half: 3
Half: 7
Conclude Domino
Begin Domino
Half: 1
...
Begin Domino
Half: 0
Half: 2
Conclude Domino
Conclude List
Conclude Section
Conclude File
```

Figure 2: Truncated Output of Scanner for File in Figure 1

For a correctly formatted file in Part 2, the output would be: `d dominoes have been played` where `d` is the number of dominoes that have been played in the **Trains** . For the file in Figure 1, the output would be
`14 dominoes have been played`.

2.2 Invalid Syntax & Semantics in Parsing

For invalid encodings in Part 2, a message describing the error should be displayed. For a syntax error (violation of the syntax rules), the output
`Notification: Problem on Line L` should be displayed, where `L` is the line number where the unrecognized token was found. For that error, the parser should stop processing the file. For a semantic rule violation, the output
`Notification: Semantic Problem R` should be displayed, where `R` is the number of the rule (from List 1.3) that was violated, but parsing should continue.

Syntax errors in Part 2 should be reported in the `syntaxError` method of `csce322Homework01Part02Error.java`.

3 Naming Conventions

The ANTLR file for the first part of the assignment should be named `csce322Homework01Part01.g4`. The ANTLR file for the second part of the assignment should be named `csce322Homework01Part02.g4`. Both grammars should contain a start rule named `dominoes`. The Java file for the second part of the assignment should be named `csce322Homework01Part02Error.java`.

4 webgrader

The webgrader is available for this assignment. You can test your submitted files before the deadline by submitting them on webhandin and going to http://cse.unl.edu/~cse322/grade, choosing the correct assignment and entering your `cse.unl.edu` credentials

The script should take approximately 2 minutes to run and produce a PDF.

4.1 The Use of `diff`

Because Part 1 of this assignment only depends on the symbols in the file, the order in which they are displayed should not be submission dependent. Therefore, `diff` will be used to compare the output of a particular submission against the output of the solution implementation. For Part 2, order and repeated output lines will not influence `diff`.

5 Point Allocation

Component	Points
Part 1	35
Part 2	65
Total	100

6 External Resources

ANTLR
Getting Started with ANTLR v4
ANTLR 4 Documentation
Overview (ANTLR 4 Runtime 4.7 API)

7 Commands of Interest

```
alias antlr4='java -jar /path/to/antlr-4.7.1-complete.jar'
alias grun='java org.antlr.v4.gui.TestRig'
export CLASSPATH="/path/to/antlr-4.7.1-complete.jar:$CLASSPATH"
antlr4 /path/to/csce322Homework01Part0#.g4
javac -d /path/for/.classfiles /path/to/csce322Homework01Part0#*.java
java /path/of/.classfiles csce322Homework01Part02Driver /path/to/inputfile
grun csce322Homework01Part0# dominoes -gui
grun csce322Homework01Part0# dominoes -gui /path/to/inputfile
grun csce322Homework01Part0# dominoes
grun csce322Homework01Part0# dominoes /path/to/inputfile
```