Homework 01 (Due: Friday, September 29, 2017, 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 an elaborate maze. 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: **directions** and **maze** . Each section is enclosed by start and end tags (`/*` and `*/`, respectively).
- **directions** is an asterik-separated (`*`) list of directions that appear between `//` and `-->` tokens. Valid **directions** are `u`, `d`, `l`, and `r`.
- **maze** contains a two-dimensional array of entries that uses `x`, `-`, `g`, and numerical symbols to encode the state of the maze. Rows will be ended with a `|` and the **maze** will be begun with a `<<` and ended with a `>>`.

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

```
  @maze    /*
   <<
xxxxxxxxxxxxxxx      |
x--1-x---x-xg2x   |
x----x-x------x     |
x-xx----------x  |
x-x---xx-x--xxx      |
xxxxxxxxxxxxxxx
 >>
     */
  @directions     /*
   //      r    *       d   *   l *     l     *      u *    u     *       u     *         l     *      l     *    d *      l     *       r     *     l            -->
  */
```

Figure 1: A properly formatted Elaborate Maze Format (emf) 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 Elaborate Maze Format 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 Elaborate Maze Format encoding, the types of tokens that you will need to consider are given in Table 1.

Type	Form
Section Beginning	`/*`
Section Ending	`*/`
Section Title	`@maze` and `@directions`
Move Symbol	`u`, `d`, `l`, and `r`
Maze Symbol	`x`, `-`, `g`, or one or more numerical symbols
Numerical Symbol	`0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, or `9`
Row Ending	`|`
Maze Beginning	`<<`
Maze Ending	`>>`
Moves Beginning	`//`
Moves Ending	`-->`
White Space (to be ignored)	spaces, tabs, newlines

Table 1: Tokens to Consider

1.2.1 Invalid Encodings

For invalid Elaborate Maze encodings, the output `DANGER: Line L contains the symbol T.` should display. `T` would be the symbol read and `L` would be the line of input where the 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 Elabor Maze encoding. In addition to the rules for scanning, there are several parsing rules:

- Each section appears once and only once. The sections may appear in either **directions** /**maze** or **maze** /**directions** order.
- There must be more than five (5) rows in a valid **maze** .
- There must be more than five (5) columns in a valid **maze** .

  You may assume that each row has the same number of columns, and each column has the same number of rows.
- There must be more than four (4) locations in the **directions** section.

The semantics of a properly formatted Elaborate Maze File encoding are:

1. The **maze** must have 2 to 4 players
2. The number of **directions** must be more than two (2) times as large as the number of players in the **maze**
3. The **directions** must contain every type of move (`u`, `d`, `l`, `r`) at least once
4. Barriers (`x`) may not make up more than 70% of locations in the **maze** .
5. **Extra Credit (10 points or Honors contract)**: No player in the **maze** may start at a location where they can move in more than three (3) directions.

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

```
Sector: @maze
Section Inception: /*
Maze Inception: <<
Maze Symbol: x
...
Maze Symbol: x
Row Discontinuation: |
Maze Symbol: x
Maze Symbol: -
Maze Symbol: -
Number: 1
Maze Symbol: -
...
Number: 2
Maze Symbol: x
Row Discontinuation: |
Maze Symbol: x
...
Maze Symbol: x
Maze Discontinuation: >>
Section Discontinuation: */
Sector: @directions
Section Inception: /*
List Inception: //
Direction: r
Direction: d
...
Direction: r
Direction: l
List Discontinuation: -->
Section Discontinuation: */
File Discontinuation
```

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

For a correctly formatted file in Part 2, the output would be: `Your maze has p players.` where `p` is the number of players in the **maze** . The output would be
`Your maze has 2 players.` for the file in Figure 1.

2.1.1 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
`DANGER: Line L contains the symbol T.` should be displayed, where `L` is the line number where the unrecognized token `T` was found. For that error, the parser should stop processing the file. For a semantic rule violation, the output
`ALERT: Violation of Semantic Rule 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 `elaborate`. 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 the form of the output has been specified, `diff` will be used to confirm that a program's output is correct. Output must match exactly.

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 4.6 API)

7 Commands of Interest

```
alias antlr4='java -jar /path/to/antlr-4.7-complete.jar'
alias grun='java org.antlr.v4.gui.TestRig'
export CLASSPATH="/path/to/antlr-4.7-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 csce322homeWork01partt0# elaborate -gui
grun csce322homeWork01part0# elaborate -gui /path/to/inputfile
grun csce322homeWork01partt0# elaborate
grun csce322homeWork01part0# elaborate /path/to/inputfile
```