Homework 01 (Due: Friday, September 25, 2020, 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 the Boss Puzzle. 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 webhandin.

1.1 File Specification

- The file contains two (2) labeled sections: **Moves** and **Puzzle** . Each section appears once (and only once) in each file. Each section is enclosed by start and end tags (`<<alpha` and `omega>>`, respectively). The value of the section is set by the `=` assignment operator.
- **Moves** a is list of values that appear as an ampersand-separated (&) list in brackets (`[` and `]`).
- **Puzzle** is a two-dimensional array of ampersand-separated entries that uses a numerical alphabet to encode the current state of the puzzle. The 0 corresponds to the blank space in the puzzle. Rows will be ended with a `$` and the **Puzzle** will be ended with a `}` (and is begun with a `{`).

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

```
<<alpha  Moves    =   [       1 &   1]    omega>>
<<alpha  Puzzle     =    {    23 &  37 &   5 &  10 &  20 &  12$
 17 &  51 &  15 &  54 &   1 &  33$
 44 &   3 &  55 &  35 &  30 &  29$
 36 &  24 &  58 &  42 &   4 &  40$
 19 &  50 &   0 &  31 &  56 &   6$
 47 &  57 &  46 &  13 &  14 &  43$
 16 &   2 &  25 &  59 &  53 &  41$
 11 &  39 &   7 &  45 &  38 &  28$
  9 &  32 &  49 &  52 &  27 &  22$
 48 &  34 &   8 &  18 &  26 &  21 }
omega>>
```

Figure 1: A properly formatted Boss Puzzle 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 Boss Puzzle 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 Boss Puzzle encoding, the types of tokens that you will need to consider are given in Table 1.

Type	Form
Beginning of Section	`<<alpha`
End of Section	`omega>>`
Type of Section	`Moves` and `Puzzle`
Value Assignment	`=`
Numerical Symbol	`0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`
Puzzle Symbol	1 or more Numerical Symbols
Moves Symbol	`u`, `d`, `l`, `r`
Row Separator	`$`
Beginning of Board	`{`
End of Board	`}`
Beginning of List	`[`
End of List	`]`
White Space (to be ignored)	spaces, tabs, newlines

Table 1: Tokens to Consider

1.2.1 Invalid Encodings

For invalid Boss Puzzle encodings, the output `Line L Contains Unrecognized Token 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 Boss Puzzle encoding. In addition to the rules for scanning, there are several parsing rules:

- Each section appears once and only once. The **Moves** and **Puzzle** sections may appear in either **Moves** /**Puzzle** or **Puzzle** /**Moves** order.
- There must be more than one move (which is encoded in **Moves**)
- The **Puzzle** must contain more than one row and more than one column

The semantics of a properly formatted Boss Puzzle encoding are:

1. Each **Puzzle** Symbol must be greater than or equal to zero (0) and less than or equal to the number of tiles in the **Puzzle** .
2. The number of rows in the **Puzzle** must be greater than two (2) and less than ten (10).
3. The number of columns in the **Puzzle** must be greater than two (2) and less than ten (10).
4. **Extra Credit (10 points or Honors contract)**: Determining if a given Boss Puzzle is solvable is easily done. A puzzle must be solvable.

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

```
Start Section: <<alpha
Section: Moves
Assignment: =
Move: l
Move: l
End Section: omega>>
Start Section: <<alpha
Section: Puzzle
Assignment: =
Start Puzzle: {
Tile: 23
Tile: 37
Tile: 5
Tile: 10
Tile: 20
Tile: 12
End Row: $
Tile: 17
...
Tile: 22
End Row: $
Tile: 48
Tile: 34
Tile: 8
Tile: 18
Tile: 26
Tile: 21
End Puzzle: }
End Section: omega>>
End of File
```

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

For a correctly formatted file in Part 2, the output would be: `The puzzle has t tiles.` where `t` is the total number of tiles in the **Puzzle** . For the file in Figure 1, the output would be `The puzzle has 59 tiles.` (if Rule 2 hadn't been violated).

2.1.1 Invalid Syntax & Semantics in Parsing

For invalid Boss Puzzle encodings in Part 2, a message describing the parsing error should be displayed. For an unrecognized token (not in the alphabet of tokens), the output `Line L Contains Problem(s)` should be displayed, where `L` is the line number where the problem occurred. On a syntax error, the parser should stop processing the file. For a semantic violation, the output `Semantic Error: Rule R Violated` 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 `csce322hmwrk01prt02error.java`.

3 Naming Conventions

The ANTLR file for the first part of the assignment should be named `csce322hmwrk01prt01.g4`. The ANTLR file for the second part of the assignment should be named `csce322hmwrk01prt02.g4`. Both grammars should contain a start rule named `boss`. The Java file for the second part of the assignment should be named `csce322hmwrk01prt02error.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, submission output will be sorted and only unique lines compared against the solution, so the order of output and duplication will not be a factor.

5 Point Allocation

Component	Points
Part 1	
Test Cases	2×10
Compilation	15
Total	35
Part 2	
Test Cases	3×15
Compilation	20
Total	65
Total	100

6 External Resources

ANTLR
Getting Started with ANTLR v4
ANTLR 4 Documentation
Overview (ANTLR 4 Runtime 4.8 API)

7 Commands of Interest

```
alias antlr4='java -jar /path/to/antlr-4.8-complete.jar'
alias grun='java org.antlr.v4.runtime.misc.TestRig'
export CLASSPATH="/path/to/antlr-4.8-complete.jar:$CLASSPATH"
antlr4 /path/to/csce322hmwrk01prt0#.g4
javac -d /path/for/.classfiles /path/to/csce322hmwrk01prt0#*.java
```

```
java /path/of/.classfiles csce322hmwrk01prt02driver /path/to/inputfile
grun csce322hmwrk01prt0# boss -gui
grun csce322hmwrk01prt0# boss -gui /path/to/inputfile
grun csce322hmwrk01prt0# boss
grun csce322hmwrk01prt0# boss /path/to/inputfile
```