Homework 01 (Due: Friday, September 28, 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 UNO. 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 thre (3) labeled sections: **Hands** , **Deck** , and **Discard** . Each section is enclosed by start and end tags (`>>` and `<<`, respectively).
- **Deck** and **Discard** are space-separated lists of colors and symbols that appear between `^` and `$` tokens. Valid cards are colors `r`, `b`, `g`, `y`, and `w` followed by valid symbols (numbers 0 through 9) and `r`, `s`, `d`, and `-`.
- **Hands** contains a two-dimensional array of space-separated entries that use the same encoding as **Deck** and **Discard** to encode the hands of cards that each player holds. Rows will be ended with a `*` and the **Hands** will be begun with a `{` and ended with a `}`.

An example of a properly formatted file is shown at https://cse.unl.edu/~cse322/homework/01/test01.uno.

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 UNO 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 UNO encoding, the types of tokens that you will need to consider are given in Table 1.

Type	Form
Section Beginning	`>>`
Section Ending	`<<`
Section Title	`!hands`, `!deck` and `!discard`
Color Symbol	`r`, `b`, `g`, `y`, and `w`
Value Symbol	`0` through `9` and `r`, `s`, `d`, and `-`
Row Ending	`*`
Hands Beginning	`{`
Hands 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 UNO encodings, the output `NOTICE: Trouble on Line L` should display. `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 UNO 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 **Hands /Discard /Deck** or **Hands /Deck /Discard** order.
- Each hand must have at least one card.
- There must be at least two hands in the **Hands** section.
- The **Deck** and **Discard** sections must each contain at least one card.

The semantics of a properly formatted UNO encoding are:

1. Each game must have between two (2) and ten (10) players (hands)
2. No more than 10% of the possible cards have been played (placed in the **Discard** section)
3. **Extra Credit (**10 **points or Honors contract)**: No player has four (4) or more cards than any other player.

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 **??**, the output would have the form of the output presented in Figure 1

```
Area: !hands
Start of Section
Start of Hands
Card: b2
Card: b6
Card: b1
Card: y0
Card: y3
End of Hand
Card: y7
...
Card: r6
End of Hands
End of Section
Area: !discard
Start of Section
Start of List
Card: g2
...
Card: y4
End of List
End of Section
Area: !deck
Start of Section
Start of List
Card: rs
...
Card: b4
End of List
End of Section
End of UNO File
```

Figure 1: Truncated Output of Scanner for File in Figure **??**

For a correctly formatted file in Part 2, the output would be: `c cards have been played.` where `c` is the number of cards that have been played. For the file in Figure **??**, the output would be `9 cards have been played.` (if you were not attempting extra credit).

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
`NOTICE: Trouble on Line L` should be displayed, where `L` is the line number where the problem was found. For that error, the parser should stop processing the file. For a semantic rule violation, the output `TROUBLE: Semantic 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 `csce322h0mework01part02error.java`.

3 Naming Conventions

The ANTLR file for the first part of the assignment should be named `csce322h0mework01part01.g4`. The ANTLR file for the second part of the assignment should be named `csce322h0mework01part02.g4`. Both grammars should contain a start rule named `uno`. The Java file for the second part of the assignment should be named `csce322h0mework01part02error.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.

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 4.7.1 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/csce322h0mework01part0#.g4
javac -d /path/for/.classfiles /path/to/csce322h0mework01part0#*.java
java /path/of/.classfiles csce322h0mework01part02driver /path/to/inputfile
grun csce322h0mework01part0# uno -gui
grun csce322h0mework01part0# uno -gui /path/to/inputfile
grun csce322h0mework01part0# uno
grun csce322h0mework01part0# uno /path/to/inputfile
```