CS-5340/6340, Programming Assignment #2
Due: Friday, October 7, 2016 by 11:00pm

Your task is to build a syntactic parser from scratch using the **CKY chart parsing algorithm**. Your parser should accept two input files: (1) a grammar file, and (2) a sentences file. Your program should be named **"parser"** and should accept the files as command-line arguments in the following order:

parser <grammar_file> <sentences_file>

The Grammar File

The grammar file will consist of context-free grammar rules in CNF form. Therefore you can assume that each rule consists of exactly two non-terminal symbols on its right-hand side, or exactly one terminal symbol on its right-hand side. The left-hand side of each rule will be separated from the right-hand side by the symbols "->". Each rule will be on a separate line.

Here is a sample grammar file:

```
S -> NP VP
S -> VP NP
NP -> I
NP -> fish
NP -> trust
NP -> shrinks
VP -> fish
VP -> shrinks
VP -> trust
```

The Sentences File

The sentences file will contain sentences that you should give to your parser as input. The words will be separated by whitespace. Each sentence will be on a separate line.

A sample sentences file is shown below:

```
I fish
trust shrinks
```

Your program should treat the words as **case sensitive** (e.g., the words "dog", "Dog", and "DOG" are *different* words).

CKY Parsing Algorithm

You should implement the CKY parsing algorithm, as outlined on the lecture slides and in the textbook. Your parser should identify all possible parses for each sentence with respect to the given grammar, or determine that no legal parses can be generated.

Output Specifications

For each sentence in the input file, your program should print three things:

1. **PARSING SENTENCE:** <sentence>
2. **NUMBER OF PARSES FOUND: <number>**
   The number of parses corresponds to the number of S constituents found in cell[1,N] of the chart, where N is the length of the input sentence.
3. **CHART:** followed by the contents of each cell in the chart on a separate line.

   For an NxN matrix, the CKY algorithm only uses the cells for the [i][i] diagonal and above that, so please only print the cells used by CKY in your output. Print the chart entries for the rows from top to bottom. Within a row, print the entries from left to right (i.e., corresponding to reading the sentence from left to right).

   For each cell[r][c] in the chart, print the list of constituents (non-terminals) that have been successfully produced by the parser for words r through c. Important: please print the non-terminals in alphabetical order! If NO constituents have been produced, then print a hyphen (-).

For example, given a sentence of length 3, your output should be formatted like this:

```
PARSING SENTENCE: <sentence>
NUMBER OF PARSES FOUND: <num>
CHART:
  chart[1,1]: <entries>
  chart[1,2]: <entries>
  chart[1,3]: <entries>
  chart[2,2]: <entries>
  chart[2,3]: <entries>
  chart[3,3]: <entries>
```

Here is the output for the grammar and sentences shown earlier:

```
PARSING SENTENCE: I fish
NUMBER OF PARSES FOUND: 1
CHART:
  chart[1,1]: NP
  chart[1,2]: S
  chart[2,2]: NP VP

PARSING SENTENCE: trust shrinks
NUMBER OF PARSES FOUND: 2
CHART:
  chart[1,1]: NP VP
  chart[1,2]: S S
  chart[2,2]: NP VP
```

Grading Criteria

Your program will be graded based on new input files! **So please test your program thoroughly to evaluate the generality and correctness of your code!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on new input.

Please exactly follow the formatting instructions specified in this assignment. We will deduct points if you fail to follow the specifications because it makes our job much more difficult to grade programs that do not conform to the same standards.

IMPORTANT: You may not use ANY external software packages or dictionaries to complete this assignment except for *basic* libraries. For example, libraries for general I/O handling, math functions, and regular expression matching are ok to use. But you may not use libraries or code from any NLP-related software packages, or external code that performs any functions specifically related to syntactic parsing. All submitted code *must be your own.*

If you are unsure about the acceptability of anything, please email us at `teach-cs5340@list.eng.utah.edu`

ELECTRONIC SUBMISSION INSTRUCTIONS

You need to submit 4 things:

1. The source code files for your program. Be sure to include all files that we will need to compile and run your program!

   REMINDER: your program **must** be written in Python or Java, and it **must** compile and run on the Linux-based CADE (lab1 or lab2) machines! We will not grade programs that cannot be run on the Linux-based CADE machines.

2. An executable *shell script* named `parser.sh` that contains the exact commands needed to compile and run your parser program on the data files provided on the cs5340 web page. We should be able to execute this file on the command line, and it will compile and run your code. For example, if your code is in Python and does not need to be compiled, then your script file might look like this:

   ```
   python parser.py grammar.txt sentences.txt
   ```

   If your code is in Java, then your script file might look something like this:

   ```
   javac Parser/*.java
   java Parser/MainClass grammar.txt sentences.txt
   ```

3. A `README.txt` file that includes the following information:

   - Which CADE machine you tested your program on
     (this info may be useful to us if we have trouble running your program)
   - Any known idiosyncracies, problems, or limitations of your program.

4. Submit one trace file called **parser.trace** that shows the output of your program when given the input files on the CS-5340 web page.

You can generate a trace file in (at least) 3 different ways: (1) print your program's output to a file called `parser.trace`, (2) print your program's output to standard output and then pipe it to a file (e.g., `parser grammar.txt sentences.txt > parser.trace`), or (3) print your output to standard output and use the unix *script* command before running your program on the test files. The sequence of commands to use is:

```
script parser.trace
parser grammar.txt sentences.txt
exit
```

This will save everything that is printed to standard output during the session to a file called `parser.trace`.

To turn in your files, the CADE provides a web-based facility for electronic handin, which can be found here:

`https://webhandin.eng.utah.edu/`

Or you can log in to any of the CADE machines and issue the command:

`handin cs5340 parser <filename>`

HELPFUL HINT: you can get a listing of the files that you've already turned in via electronic handin by using the 'handin' command without giving it a filename. For example:

`handin cs5340 parser`

will list all of the files that you've turned in thus far. If you submit a new file with the same name as a previous file, the new file will overwrite the old one.