

CSE 4714 / 6714 – Theory and Implementation of Programming Languages

Lab 2 – Parsing

Create a recursive descent parser for a subset of English sentences. The following EBNF rules that describe the syntax is adapted from <http://jeffe.cs.illinois.edu/teaching/algorithms/models/05-context-free.pdf>.

Grammar Productions	First Token Set
<code><sentence> → <noun phrase> <verb phrase> <noun phrase></code>	<code>{ ARTICLE, POSSESSIVE }</code>
<code><noun phrase> → <adjective phrase> NOUN</code>	<code>{ ARTICLE, POSSESSIVE }</code>
<code><adjective phrase> → (ARTICLE POSSESSIVE) ADJECTIVE</code>	<code>{ ARTICLE, POSSESSIVE }</code>
<code><verb phrase> → VERB ADVERB <verb phrase></code>	<code>{ VERB, ADVERB }</code>

Note: Symbols shown in **red** are EBNF symbols, not actual symbols expected.

The attached rules.l file already contains the regular expressions for the accepted parts of speech: NOUN, ARTICLE, POSSESSIVE, ADJECTIVE, VERB, and ADVERB.

See the attached input and output examples for the output of the parser.

Syntax errors are found in two ways:

- If a parsing function is called and the next input token is not a member of its first token set, a syntax error has been found. For example, if you start parsing a `<verb phrase>` and the very first token is not a VERB or an ADVERB, you found a syntax error and the input program is not syntactically correct.
- If the next input token is not what is expected at any point in the parsing of the input program, a syntax error has been found. For example, if your program starts parsing an `<adjective phrase>` and finds that the first token is an ARTICLE, the very next token has to be an ADJECTIVE. If the next token is anything else, you found a syntax error.

If an error is found in the input, the parser should print an error message and halt. Expected error messages are:

- `"<sentence> did not start with an article or possessive."`
- `"<noun phrase> did not start with an article or possessive."`
- `"<noun phrase> did not have a noun."`
- `"<adjective phrase> did not start with an article or possessive."`
- `"<adjective phrase> did not have an adjective."`
- `"<verb phrase> did not start with a verb or an adverb."`

Use the files in Lab_2_Starting_Point.zip as a starting point for your parser.

Samples

See *.in for example input files. The expected outputs are in corresponding *.correct files.

HINT: Use diff to compare the output of your program to the expected output files. This is much easier than trying to compare the files manually. The following is what I like to use:

```
$ F=input1; cat $F.in; parse $F.in | diff - $F.correct
the little dog ate my green trousers
```

Deliverable

The deliverable is a zip file of the files needed to build and execute your parser (rules.l, parser.h, driver.cpp, parser.cpp, makefile, etc.). Create a zip file named *netid_lab_2.zip* and upload that file to the assignment. Do not include any files generated by the makefile in your submission. For example, your submission should not include any .o or .exe files.

Feel free to assist other students on this lab assignment. If you need additional help, please contact the TA or attend their office hours.

The sample input / output files provided with the assignment are a good starting point for testing your program. You are responsible for testing your program so that the lexical analyzer will function as expected with *any* input.

Initial submissions that are late will be penalized 10 points for each day past the due date/time. The timestamp of your last submission will be used to calculate the late penalty.

Any code that does not compile / run in the Windows Subsystem for Linux will receive a grade of 0.

Hint

Do not try to write the complete parser before starting to debug your code. Start with the easier statements first, such as <adjective phrase>. Then add the additional rules after confirming that the simplified versions work as expected.