CMPSC 461: Programming Language Concepts

Spring 2018

Programming assignment 1: Recursive Descent Parsing
Total: 20 points. Due on Feb 5th at 12:20pm in Canvas.

Your assignment is to use Java to write a recursive descent parser for a simplified HTML language. The lexical syntax is specified by regular expressions:

Token	Extended regular expression definition
STRING	(LETTER | DIGIT)+
KEYWORD	<body> | </body> | | | | | | | |

In the above, LETTER is any lower or upper-case letter and DIGIT is any digit. An arbitrary number of whitespace can appear between tokens.

You may already know that ... is the tag for bolded text in HTML, ... for italicized text, <ul>... for an unordered list, and <li>... for a list item.

Note in the above syntax we use a notation for non-terminals that is different from the notation we used in lectures. The reason is that symbols < and > are terminals in the HTML language; so we can no longer use the notation <A> for non-terminals. Instead, we use names in upper-case letters for non-terminals. Therefore, in the above token syntax, KEYWORD is a non-terminal, while <body> is a string of terminals that starts with terminal < and ends with terminal >.

Using the same notation, the syntax of the simplified HTML language is specified by the following E-BNF grammar, where WEBPAGE is the start non-terminal:

```
WEBPAGE -> <body> { TEXT } </body>
TEXT -> STRING | <b> TEXT </b> | <i> TEXT </i> | <ul> { LISTITEM } </ul>
LISTITEM -> <li> TEXT </li>
```

Note that { and } are meta-symbols in E-BNF.

An example expression in the language is as follows:

```
<body> google <b><i><b> yahoo</b></i></b></body>
```

This programming project is broken down into the following series of tasks.

1. (3 points) Write a class Token that can store the value and category of any token.

2. (7 points) Develop a lexical analyzer. You should start by drawing on paper a finite state automaton that can recognize types of tokens and then convert the automaton into code. Name the class Lexer and ensure that it has a public method nextToken() which returns a Token. Lexer should have a constructor with the signature Lexer(String input), where input is the string representing the query to be parsed. If nextToken() is called when no input is left, then it should return a token with category EOI (EndOfInput). If the next terminal string is not considered a token by the lexical syntax, then return a token with category Invalid.

3. (10 points) Write a syntactic analyzer which parses the tokens given by Lexer using the recursive descent technique. Name this class Parser. Like Lexer, Parser should have a public constructor with the signature Parser(String input). This input string should be used to create a Lexer object. There should also be a public method run() that will start the parse.

   As you parse the input, Parser should output (by writing to System.out) the token that was matched in a new line with indentation reflecting the nesting structure. In particular, when there is a token nested inside a tag such as `<body>`, the token's indentation should be two spaces more than the indentation of the outer tag. An example will be provided shortly.

   Hint: to have proper indentation, one approach is to make your parser methods for non-terminals `WEBPAGE`, `TEXT`, and `LISTITEM` take an indentation level as a parameter. When a parser method is invoked inside another parser method, the indentation level should be increased by one.

   Whenever the parser comes across a token that does not fit the grammar, it should output a message of the form "Syntax error: expecting expected-token-category; saw token" and immediately exit.

Note that for this assignment you are allowed to base your code on the example lexer and parser we discussed in class.

In order to test your file, you will have to create a Test class with a main() method that creates one or more Parser objects using different query strings and then calls the run() method on each object. For example, the code:

```
<body>
  google
  <b>
    <i>
      <b>
        yahoo
      </b>
    </i>
  </b>
</body>
```

Figure 1: Sample output.

```
Parser parser =
  new Parser ("<body> google <b><i><b> yahoo</b></i></b></body>");
parser.run();
```

should have the output given in Figure 1.

Submission format: The electronic version of your program files must be submitted through Canvas.

Make sure that your code can be compiled and run on those Linux machines in the Westgate W204 lab (cse-p204inst01.cse.psu.edu to cse-p204inst38.psu.edu). We will grade your submission on those machines.

Make sure your code is well tested. We provided one test for your reference, but you should design your own test cases to test your code.

Please zip all your .java and .class files into one single file and submit the zipped file. Please ensure that each of your files has a comment that includes your name, Penn State network user id, and a description of the purpose of the file. Also say which version of the Java you are using (e.g., Java 1.7) and what operating system under which you compiled your program. Please include comments for each method, as well as any complicated logic within methods.