Weekly Reading. Ch. 1: Sections 1.7; Ch. 3: Sections 3.1–3.2

Terms and concepts you should be familiar with:

- Syntax vs Semantics
- Sentence (statement) vs Token vs Lexeme
- Grammar (production) rules including left-hand sides, right-hand sides, terminal symbols, non-terminal symbols, start symbols
- Grammar derivations

Part 1: Exercises. Complete the questions below and and in your answers with your hard-copy.

(a). Define a grammar that will generate all pairs of matched curly braces. As examples, the strings “{}”, “{{}”,”{"{}}{}{}}” should be in the language defined by your grammar but “{{}” and “{{}}” should not. Give two derivations of strings (of your choice) using your grammar.

(b). Define a grammar for Boolean formulas over variables. Assume variables consist of a single lower-case letter. Let “and” denote conjunction, “or” denote disjunction, “not” denote negation, and square brackets []’s denote (optional) evaluation order. Your grammar should ONLY allow well-formed Boolean formulas. The following are examples of well-formed formulas (one per line).

\[
\begin{align*}
\text{p} \\
\text{p or q} \\
\text{q or [p and not q]} \\
\text{[p and not [q or r] and [q and not s]]}
\end{align*}
\]

(c). Consider the following simple Python function.

```python
def f(x, y):
    if x < y:
        return y - x
    else:
        return x - y
```

Start the Python REPL (i.e., run the `python` program from the command line), enter the code above in the REPL, then do the following to obtain the corresponding (CPython) bytecode for the function.
>>> import dis
>>> dis.dis(f)

Note that `import dis` (within CPython, which is the standard implementation of Python) loads the disassembler module and `dis.dis` calls the disassemble function `dis` within the module (which produces the corresponding bytecode). Give the bytecode produced and a brief description of what you think the bytecode for the example is saying. Use https://docs.python.org/3/library/dis.html as a reference as needed.

(d). Consider the following simple Java program:

```java
public class BCTest {
    public static int f(int x, int y) {
        if(x < y)
            return y - x;
        else
            return x - y;
    }
    public static void main(String[] args) {
        System.out.println(f(1, 2));
    }
}
```

Compile the file (via `javac`) and then run the `javap -c` command on the class file to view its corresponding bytecode. Look up (e.g., here: https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings) what each of the bytecode instructions do (e.g., `aload_0`, `invespecial`, `invoke_virtual`, and so on). Give the bytecode produced and a brief description of what you think the bytecode for the example is saying.

**Part 2: MyPL Lexical Analyzer.** Implement a MyPL lexical analyzer (lexer) using Java. You will be provided basic starter code via GitHub Classroom for the assignment, including a `Lexer.java` file (where the majority of code needs to be written), a `Token.java` file, a `TokenType.java` file, a `MyPLException.java` file, and a `HW2.java` file (test driver). Note that you must use the following token types for your implementation (which are defined in `TokenType.java` file).

```
COMMA    DOT    PLUS    MINUS    MULTIPLY    DIVIDE
MODULO   EQUAL  GREATER_THAN GREATER_THAN_EQUAL LESS_THAN LESS_THAN_EQUAL
NOT_EQUAL LPAREN RPAREN ASSIGN EOS INT_VAL
DOUBLE_VAL CHAR_VAL STRING_VAL BOOL_VAL INT_TYPE BOOL_TYPE
DOUBLE_TYPE CHAR_TYPE STRING_TYPE TYPE AND OR
NOT      NEG     WHILE     FOR TO DO
IF       THEN    ELSE      ELIF END FUN
VAR      SET     RETURN   NEW NIL ID
```

Your `HW2.java` program should take a source file written in MyPL and output the set of tokens in the file. It should be implemented such that it passes the file to a `Lexer` object, and then calls the `next_token()` function repeatedly until the end of file (stream) is reached, printing to standard out each returned token. For example, given this simple program (stored in `p1.mipl`):
print("Hello World!")

Running your test program should print the following.

```
$ java HW2 p1.mypl
ID 'print' 1:1
LPAREN '(' 1:6
STRING_VAL 'Hello World!' 1:7
RPAREN ')' 1:21
EOS '' 2:0
```

In particular, each line should consist of the token type, the lexeme, and the location, where \(x:y\) stands for line \(x\) and column \(y\). As part of this assignment, you must also develop additional test cases beyond those provided. Note that your tests must include both the input files and the output produced after running your lexer over the files. Be sure to include examples that are syntactically incorrect (from the perspective of what the lexer checks in terms of basic tokens). As an example, the following source file

```
"hello
world!"
```

should output

```
Lexer error: found newline within string at line 1 column 7
```

**What to Turn In:** You must hand in the following by the due date for your assignment to be considered complete.

- A cover sheet with your name, the assignment number, and the date filled in
- A hard copy print out of your test files
- A hard copy print out showing your code runs correctly over all tests
- A hard copy print out of your “discussion” write up (see cover sheet)
- All program source code submitted through GitHub (instructions provided separately)