Prof. Jingke Li (FAB120-06, lij@pdx.edu), Tue 10:00-11:50 @SRTC 155, Labs: Tue 14:00-15:50 & Wed 12:00-13:50 @FAB 88-10

Homework Assignment 1: Lexer

(Due 10/23/14 @ 11:59pm)

In this assignment, you are going to implement two versions of lexer for a small language, "miniJava": one through manual implementation and the other with the lexer-generation tool, JavaCC. This assignment builds on top of Lab 2. If you have not attended Lab 2 for any reason, please go through its materials first. The assignment carries a total of 100 points.

Preparation

Download the zip file "`hw1.zip`" from D2L. After unzipping, you should see a `hw1` directory with the following items:

- `hw1.pdf` — this document
- `mjLexicalRules.pdf` — miniJava's lexical rules
  This document provides the base for the lexer implementation.
- `mjTokenConstants.java` — suggested internal token code for the hand-written lexer
- `Lexer0.jj`, `Lexer0.java` — sample starting versions for the two lexers
- `Makefile` — for compiling the lexers
  (*Usage:* "`make lexer1`", "`make lexer2`", or just "`make`" for both)
- `runl1`, `runl2` — two scripts for running the tests
  (*Usage:* "`./runl1 tst/*.txt`", "`./runl2 tst/*.txt`")
- `tst/` — a set of test inputs (incomplete coverage)

Your Tasks

1. **Implementing a lexer using JavaCC.** (45 points)

   Name your JavaCC lexer program `Lexer1.jj`. Include the name `Lexer1` as the package name in the header section of the program (see `Lexer0.jj`):

   ```
   PARSER_BEGIN(Lexer1)
   public class Lexer1 {
     ...
   }
   PARSER_END(Lexer1)
   ```

2. **Implementing a lexer manually.** (45 points)

   Name your hand-written lexer `Lexer2.java`. You are free to organize this program in any way you see fit, but here is a top-level structure used in Lab 2 lexer programs (see `Lexer0.java`):

```
public class Lexer2 implements mjTokenConstants {

  static class LexError extends Exception {
    int line;
    int column;
    ...
  }

  static class Token {
    int kind;           // token code
    int line;           // line number of token's first char
    int column;         // column number of token's first char
    String lexeme;      // lexeme string
    ...
  }

  public static void main(String [] args) {
    ...
  }
}
```

3. **Developing four testing inputs.** (10 points)

   The provided test inputs, `testinput*.txt`, cover only correct tokens. Your third task is to develop four new test inputs for testing the four types of lexical errors (see below). Name your files `testerror1.txt` — `testerror4.txt`.

Requirements

The following are the common requirements for both the hand-written and the JavaCC-generated lexers.

- Recognize all the tokens specified in the file, `mjLexicalRules.pdf`.
- Display tokens from an input stream in the following format:
  - One token per line.
  - Each line starts with a pair `(linNum,colNum)`, where the two numbers are the starting line and column numbers of the token. Then,
    - . for an `ID` token, display `ID(lexeme);`
    - . for an `INTLIT` token, display `INTLIT(lexeme's value);`
      (*e.g.* for the input sequence, `00123`, the display should be `INTLIT(123)`)
    - . for an `STRLIT` token, display `STRLIT(lexeme);`
    - . for any other token, display `lexeme`.
  - After all tokens are displayed, show one last line: `Total: tknCnt tokens`.

  Exact spacing among items within a line is not required.

Here is an example:

Input:

```
int i = 00023; "Hello World!"
The end.
```

Output:

```
(1,1)   int
(1,5)   ID(i)
(1,7)   =
(1,9)   INTLIT(23)
(1,14)  ;
(1,16)  STRLIT("Hello World!")
(2,1)   ID(The)
(2,5)   ID(end)
(2,8)   .
Total: 9 tokens
```

- Detect all lexical errors. For miniJava, there are four possible types of lexical errors:
  - illegal characters
  - integer overflow
  - unterminated strings
  - unterminated comments

  Your lexers' error message should indicate the type of lexical error. Optionally, it may provide additional information, such as line and column numbers. (But this is not required.)

Minimum Requirement for Passing The minimum requirement for passing this assignment is that both of your lexers compile on the CS Linux system without error and each can correctly recognize at least one token. By satisfying this requirement, you will secure 10 points on the assignment.

Submission

Submit a single zip file, `hw1sol.zip`, containing, `Lexer1.jj`, `Lexer2.java` (and `mjTokenConstatnts.java` if you used it), and `testerror[1-4].txt` through the "Dropbox" on the D2L class website. You don't need to encode your name in the zip file name; D2L automatically does that. *But don't forget to include your name in your programs.*.