Assignment Description

In this programming assignment, you will implement the scanner for the IC compiler. The IC language specification document is available on the course web page.

What to Implement

You will implement the scanner using JFlex. You will also build a driver program for the scanner and a test suite. You are required to implement the following:

- `class Token`. The lexer returns an object of this class for each token. The `Token` class must contain at least the following information:

  - `line`, the line number where the token occurs in the input file;
  - `id`, an integer identifier for the token; and
  - `value`, an arbitrary object holding the specific value of the token (e.g. the character string, or the numeric value).

  The numeric identifiers for all of the tokens must be placed in a file `sym.java` containing a class `sym` with the following structure:

  ```
  public class sym {
    public static final int IDENTIFIER = 0;
    public static final int LESS_THAN = 1;
    public static final int INTEGER = 2;
    ...
  }
  ```

  Note: in the next assignment, this file will be automatically generated by the parser generator `java_cup`.

- Lexer specification: `IC.lex`. Using this specification as input to JFlex must produce a file `Lexer.java` containing the lexical analyzer generator:

                              java JFlex.Main IC.lex

  The generated scanner will produce objects of the class `Token`.

- `class Compiler`. This will be the main class of your compiler at the end of the semester. At this point, this class is just a testbed for your scanner. It takes a single filename as an argument, it reads that file, breaks it into tokens, and successively calls the `next_token` method of the generated scanner to print a representation of the file as a series of tokens to the standard output, one token per line. The format of each line is

                              LINE: ID(VALUE)

  where `LINE` is the line number of the token, `ID` is the token identifier, and `(VALUE)` is the value of the token. If the token has no value omit the parentheses. The last line should be `LINE: EOF`. Use the following naming convention for the token printouts:
  `=` ASSIGN, `boolean` BOOLEAN, `break` BREAK, `class` CLASS, class identifier CLASS_ID, `,` COMMA, `continue` CONTINUE, `/` DIVIDE, `.` DOT, `==` EQUAL, `extends` EXTENDS, `else` ELSE, `false` FALSE, `>` GT, `>=` GTE, identifier ID, `if` IF, `int` INT, integer INTEGER, `&&` LAND, `[` LB, `(` LP, `{` LCBR, `length` LENGTH, `new` NEW, `!` LNEG, `||` LOR, `<` LT, `<=` LTE, `−` MINUS, `%` MOD, `*` MULTIPLY, `!=` NEQUAL, `null` NULL, `+` PLUS, `]` RB, `}` RCBR, `return` RETURN, `)` RP, `;` SEMI, `static` STATIC, `string (keyword)` STRING, `quoted string` QUOTE, `this` THIS, `true` TRUE, `void` VOID, `while` WHILE,

  At the command line, your program must be invoked with exactly one argument:

                              java IC.Compiler <file.ic>

- **class LexicalError**. Your scanner should also detect and report any lexical analysis errors it may encounter. You must implement an exception class for lexical errors, which contains at least the line number where the error occurred and an error message. Whenever the program encounters a lexical error, the scanner must throw a `LexicalError` exception and the main method must catch it and terminate the execution. Your program must always report the first lexical error in the file.

Code Structure: All of the classes you write should be in or under the package `IC`, containing the following:

- the class `Compiler` containing the main method;

- the `IC.Parser` sub-package, containing the `Lexer`, `sym` classes, and `LexicalError` class.

The course web-site contains a zipped file with the correct directory and file structure. The zip file also contains a `build.xml` file that performs basic operations such as compiling, cleaning, etc.

Testing the scanner: We expect you to perform your own testing of the scanner. You should develop a thorough test suite that tests all legal tokens and as many lexical errors as you can think of. We will test your scanner against our own test cases – including programs that are lexically correct, and also programs that contain lexical errors. You may find it useful to go over the ASCII table to make sure your scanner doesn't miss cases. Make sure you handle comments and quoted strings appropriately.

Other tools: You may consider the automation of this process using makefiles, shell scripts or other similar tools. We recommend using the Apache Ant tool, which is a kind of a platform-independent make utility (JFlex and JavaCup come with Ant tasks).

We highly recommend using the Eclipse IDE. Eclipse has many useful features such as code navigation, text completion, unit testing, and debugging. All of these can significantly help increase your productivity in this project.

What to turn in

You must turn in your code electronically in the team account on the due date, including a documentation write-up. **Please include only the source files in your submission, not the compiled class files or any other temporary files.**

As in any other large program, much of the value in a compiler is in how easily it can be maintained. For this reason, a high value will be placed here on both clarity and brevity – both in documentation and code. Make sure your code structure is well-explained in your write-up and in your javadoc documentation.

Turn in a document PA1-DOC.XXX (XXX can be one of txt/doc/pdf) with the following information:

- A brief, clear, and concise (1–2 pages) description of your code structure and testing strategy. Include in this document a list of regular expressions for all the tokens that your scanner recognizes.

- a description of the class hierarchy in your src directory, brief descriptions of the major classes, any known bugs, and any other information that we might find useful when grading your assignment. Documented bugs will be graded more forgivingly than non-documented bugs.

Electronic Submission Instructions.   Please organize your top-level directory structure as follows:

- `src` - all of your source code.

- `doc` - any documentation, including your write-up (PA1-DOC).

- `test` - any test cases you used in testing your project.

Note: Failure to submit your assignment in the proper format may result in deductions from your grade.