Assignment Description

In this programming assignment, you will have to implement the lexical analysis for the IC language, defined on the web page at `http://www.cs.cornell.edu/courses/cs412/2004sp/ic/icsspec.ps`. You will build the lexer using the JLex lexical analyzer generator. Detailed documentation and examples for JLex can be found on the JLex web page at: `http://www.cs.princeton.edu/~appel/modern/java/JLex`.

What to implement

You have to implement a lexical analyzer using Jlex, as well as a testbed program for the lexer, and an error-handling mechanism. 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:
  - `id`, an integer identifier for the token;
  - `value`, an arbitrary object holding the specific value of the token (e.g. the character string, or the numeric value);
  - `line`, an integer representing the line number where the token occurs in the input file.

  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;
    ...
  }
  ```

- `Lexer` specification. Compiling this specification with JLex must produce the `Lexer.java` file containing the lexical analyzer generator:

  `java JLex.Main Lexer`

- `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 lexer. 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 lexer to print a representation of the file as a series of tokens to the standard output, one token per line. Your output must include the following information: the token identifier, the value of the token (if any), and the line number for that token. At the command line, your program must be invoked with exactly one argument:

  `java IC.Compiler <file.ic>`

- `class LexicalError`. Your lexer 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 lexer 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.Lexer` sub-package, containing the `Lexer` and `sym` classes;
- the `IC.Error` sub-package, containing the `LexicalError` class;

Testing the lexer: We expect you to perform your own testing of the lexer. 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 lexer against our own test cases – including programs that are lexically correct, and also programs that contain lexical errors.

Other tools: It is recommended that you start using the CVS system. This is a useful tool for managing the concurrent code development by multiple persons. Such a tool will become more useful in the following assignments, which will be significantly larger than this first assignment. You should therefore use this assignment as a chance to set up your code production and testing process. You may also consider the automation of this process using makefiles, shell scripts or other similar tools.

What to turn in

You must turn in your code electronically using the Course Management System (CMS) on the due date, and submit your a short write-up the next day in class. You must submit your source code as a tarball `pa1.tar.gz` using CMS, anytime on the due date (i.e. until 11:59pm). Please include only the source files in your submission, not the compiled class files. You must provide a Makefile for compiling your sources and generating javadoc documentation.

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 on paper:

- A brief, clear, and concise document (at most 2 pages) describing the your code structure and testing strategy. Include a list of regular expressions for all the tokens that your lexer recognizes. Make sure you mention any known bugs and other information that we might find useful when grading your assignment.
- Feedback – please provide your overall thoughts about the assignment: how much time you spent on it, what was the most difficult or more interesting part, and how you think it could be made better.

Turn in electronically:

- All of your source code and test cases.
- A Makefile for: a) compiling the sources to class files; and b) generating javadoc documentation.

The structure of your working directory should be organized as follows:

- `/src` - all of your source code, containing the `/IC` directory and your `Makefile`.
- `/test` - any test cases you used in testing your project.
- `/tools` - containing the `JLex` and `java_cup` tools that you will use in the project.
- `/classes` - the class files generated by compiling your source code.
- `/doc` - generated javadoc documentation.

Your electronic submission should contain only the `src` and `test` directories. Do not include `JLex` or `java_cup` code in your submission. Assume instead that java will find their class files using the `CLASSPATH` variable. Failure to submit your assignment in the proper format may result in deductions from your grade.