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. You will build the scanner using the JFlex lexical analyzer generator. Examples and documentation for this tool can be found on the JFlex home page. Check the project resources section of the CS412 web page for a link to JFlex.

What to implement

You will implement the lexical analyzer using JFlex. You will also build a driver program for the lexer, 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:
  - `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`, 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;
    ...
  }
  ```

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

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

  ```
  java JFlex.Main Lexer
  ```

  The generated scanner will produce `Token()` objects.

- `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>
  ```

- *Error Handling.* Your lexer should also detect and report any lexical analysis errors it may encounter. When a lexical error occurs, and exception will be thrown. The main method must catch it, print an error message that clearly describes the cause of error and the place where it has occurred, and then terminate the execution.

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;

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.

We highly recommend using the Eclipse IDE. Eclipse has integrated CVS support, as well as many other 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 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 11pm). *Please include only the source files in your submission, not the compiled class 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 electronically:

- A brief, clear, and concise document (1-2 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.
- All of your source code and test cases.

To make grading easier, your submission should unzip to a directory `groupX`, where `X` is your group number. The directory will contain two subdirectories:

- `/src` - all of your source code, containing the `/IC` directory and your `Makefile`.
- `/test` - any test cases you used in testing your project.
- `/writeup` - containing the your writeup file.

Please do not submit class files or javadoc files – we will generate them from your sources. Do not include `JLex` or `java_cup` code in your submission. Assume instead that java will find their class files using the `CLASSPATH` variable. You can use additional sub-directories for your own (e.g., `doc`, `classes`, `tools`, etc), but do not submit those. Failure to submit your assignment in the proper format may result in deductions from your grade.