Milestone 4

In this milestone, you will be extending your compiler created throughout the previous 3 milestones by adding some features related to error handling.

Error Reporting:

In this part, you will be enhancing error reporting in your compiler. Only two types of errors are currently supported: syntax and semantic errors. Any reported error must specify the type of the error, the location of the error in the input file (line and character), and the line that causes the error must be printed in the description of the error.

Syntax Errors (Manual Part Only):

For simplicity, Syntax Error reporting will not be required in CUP generated parsers.

Syntax errors occur either when you fail to match an expected token or you fail to match a grammar rule. In the first case, you are required to print the expected token, the available token, the line that has this token. For example:

```
if{x==1)

Syntax Error at line 10 char 3:
Expected '('
Found '{'
In: If{x==1)
```

In the second case, print an error message that explains what rule you fail to match. For example:

```
static int gcd(int x, y) {

Syntax Error at line 5 char 23:
Cannot match 'y' to a type in:
static int gcd(int x, y) {
```

Semantic Errors:

Semantic errors will be the same as milestone 3 with the extra requirements in the error report. For example:

```
static int avg(int x, int y) {
    int sum;
    sum = x + z;
```

```
Semantic Error at line 7 char 15:
Variable 'z' was not declared in this scope.
In:    sum = x + z;
```

Semantic errors are handled in the parse tree, so it should be independent from the type of parser used to generate the tree.

Error Correction (Manual and Automated):

In this part, you will be handling all possible lexical errors and correcting them. Any corrected lexical error must be accompanied by a warning printed and, similar to error reporting, the line number, the character, and line causing the error must be specified.

An example of a warning is as follows.

```
if( x == 0 & y == 1)

WARNING at line 20 char 12:
& converted to &&
In: if( x == 0 & y == 1)
```

The following are the possible errors you can handle:

- Single `'&'` and `'|'`: to be converted to `'&&'` and `'||'`.
- Invalid input is to be ignored and be treated like a white space, but with a warning printed.
- An invalid string literal is to be automatically closed and the string is returned as a token. For example, a statement like:
  ```
  s = "This is a string
  ;
  ```
  This statement should return the string token correctly and will pass syntax analysis with no problems.
- A dot at the beginning or an end of a number (ex: `.123` or `123.`)is to be counted as an invalid input and is to be ignored.

All error handling is to be implemented in the manual and the automated lexer.

Submission:

Deadline: May 27th 2012
Submission will be on:
http://ams.g-beehive.com/

Submit one Zip file containing the following two folders:

- **`Manual:`** it should include all your classes for lexical, syntax, and semantic analysis.
    - Include the provided class `Main.java` after editing it to run your compiler. Follow the instructions in `Main.java` such that it would work with command line arguments.
- Automated: it should include you JLex and CUP files and your semantic analysis classes.
    - Include the provided class `Main.java` after editing it to run your compiler. Follow the instructions in `Main.java` such that it would work with command line arguments.

Notes:

- Remove all package declaration from all your Java, JLex, and CUP files.
- Any Java code that does not compile will not be corrected.