Lexer

Tokenizing stream of characters using regular expressions

Lex

Lex is a program that generates lexical analyzer or scanner.

Structure of lex file (*.lex)

- Definition
  - define macros, import headers and etc.
- Rules
  - define regular expressions and the associated C / java / scala code block
- C code (lex or flex) / java code (jflex)
  - define utility functions which are accessible by rule code blocks

Concerning lex

- `YYINITIAL` : initial lexical state of the scanner
- `yylex()` : special function that returns the matched token
- `yybegin(<state>)` : "goto" <state>
- `%foo` : this is directive (complete list of available directives)
  - `%line` : turns on line number counter so we could use a special function `yyline()`
  - `%class` : name of generated lexer class
  - `%type` : type of returned tokens in each code block
  - `%implements` : generated class implements
  - `%state` : defines new lexical state
- `[^]` : matches all characters not listed in the class. This is used to catch errors.

Lab assignment

- In this lab we will tokenize math expressions (parsing is next week!).
- Think about the tokens we may need.
- Source file:

```
0
72
2.4
0.23
~33
1 + 2
2 - 1
3 * 4.1 - 2
1 + ~2 - ~3
42 + ~~~~~5
(3 - 2) * 5
```

Where to start

Checkout the first two lines of `Cal.y`

- To view all possible tokens to return to we should look at `Cal.y`
  - we can see all possible tokens, such as `ADD`, `SUB` and etc
  - we also see a `OPERAND` token which takes a `Double` as an argument
  - we can capture the "lexeme" by using `yytext()` method

Complete lex file

```
import java_cup.runtime.*;

%%

%class Lexer
%unicode
%cup
%line
%column

%{
  StringBuffer string = new StringBuffer();

  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }
  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);
  }
%}

LineTerminator = \r|\n|\r\n
InputCharacter = [^\r\n]
WhiteSpace     = {LineTerminator} | [ \t\f]

/* comments */
Comment = {EndOfLineComment}

EndOfLineComment     = "//" {InputCharacter}* {LineTerminator}?

DecIntegerLiteral = 0 | [1-9][0-9]*

%state STRING

%%

<YYINITIAL> "abstract"           { return symbol(sym.ABSTRACT); }
<YYINITIAL> "boolean"            { return symbol(sym.BOOLEAN); }
<YYINITIAL> "break"              { return symbol(sym.BREAK); }

<YYINITIAL> {
  /* literals */
  {DecIntegerLiteral}            { return symbol(sym.INTEGER_LITERAL); }
  \"                             { string.setLength(0); yybegin(STRING); }

  /* operators */
  "="                            { return symbol(sym.EQ); }
  "=="                           { return symbol(sym.EQEQ); }
  "+"                            { return symbol(sym.PLUS); }

  /* comments */
  {Comment}                      { /* ignore */ }

  /* whitespace */
  {WhiteSpace}                   { /* ignore */ }
}

<STRING> {
  \"                             { yybegin(YYINITIAL);
                                   return symbol(sym.STRING_LITERAL,
                                   string.toString()); }
  [^\n\r\"\\]+                   { string.append( yytext() ); }
  \\t                            { string.append('\t'); }
  \\n                            { string.append('\n'); }

  \\r                            { string.append('\r'); }
  \\\"                           { string.append('\"'); }
  \\                             { string.append('\\'); }
}

/* error fallback */
[^]                              { throw new Error("Illegal character <"+
                                                   yytext()+">"); }
```