Ted Timmons
CS321 assignment 2

Parts 1 and 2 of the assignment are combined in parser.jflex; part 3 is in htmlparser.jflex + and color/Highlighter.java. Sources for "borrowed" code segments are given.

Test files are shown in the typescript, as well as compilation results. jflex-generated code + is not included per email instructions that it wasn’t required.

```
$ PS1='$ '
$ jflex --nobak parser.jflex && javac Lexer.java
Reading "parser.jflex"
Constructing NFA : 615 states in NFA
Converting NFA to DFA :
.....................................................................................+
.....................................................................................+
.....................................................................................+
.....................................................
349 states before minimization, 284 states in minimized DFA
Writing code to "Lexer.java"
$ cat tests/question3-sample
String s = "hello world!"; // this is a sample comment
int i = 666;
int y = 2147483648;

while (i > 0 || i <= 0) {
  if (s == true) { return false; }
}

/* this is a sample comment too */
$ java Lexer tests/question3-sample
rejecting number larger than MAXINT: 2147483648 --

$ cat tests/question4-test
foo >= bar
foo <= bar
foo += bar
foo -= bar
$ java Lexer tests/question4-test
$ cat tests/question5-actual-test
"1 hello world!"
"2 hello \\ world!"
"3 hello \" world!"
"4 hello \n world!"
"5 hello \\\"\n world!"
$ java Lexer tests/question5-actual-test
$ cat tests/question5-actual-test-basic
"hi"
foo = bar;
zzb
$ java Lexer tests/question5-actual-test-basic
$ cat tests/question5-actual-test-failure
hi there!"
$ java Lexer tests/question5-actual-test-failure

$ echo "this didn't fail as it went into STRING mode and didn't come back out. Not sure if +
this is a feature or a bug. (should it be caught here or not until syntax/semantic analysis?)"
this didn't fail as it went into STRING mode and didn't come back out. Not sure if this is a +
feature or a bug. (should it be caught here or not until syntax/semantic analysis?)
$ cat tests/question5-test
public foo
abstract foo
protected foo
private foo
$ java Lexer tests/question5-test


(end of tests)
```

```
// Based on the "simple example" in the JFlex manual:
// http://jflex.de/manual.html

/* JFlex example: part of Java language lexer specification */
//import java_cup.runtime.*;
//import compiler.*;

/**
 * This class is a simple example lexer.
 */


%%

%class Lexer
%public
//not implementing compiler.* either. %extends SourceLexer
//Not implementing the Mjc* stuff. %implements MjcTokens
%unicode
%line
%column
%standalone

%{

    StringBuffer string = new StringBuffer();

    int ENDINPUT = 0;
    int BOOLEAN = 1;
    int CAND = 2;
    int CLASS = 3;
    int COR = 4;
    int ELSE = 5;
    int EQEQ = 6;
    int EXTENDS = 7;
    int FALSE = 8;
    int IDENT = 9;
    int IF = 10;
    int INT = 11;
    int INTLIT = 12;
    int NEQ = 13;
    int NEW = 14;
    int NULL = 15;
    int RETURN = 16;
    int STATIC = 17;
    int SUPER = 18;
    int THIS = 19;
    int TRUE = 20;
    int VOID = 21;
    int WHILE = 22;
    int GTEQ = 23;
    int LTEQ = 24;
    int PLUSEQ = 25;
    int MINUSEQ = 26;
    int PRIVATE = 27;
    int PROTECTED = 28;
    int PUBLIC = 29;
    int ABSTRACT = 30;
    int STRLIT = 31;
    // '!' (code=33)
    // '&' (code=38)
    // '(' (code=40)
    // ')' (code=41)
```

```
    // '*' (code=42)
    // '+' (code=43)
    // ',' (code=44)
    // '-' (code=45)
    // '.' (code=46)
    // '/' (code=47)
    // ';' (code=59)
    // '<' (code=60)
    // '=' (code=61)
    // '>' (code=62)
    // '^' (code=94)
    // '{' (code=123)
    // '|' (code=124)
    // '}' (code=125)

%}

lineterminator = \r|\n|\r\n
inputcharacter = [^\r\n]
whitespace     = {lineterminator} | [ \t\f]

comment = {traditionalcomment} | {endoflinecomment}

traditionalcomment   = "/*" [^*] ~"*/" | "/*" "*"+ "/"
endoflinecomment     = "//" {inputcharacter}* {lineterminator}
// unused: commentcontent       = ( [^*] | \*+ [^/*] )*

// Some of these macros are from/inspired by the following:
// http://users.csc.calpoly.edu/~gfisher/classes/330/examples/jflex/pascal.jflex
// changed [A-Za-z] to :jletter: to conform with Unicode.

letter          = [:letter:]
digits          = ([0-9])*
alphanumeric    = [:jletterdigit:]
other_id_char   = [_]
identifier      = {letter}({alphanumeric}|{other_id_char})*
integer         = +
214748364[0-7]|21474836[0-3][0-9]|2147483[0-5][0-9][0-9]|214748[0-2][0-9][0-9][0-9]|21474[0-7]+
[0-9][0-9][0-9][0-9]|2147[0-3][0-9][0-9][0-9][0-9][0-9]|214[0-6][0-9][0-9][0-9][0-9][0-9][0-9]+
|21[0-3][0-9][0-9][0-9][0-9][0-9][0-9][0-9]|20[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]|1[0-9][+
0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]|[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]||[0-9][0-+
9][0-9][0-9][0-9][0-9][0-9][0-9]||[0-9][0-9][0-9][0-9][0-9][0-9][0-9]||[0-9][0-9][0-9][0-9][0-9]+
[0-9]||[0-9][0-9][0-9][0-9][0-9]||[0-9][0-9][0-9][0-9]||[0-9][0-9][0-9]||[0-9][0-9]||[0-9]

%state STRING


%%
  "("     { return '('; }
  ")"     { return ')'; }
  "{"     { return '{'; }
  "}"     { return '}'; }
  ";"     { return ';'; }
  ","     { return ','; }
  "."     { return '.'; }

  "="     { return '='; }
  "=="    { return EQEQ; }

  ">"     { return '>'; }
  ">="    { return GTEQ; }
  "<"     { return '<'; }
  "<="    { return LTEQ; }
```

```
  "!"     { return '!'; }
  "!="    { return NEQ; }
  "&"     { return '&'; }
  "&&"    { return CAND; }
  "|"     { return '|'; }
  "||"    { return COR; }
  "^"     { return '^'; }
  "+"     { return '+'; }
  "+="    { return PLUSEQ; }
  "-"     { return '-'; }
  "-="    { return MINUSEQ; }
  "*"     { return '*'; }
  "/"     { return '/'; }

  // reserved identifiers
  "boolean"     { return BOOLEAN; }
  "class"       { return CLASS; }
  "else"        { return ELSE; }
  "extends"     { return EXTENDS; }
  "if"          { return IF; }
  "int"         { return INT; }
  "new"         { return NEW; }
  "return"      { return RETURN; }
  "static"      { return STATIC; }
  "super"       { return SUPER; }
  "this"        { return THIS; }
  "void"        { return VOID; }
  "while"       { return WHILE; }
  "null"        { return NULL; }
  "true"        { return TRUE; }
  "false"       { return FALSE; }
  "private"     { return PRIVATE; }
  "protected"   { return PROTECTED; }
  "public"      { return PUBLIC; }
  "abstract"    { return ABSTRACT; }

// string matching taken from:
//   http://linuxgazette.net/issue41/lopes/lopes.html
//   http://jflex.de/manual.html

<YYINITIAL> {
  \"            { string.setLength(0); yybegin(STRING); }

  {comment}     { /* throw away comments */ }
  {identifier} { return IDENT; }


  // jflex regex solution; accepts anything that fits.
  {integer}     { return INTLIT; }
  // if the digit wasn't matched in {integer}, it's too big. Fail.
  {digits}      { System.out.println("rejecting number larger than MAXINT: " + yytext() + " +
--\n"); }
  {whitespace} { /* ignore whitespace */ }
}

<STRING> {
          \"            { yybegin(YYINITIAL); return STRLIT; }
        [^\n\"\\]+    { string.append( yytext() ); }
        \\n           { string.append('\n'); }
        \\\"          { string.append('\"'); }
        \\            { string.append('\\'); }
}
```

```
  // match whatever is left and complain about it.
  .            { System.out.println("Illegal char, '" + yytext() +
                   "' line: " + yyline + ", column: " + yychar); }
```

```
$ jflex --nobak -d color htmlparser.jflex && javac color/Highlighter.java +
color/HTMLLexer.java && java color.Highlighter tests/question3-sample > +
tests/question3-sample.html
Reading "htmlparser.jflex"
Constructing NFA : 525 states in NFA
Converting NFA to DFA :
.....................................................................................+
.....................................................................................
192 states before minimization, 99 states in minimized DFA
Writing code to "color/HTMLLexer.java"
```

```
String s = "hello world!"; // this is a sample comment
int i = 666;
int y = 2147483648;

while (i > 0 || i <= 0) {
  if (s == true) { return false; }
}

/* this is a sample comment too */
```

```
<html><head><title>My Syntax Colored Web Page</title><style type="text/css">body     +
{white-space:pre;font-family:"Lucida Console","Courier New",Monotype} .keyword {color:blue} +
.comment {color: cyan} .literal {color:green} .invalid {color:red} </style></head><body>
String s = "hello world!"; <span class="comment">// this is a sample comment
</span><span class="keyword">int</span> i = <span class="literal">666</span>;
<span class="keyword">int</span> y = <span class="invalid">2147483648</span>;

<span class="keyword">while</span> (i &gt; <span class="literal">0</span> || i &lt;= <span +
class="literal">0</span>) {
  <span class="keyword">if</span> (s == <span class="keyword">true</span>) { <span +
class="keyword">return</span> <span class="keyword">false</span>; }
}

<span class="comment">/* this is a sample comment too */</span>
</body></html>
```

```
// Based on the "simple example" in the JFlex manual:
// http://jflex.de/manual.html

package color;
%%

%class HTMLLexer
%public
%unicode
%line
%column
%standalone

%{
    StringBuffer string = new StringBuffer();

    // helper functions (faux macros, really)
    void print(char c) {
      System.out.print(c);
    }
    void print(char c, String label) {
      System.out.print("<span class=\"" + label + "\">" + c + "</span>");
    }
    void print(String s) {
      System.out.print(s);
    }
    void print(String s, String label) {
      System.out.print("<span class=\"" + label + "\">" + s + "</span>");
    }
    //System.out.println("<html><head><title>My Syntax Colored Web Page</title><style +
type=\"text/css\">body     {white-space:pre;font-family:\"Lucida Console\",\"Courier +
New\",Monotype} .keyword {color:blue} .comment {color:cyan} .literal {color:green} .invalid +
{color:red} </style></head><body>");
    //System.out.println("</body></html>");

%}

lineterminator = \r|\n|\r\n
inputcharacter = [^\r\n]

comment = {traditionalcomment} | {endoflinecomment}

traditionalcomment   = "/*" [^*] ~"*/" | "/*" "*"+ "/"
endoflinecomment     = "//" {inputcharacter}* {lineterminator}
// unused: commentcontent       = ( [^*] | \*+ [^/*] )*

// Some of these macros are from/inspired by the following:
// http://users.csc.calpoly.edu/~gfisher/classes/330/examples/jflex/pascal.jflex
// changed [A-Za-z] to :jletter: to conform with Unicode.

digits          = ([0-9])*
integer         = +
214748364[0-7]|21474836[0-3][0-9]|2147483[0-5][0-9][0-9]|214748[0-2][0-9][0-9][0-9]|21474[0-7]+
[0-9][0-9][0-9][0-9]|2147[0-3][0-9][0-9][0-9][0-9][0-9]|214[0-6][0-9][0-9][0-9][0-9][0-9][0-9]+
|21[0-3][0-9][0-9][0-9][0-9][0-9][0-9][0-9]|20[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]|1[0-9][+
0-9][0-9][0-9][0-9][0-9][0-9][0-9]|[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]|[0-9][0-+
9][0-9][0-9][0-9][0-9][0-9][0-9]|[0-9][0-9][0-9][0-9][0-9][0-9][0-9]|[0-9][0-9][0-9][0-9][0-9]+
[0-9]|[0-9][0-9][0-9][0-9][0-9]|[0-9][0-9][0-9][0-9]|[0-9][0-9][0-9]|[0-9][0-9]|[0-9]

%state STRING


%%
```

```
  ">"     { print("&gt;"); }
  "<"     { print("&lt;"); }
  "&"     { print("&amp;"); }

  // reserved identifiers
  "boolean"     { print(yytext(), "keyword"); }
  "class"       { print(yytext(), "keyword"); }
  "else"        { print(yytext(), "keyword"); }
  "extends"     { print(yytext(), "keyword"); }
  "if"          { print(yytext(), "keyword"); }
  "int"         { print(yytext(), "keyword"); }
  "new"         { print(yytext(), "keyword"); }
  "return"      { print(yytext(), "keyword"); }
  "static"      { print(yytext(), "keyword"); }
  "super"       { print(yytext(), "keyword"); }
  "this"        { print(yytext(), "keyword"); }
  "void"        { print(yytext(), "keyword"); }
  "while"       { print(yytext(), "keyword"); }
  "null"        { print(yytext(), "keyword"); }
  "true"        { print(yytext(), "keyword"); }
  "false"       { print(yytext(), "keyword"); }
  "private"     { print(yytext(), "keyword"); }
  "protected"   { print(yytext(), "keyword"); }
  "public"      { print(yytext(), "keyword"); }
  "abstract"    { print(yytext(), "keyword"); }

  {comment}     { print(yytext(), "comment"); }
  {integer}     { print(yytext(), "literal"); }
  {digits}      { print(yytext(), "invalid"); }

  // match whatever is left
  .             { print(yytext()); }
```

```
package color;
import color.HTMLLexer;


//public class Highlighter extends HTMLLexer {
public class Highlighter {

  // code gratiutously stolen from the jflex output.

  public static void main(String argv[]) {
    if (argv.length == 0) {
      System.out.println("Usage : java HTMLLexer <inputfile>");
    }
    else {

      // not inserting newlines because they aren't required
      //  and it makes our typescripts shorter.
      System.out.println("<html><head><title>My Syntax Colored Web Page</title><style +
type=\"text/css\">body     {white-space:pre;font-family:\"Lucida Console\",\"Courier +
New\",Monotype} .keyword {color:blue} .comment {color: cyan} .literal {color:green} .invalid +
{color:red} </style></head><body>");

      try {
        HTMLLexer scanner = new HTMLLexer( new java.io.FileReader(argv[0]) );

        scanner.yylex();
      }
      catch (java.io.FileNotFoundException e) {
        System.out.println("File not found : \""+argv[0]+"\"");
      }
      catch (java.io.IOException e) {
        System.out.println("IO error scanning file \""+argv[0]+"\"");
        System.out.println(e);
      }
      catch (Exception e) {
        System.out.println("Unexpected exception:");
        e.printStackTrace();
      }

      // finish our HTML.
      System.out.println("</body></html>");
    }
  }


}
```