Problem 1

In this assignment, we begin the compiler project by implementing a lexical analyzer for MiniJava. To find out details of MiniJava's tokens, you will need to refer to the Appendix on pages 484-486.

We will not be using the tools JavaCC or SableCC which are discussed (briefly!) in Chapter 2. Instead we will be using tools called JLex and CIP; these are Java versions of the classic Unix tools, lex and yacc. You can find the JLex Manual at the course Moodel page.

As mentioned in the syllabus, you need to set some UNIX environment variables to access JLex. Add the following lines to your `.cshr` files:

```
setenv JAVA_HOME /depot/JSE-1.8
setenv CLASSPATH "/homes/smithg/compiler:."
```

My `/homes/smithg/compiler/minijava/chap2` directory contains a number of files to help you; you can copy them to your directory with the command

```
cp -rp /homes/smithg/compiler/minijava/chap2 .
```

The only file you need to modify is `parse/MiniJava.lex`, which contains the JLex specification. Once you have completed `MiniJava.lex`, you can run JLex and compile everything simply by typing

```
make
```

from `chap2`; and then you can run your lexer by typing

```
java parse.Main test.java
```

Please upload your `MiniJava.lex` file to the Moodle web site, and also please prepare a hard copy to submit in class. As always, it would be nice to include printouts of some sample executions.
Here are a few more tips:

- You should treat everything written in **boldface** in the grammar on page 485 as a reserved word. You may also wish to look at `parse/sym.java`, which defines constants for all of the token that you must return. (It also defines a constant for `error`, but this is not a real token; it is used by CUP for error recovery.) Note also that you should not recognize **extends**, which is supported only when MiniJava is augmented with inheritance in Chapter 14.
- Contrary to what is said on page 484, Java /* */ comments may *not* be nested.
- To interface with CUP, your lexer must return token with class `java_cup/runtime\Symbol`. You can create such objects by calling the class constructor or by using the handy method `tok` defined in `MiniJava.lex`. Notice that a `Symbol` has a `value` field of class `Object`, which holds the semantic value of the token. For most MiniJava tokens, `value` should be null, but for an `ID` it should be a `String`, and for an `INTEGER_LITERAL` it should be an `Integer`.
- For printing nice error messages, the class `errormsg/ErrorMsg` provides an `error` method that takes two parameters: an `int pos`, which tells where in the source file the error occured, and a `String msg`. To get error positions, using the JLex variable `yychar`, which gives the position of the last token matched.
- You should use JLex *lexical states* to deal with /* */ comments.
- To test your lexer, you can try out the file `test.java` and compare your output with mine in `test.out`.
- We won't be using CUP until Chapter 3, but some of the code included in the `chap2/` skeleton files deals with the interface between JLex and CUP. If you are curious about this, you can look at the CUP Manual on the course Moodle page.
- This assignment is worth 10 points.

Listing 1: MiniJava.lex

```
// Skeleton MiniJava Lexical Analyzer Specification

package parse;

%%

%implements java_cup.runtime.Scanner
%function next_token
%type java_cup.runtime.Symbol
%char

%state COMMENT

%{
private errormsg.ErrorMsg errorMsg;

private java_cup.runtime.Symbol tok(int kind, Object value) {
  return new java_cup.runtime.Symbol(kind, yychar, yychar+yylength(), value);
}

Ylex(java.io.InputStream s, errormsg.ErrorMsg e) {
  this(s);
  errorMsg=e;
}

%}

%eofval{
{
  return tok(sym.EOF, null);
}
%eofval}

%%

<YYINITIAL> "("                     {return tok(sym.LPAREN, null);}
<YYINITIAL> ")"                     {return tok(sym.RPAREN, null);}
<YYINITIAL> "{"                     {return tok(sym.LBRACE, null);}
<YYINITIAL> "}"                     {return tok(sym.RBRACE, null);}
<YYINITIAL> "["                     {return tok(sym.LBRACK, null);}
<YYINITIAL> "]"                     {return tok(sym.RBRACK, null);}

<YYINITIAL> ","                     {return tok(sym.COMMA, null);}
<YYINITIAL> "<"                     {return tok(sym.LT, null);}
<YYINITIAL> "."                     {return tok(sym.DOT, null);}
<YYINITIAL> ";"                     {return tok(sym.SEMICOLON, null);}
<YYINITIAL> "="                     {return tok(sym.ASSIGN, null);}
<YYINITIAL> "+"                     {return tok(sym.PLUS, null);}
<YYINITIAL> "-"                     {return tok(sym.MINUS, null);}
<YYINITIAL> "*"                     {return tok(sym.TIMES, null);}
<YYINITIAL> "&&"                    {return tok(sym.AND, null);}
<YYINITIAL> "!"                     {return tok(sym.EXCLAMATION, null);}

<YYINITIAL> "false"                 {return tok(sym.FALSE, null);}
<YYINITIAL> "true"                  {return tok(sym.TRUE, null);}
<YYINITIAL> "System.out.println"    {return tok(sym.PRINTLN, null);}
<YYINITIAL> "new"                   {return tok(sym.NEW, null);}
<YYINITIAL> "this"                  {return tok(sym.THIS, null);}
<YYINITIAL> "length"                {return tok(sym.LENGTH, null);}
<YYINITIAL> "while"                 {return tok(sym.WHILE, null);}
<YYINITIAL> "else"                  {return tok(sym.ELSE, null);}
<YYINITIAL> "if"                    {return tok(sym.IF, null);}
<YYINITIAL> "boolean"               {return tok(sym.BOOLEAN, null);}
<YYINITIAL> "int"                   {return tok(sym.INT, null);}
<YYINITIAL> "return"                {return tok(sym.RETURN, null);}
<YYINITIAL> "String"                {return tok(sym.STRING, null);}
<YYINITIAL> "main"                  {return tok(sym.MAIN, null);}
<YYINITIAL> "void"                  {return tok(sym.MAIN, null);}
<YYINITIAL> "static"                {return tok(sym.STATIC, null);}
<YYINITIAL> "public"                {return tok(sym.PUBLIC, null);}
<YYINITIAL> "class"                         {return tok(sym.CLASS, null);}
```

```
<YYINITIAL> "/*"                    {yybegin(COMMENT);}

<COMMENT>   ([^*]|\*+[^*/])*\**\*/  {yybegin(YYINITIAL);}

<YYINITIAL> "//"[^\n]*\n            { }

<YYINITIAL> [\ \t\n]+                   { }

<YYINITIAL> [a-zA-Z][a-zA-Z0-9_]*   {return tok(sym.ID, yytext());}

<YYINITIAL> [0-9]+                  {return tok(sym.INTEGER_LITERAL, yytext());}

<COMMENT>   ([^*]|\*+[^*/])*\**[^*/] {errorMsg.error(yychar,
                                   "unclosed␣comment...");}
<YYINITIAL> .                           {errorMsg.error(yychar,
                                                "unmatched␣input:␣" + yytext());}
```

Listing 2: test.java

```
/* Test the MiniJava lexical analyzer. */
int foo;
System.out.println();
if (!cat && new < 17) {
  // a comment
  $ - this
}
/* Finally we have an unclosed comment...
x = 5
```

Listing 3: test.out

```
INT (42,45) null
ID (46,49) foo
SEMICOLON (49,50) null
PRINTLN (51,69) null
LPAREN (69,70) null
RPAREN (70,71) null
SEMICOLON (71,72) null
IF (73,75) null
LPAREN (76,77) null
EXCLAMATION (77,78) null
ID (78,81) cat
AND (82,84) null
NEW (85,88) null
LT (89,90) null
INTEGER_LITERAL (91,93) 17
RPAREN (93,94) null
LBRACE (95,96) null
test.java:6.3: unmatched input: $
  $ - this
  ^
MINUS (116,117) null
THIS (118,122) null
RBRACE (123,124) null
test.java:8.3: unclosed comment...
/* Finally we have an unclosed comment...
  ^
EOF (174,174) null
```

Listing 4: test2.java

```
/** Test the MiniJava lexical analyzer. **/
int foo;
System.out.println();
if (!cat && new < 17) {
  // a comment
  $ - this
}
/* Finally we have an unclosed comment...
x = 5
```

Listing 5: test2.out

```
INT (44,47) null
ID (48,51) foo
SEMICOLON (51,52) null
PRINTLN (53,71) null
LPAREN (71,72) null
RPAREN (72,73) null
SEMICOLON (73,74) null
IF (75,77) null
LPAREN (78,79) null
EXCLAMATION (79,80) null
ID (80,83) cat
AND (84,86) null
NEW (87,90) null
LT (91,92) null
INTEGER_LITERAL (93,95) 17
RPAREN (95,96) null
LBRACE (97,98) null
test2.java:6.3: unmatched input: $
  $ - this
  ^
MINUS (118,119) null
THIS (120,124) null
RBRACE (125,126) null
test2.java:8.3: unclosed comment...
/* Finally we have an unclosed comment...
  ^
EOF (176,176) null
```