University of Puerto Rico
Mayagüez Campus
College of Engineering
Department of Electrical and Computer Engineering

ICOM4029 – Compilers
Professor: Bienvenido Vélez
Technical Assistant: René D. Badía

Laboratory 2 – Lexical Analysis

The function of a lexical analyzer (lexer) is to take an input stream of characters and break it into tokens. *JLex* is a Java-based lexical analyzer generator based on *Lex* (C-based). Its output is a Java source program that contains the necessary functions to perform a particular lexical analysis.

I. Preparation

Include the Java binary directory in the path:

```
      export PATH=$PATH:/opt/java/bin
```

You may want to add the "`/opt/java/bin`" directory to the path declaration in `.bash_profile` on your home directory as well so it is included every time you log in. Make sure the "`export`" line looks like this:

```
      export PATH=$PATH:~icom4029/cool/bin:~:/opt/java/bin
```

Download or browse to the JLex manual at:
`http://www.cs.princeton.edu/~appel/modern/java/JLex/current/manual.html`

II. JLex File Syntax

```
<user code>
%%
<JLex directives & declarations>
%%
<regular expression rules>
```

The code in <user code> is copied at the top of the lexical analyzer source file (useful, for example, for Java "`import`" statements). JLex directives include special commands interpreted by JLex to state some configuration or behavior, code that should be written to the generated Java source, and declarations of tokens. The regular expression rules state what action the analyzer should perform when it finds a particular token.

II. Example 1

Create a file named "`ex1.lex`" with the following code:

```
%%
%{
  int words = 0;
  int ints = 0;
  int nonInts = 0;
%}
%eof{
  System.out.println("Words found: " + words);
  System.out.println("Integers found: " + ints);
  System.out.println("Non-integers found: " + nonInts);
%eof}
```

```
%integer

BLANKCHAR = (" "|\t|\n|\r)
WHITESPACE = {BLANKCHAR}+
DIGIT = [0-9]
SIGN = ("+"|"-")
UNSIGNEDINT = {DIGIT}+
INT = ({SIGN}?{UNSIGNEDINT})
NUMBER = ({INT}|{INT}"."{UNSIGNEDINT})
LOWERCHAR = [a-z]
UPPERCHAR = [A-Z]
CHAR = ({LOWERCHAR}|{UPPERCHAR})
WORD = {CHAR}+
%%
{WHITESPACE} {;}
{WORD} { System.out.println("Found the word: " + yytext());
  words++; }
{INT} { System.out.println("Found the integer: " + yytext());
  ints++; }
{NUMBER} { System.out.println("Found the non-integer: " + yytext());
  nonInts++; }
. {;}
```

Now we need to create a file for the main Java program that will use the Yylex class created by JLex. Create a new file named “`Lexer.java`” with the following code:

```
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Lexer{
    public static void main(String[] args){
      for (int i = 0; i < args.length; i++) {
          FileReader file = null;
          try {
            file = new FileReader(args[i]);
            Yylex lexer = new Yylex(file);
            int retval = lexer.yylex();
            while (retval != -1) {
                retval = lexer.yylex();
            }
          } catch (FileNotFoundException ex) {
            System.out.println("Could not open input file " + args[i]);
          } catch (IOException ex) {
            System.out.println("Unexpected exception in lexer");
          }
      }
    }
}
```

Write a file named “`input1.txt`” for use as input:

```
12 hello
294.34 hOuSe -54
This 43.01 DOG +1267
caT
```

In order to test the lexical analyzer we first need to use JLex to create the Java source file for the analyzer:

```
	jlex ex1.lex
```

This will create a file named "`ex1.lex.java`" which we will now need to compile using the Java compiler:

```
	javac ex1.lex.java Lexer.java
```

This will create the "Lexer.class" class file which can now be run using:

```
	java Lexer <input-file>
```

where `<input-file>` is the name of the text file which will be used as input for the lexical analyzer. Try:

```
	java Lexer input1.txt
```

The output should state that it found 5 words, 3 integer numbers, and 2 non-integers.

III. Example 2

Create a file named "`ex2.lex`" with the following code:

```
%%
%{
  int dates = 0;
  int times = 0;
  int stdNums = 0;
  int telephones = 0;
  int icomClasses = 0;
%}
%eof{
  System.out.println("Total times found: " + times);
  System.out.println("Total dates found: " + dates);
  System.out.println("Total student numbers found: " + stdNums);
  System.out.println("Total telephones found: " + telephones);
  System.out.println("Total ICOM classes found: " + icomClasses);
%eof}
%integer

BLANKCHAR = (" "|\t|\n|\r)
WHITESPACE = {BLANKCHAR}+
DIGIT = [0-9]
TWODIGIT = {DIGIT}{DIGIT}
TWOPLACE = ({TWODIGIT}|{DIGIT})
THREEDIGIT = {DIGIT}{TWODIGIT}
FOURDIGIT = {TWODIGIT}{TWODIGIT}
TIME = ({TWOPLACE}":"{TWODIGIT})
DATE = ({TWOPLACE}"/"{TWOPLACE}"/"{FOURDIGIT})
STDNUM = ({THREEDIGIT}"-"{TWODIGIT}"-"{FOURDIGIT})
TELEPHONE = ({THREEDIGIT}"-"{THREEDIGIT}"-"{FOURDIGIT})
ICOMCLASS = ("ICOM"{FOURDIGIT})
%%
{WHITESPACE} {;}
{TIME} { System.out.println("Found the time: " + yytext());
  times++; }
{DATE} { System.out.println("Found the date: " + yytext());
  dates++; }
{STDNUM} { System.out.println("Found the student number: " + yytext());
  stdNums++; }
{TELEPHONE} { System.out.println("Found the telephone: " + yytext());
  telephones++; }
{ICOMCLASS} { System.out.println("Found the ICOM class: " + yytext());
```

```
  icomClasses++; }
. {;}
```

Write the file "`input2.txt`":

```
ICOM4029 10:35 77-78-9999
802-99-9999 INEL3125 9/12/2003 9:23 ICOM1c34
12/25/2004 787-787-7878 802-12-1234
1:35 ICOM5016 123-123-12345 777-777-7777
```

Generate the lexer with: `jlex ex2.lex`
Compile it with: `javac ex2.lex.java Lexer.java`
Run it with: `java Lexer input2.txt`
The output should state that it found 3 times, 2 dates, 2 student numbers, 3 telephones, and 2 ICOM classes.

IV. Regular Expressions and Finite Automatas

Regular expression: `(01|1)*`

- Converting from Reg. Exp. To NFA
- Converting from Reg. Exp. To DFA

V. Practice Exercises

1. Write a lexer that takes an algebraic expression as input and returns the corresponding token-lexeme pairs.
   Example:
   Input Text: `x = 5 * -3`
   Output:
   ```
   VARIABLE, x
   EQUALITY, =
   INTEGER, 5
   MULTIPLY, *
   INTEGER, -3
   ```

2. Encrypt a message by using the following algorithm:

   Replace each character with the letter that is one position after it in the alphabet followed by the one that is one position before it. ('*z*' being one position before '*a*', and '*a*' one position after '*z*').
   Example:
   Input Text: `This is a secret message`
   Output: `USigjhtr jhtr bz trfddbsqfdus nlfdtrtrbzhffd`

3. Write the lexer to decrypt a message encrypted by the program above (2).

4. Write the NFA for the language defined by the regular expression:
   `(10|0)*1`