# Applications of Regular Expressions

BY —

AMIT DHOMNE

# Index

# Regular Expression



- Regular expression is defined as a sequence of characters .
- A regular expression, often called a **pattern**, it is an expression used to specify a set of strings required for a particular purpose
- Regular Expression operators

| X Y concatination | X followed by Y |
|---|---|
| X \| Y alteration | X or Y |
| X* Kleen closure | Zero or more occurrences of X |
| X + | One or more occurrences of X |
| (X) | used for grouping X |

# Examples

- L(**001**) = {001}
- L(**0+10***) = { 0, 1, 10, 100, 1000, 10000, … }
- L(**0*10***) = {1, 01, 10, 010, 0010, …}  i.e. {w | w has exactly a single 1}
- L(∑∑)* = {w | w is a string of even length}
- L((**0(0+1)**)*) = { ε, 00, 01, 0000, 0001, 0100, 0101, …}
- L((**0+ε)(1+ ε**)) = {ε, 0, 1, 01}
- L(1Ø) = Ø  ; concatenating the empty set to any set yields the empty set.
- Rε = R
- R+Ø = R

# Applications of Regular Expression in Unix

- ✓ Basically regular expressions are used in search tools
- ✓ Text file search in Unix (tool: egrep)
- ✓ This command searches for a text pattern in the file and list the file names containing that pattern .

# Example:

- Standard egrep command looks like:

  egrep<flags>’<regular expression>’<file name>

  Some common flags are:
  - ✓ -c for counting the number of successful matches and not printing the actual matches
  - ✓ -i to make the search case insensitive
  - ✓ -n to print the line number before each match printout
  - ✓ -v to take the complement of the regular expression (i.e. return the lines which don't match)
  - ✓ -l to print the filenames of files with lines which match the expression.

# Example:

- ✓ File create

Sam
Dexter
John
Raman

By implementing egrep command :

% egrep ‘n’ create

John

Raman

%

- ✓ File create

Dexter
John
Alice
Raman

By implementing egrep command: % egrep ‘o.*h’ create

john

%

# Few more related examples

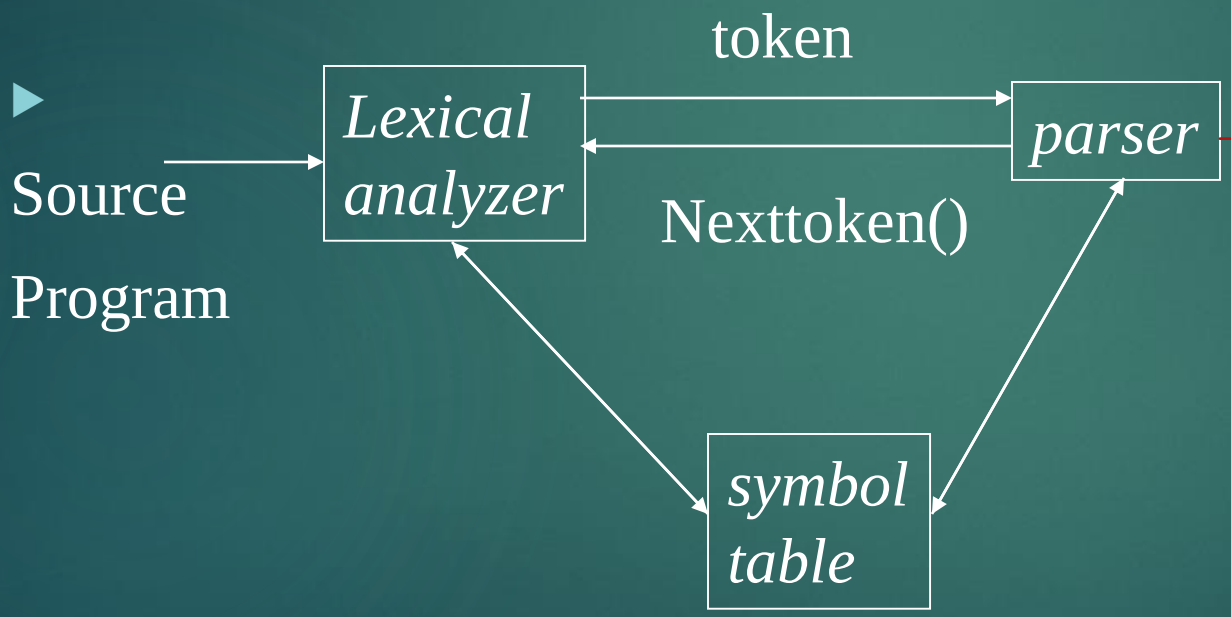
✓ egrep -c '^1|01$' lots_o_bits
count the number of lines in lots_o_bits which either start with 1 or end with 01

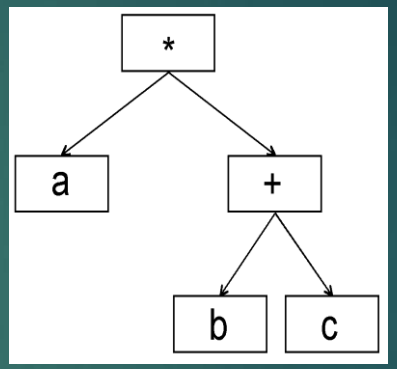
✓ egrep -c '10*10*10*10*10*10*10*10*10*10*1' lots_o_bits
count the number of lines with at least eleven 1's

✓ egrep -i '\<the\>' myletter.txt
list all the lines in myletter.txt containing the word  insensitive of case.

# Regular Expressions in Lexical analysis

- Interaction of Lexical analyzer with parser




Symbol Table

| | Lexeme | Token Code |
|---|---|---|
| 1 | a | ID |
| 2 | b | ID |
| 3 | c | ID |
| | ..... | ..... |

# Lexical Analysis

- ✓ The Lexical Analyzer (lexer) reads source code and generates a stream of tokens
- ✓ What is a token?
  - Identifier
  - Keyword
  - Number
  - Operator
  - Punctuation
- ✓ Tokens can be described using regular expressions!

**pattern**: The rule describing how a token can be formed.

Ex: identifier: ([a-z]|[A-Z]) ([a-z]|[A-Z]|[0-9])*

- How to specify tokens:
  - all the basic elements in a language must be tokens so that they can be recognized.

```
main() {
    int i, j;
    for (I=0; I<50; I++) {
        printf("I = %d", I);
    }
}
```

  - Token types: constant, identifier, reserved word, operator and misc. symbol.

# Regular expressions in Java:



- Regular expressions are a language of string patterns built in to most modern programming languages, including Java 1.4 onwards; they can be used for: searching, extracting, and modifying text.
- The Java package java.util.regex contains classes for working with regexps in Java
- Character Classes
  - Character classes are used to define the content of the pattern.
- Example, what should the pattern look for?
  - . Dot, any character (may or may not match line terminators, read on)
  - \d A digit: [0-9]
  - \D A non-digit: [^0-9]
  - \s A whitespace character: [ \t\n\x0B\f\r]
  - \S A non-whitespace character: [^\s]
  - \w A word character: [a-zA-Z_0-9]
  - \W A non-word character: [^\w]
- However; notice that in Java, you will need to “double escape” these backslashes.
- String pattern = "\\d \\D \\W \\w \\S \\s";

# Quantifiers



- Quantifiers can be used to specify the number or
- length that part of a pattern should match or repeat.
- A quantifier will bind to the expression group to its immediate left.
- * Match 0 or more times
- + Match 1 or more times
- ? Match 1 or 0 times
- {n} Match exactly n times
- {n,} Match at least n times

# Matching/Validating



- ✓ Regular expressions make it possible to find all instances of text that match a certain pattern, and return a Boolean value if the pattern is found/not found.

**Sample code**

```
public class ValidateDemo {
        public static void main(String[] args) {
                List<String> input = new ArrayList<String>();
                input.add("123-45-6789");
                input.add("9876-5-4321");
                input.add("987-65-4321 (attack)");
                input.add("987-65-4321 ");
                input.add("192-83-7465");
                                        for (String ssn : input) {
                        if (ssn.matches("^(\\d{3}-?\\d{2}-?\\d{4})$")) {
                                System.out.println("Found good SSN: " +
ssn);
                        }
```

OUTPUT:

Found good SSN: 123-45-6789</br>

Found good SSN: 192-83-7465

# Extracting/Capturing



- ✓ Specific values can be selected out of a large complex body of text.
- ✓ These values can be used in the application.
- ✓ Sample Code

```
public class ExtractDemo {
public static void main(String[] args) {
String input = "I have a cat, but I like my dog better.";
Pattern p = Pattern.compile("(mouse|cat|dog|wolf|bear|human)");
Matcher m = p.matcher(input);
List<String> animals = new ArrayList<String>();
while (m.find()) {
System.out.println("Found a " + m.group() + ".");
animals.add(m.group());
```

This produces the following output:

Found a cat
Found a dog

# References


- http://www.cs.columbia.edu/~tal/3261/fall07/handout/egrep_mini-tutorial.htm
- http://en.wikipedia.org/wiki/Regular_expression
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.4212&rep=rep1&type=pdf
- http://www.ocpsoft.org/opensource/guide-to-regular-expressions-in-java-part-1/