

Regular expressions in JFlex

Regular expressions describe sequences of ASCII characters using a set of operators:

"\[]^-?.*+|()\$/{}%<>
Letters and numbers in the input string are described by the characters themselves:

"the regular expression val1 matches the input sequence 'v' 'a' 'l' '1'

Non alphabetical characters must be written in quotation marks, to avoid ambiguities with operators:

"the regular expression xyz"++" matches the input sequence 'x' 'y' 'z' '+' '+'

5

7

Regular expressions in JFlex
....continues...

To include the character – in a character class, it must be either the first or the last character within the brackets:

the expression [-+0-9] matches a digit or a +/- sign in the input string.

The character ^ at the beginning of a character class identifies a "negated character class", i.e. a list of characters to be excluded

the expression [^0-9] matches any character except digits.

The symbol . (dot) identifies any character except newline.

4

Regular expressions in JFlex

...continues...

- The newline character is described by the following regular expression
 - $\n \mid \r \mid \r \n$ (\r line feed \n carrige return)
 - JFlex is written in Java, as a consequence generated scanners must be portable on Windows, Linux and Mac OS operating systems
 - Note:

 - -> matches one or more newlines: \n\n\r\r
- The symbol \t identifies the tabulation character.
- The operator ? Indicates that the preceding expression is optional
 - the expression ab?c matches both ac and abc.

Laboratory N° 1



8

Regular expressions in JFlex

...continues...

- The operator * indicates that the preceding expression can be repeated 0 or more times:
 - the expression ab*c matches all the sequences starting with a, terminating with c and with any number of b's in between
- The operator + indicates that the preceding expression can be repeated 1 or more times:
 - the expression ab+c matches all the sequences starting with a, terminating with c and with at least 1 b in between.
 - abc. abbc. abbbc : OK
 - ac · NOIII



9

11

Regular expressions in JFlex

...continues...

- The operator {n} represents n repetitions of the precedent regular expression:
 - ab{3}c matches the sequence abbbc
- The operator {n,m} represents a repetition of the precedent regular expression between a minimum of n and a maximum of m
 - ab{2,4}c matches the sequences abbc, abbbc and abbbbc
- The operator | represents two alternative expressions:
 - ab|cd matches both the sequences ab and cd.
- Parentheses are used to express or modify operators priority:
 - (ab|cd+)?ef matches sequences such as ef, abef, cdddef



Laboratory Nº 1

Regular expressions in JFlex

...continues...

- Unsigned integer
 - **[**0-9]+
- Unsigned integer without leading zeros
 - **[1-9][0-9]**
- Signed integer
 - **■** ("+"|"-")? [0-9]+
- Floating point number
 - $\blacksquare \ ("+"|"-")? \ (\ [1-9][0-9]^* \ "." \ [0-9]^* \) \ \ | \ \ ("." \ [0-9]+ \) \ \ | \ \ (\ 0 \ "." \ [0-9]^* \)$

Single quotation marks allow to distinguish an input character $\,("+")$ from an operator (+).



10

Structure of a JFlex source file

- A JFlex source file has three distinct sections separated by '%%'.
 - The first section (code section) contains the user code and can be empty.
 - The second section (declarations section) contains option and declarations
 - The third section (rules section) contains the lexical rules in the form of regular_expression action pairs.

Code section

Declarations section

%%

12

Rules section



Code Section

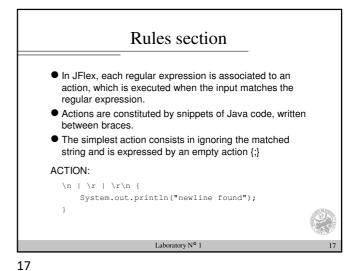
- All the code lines present in this section are copied without any modification in the generated scanner.
- Usually, import statement for Java libraries that will be used in the next sections are inserted here.
- Examples:

import java.io.*; (if one wishes to use the Java I/O library)

import java_cup.runtime.*; (for compatibility with the Cup parser generator)



Declarations section To simplify the use of complex or repetitive regular expressions, it is possible to define identifiers for sub-expressions. Example: definition of a signed integer: integer = [+-]?[1-9][0-9]* The sub-expression can then be used in the rules section or directly in the declaration section, writing its name between braces: {integer} { System.out.print("integer found\n"); } Java code can be included in the declarations section by writing it between '%{' and '%}'.



Laboratory Nº 1

Scanner methods and fields accessible in actions

- Returns the matched string (that is saved in a internal buffer):

 string untert()
 - string yytext()

15

- The number of matched character is returned by the method:
 - $\quad \blacksquare \text{ int } \textbf{yylength}()$
- Returns the character at position pos.

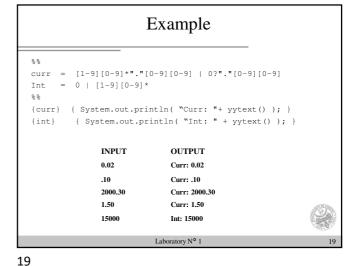
• See also %init{ ... %init} and %eof{ ...%eof}

- char yycharat(int pos)
- Contains the current line and column of input file, respectively.
 Those variables have a meaningful value only if <u>%line</u> and <u>%column</u> directives are declared.
 - int yyline
 - int yycolumn
- contains the current character count in the input (starting with 0, only active with the %char directive)
 - int yychar

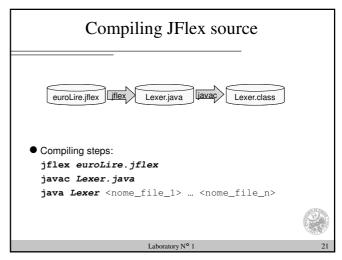
18

Laboratory Nº 1

...........



Compiling JFlex source FILE: euroLire.jflex : %class Lexe: %standalone curr = [1-9][0-9]*"."[0-9][0-9] | 0?"."[0-9][0-9]Int = 0 | [1-9][0-9]* 용용 {curr} { System.out.println("Curr: "+ yytext()); } { System.out.println("Int: " + yytext()); } • %standalone: generates the main method ■ The main method accepts as input the list of file to be scanned. ■ NB; with %standalone option, the default Jflex behaviour is to print unmatched characters to stdout. Use . (dot) regular expression to manage • %class Lexer: the generated class is named Lexer.java



21

Ambiguous Source Rules

- JFlex can handle ambiguous specifications
- There are two main sources of ambiguity:
 - the initial part of character sequences matched by one regular expression is also matched by another regular expression
 - the same character sequence is matched by two distinct regular
- The first case is handled by always selecting the regular expression that gives the longest match.
- Among rules which matched the same number of characters, the rule specified first in the source file is preferred.



Laboratory Nº 1

23

Ambiguous Source Rules

- Given the rules for handling ambiguous specifications, when analyzing a programming language it is necessary to define first the rules for keywords, and then for identifiers.
- The longest match rule can result in unwanted behaviour:

\".*\" { System.out.println("QUOTED_STRING");}

tries to match the second single quotation mark as far as possible (since longest matches are preferred); hence, given the following input string

"first" quoted string here, "second" here

it will match 36 characters instead of 7.

A better regular expression is the following:

```
{ System.out.println( "QUOTED_STRING" );
{ System.out.println( "QUOTED_STRING" ); }
```



Laboratory Nº 1

24

22

25

Beginning and end of line

- The character '^' at the beginning of a regular expression indicates that the sequence must be found at the beginning of the line.
 - This means that either the character sequence is at the beginning of the input stream, or that the last character previously read was a newline.
- The character '\$' at the end of a regular expression indicates that the sequence must be followed by a newline character.
- By default, the newline is not matched by the regular expression, and thus must be matched by another rule
 - The characters 'e' 'n' 'd' at the end of the line ■ end\$
 - \r | \n | \r\n Matches the newline



27

26

It could be useful to limit the validity of a regular expression to a determined context.

Context

Example

{ System.out.println("FOR_CMD"); }

Given the input string "format", the scanner will print

■ Preferring the second rule to the first, because it gives a longer

■ Preferring the second rule to the third, because it comes before

{ System.out.println("FORMAT_CMD"); }

{ System.out.println("GENERIC_ID"); }

Given the source file

FORMAT_CMD,

in the source file

match

for

format

- There are different mechanisms to specify sensitivity to the left context (i.e., the string that precedes the sequence being matched) and to the right context (i.e., the string that follows the sequence being matched).
- Special techniques are used to handle the beginning and the end of a line.



Sensitivity to the right context

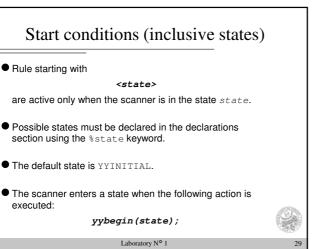
- The binary operator '/' separates a regular expression from its right context.
- Therefore, the expression

ab/cd

matches the string "ab", but if and only if is followed by the string "cd".

- The characters forming the right context are read from the input file, but are not part of the matched string. A suitable buffer is defined by JFlex to hold such characters.
- NB: The expression ab\$ is equivalent to ab / (\n | \r | \r\n).





Start conditions (inclusive states)
....continues...

• When a state is activated, the state rules are added (inclusive or) to the other scanner base rules.

• A state is active until another state is activated. To return to the initial condition, one must activate the initial state by means of the statement

Yybegin (YYINITIAL);

• A rule can be preceded by one or more state names, separated by a comma, to indicate that it is active in each of the states.

29

31

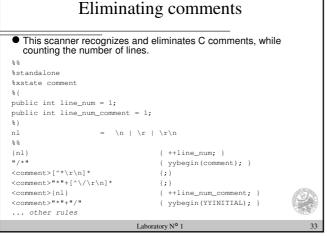
30

32

34

Example The following program handles pseudo-comments of the form // \$var+ %% %state comment %% <comment>\\$[a-zA-Z]+[-+] {process(yytext());} "//" {yybegin(comment);} \n|\r|\r\n {yybegin(YYINITIAL);} " " {;} /* ignore blanks*/ \t {;} /* and tabs */ ... /* other rules */

Combining more than one scanner (exclusive states) A set of rules can be grouped within an exclusive state as well. When the scanner enters an exclusive state: default rules are disabled, only the rules explicitly defined for the state are active. In this way, "mini-scanner" that deal with special sections of the input stream, such as comments or strings, can be defined. The %xstate keyword defines an exclusive state.



End of file rule

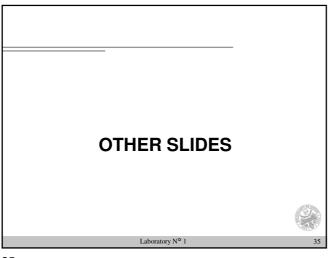
The special rule <<EOF>> introduces the action to be performed when the end of file is reached.

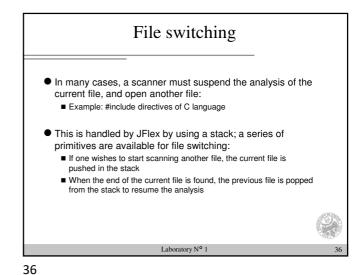
<<EOF>>
{System.out.println(line_num+" "+line_num_comment); return YYEOF;}

This rule can be useful, coupled with start conditions, to detect unbalanced parentheses (or braces, brackets, quotation marks, ...):

'" { yybegin(quote); }
...

<quote><<EOF>> { System.out.println("EOF in string"); }





35

