Lab1

Problem

Check if the string belongs to the given regular expression or not.

Example

If Regualr expressions :

- (0|1)*|0*1*
- (1|0)*(101)+(1|0)*(001)*

then tested Strings:

- 010101010101010101010
- 00111011011010000101010001

Idea used

- For this problem we used java builtin classes **Pattern** and **Matcher**.
- Pattern class is used to compile the regular expression.

  // Pattern pattern = Pattern.*compile*(regular_expression);

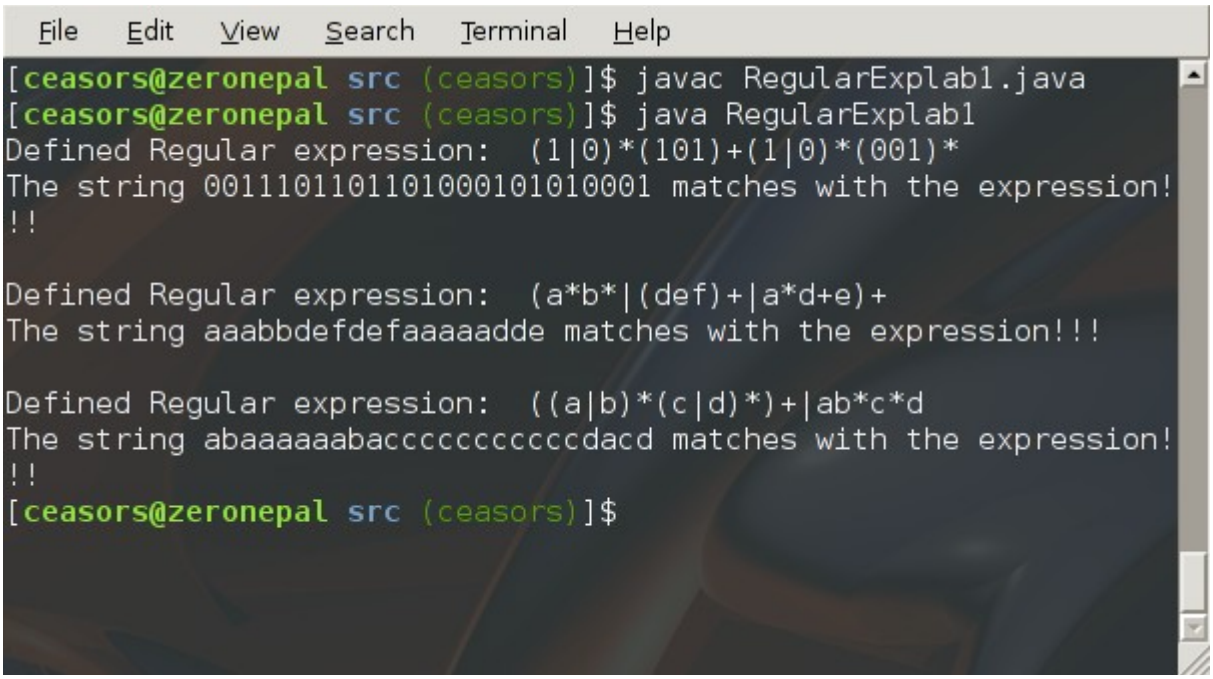
- Matcher class is used to check if string to be checked matches the pattern.

  // Matcher matcher = pattern.matcher(test_String);

Implementation

- Define a test string to be checked.
- Take regular expressions and compile them as a pattern
- Define a matcher of that pattern.
- See if the pattern matches or not using if-else condition. And output accordingly. i.e,

```
if (matcher.matches()) {
        System.out.println("The string " + testStr
                                        + " matches with the expression!!!\n");
    } else {
        System.out.println("The string " + testStr
        + "  does not matches with the regular expression defined.\n");
      }
```



Sample Output Screen Shot for problem: Lab1

Lab2

Problem

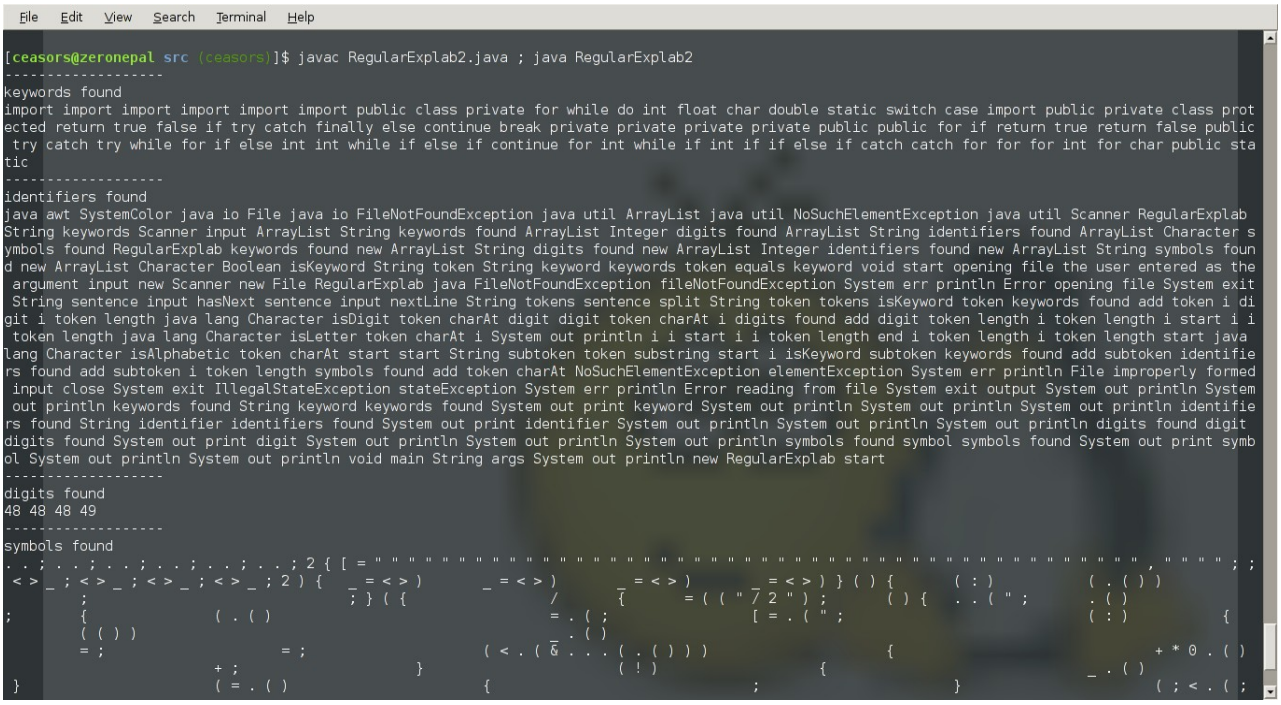
Design a scanner that could generate tokens out of a java file i.e, a source code.

Idea used

- The program is designed to analyse the tokens that are contined in a java code so the final output is the list of tokens found in a source file grouped as identifiers, symbols, keywords, digits.
- The program is fed the source code the program is made up of.
- The java's inbuilt class Scanner is also imported in this problem.

Implementation

- Read the inputs source file up to the end of file. Here, a source code of the porgram itself is a source to the program to find out tokens.
- Read a line at a time
- Split each lexeme seperated by the space.
- Check each lexeme if it could be a keyword, a identifier, a digit or a symbol.



Sample Output Screen Shot for problem: Lab2

Lab3

Problem

Read input strings from a file and check to see if they belong to the given regular expression or not.

Idea used

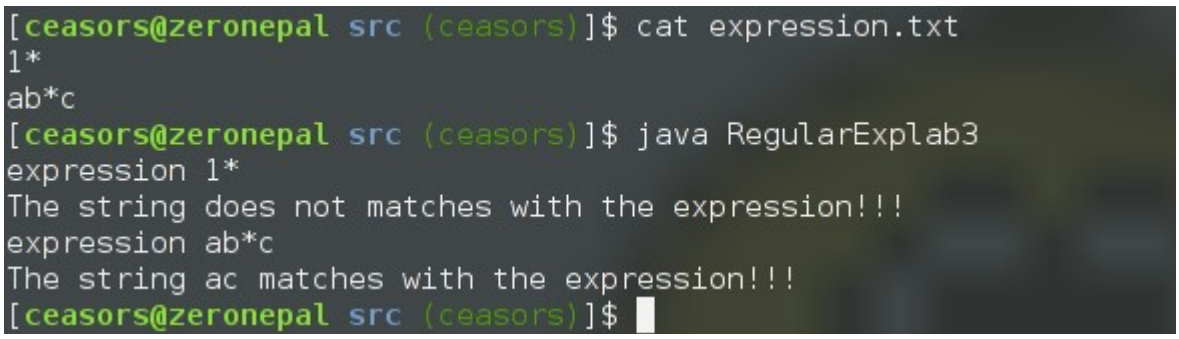
- Store input strings in a file in different line
- Check for matches for each string using ‘String.matches(Regex r)’ function.

Implementation

- Read the inputs strings from the file one line at a time.
- Run a loop for all these lines.
- Inside the loop check for matches.
- Example:

```
for(String s : strings){
        s.matches(reg);              //reg is the given regular expression
}
```

- If they match notify the user that they belong.



Sample Output Screen Shot for problem: Lab3

Lab 4

Problem

Remove the immediate left recursion of a grammer with general form.

Idea

- Determine the alpha, beta from the input string. [General form A->A[(alpha) | (beta)]
- Then change the form into:
  A->(beta)A'
  A'->(alpha)A' | *(Epsilon)*

Implementaion

- Get the input from user.
- Check to see if the input is left recursive or not.
  - This is done by comparing the initial value in the array and the one after the '>'symbol.

If recursive then continue with the following steps:

- Since the program is designed for general form so first of all we chunked the input production into a jagged array holding seperate possible production rules seperated by the or '|' symbol. For this we counted the number of '|' symbols present in the production and made a for loop run for number of '|' symbols present plus 1. By this jagged array we could easily handel the all input productions and check if immediate left recursion exists.
- Then using a for loop we iterate over those chunks to adjust alpha or beta case of input. For this we implemented two array dynamically created using malloc and our sample output looks like this: