SS ASSIGNMENT -05

ROLL NO: U21CS052
NAME : PANCHAL GUNGUN PARESH

IDENTIFY AND REMOVE LEFT RECURSION

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

class NonTerminal {

  private String name;
  private ArrayList<String> rules;

  public NonTerminal(String name) {
    this.name = name;
    rules = new ArrayList<>();
  }

  public void addRule(String rule) {
    rules.add(rule);
  }

  public void setRules(ArrayList<String> rules) {
    this.rules = rules;
```

```
  }

  public String getName() {
    return name;
  }

  public ArrayList<String> getRules() {
    return rules;
  }

  public void printRule() {
    System.out.print(name + " -> ");
    for (int i = 0; i < rules.size(); i++) {
      System.out.print(rules.get(i));
      if (i != rules.size() - 1)
System.out.print(" | ");
    }
    System.out.println();
  }
}

class Grammar {

  private ArrayList<NonTerminal> nonTerminals;

  public Grammar() {
```

```
    nonTerminals = new ArrayList<>();
  }

  public void addRule(String rule) {
    boolean nt = false;
    String parse = "";

    for (int i = 0; i < rule.length(); i++) {
      char c = rule.charAt(i);
      if (c == ' ') {
        if (!nt) {
          NonTerminal newNonTerminal = new
NonTerminal(parse);
          nonTerminals.add(newNonTerminal);
          nt = true;
          parse = "";
        } else if (parse.length() != 0) {
          nonTerminals.get(nonTerminals.size() -
1).addRule(parse);
          parse = "";
        }
      } else if (c != '|' && c != '-' && c !=
'>') {
        parse += c;
      }
    }
```

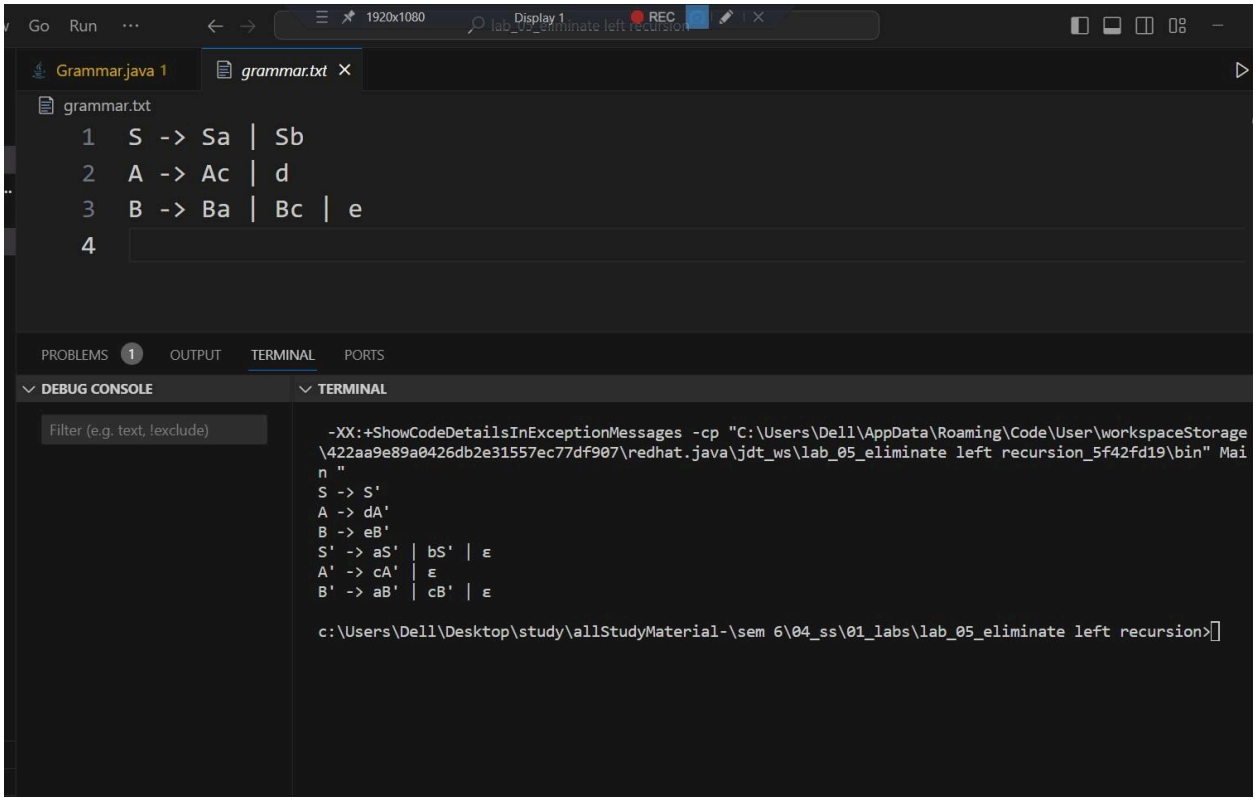
```
    if (parse.length() != 0) {
      nonTerminals.get(nonTerminals.size() -
1).addRule(parse);
    }
  }

  public void inputData(String filename) {
    // addRule("S -> Sa | Sb | c | d | A");
    // addRule(" A-> Af | d");
    try {
      Scanner sc = new Scanner(new
File(filename)); // Read from the specified file
      while (sc.hasNextLine()) {
        String rule = sc.nextLine();
        addRule(rule);
      }
      sc.close();
    } catch (FileNotFoundException e) {
      System.err.println("File not found: " +
e.getMessage());
    }
    /*
    } */

  }
```

```
  public void solveNonImmediateLR(NonTerminal A,
NonTerminal B) {
    String nameA = A.getName();
    String nameB = B.getName();

    ArrayList<String> rulesA = new ArrayList<>();
    ArrayList<String> rulesB = new ArrayList<>();
    ArrayList<String> newRulesA = new
ArrayList<>();
    rulesA = A.getRules();
    rulesB = B.getRules();

    for (String rule : rulesA) {
      if (rule.substring(0,
nameB.length()).equals(nameB)) {
        for (String rule1 : rulesB) {
          newRulesA.add(rule1 +
rule.substring(nameB.length()));
        }
      } else {
        newRulesA.add(rule);
      }
    }
    A.setRules(newRulesA);
  }
```

```
  public void solveImmediateLR(NonTerminal A) {
    String name = A.getName();
    String newName = name + "'";

    ArrayList<String> alphas = new ArrayList<>();
    ArrayList<String> betas = new ArrayList<>();
    ArrayList<String> rules = A.getRules();
    ArrayList<String> newRulesA = new
ArrayList<>();
    ArrayList<String> newRulesA1 = new
ArrayList<>();

    rules = A.getRules();

    // Checks if there is left recursion or not
    for (String rule : rules) {
      if (rule.substring(0,
name.length()).equals(name)) {

alphas.add(rule.substring(name.length()));
      } else {
        betas.add(rule);
      }
    }

    // If no left recursion, exit
```

```
    if (alphas.size() == 0) return;

    if (betas.size() == 0)
newRulesA.add(newName);

    for (String beta : betas) newRulesA.add(beta
+ newName);

    for (String alpha : alphas)
newRulesA1.add(alpha + newName);

    // Amends the original rule

    A.setRules(newRulesA);
    newRulesA1.add("\u03B5");

    // Adds new production rule
    NonTerminal newNonTerminal = new
NonTerminal(newName);
    newNonTerminal.setRules(newRulesA1);
    nonTerminals.add(newNonTerminal);
  }

  public void applyAlgorithm() {
    int size = nonTerminals.size();
    for (int i = 0; i < size; i++) {
```

```
      for (int j = 0; j < i; j++) {
        solveNonImmediateLR(nonTerminals.get(i),
nonTerminals.get(j));
      }
      solveImmediateLR(nonTerminals.get(i));
    }
  }

  void printRules() {
    for (NonTerminal nonTerminal : nonTerminals)
{
      nonTerminal.printRule();
    }
  }
}

class Main {

  public static void main(String[] args) {
    Grammar grammar = new Grammar();
    grammar.inputData("grammar.txt");
    grammar.applyAlgorithm();
    grammar.printRules();
  }
}
```


Go Run
1920x1080
Display 1
REC
Grammar.java 1
grammar.txt
grammar.txt
1 S -> Sa | Sb
2 A -> Ac | d
3 B -> Ba | Bc | e
4
PROBLEMS 1
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
Filter (e.g. text, !exclude)
TERMINAL
-XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Dell\AppData\Roaming\Code\User\workspaceStorage
\422aa9e89a0426db2e31557ec77df907\redhat.java\jdt_ws\lab_05_eliminate left recursion_5f42fd19\bin" Mai
n "
S -> S'
A -> dA'
B -> eB'
S' -> aS' | bS' | ε
A' -> cA' | ε
B' -> aB' | cB' | ε
c:\Users\Dell\Desktop\study\allStudyMaterial-\sem 6\04_ss\01_labs\lab_05_eliminate left recursion>