```
EXPERIMENT -01
import java.util.Scanner;
public class exp1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the String:");
        String str = sc.nextLine();
        dfa(str);
        sc.close();
    }
    static void dfa(String data) {
        int n = data.length();
        int q = 0;
        StringBuilder statePath = new StringBuilder();
      statePath.append(q);
        for (int i = 0; i < n; i++) {
            if (data.charAt(i) != 'a' && data.charAt(i) != 'b' && data.charAt(i)
!= 'c') {
                System.out.println("Given string is invalid.");
                return;
            }
            switch (q) {
                case 0:
                    q = (data.charAt(i) == 'a') ? 1 : 0;
                    break;
                case 1:
                    q = (data.charAt(i) == 'b') ? 2 : (data.charAt(i) == 'a') ?
1 : 0;
                    break;
                case 2:
                    q = (data.charAt(i) == 'c') ? 3 : (data.charAt(i) == 'a') ?
1 : 0;
                    break;
                case 3:
                    q = (data.charAt(i) == 'a') ? 1 : 0;
                    break;
            }
            statePath.append(" -> q").append(q);
        }
        System.out.println("Path of States: q" + statePath.toString());

        if (q == 3)
            System.out.println("Accepted");
        else
            System.out.println("Not Accepted");
    }
}
================================================================================
==================================================
EXPERIMENT-02
import java.util.regex.*;
import java.util.Scanner;

public class exp2 {
    private static final String K = "int|float|char|else|if|while|return|
system";
    private static final String I = "[a-zA-Z][a-zA-Z0-9]*";
    private static final String N = "\\d+";
    private static final String O = "[+\\-*/=<>!]+";
    private static final String S = "[();{}\\[\\]]";
    private static final String LIT = "\"[^\"]*\"";

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
```

```
        String op;

        do {
            System.out.print("Enter the input String: ");
            String in = sc.nextLine();
            System.out.println("Input String: " + in);
            int tokens = tokenize(in);
            System.out.println("Total tokens: " + tokens);
            System.out.print("Do you want to perform another operation (yes/no):
");
            op = sc.next();
            sc.nextLine();
        } while (op.equalsIgnoreCase("yes"));

        sc.close();
    }

    public static int tokenize(String in) {
        String p = String.format("(%s)|(%s)|(%s)|(%s)|(%s)|(%s)", K, I, N, O, S,
LIT);
        Pattern cp = Pattern.compile(p);
        Matcher m = cp.matcher(in);
        int tc = 0;

        while (m.find()) {
            if (m.group(1) != null)
                System.out.println("Keyword: " + m.group(1));
            else if (m.group(2) != null)
                System.out.println("Identifier: " + m.group(2));
            else if (m.group(3) != null)
                System.out.println("Number: " + m.group(3));
            else if (m.group(4) != null)
                System.out.println("Operator: " + m.group(4));
            else if (m.group(5) != null)
                System.out.println("Separator: " + m.group(5));
            else if (m.group(6) != null)
                System.out.println("String Literal: " + m.group(6));
            tc++;
        }
        return tc;
    }
}
================================================================================
============================
EXPERIMENT-03
import java.util.Scanner;

public class exp3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the production (e.g., A->Ad|Ae|aB|ac):");
        String production = sc.nextLine().trim();
        eliminateLeftRecursion(production);
        sc.close();
    }

    public static void eliminateLeftRecursion(String production) {
        String[] parts = production.split("->");
        char nonTerminal = parts[0].charAt(0);
        String[] choices = parts[1].split("\\|");

        StringBuilder alpha = new StringBuilder();
        StringBuilder beta = new StringBuilder();
```

```
        for (String choice : choices) {
            choice = choice.trim();
            if (choice.startsWith("" + nonTerminal)) {

                alpha.append(choice.substring(1).trim()).append(" | ");
            } else {

                beta.append(choice.trim()).append(" | ");
            }
        }

        if (alpha.length() > 0) {
            alpha.setLength(alpha.length() - 3);
        }
        if (beta.length() > 0) {
            beta.setLength(beta.length() - 3);
        }


        if (alpha.length() > 0) {
            System.out.println(nonTerminal + " -> " + beta + nonTerminal + "'");
            System.out.println(nonTerminal + "' -> " + alpha + nonTerminal+"'|
epsilon");
        } else {
            System.out.println(nonTerminal + " is not left recursive.");
        }
    }
}
================================================================================
========================================
EXPERIMENT-04
import java.util.*;

public class exp4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the production (e.g., A->aB|aC|bD):");
        String production = sc.nextLine().trim();
        leftFactor(production);
        sc.close();
    }

    public static void leftFactor(String production) {
        String[] parts = production.split("->");
        char nonTerminal = parts[0].charAt(0);
        String[] choices = parts[1].split("\\|");

        // Map to store common prefixes and their corresponding productions
        Map<String, List<String>> prefixMap = new HashMap<>();

        // Group productions by their common prefixes
        for (String choice : choices) {
            choice = choice.trim();
            String prefix = getCommonPrefix(choice, choices);
            if (!prefix.isEmpty()) {
                prefixMap.putIfAbsent(prefix, new ArrayList<>());

prefixMap.get(prefix).add(choice.substring(prefix.length()).trim());
            }
        }

        // Output the left-factored grammar
        if (!prefixMap.isEmpty()) {
```

```
            System.out.println(nonTerminal + " -> " + String.join(" | ",
prefixMap.keySet()) + nonTerminal + "'");
            for (Map.Entry<String, List<String>> entry : prefixMap.entrySet()) {
                String prefix = entry.getKey();
                List<String> suffixes = entry.getValue();
                StringBuilder suffixBuilder = new StringBuilder();
                for (String suffix : suffixes) {
                    suffixBuilder.append(suffix).append(" | ");
                }
                // Remove the last " | " if present
                if (suffixBuilder.length() > 0) {
                    suffixBuilder.setLength(suffixBuilder.length() - 3);
                }
                System.out.println(nonTerminal + "' -> " +
suffixBuilder.toString());
            }
        } else {
            System.out.println(nonTerminal + " has no common prefixes to
factor.");
        }
    }

    // Function to find the common prefix of a production with others
    private static String getCommonPrefix(String choice, String[] choices) {
        String prefix = "";
        for (String otherChoice : choices) {
            otherChoice = otherChoice.trim();
            if (otherChoice.equals(choice)) continue; // Skip itself
            int minLength = Math.min(choice.length(), otherChoice.length());
            StringBuilder common = new StringBuilder();
            for (int i = 0; i < minLength; i++) {
                if (choice.charAt(i) == otherChoice.charAt(i)) {
                    common.append(choice.charAt(i));
                } else {
                    break;
                }
            }
            if (common.length() > prefix.length()) {
                prefix = common.toString();
            }
        }
        return prefix;
    }
}
```

```
================================================================================
====================================
```

EXPERIMENT-05

```
import java.util.*;
import java.io.*;
class exp5
{
static char ntermnl[],termnl[];
static int ntlen,tlen;
static String grmr[][],fst[],flw[];
public static void main(String args[]) throws IOException
{
String nt,t;
int i,j,n;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the non-terminals");
nt=br.readLine();
ntlen=nt.length();
ntermnl=new char[ntlen];
ntermnl=nt.toCharArray();
```

```
System.out.println("Enter the terminals");
t=br.readLine();
tlen=t.length();
termnl=new char[tlen];
termnl=t.toCharArray();
System.out.println("Specify the grammar(Enter 9 for epsilon production)");
grmr=new String[ntlen][];
for(i=0;i<ntlen;i++)
{
System.out.println("Enter the number of productions for "+ntermnl[i]);
n=Integer.parseInt(br.readLine());
grmr[i]=new String[n];
System.out.println("Enter the productions");
for(j=0;j<n;j++)
grmr[i][j]=br.readLine();
}
fst=new String[ntlen];
for(i=0;i<ntlen;i++)
fst[i]=first(i);
System.out.println("First Set");
for(i=0;i<ntlen;i++)
System.out.println(removeDuplicates(fst[i]));
flw=new String[ntlen];
for(i=0;i<ntlen;i++)
flw[i]=follow(i);
System.out.println("Follow Set");
for(i=0;i<ntlen;i++)
System.out.println(removeDuplicates(flw[i]));
}
static String first(int i)
{
int j,k,l=0,found=0;
String temp="",str="";
for(j=0;j<grmr[i].length;j++)
{
for(k=0;k<grmr[i][j].length();k++,found=0)
{
for(l=0;l<ntlen;l++)
{
if(grmr[i][j].charAt(k)==ntermnl[l])
{
str=first(l);
if(!(str.length()==1 && str.charAt(0)=='9'))
temp=temp+str;
found=1;
break;
}
}
if(found==1)
{
if(str.contains("9"))
continue;
}
else
temp=temp+grmr[i][j].charAt(k);
break;
}
}
return temp;
}
static String follow(int i)
{
char pro[],chr[];
String temp=" ";
```

```
int j,k,l,m,n,found=0;
if(i==0)
temp="$";
for(j=0;j<ntlen;j++)
{
for(k=0;k<grmr[j].length;k++)
{
pro=new char[grmr[j][k].length()];
pro=grmr[j][k].toCharArray();
for(l=0;l<pro.length;l++)
{
if(pro[l]==ntermnl[i])
{
if(l==pro.length-1)
{
if(j<i)
temp=temp+flw[j];
}
else
{
for(m=0;m<ntlen;m++)
{
if(pro[l+1]==ntermnl[m])
{
chr=new char[fst[m].length()];
chr=fst[m].toCharArray();
for(n=0;n<chr.length;n++)
{
if(chr[n]=='9')
{
if(l+1==pro.length-1)
temp=temp+follow(j);
else
temp=temp+follow(m);
}
else
temp=temp+chr[n];
}
found=1;
}
}
if(found!=1)
temp=temp+pro[l+1];
}
}
}
}
}
return temp;
}
static String removeDuplicates(String str)
{
int i;
char ch;
boolean seen[] = new boolean[256];
StringBuilder sb = new StringBuilder(seen.length);
for(i=0;i<str.length();i++)
{
ch=str.charAt(i);
if (!seen[ch])
{
seen[ch] = true;
sb.append(ch);
}
```

```
}
return sb.toString();
}
}
================================================================================
==================================
EXPERIMENT-06
import java.util.*;

public class exp6 {
    static Map<Character, List<String>> grammar = new HashMap<>();
    static String input;
    static int i = 0;

    static boolean parse(char nonTerminal) {
        int backtrack = i;
        for (String prod : grammar.get(nonTerminal)) {
            i = backtrack;
            boolean success = true;
            for (char symbol : prod.toCharArray()) {
                if (symbol == '@') continue;
                else if (Character.isUpperCase(symbol)) success &=
parse(symbol);
                else if (i < input.length() && input.charAt(i) == symbol) i++;
                else { success = false; break; }
            }
            if (success) return true;
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of productions:");
        int n = sc.nextInt();
        sc.nextLine();
        System.out.println("Enter productions (Use '@' for epsilon, e.g., A->aA|
@):");
        for (int j = 0; j < n; j++) {
            String[] rule = sc.nextLine().split("->");
            grammar.put(rule[0].charAt(0), Arrays.asList(rule[1].split("\\|")));
        }
        System.out.println("Enter the string to check:");
        input = sc.next() + "$";
        System.out.println(parse('E') && i == input.length() - 1 ? "String is
accepted" : "String is rejected");
    }
}
================================================================================
========================================
EXPERIMENT-07
import java.util.Scanner;
class ProductionRule {
    String left, right;
    ProductionRule(String left, String right) {
        this.left = left;
        this.right = right;
    }
}
public class exp7 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of production rules: ");
        int ruleCount = scanner.nextInt();
```

```
        scanner.nextLine();
        ProductionRule[] rules = new ProductionRule[ruleCount];
        System.out.println("Enter the production rules (in the form 'left-
>right'): ");
        for (int i = 0; i < ruleCount; i++) {
            String[] temp = scanner.nextLine().split("->");
            rules[i] = new ProductionRule(temp[0], temp[1]);
        }
        System.out.print("Enter the input string: ");
        String input = scanner.nextLine();
        String stack = "";
        int i = 0;
        System.out.println("Stack\tInputBuffer\tAction");
        while (true) {
            if (i < input.length()) {
                char ch = input.charAt(i++);
                stack += ch;
                System.out.printf("%s\t%s\t\tShift %c\n", stack,
input.substring(i), ch);
            }
            boolean reduced = false;
            for (ProductionRule rule : rules) {
                int index = stack.indexOf(rule.right);
                if (index != -1) {
                    stack = stack.substring(0, index) + rule.left +
stack.substring(index + rule.right.length());
                    System.out.printf("%s\t%s\t\tReduce %s->%s\n", stack,
input.substring(i), rule.left, rule.right);
                    reduced = true;
                    break;
                }
            }
            if (stack.equals(rules[0].left) && i == input.length()) {
                System.out.println("\nAccepted");
                break;
            }
            if (i == input.length() && !reduced) {
                System.out.println("\nNot Accepted");
                break;
            }
        }
        scanner.close();
    }
}
```

```
================================================================================
==================================
```

```
EXPERIMENT-08
import java.util.Scanner;
public class exp8 {
public static void main(String[] args) {
char[] stack = new char[20];
char[] ip = new char[20];
char[][][] opt = new char[10][10][1];
char[] ter = new char[10];
int i, j, k, n, top = 0, col = 0, row = 0;
Scanner scanner = new Scanner(System.in);
for (i = 0; i < 10; i++) {
stack[i] = 0;
ip[i] = 0;
for (j = 0; j < 10; j++) {
opt[i][j][0] = 0;
}
}
System.out.print("Enter the no. of terminals:");
```

```
n = scanner.nextInt();
System.out.print("\nEnter the terminals:");
ter = scanner.next().toCharArray();
System.out.println("\nEnter the table values:");
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
System.out.printf("\nEnter the value for %c %c:", ter[i], ter[j]);
opt[i][j] = scanner.next().toCharArray();
}
}
System.out.println("\nOPERATOR PRECEDENCE TABLE:");
for (i = 0; i < n; i++) {
System.out.print("\t" + ter[i]);
}
System.out.println();
for (i = 0; i < n; i++) {
System.out.println();
System.out.print(ter[i]);
for (j = 0; j < n; j++) {
System.out.print("\t" + opt[i][j][0]);
}
}
stack[top] = '$';
System.out.print("\nEnter the input string:");
String input = scanner.next();
ip = input.toCharArray();
i = 0;
System.out.println("\nSTACK\t\t\tINPUT STRING\t\t\tACTION");
System.out.print("\n" + String.valueOf(stack) + "\t" + input + "\t\t");
while (i <= input.length()) {
for (k = 0; k < n; k++) {
if (stack[top] == ter[k])
col = k;
if (ip[i] == ter[k])
row = k;
}
if ((stack[top] == '$') && (ip[i] == '$')) {
System.out.println("String is accepted");
break;
} else if ((opt[col][row][0] == '<') || (opt[col][row][0] == '=')) {
stack[++top] = opt[col][row][0];
stack[++top] = ip[i];
System.out.println("Shift " + ip[i]);
i++;
} else {
if (opt[col][row][0] == '>') {
while (stack[top] != '<') {
--top;
}
top = top - 1;
System.out.println("Reduce");
} else {
System.out.println("\nString is not accepted");
break;
}
}
System.out.println();
for (k = 0; k <= top; k++) {
System.out.print(stack[k]);
}
System.out.print("\t\t\t");
for (k = i; k < input.length(); k++) {
System.out.print(ip[k]);
}
```

```
System.out.print("\t\t\t");
}
}
}
================================================================================
=====================================
```