Exp1:

import java.util.\*;

public class Main

{

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.println("1. Enter the string");

System.out.println ("2. Exit");

System.out.println("Enter a choice");

int n = sc.nextInt();

while (n != 2){

System.out.println("Enter the string :");

String s = sc.next();

if (s.endsWith("abc")){

System.out.println(s + " " + "is Accepted");

}else{

System.out.println(s + " " + "is Not Accepted");

}

System.out.println("1. Enter the string\n2. Exit");

System.out.println("Enter a choice");

n = sc.nextInt();

}

}

}

Output:

1. Enter the string

2. Exit

Enter a choice

1

Enter the string:

aabbabc

aabbabc is Accepted

Exxp2:  
import java.util.\*;

public class Exp2

{

public static void main(String[] args){

ArrayList<String> keywords = new ArrayList<>(Arrays.asList(

"if", "else", "while", "for", "int", "float", "double", "char", "String", "boolean" ));

ArrayList<String> operators = new ArrayList<>(Arrays.asList(

"+", "-", "\*", "/", "=", ">", "<", "!", "&", "|" ,"++","--" ));

ArrayList<String> delimiters = new ArrayList<>(Arrays.asList(

"(", ")", "{", "}", "[", "]", ",", ";"));

Scanner sc = new Scanner(System.in);

System.out.println("Enter program with single spaces");

String input = sc.nextLine();

String[] arr = input.split(" ");

int len = arr.length;

String[] ans = new String[len];

for (int i = 0; i < len; i++){

if (keywords.contains(arr[i])) {

ans[i] = "keyword";

}else if (operators.contains(arr[i])) {

ans[i] = "operator";

}else if (delimiters.contains(arr[i])) {

ans[i] = "delimiter";

}else if (isIdentifier(arr[i])){

ans[i] = "identifier";

}else if (isLiteral(arr[i]))

{

ans[i] = "literal";

}

else{

ans[i] = "unknown";

}

}

for (int i = 0; i < len; i++){

System.out.println(arr[i] + ": " + ans[i]);

}

}

private static boolean isIdentifier(String str){

if (Character.isDigit(str.charAt(0))){

return false;

}

for (char c : str.toCharArray()){

if (!Character.isLetterOrDigit(c) && c != '\_'){

return false;

}

}

return true;

}

private static boolean isLiteral(String str){

try {

Integer.parseInt(str);

return true;

}

catch (NumberFormatException e1){

try{

Double.parseDouble(str);

return true;

}

catch (NumberFormatException e2)

{

return false; } } } }

Sample output:

Enter program with single spaces

java 123 { main int

java: identifier

123: literal

{: delimiter

main: identifier

int: keyword

Exp3:

import java.util.Scanner;

public class LeftRecursionElimination{

public static void main(String[] args){

Scanner scanner = new Scanner(System.in);

System.out.print("Enter Number of Productions: ");

int num = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.println("Enter the grammar as A -> Aa / b:");

for (int i = 0; i < num; i++){

String production = scanner.nextLine().trim();

eliminateLeftRecursion(production);

}

scanner.close();

}

public static void eliminateLeftRecursion(String production){

String[] parts = production.split("->");

char nonTerminal = parts[0].charAt(0);

String[] choices = parts[1].split("/");

System.out.println("GRAMMAR: " + production); // Checking for left recursion

if (choices[0].startsWith("" + nonTerminal)){

String beta = choices[0].substring(1); // Extracting beta from the first choice

System.out.println(nonTerminal + " is left recursive.");

// Printing reduced grammar

System.out.println(nonTerminal + " -> " + choices[1].trim() + nonTerminal + "'");

System.out.println(nonTerminal + "' -> " + beta + nonTerminal + "' / epsilon");

}

else{

System.out.println(nonTerminal + " is not left recursive."); } } }

Sample Output:

Enter Number of Productions: 1

Enter the grammar as A -> Aa / b:

E->E+T/T

GRAMMAR: E->E+T/T

E is left recursive.

E -> TE'

E' -> +TE' / epsilon

Exp4:  
import java.util.\*;

public class LeftFactoring{

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

System.out.println("Enter the number of productions:");

int n = sc.nextInt();

sc.nextLine();

String[] productions = new String[n];

System.out.println("Enter the productions:");

for (int i = 0; i < n; i++) {

productions[i] = sc.nextLine();

}

eliminateLeftFactoring(productions);

}

private static void eliminateLeftFactoring(String[] productions){

boolean leftFactored = false;

for (String production : productions){

String[] parts = production.split("->");

String lhs = parts[0].trim();

String[] rhs = parts[1].split("\\|");

String prefix = findCommonPrefix(rhs);

if (!prefix.isEmpty()) {

leftFactored = true;

System.out.println(lhs + "->" + prefix + lhs + "'");

List<String> newRhs = new ArrayList<>();

for (String r : rhs) {

if (r.startsWith(prefix)) {

String suffix = r.substring(prefix.length()).trim();

if (suffix.isEmpty()) {

suffix = "";

}

newRhs.add(suffix);

} else {

newRhs.add(r); } }

System.out.println(lhs + "'->" + String.join("|", newRhs)); } }

if (!leftFactored) {

System.out.println("Given productions do not have left factoring"); } }

private static String findCommonPrefix(String[] rhs) {

String prefix = rhs[0];

for (int i = 1; i < rhs.length; i++) {

while (rhs[i].indexOf(prefix) != 0) {

prefix = prefix.substring(0, prefix.length() - 1);

if (prefix.isEmpty()) {

return ""; } } }

return prefix; } }

Sample output:

Enter the number of productions:

1

Enter the productions:

S->Sa|Sb|Sc

S->SS'

S'->a|b|c

Exp5:

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Scanner;

import java.util.Set;

public class FirstFollow {

static String first[], follow[], grammar[][];

static List<String> nonTerminals = new ArrayList<>();

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no of productions");

int n = sc.nextInt();

grammar = new String[n][2];

System.out.println("Enter the productions that are free from Left Recursion");

sc.nextLine();

for (int i = 0; i < n; i++) {

String s = sc.nextLine();

String p[] = s.split("->");

nonTerminals.add(p[0].trim());

grammar[i][0] = p[0].trim();

grammar[i][1] = p[1];

}

first = new String[n];

follow = new String[n];

for (int i = 0; i < n; i++) {

first[i] = calculateFirst(i);

}

System.out.println("First sets:");

for (int i = 0; i < n; i++) {

System.out.println(nonTerminals.get(i) + " : " + print(first[i]));

} for (int i = 0; i < n; i++) {

follow[i] = calculateFollow(i);

}

System.out.println("Follow sets:");

for (int i = 0; i < n; i++) {

System.out.println(nonTerminals.get(i) + " : " + print(follow[i]));

}

} static String print(String s) {

StringBuilder sb = new StringBuilder();

sb.append('{');

for (char c : s.toCharArray()) {

if (sb.indexOf(String.valueOf(c)) == -1) {

sb.append(c).append(",");

}

} if (sb.charAt(sb.length() - 1) == ',') {

sb.deleteCharAt(sb.length() - 1);

}

sb.append('}');

return sb.toString();

} static String calculateFirst(int i) {

String s[] = grammar[i][1].split("\\|");

String temp = "";

for (String p : s) {

if (!nonTerminals.contains(String.valueOf(p.charAt(0)))) {

temp += p.charAt(0); // Terminal

} else {

temp += calculateFirst(nonTerminals.indexOf(String.valueOf(p.charAt(0))));

}

}

return temp;

} static String calculateFollow(int i) {

Set<Character> temp = new HashSet<>();

if (i == 0) {

temp.add('$');

} for (int idx = 0; idx < grammar.length; idx++) {

String[] rhs = grammar[idx][1].split("\\|");

for (String p : rhs) {

int nonTerminalPos = p.indexOf(nonTerminals.get(i).charAt(0));

if (nonTerminalPos != -1 && nonTerminalPos < p.length() - 1) {

char nextSymbol = p.charAt(nonTerminalPos + 1);

if (!nonTerminals.contains(String.valueOf(nextSymbol))) {

temp.add(nextSymbol);

} else {

String firstNext = first[nonTerminals.indexOf(String.valueOf(nextSymbol))];

for (char c : firstNext.toCharArray()) {

if (c != 'e') {

temp.add(c); } }

if (firstNext.contains("e")) {

String followNext = follow[nonTerminals.indexOf(String.valueOf(nextSymbol))];

for (char c : followNext.toCharArray()) {

temp.add(c); } } } }

if (nonTerminalPos != -1 && nonTerminalPos == p.length() - 1) {

String followOfNonTerminal = follow[nonTerminals.indexOf(grammar[idx][0])];

for (char c : followOfNonTerminal.toCharArray()) {

temp.add(c); } } } }

StringBuilder ans = new StringBuilder();

for (char c : temp) {

ans.append(c);

}

return ans.toString(); } }

Enter the no of productions:

3

S -> A B

A -> a | e

B -> b

S : {a}

A : {a,e}

B : {b}

S : {$}

A : {b}

B : {$}

Exp6:

import java.util.\*;

public class Exp6 {

static String first[],follow[],grammar[][];

static List<String> nonTerminals=new ArrayList<>();

static Map<String,Map<Character,String>> table=new HashMap<>();

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter the no of productions");

int n=sc.nextInt();

grammar=new String[n][2];

System.out.println("Enter the productions that are free from Left Recursion(Use '@'Symbol for Epsilon)");

sc.nextLine();

for(int i=0;i<n;i++){

String s=sc.nextLine();

String p[]=s.split("->");

nonTerminals.add(p[0].trim());

grammar[i][0]=p[0].trim();

grammar[i][1]=p[1];

}

for(String s:nonTerminals)

table.put(s, new HashMap<>());

first=new String[n];follow=new String[n];

for(int i=0;i<n;i++)

first[i]=calculateFirst(i);

for(int i=0;i<n;i++)

follow[i]=calculateFollow(i);

for(int i=0;i<n;i++)

if(first[i].contains("@"))

for(char c:follow[i].toCharArray())

table.get(grammar[i][0]).put(c,"@");

System.out.println("Enter the String to Check whether it is accepted or not");

String s=sc.nextLine();

if(check(s+"$"))

System.out.println("String is accepted");

else

System.out.println("String is rejected");

}

static String calculateFirst(int i){

String s[]=grammar[i][1].split("\\|"),temp="";

for(String p:s){

if(!nonTerminals.contains(p.charAt(0)+""))

temp+=p.charAt(0);

else{

String t=calculateFirst(nonTerminals.indexOf(p.charAt(0)+""));

if(t.contains("@")){

if(!nonTerminals.contains(p.charAt(1)+""))

temp+=p.charAt(1);

else

temp+=calculateFirst(nonTerminals.indexOf(p.charAt(1)+""));

}

temp+=t;

}

table.get(grammar[i][0]).put(temp.charAt(temp.length()-1),p);

}

return temp;

}

static String calculateFollow(int i){

Set<Character> temp=new HashSet<>();

if(i==0)

temp.add('$');

for(int idx=0;idx<grammar.length;idx++) {

if(grammar[idx][0]==nonTerminals.get(i))

continue;

String s[]=grammar[idx][1].split("\\|");

for(String p:s) {

String nt=nonTerminals.get(i);

if(p.contains(nt)) {

if(p.indexOf(nt)==p.length()-1) {

String t=follow[nonTerminals.indexOf(grammar[idx][0])];

if(t==null)

t=calculateFollow(nonTerminals.indexOf(grammar[idx][0]));

for(char c:t.toCharArray())

temp.add(c);

}

else {

int x=p.indexOf(nt);

if(!nonTerminals.contains(p.charAt(x+1)+""))

temp.add(p.charAt(x+1));

else {

if(first[nonTerminals.indexOf(p.charAt(x+1)+"")].contains("@")) {

String t=first[nonTerminals.indexOf(p.charAt(x+1)+"")];

for(char c:t.toCharArray())

temp.add(c);

temp.remove('@');

t=follow[nonTerminals.indexOf(grammar[idx][0])];

if(t==null)

t=calculateFollow(nonTerminals.indexOf(grammar[idx][0]));

for(char c:t.toCharArray())

temp.add(c);

}

else {

String t=first[nonTerminals.indexOf(p.charAt(x+1)+"")];

for(char c:t.toCharArray())

temp.add(c); } } } } } }

String ans="";

for(char x:temp)

ans+=x;

return ans;

}

static boolean check(String s) {

Stack<Character> stk=new Stack<>();

stk.add('$');

stk.add(nonTerminals.get(0).charAt(0));

int i=0;

while(i<s.length()) {

if(stk.isEmpty())

return false;

if(stk.peek()==s.charAt(i)){

stk.pop();

i++;

continue;

}

char c=stk.pop();

String t=table.get(c+"").get(s.charAt(i));

if(t=="@")

continue;

for(int idx=t.length()-1;idx>=0;idx--)

stk.push(t.charAt(idx));

} if(stk.isEmpty())

return true;

return false;

}

}

Sample Output:

Enter the no of productions

5

Enter the productions that are free from Left Recursion(Use '@' Symbol for Epsilon)

E->TA

A->+TA|@

T->FB

B->\*FB|@

F->(E)|i

Enter the String to Check whether it is accepted or not

i+i\*i

String is accepted

Exp7:

import java.util.Scanner;

public class Exp7 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String input, stack = "";

int ruleCount;

System.out.println("Enter the number of production rules: ");

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.println("Enter the input string: ");

input = scanner.nextLine();

int i = 0;

System.out.println("Stack\tInputBuffer\tAction");

while (true) {

if (i < input.length()) {

char ch = input.charAt(i);

i++;

stack += ch;

System.out.print(stack + "\t");

System.out.print(input.substring(i) + "\t\tShift " + ch + "\n");

}

for (int j = 0; j < ruleCount; j++) {

int substringIndex = stack.indexOf(rules[j].right);

if (substringIndex != -1) {

stack = stack.substring(0, substringIndex) + rules[j].left;

System.out.print(stack + "\t");

System.out.print(input.substring(i) + "\t\tReduce " + rules[j].left + "->" +

rules[j].right + "\n");

j = -1;

}

}

if (stack.equals(rules[0].left) && i == input.length()) {

System.out.println("\nAccepted");

break;

}

if (i == input.length()) {

System.out.println("\nNot Accepted");

break;

}

}

scanner.close(); } }

class ProductionRule {

String left;

String right;

ProductionRule(String left, String right) {

this.left = left;

this.right = right; } }

Sample output:

Enter the number of production rules: 4

Enter the production rules (in the form 'left->right'):

E->E+E

E->E\*E

E->(E)

E->i

Enter the input string: i\*i+i

Stack InputBuffer Action

i \*i+I Shift i

E \*i+i Reduce E->i

E\* i+i Shift \*

E\*i +i Shift i

E\*E +i Reduce E->i

E + i Reduce E->E\*E

E+ i Shift +

E+i Shift i

E+E Reduce E->i

E Reduce E->E+E

Accepted

Exp8:

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");

}

}

}

Sample output:

Enter the no. of terminals: 2

Enter the terminals: + \*

Enter the table values:

Enter the value for + +: <

Enter the value for + \*: <

Enter the value for \* +: >

Enter the value for \* \*: <

Enter the input string: a+a

OPERATOR PRECEDENCE TABLE:

+ \*

+ < <

\* > <

Enter the input string: a+a

STACK INPUT STRING ACTION

$ a+a Shift a

a$ +a Shift +

a+a$ Shift a

a+a$a$ Reduce

$ $ String is accepted

Exp9:

import java.util.Scanner;

public class Exp9 {

static int[] stack;

static int top, n;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

top = -1;

System.out.println("Enter the size of stack[MAX=100]: ");

n = scanner.nextInt();

if (n <= 0) {

System.out.println("Invalid stack size.");

return;

}

stack = new int[n];

System.out.println("\n\tStack Operations:");

System.out.println("\t--------------------------");

System.out.println("\t1. Push");

System.out.println("\t2. Pop");

System.out.println("\t3. Display");

System.out.println("\t4. EXIT");

int choice;

do {

System.out.println("\nEnter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1: push(scanner);

break;

case 2: pop();

break;

case 3: display();

break;

case 4: System.out.println("\nEXIT");

break;

default:

System.out.println("Please enter a valid choice.");

}

} while (choice != 4);

scanner.close();

}

static void push(Scanner scanner) {

if (top >= n - 1) {

System.out.println("\nStack overflow");

} else {

System.out.println("Enter a value to be pushed: ");

int x = scanner.nextInt();

top++;

stack[top] = x;

}

}

static void pop() {

if (top == -1)

System.out.println("\nStack underflow");

else {

System.out.println("\nThe popped element is " + stack[top]);

top--;

}

}

static void display() {

if (top >= 0) {

System.out.println("\nThe elements in the stack are:");

for (int i = top; i >= 0; i--)

System.out.println(stack[i]);

System.out.println("\nSelect next choice");

} else

System.out.println("\nThe stack is empty."); } }

Sample Output:

Enter the size of stack[MAX=100]: 10

Stack Operations:

1. Push

2. Pop

3. Display

4. EXIT

Enter your choice: 1

Enter a value to be pushed: 5

Exp10:

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

import java.util.Stack;

class Instruction {

String op;

String arg1;

String arg2;

String result;

Instruction(String op, String arg1, String arg2, String result) {

this.op = op;

this.arg1 = arg1;

this.arg2 = arg2;

this.result = result;

}

@Override

public String toString() {

return result + " = " + arg1 + " " + op + " " + arg2;

}

}

class IntermediateCodeGenerator {

private List<Instruction> instructions;

private Stack<String> operands;

private int tempCount;

IntermediateCodeGenerator() {

instructions = new ArrayList<>();

operands = new Stack<>();

tempCount = 0;

}

public List<Instruction> generate(String expression) {

Stack<Character> operators = new Stack<>();

StringBuilder operand = new StringBuilder();

for (int i = 0; i < expression.length(); i++) {

char token = expression.charAt(i);

if (Character.isWhitespace(token)) {

continue;

}

if (Character.isLetterOrDigit(token)) {

operand.append(token);

if (i == expression.length() - 1 || !Character.isLetterOrDigit(expression.charAt(i + 1))) {

operands.push(operand.toString());

operand.setLength(0);

}

} else if (token == '(') {

operators.push(token);

} else if (token == ')') {

while (!operators.isEmpty() && operators.peek() != '(') {

processOperator(operators.pop());

}

operators.pop();

} else if (isOperator(token)) {

while (!operators.isEmpty() && precedence(token) <= precedence(operators.peek())) {

processOperator(operators.pop());

}

operators.push(token);

} else {

System.out.println("Invalid character encountered: " + token);

return null; } }

while (!operators.isEmpty()) {

processOperator(operators.pop());

} return instructions; }

private void processOperator(char operator) {

String operand2 = operands.pop();

String operand1 = operands.pop();

String result = newTemp();

instructions.add(new Instruction(String.valueOf(operator), operand1, operand2, result));

operands.push(result);

}

private String newTemp() {

return "t" + tempCount++;

}

private boolean isOperator(char token) {

return token == '+' || token == '-' || token == '\*' || token == '/';

}

private int precedence(char operator) {

switch (operator) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

default:

return -1; } } }

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter an arithmetic expression:");

String expression = scanner.nextLine();

IntermediateCodeGenerator icg = new IntermediateCodeGenerator();

List<Instruction> code = icg.generate(expression);

if (code != null) {

System.out.println("\nIntermediate Code:");

for (Instruction instr : code) {

System.out.println(instr);

}

}

scanner.close();

}

}

Sample output:

a\*b+c/d-e/f+g\*h

Intermediate Code:

t0 = a \* b

t1 = c / d

t2 = t0 + t1

t3 = e / f

t4 = t2 - t3

t5 = g \* h

t6 = t4 + t5