**PRACTICAL 01**

**Aim**: Write a Program to convert the given NDFA to DFA.

**Conversion.java**

package NtPd;

import java.util.StringTokenizer;

public class Conversion

{

String Nonterminal[];

String Terminal[];

String TT[][];

String STT[][];

String Prod[][];

int STTsize;

public Conversion(String Nonterminal[], String Terminal[], String TT[][], String Prod[][], String STT[][])

{

this.Nonterminal = Nonterminal;

this.Terminal = Terminal;

this.TT = TT;

this.Prod = Prod;

this.STT = STT;

}

public void PrintTransistionTable(String TT[][], String P[][], String T[], String NT[])

{

int i, j, k, m;

TT[0][0] = "-";

for (i = 1; i < 4; i++)

{

TT[0][i] = T[i - 1];

STT[0][i] = T[i - 1];

}

for (i = 1; i < 5; i++)

{

TT[i][0] = NT[i - 1];

STT[i][0] = NT[i - 1];

}

for (i = 1; i <= 4; i++)//for TT row cpunt

{

for (j = 0; j <= 6; j++)//for colum Production row count

{

if (TT[i][0] == P[j][0])

{

for (k = 1; k <= 3; k++)

{

for (m = 0; m <= 6; m++)

{

if (TT[0][k] == P[m][1] && TT[i][0] == P[m][0])

{

if (TT[i][k] == "-")

{

TT[i][k] = P[m][2];

}

else if (TT[i][k] != P[m][2])

{

String str = TT[i][k]; //take previous values stored in transistion table

if (str.length() < 2)

{

TT[i][k] = TT[i][k] + "," + P[m][2];

}}}}}}}}

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

System.out.println("\t" + "===Transistion Table===");

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

for (i = 0; i < 5; i++)

{

for (j = 0; j < 4; j++)

{

System.out.print("\t" + TT[i][j]);

}

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

}

STT[0][0] = "-";

for (i = 0; i < 4; i++)

{

STT[1][i] = TT[1][i];

}

STTsize = 2;

}

public void subTransTable()

{

boolean addRow = false;

for (int i = 1; i <= 3; i++)

{

for (int j = 1; j < STTsize; j++)

{

if (STT[j][0].equals(STT[j][i]) == false && STT[j][i].equals("-") == false)

{

String st = STT[j][i];

for (int z = 1; z < STTsize; z++)

{

if (st.equals(STT[z][0]))

{

addRow = false;

z = STTsize + 1;

}

else

{

addRow = true;

}

}

if (addRow)

{

String transt[][] = getRow(st);

for (int k = 0; k < 4; k++)

{

STT[STTsize][k] = transt[0][k];

}

addRow = false;

}

}

}

}

STTsize++;

if (STTsize < 11)

{

subTransTable();

}

}

public void displaySTT()

{

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

System.out.println("\t" + "===Sub Transistion Table===");

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

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 4; j++)

{

System.out.print("\t" + STT[i][j]);

}

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

}

}

public String[][] getRow(String ST)

{

String transt[][] = new String[1][4];

StringTokenizer StrTok = new StringTokenizer(ST, ",");

ST = "";

while (StrTok.hasMoreTokens())

{

ST = ST + StrTok.nextToken();

}

for (int i = 0; i < 4; i++)

{

transt[0][i] = "";

}

for (int i = 1; i < 5; i++)

{

for (int k = 0; k < ST.length(); k++)

{

if (TT[i][0].equals(ST.substring(k, k + 1)))

{

for (int j = 0; j < 4; j++)

{

if (k == 0 || transt[0][j].equals("-"))

{

transt[0][j] = TT[i][j];

}

else if (TT[i][j].equals("-") == false)

{

transt[0][j] = transt[0][j] + "," + TT[i][j];

}

}

if (k == ST.length())

{

i = 5;

}

}

}

}

return transt;

}

public void PrintProduction(String lProd[][])

{

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

System.out.println("\tConversion of NDFA to DFA:");

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

System.out.println("\tProduction are:\n");

for (int i = 0; i < 7; i++)

{

System.out.println("\t" + lProd[i][0] + " => " + lProd[i][1] + "" + lProd[i][2]);

}

}

public static void main(String[] args)

{

String NonTerminal[] = {"A", "B", "C", "D"};

String Terminal[] = {"a", "b", "c"};

String TT[][] = {{"-", "-", "-", "-"},

{"-", "-", "-", "-"},

{"-", "-", "-", "-"},

{"-", "-", "-", "-"},

{"-", "-", "-", "-"}};

String STT[][] = new String[10][4];

String Prod[][] = {{"A", "a", "A"},

{"A", "c", "B"},

{"B", "b", "A"},

{"B", "b", "B"},

{"B", "a", "C"},

{"C", "a", "D"},

{"D", "c", "C"}};

Conversion c = new Conversion(NonTerminal, Terminal, TT, Prod, STT);

c.PrintProduction(Prod);

c.PrintTransistionTable(TT, Prod, Terminal, NonTerminal);

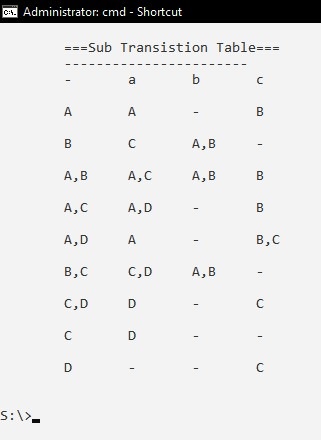
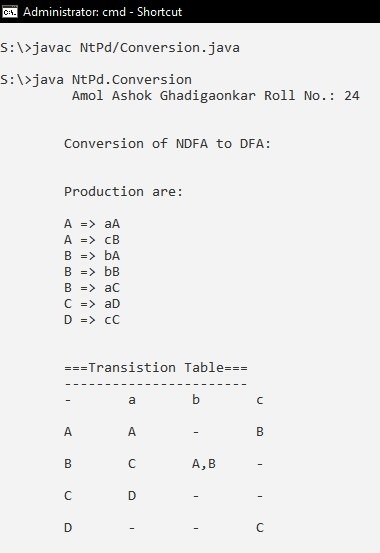
c.subTransTable();

c.displaySTT();

}

}

Output:

****

**PRACTICAL 02**

**Aim:** Write a program to convert Left Linear Grammar to Right Linear Grammar.

***RightToLeft.java***

package NtPd;

public class RightToLeft

{

char tmp;

int i;

String prod[][] = {{"S", "", "B"},

{"B", "b", "D"},

{"D", "b", "B"},

{"S", "a", "B"},

{"B", "a", ""}};

RightToLeft()

{}

void PrintNonTerminal()

{

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

System.out.println("\tNon terminals are : " + "S,B,D");

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

}

void PrintTerminal()

{

System.out.println("\tTerminals are : " + "a,b");

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

}

void PrintProductions()

{

int i;

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

System.out.println("\tConversion of Right to left Linear Grammar:");

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

System.out.println("\tProductions are :");

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

for (i = 0; i < 5; i++)

{

if (prod[i][1] == "")

{

System.out.println("\t" + prod[i][0] + " --> " + "I" + prod[i][2]);

}

else if (prod[i][2] == "")

{

prod[i][2] = "Z";

System.out.println("\t" + prod[i][0] + " --> " + prod[i][1] + "Z");

}

else

{

System.out.println("\t" + prod[i][0] + " --> " + prod[i][1] + prod[i][2]);

}

}

}

void PrintLeft2RightLinear()

{

int i;

System.out.println("\t" + "Left Linear Grammar:");

System.out.println("\t" + "===============");

for (i = 0; i < 5; i++)

{

String tmp = prod[i][1];

prod[i][1] = prod[i][2];

prod[i][2] = tmp;

System.out.println("\t" + prod[i][0] + " --> " + prod[i][1] + prod[i][2]);

}

}

public static void main(String[] args)

{

System.out.println("\t Anand Yadav Roll No.: 11");

RightToLeft r = new RightToLeft();

r.PrintProductions();

r.PrintNonTerminal();

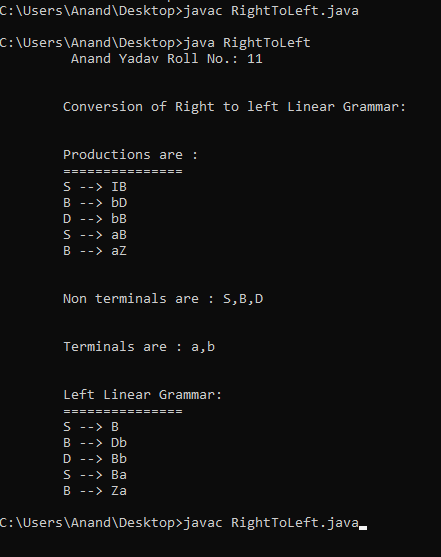
r.PrintTerminal();

r.PrintLeft2RightLinear();

}

}

Output:



**PRACTICAL 03**

**Aim**: Write a program to illustrate the generation on SPM for the input Grammar.

*SPM\_1.java*

package NtPd;

public class **SPM\_1**

{

SPM\_1()

{

}

void PrintProduction(String Prod[])

{

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

{

System.out.println("\t" + Prod[i]);

}

}

void displayMatrix(int Matrix[][], String strString)

{

System.out.print(" ");

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

{

System.out.print(" " + strString.charAt(i));

}

System.out.print("\n");

for (int i = 0; i < 7; i++)

{

System.out.println();

System.out.print(" " + strString.charAt(i) + " ");

for (int j = 0; j < 7; j++)

{

System.out.print(Matrix[i][j] + " ");

}

}

}

void displayMatrix1(String matrix[][], String strString)

{

System.out.print(" ");

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

{

System.out.print(" " + strString.charAt(i));

}

System.out.print("\n");

for (int i = 0; i < 7; i++)

{

System.out.println();

System.out.print(" " + strString.charAt(i) + " ");

for (int j = 0; j < 7; j++)

{

System.out.print(matrix[i][j] + " ");

}

}

System.out.print("\n");

}

}

***SPM\_2.java***

package NtPd;

public class **SPM\_2**

{

static int i, j, k, l = 0;

static SPM\_1 s = new SPM\_1();

static String Prod[] = new String[4];

static String NT[] = {"Z", "M", "L"};

static String T[] = {"a", "b", "(", ")"};

static String first[] = new String[4];

static String last[] = new String[4];

static String equal[] = new String[5];

static int first\_matrix[][] = new int[NT.length + T.length][NT.length + T.length];

static int last\_matrix[][] = new int[NT.length + T.length][NT.length + T.length];

static int equal\_matrix[][] = new int[NT.length + T.length][NT.length + T.length];

static int SPM[][] = new int[NT.length + T.length][NT.length + T.length];

static int A[][] = new int[NT.length + T.length][NT.length + T.length];

static int B[][] = new int[NT.length + T.length][NT.length + T.length];

static int C[][] = new int[NT.length + T.length][NT.length + T.length];

static int D[][] = new int[NT.length + T.length][NT.length + T.length];

static int E[][] = new int[NT.length + T.length][NT.length + T.length];

static int I[][] = new int[NT.length + T.length][NT.length + T.length];

static int TlastPlus\_matrix[][] = new int[NT.length + T.length][NT.length + T.length];

static int lessthan\_matrix[][] = new int[NT.length + T.length][NT.length + T.length];

static int greaterthan\_matrix[][] = new int[NT.length + T.length][NT.length + T.length];

static String spm[][] = new String[NT.length + T.length][NT.length + T.length];

static String strString = "ZbMLa()";

public static void SPM()

{

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

if (lessthan\_matrix[i][j] == 1)

{

spm[i][j] = "<";

}

else if (greaterthan\_matrix[i][j] == 1)

{

spm[i][j] = ">";

}

else if (equal\_matrix[i][j] == 1)

{

spm[i][j] = "=";

}

else

{

spm[i][j] = "0";

}

}

}

}

public static void FirstPlus(int matrix[][])

{

System.out.println("\n\n" + "First+ matrix is : ");

System.out.println();

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

A[i][j] = matrix[i][j];

}

}

for (i = 1; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

if (A[j][i] == 1)

{

for (k = 1; k < NT.length + T.length; k++)

{

A[j][k] = A[j][k] | A[i][k];

}

}

}

}

s.displayMatrix(A, strString);

}

public static void FirstStar(int first\_matrix[][])

{

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

if (i == j)

{

I[i][j] = 1;

}

else

{

I[i][j] = 0;

}

}

}

System.out.println("\n\n" + "First\* matrix is : ");

System.out.println();

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

C[i][j] = A[i][j] | I[i][j];

}

}

s.displayMatrix(C, strString);

}

public static void LastPlus(int matrix[][])

{

System.out.println("\n\n" + "Last+ matrix is : ");

System.out.println();

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

B[i][j] = last\_matrix[i][j];

}

}

for (i = 1; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

if (B[j][i] == 1)

{

for (k = 1; k < NT.length + T.length; k++)

{

B[j][k] = B[j][k] | B[i][k];

}

}

}

}

s.displayMatrix(B, strString);

}

public static void LastStar(int matrix[][])

{

System.out.println("\n\n" + "Last\* matrix is : ");

System.out.println();

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

E[i][j] = B[i][j] | I[i][j];

}

}

s.displayMatrix(E, strString);

}

public static void EqualMatrix()

{

//Finding the elements of equal

int pos = Prod[0].indexOf(">");

for (i = 0; i < Prod.length; i++)

{

String str = Prod[i].substring(pos + 1);

if (str.length() >= 2)

{

for (j = 0; j < str.length() - 1; j++)

{

equal[l] = str.charAt(j) + "" + str.charAt(j + 1);

l++;

}

}

}

//Displaying of equal elements

System.out.println();

System.out.println("\nEqual Elements ::\n");

s.PrintProduction(equal);

System.out.println();

for (i = 0; i < equal\_matrix.length; i++)

{

for (j = 0; j < equal\_matrix.length; j++)

{

equal\_matrix[i][j] = 0;

}

}

for (i = 0; i < strString.length(); i++)

{

for (j = 0; j < strString.length(); j++)

{

for (k = 0; k < equal.length; k++)

{

if (strString.charAt(i) == equal[k].charAt(0) && strString.charAt(j) == equal[k].charAt(1))

{

equal\_matrix[i][j] = 1;

}

}

}

}

s.displayMatrix(equal\_matrix, strString);

}

public static void LessThanMatrix()

{

System.out.println();

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

lessthan\_matrix[i][j] = equal\_matrix[i][0] \* A[0][j]

+ equal\_matrix[i][1] \* A[1][j]

+ equal\_matrix[i][2] \* A[2][j]

+ equal\_matrix[i][3] \* A[3][j]

+ equal\_matrix[i][4] \* A[4][j]

+ equal\_matrix[i][5] \* A[5][j]

+ equal\_matrix[i][6] \* A[6][j];

}

}

}

public static void TransposeLastPlus()

{

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

TlastPlus\_matrix[i][j] = B[j][i];

}

}

}

public static void GreaterThanMatrix()

{

System.out.println("\n\n" + "Greater than matrix is : ");

System.out.println();

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

D[i][j] = TlastPlus\_matrix[i][0] \* equal\_matrix[0][j]

+ TlastPlus\_matrix[i][1] \* equal\_matrix[1][j]

+ TlastPlus\_matrix[i][2] \* equal\_matrix[2][j]

+ TlastPlus\_matrix[i][3] \* equal\_matrix[3][j]

+ TlastPlus\_matrix[i][4] \* equal\_matrix[4][j]

+ TlastPlus\_matrix[i][5] \* equal\_matrix[5][j]

+ TlastPlus\_matrix[i][6] \* equal\_matrix[6][j];

}

}

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

greaterthan\_matrix[i][j] = D[i][0] \* C[0][j]

+ D[i][1] \* C[1][j]

+ D[i][2] \* C[2][j]

+ D[i][3] \* C[3][j]

+ D[i][4] \* C[4][j]

+ D[i][5] \* C[5][j]

+ D[i][6] \* C[6][j];

}

}

s.displayMatrix(greaterthan\_matrix, strString);

}

public static void spm\_Matrix()

{

System.out.println("\n\n" + "SPM matrix is : ");

System.out.println();

for (i = 0; i < NT.length + T.length; i++)

{

for (j = 0; j < NT.length + T.length; j++)

{

SPM[i][j] = lessthan\_matrix[i][j] | greaterthan\_matrix[i][j] | equal\_matrix[i][j];

}

}

s.displayMatrix(SPM, strString);

}

public static void main(String[] args)

{

System.out.println("\t Anand Yadav Roll No.: 11");

Prod[0] = "Z->bMb";

Prod[1] = "M->(L";

Prod[2] = "M->a";

Prod[3] = "L->Ma)";

//Displaying of productins.

System.out.println("Productions are :: \n");

s.PrintProduction(Prod);

//Finding Elements of First

for (i = 0; i < Prod.length; i++)

{

first[i] = Prod[i].charAt(0) + "" + Prod[i].charAt(3);

}

//Displaying of First elements

System.out.println("\nFirst Elements :: \n");

s.PrintProduction(first);

System.out.println("\nFirst Matrix :: \n");

for (i = 0; i < first\_matrix.length; i++)

{

for (j = 0; j < first\_matrix.length; j++)

{

first\_matrix[i][j] = 0;

}

}

for (i = 0; i < strString.length(); i++)

{

for (j = 0; j < strString.length(); j++)

{

for (k = 0; k < first.length; k++)

{

if (strString.charAt(i) == first[k].charAt(0) && strString.charAt(j) == first[k].charAt(1))

{

first\_matrix[i][j] = 1;

}

}

}

}

s.displayMatrix(first\_matrix, strString);

FirstPlus(first\_matrix);

FirstStar(first\_matrix);

//Finding the elements of Last

for (i = 0; i < Prod.length; i++)

{

last[i] = Prod[i].charAt(0) + "" + Prod[i].charAt(Prod[i].length() - 1);

}

//Displaying of Last elements

System.out.println();

System.out.println("\nLast Elements :: \n");

s.PrintProduction(last);

System.out.println("\nLast Matrix :: \n");

for (i = 0; i < last\_matrix.length; i++)

{

for (j = 0; j < last\_matrix.length; j++)

{

last\_matrix[i][j] = 0;

}

}

for (i = 0; i < strString.length(); i++)

{

for (j = 0; j < strString.length(); j++)

{

for (k = 0; k < last.length; k++)

{

if (strString.charAt(i) == last[k].charAt(0) && strString.charAt(j) == last[k].charAt(1))

{

last\_matrix[i][j] = 1;

}

}

}

}

s.displayMatrix(last\_matrix, strString);

LastPlus(last\_matrix);

LastStar(last\_matrix);

//Displaying of equal elements

EqualMatrix();

System.out.println();

//Displaying less than matrix

System.out.println("\n" + "Less than matrix is : ");

LessThanMatrix();

s.displayMatrix(lessthan\_matrix, strString);

TransposeLastPlus();

GreaterThanMatrix();

spm\_Matrix();

System.out.println("\n\nThe SPM Matrix is : \n");

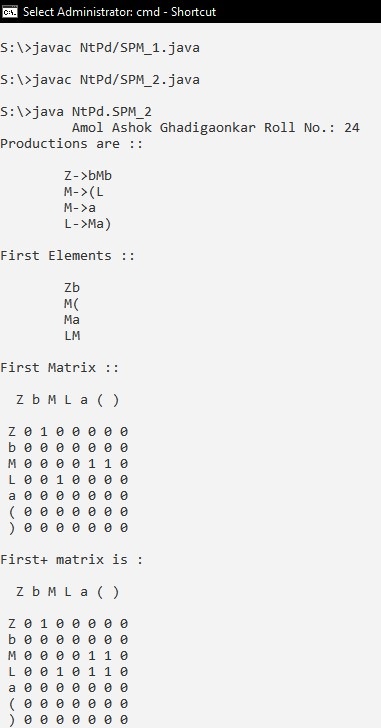
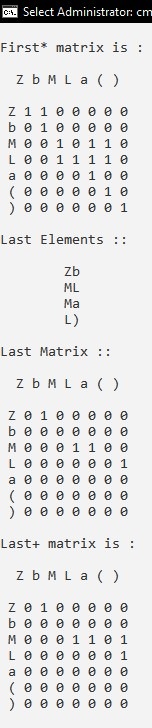
SPM();

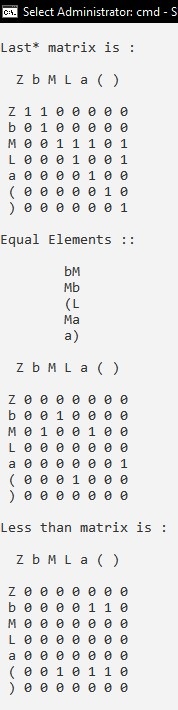
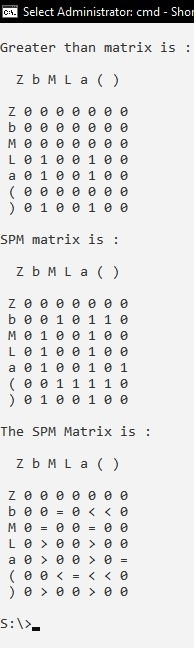
s.displayMatrix1(spm, strString);

}

}

Output:

i.  ii.

iii.iv. 

**PRACTICAL 04**

**Aim:** Write a program to illustrate the generation on OPM for the input Operator Grammar.

***OPM.java***

package NtPd;

public class OPM {

int i, j, k;

String[] prod;

String syms, nt, t;

final int LEN, NLEN, TLEN;

int[][] f;

int[][] l;

char[][] opm;

OPM(String prod[], String syms, String nt, String t, int LEN, int NLEN, int TLEN, int f[][], int l[][], char opm[][])

{

this.prod = prod;

this.syms = syms;

this.nt = nt;

this.t = t;

this.LEN = LEN;

this.NLEN = NLEN;

this.TLEN = TLEN;

this.f = f;

this.l = l;

this.opm = opm;

}

int[][] getWarshallClosure(int[][] a)

{

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

for (j = 0; j < a.length; j++) {

if (a[j][i] == 1) {

for (k = 0; k < a.length; k++) {

a[j][k] = a[j][k] | a[i][k];

}

}

}

}

return a;

}

void printGrammar() {

String grammar = "G = <{" + nt.charAt(0) + ",";

for (i = 1; i < nt.length() - 1; i++) {

grammar += nt.charAt(i) + ",";

}

grammar += nt.charAt(nt.length() - 1) + "},{" + t.charAt(0) + ",";

for (i = 1; i < t.length() - 1; i++) {

grammar += t.charAt(i) + ",";

}

grammar += t.charAt(t.length() - 1) + "},P," + nt.charAt(0) + "}>\nP = {\n\t" + prod[0] + ",";

for (i = 1; i < prod.length - 1; i++) {

grammar += "\n\t" + prod[i] + ",";

}

System.out.println(grammar + "\n\t" + prod[prod.length - 1] + "\n }");

}

public static void main(String[] args)

{

System.out.println("\t Anand yadav Roll No.: 11");

int i, j, ind, ind1;

String[] prod = {"E->E+T", "E->T", "T->T\*F", "T->F", "F->(E)", "F->i"};

String syms = "ETF+\*()i", nt = "ETF", t = "+\*()i";

final int LEN = syms.length(), NLEN = nt.length(), TLEN = t.length();

int[][] f = new int[LEN][LEN];

int[][] l = new int[LEN][LEN];

char[][] opm = new char[TLEN + 1][TLEN + 1];

OPM o = new OPM(prod, syms, nt, t, LEN, NLEN, TLEN, f, l, opm);

System.out.println("Given input grammar is:-");

o.printGrammar();

for (String p : prod) {

f[syms.indexOf(p.charAt(0))][syms.indexOf(p.charAt(3))] = 1;

l[syms.indexOf(p.charAt(0))][syms.indexOf(p.charAt(p.length() - 1))] = 1;

if (p.length() > 4 && t.contains("" + p.charAt(4))) {

f[syms.indexOf(p.charAt(0))][syms.indexOf(p.charAt(4))] = 1;

l[syms.indexOf(p.charAt(0))][syms.indexOf(p.charAt(4))] = 1;

}

}

f = o.getWarshallClosure(f);

l = o.getWarshallClosure(l);

System.out.println("\nOperator precedence matrix for the above grammar is: \n");

t = t + "$";

for (i = 0; i < TLEN; i++) {

if (f[0][NLEN + i] != 0) {

opm[TLEN][i] = '<';

}

if (l[0][NLEN + i] != 0) {

opm[i][TLEN] = '>';

}

}

for (String p : prod) {

String rhs = p.substring(3, p.length()), x, b, c = "";

if (rhs.length() >= 2) {

c = "" + rhs.charAt(2);

}

if (rhs.length() > 1) {

x = "" + rhs.charAt(0);

b = "" + rhs.charAt(1);

if (t.contains(x) && t.contains(b)) {

opm[t.indexOf(x)][t.indexOf(b)] = '=';

}

if (t.contains(x) && nt.contains(b)) {

if (t.contains(c)) {

opm[t.indexOf(x)][t.indexOf(c)] = '=';

}

}

if (nt.contains(x) && t.contains(b)) {

ind = nt.indexOf(x);

ind1 = t.indexOf(b);

for (i = 0; i < TLEN; i++) {

if (l[ind][NLEN + i] != 0) {

opm[i][ind1] = '>';

}

}

} else if (nt.contains(b) && t.contains(c)) {

ind = nt.indexOf(b);

ind1 = t.indexOf(c);

for (i = 0; i < TLEN; i++) {

if (l[ind][NLEN + i] != 0) {

opm[i][ind1] = '>';

}

}

}

if (t.contains(x) && nt.contains(b)) {

ind = t.indexOf(x);

ind1 = nt.indexOf(b);

for (i = 0; i < TLEN; i++) {

if (f[ind1][NLEN + i] != 0) {

opm[ind][i] = '<';

}

}

}

else if (t.contains(b) && nt.contains(c))

{

ind = t.indexOf(b);

ind1 = nt.indexOf(c);

for (i = 0; i < TLEN; i++)

{

if (f[ind1][NLEN + i] != 0)

{

opm[ind][i] = '<';

}

}

}

}

}

for (i = 0; i <= TLEN; i++)

{

System.out.print("\t" + t.charAt(i));

}

System.out.println();

for (i = 0; i <= TLEN; i++) {

System.out.print(t.charAt(i) + "\t");

for (j = 0; j <= TLEN; j++)

{

System.out.print(opm[i][j] + "\t");

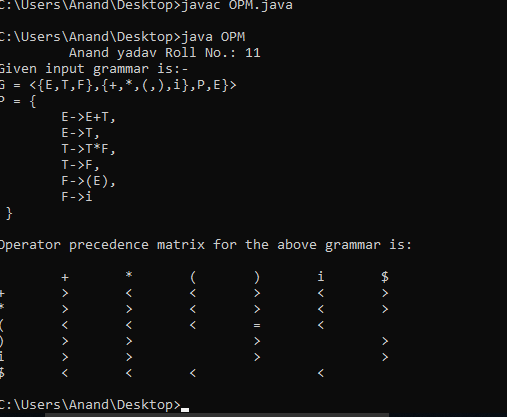
}

System.out.println();

}

}

}Output:



**PRACTICAL 05**

**Aim:** Implement a Simple Program Analyzer and Interpreter for the Straight-Line programming language.

Create several Java Classes as below:

***Slp.java***

package NtPd;

abstract class Stm

{

}

class CompoundStm extends Stm

{

Stm stm1, stm2;

CompoundStm(Stm s1, Stm s2)

{

stm1 = s1;

stm2 = s2;

}

}

class AssignStm extends Stm

{

String id;

Exp exp;

AssignStm(String i, Exp e)

{

id = i;

exp = e;

}

}

class PrintStm extends Stm

{

ExpList exps;

PrintStm(ExpList e)

{

exps = e;

}

}

abstract class Exp

{

}

class IdExp extends Exp

{

String id;

IdExp(String i)

{

id = i;

}

}

class NumExp extends Exp

{

int num;

NumExp(int n)

{

num = n;

}

}

class OpExp extends Exp

{

Exp left, right;

int oper;

final static int Plus = 1, Minus = 2, Times = 3, Div = 4;

OpExp(Exp l, int o, Exp r)

{

left = l;

oper = o;

right = r;

}

}

class EseqExp extends Exp

{

Stm stm;

Exp exp;

EseqExp(Stm s, Exp e)

{

stm = s;

exp = e;

}

}

abstract class ExpList

{

}

class PairExpList extends ExpList

{

Exp head;

ExpList tail;

public PairExpList(Exp h, ExpList t)

{

head = h;

tail = t;

}

}

class LastExpList extends ExpList

{

Exp head;

public LastExpList(Exp h)

{

head = h;

}

}

/\*Page 13 says to create this class for the interpreter in exercise 2\*/

class Table

{

String id;

int value;

Table tail;

public Table(String i, int v, Table t)

{

id = i;

value = v;

tail = t;

}

public static Table update(String i, int v, Table t)

{

return new Table(i, v, t);

}

public int lookup(String key)

{

if (id == key)

{

return value;

}

else

{

return tail.lookup(key);

}

}

}

/\*Page 14 says to define IntAndTable class for the interpreter in exercise 2\*/

class IntAndTable

{

int i;

Table t;

IntAndTable(int ii, Table tt)

{

i = ii;

t = tt;

}

}

***prog.java***

package NtPd;

class prog

{

static Stm prog =

new CompoundStm(new AssignStm("a", new OpExp(new NumExp(5), OpExp.Plus,

new NumExp(3))),

new CompoundStm(new AssignStm("b",

new EseqExp(new PrintStm(new PairExpList(new IdExp("a"),

new LastExpList(new OpExp(new IdExp("a"), OpExp.Minus,

new NumExp(1))))),

new OpExp(new NumExp(10), OpExp.Times, new IdExp("a")))),

new PrintStm(new LastExpList(new IdExp("b")))));

}

***interp.java***

package NtPd;

class interp

{

/\*Begin exercise part 2, page 12\*/

static void interp(Stm s)

{

/\* you write this part \*/

interpStm(s, null);

}

static Table interpStm(Stm s, Table t)

{

if (s.getClass() == CompoundStm.class)

{

Table cs1 = interpStm(((CompoundStm) s).stm1, t);

return interpStm(((CompoundStm) s).stm2, cs1);

}

else if (s.getClass() == AssignStm.class)

{

IntAndTable as1 = interpExp(((AssignStm) s).exp, t);

return new Table(((AssignStm) s).id, as1.i, as1.t);

}

else if (s.getClass() == PrintStm.class)

{

return print(((PrintStm) s).exps, t);

}

return null;

}

static Table print(ExpList e, Table t)

{

if (e.getClass() == PairExpList.class)

{

IntAndTable p1 = interpExp(((PairExpList) e).head, t);

System.out.print(p1.i + " ");

return print(((PairExpList) e).tail, p1.t);

}

else

{

IntAndTable p2 = interpExp(((LastExpList) e).head, t);

System.out.println(p2.i + " ");

return p2.t;

}

}

static IntAndTable interpExpList(ExpList e, Table t)

{

if (e.getClass() == PairExpList.class)

{

IntAndTable pe1 = interpExp(((PairExpList) e).head, t);

return interpExpList(((PairExpList) e).tail, pe1.t);

}

else if (e.getClass() == LastExpList.class)

{

return interpExp(((LastExpList) e).head, t);

}

//this shouldn't happen...

return null;

}

static IntAndTable interpExp(Exp e, Table t)

{

if (e.getClass() == IdExp.class)

{

return new IntAndTable(t.lookup(((IdExp) e).id), t);

}

else if (e.getClass() == NumExp.class)

{

return new IntAndTable(((NumExp) e).num, t);

}

else if (e.getClass() == OpExp.class)

{

IntAndTable oe1 = interpExp(((OpExp) e).left, t);

IntAndTable oe2 = interpExp(((OpExp) e).right, oe1.t);

switch (((OpExp) e).oper)

{

case 1:

return new IntAndTable(oe1.i + oe2.i, oe2.t);

case 2:

return new IntAndTable(oe1.i - oe2.i, oe2.t);

case 3:

return new IntAndTable(oe1.i \* oe2.i, oe2.t);

case 4:

return new IntAndTable(oe1.i / oe2.i, oe2.t);

}

}

else if (e.getClass() == EseqExp.class)

{

Table es1 = interpStm(((EseqExp) e).stm, t);

return interpExp(((EseqExp) e).exp, es1);

}

return null;

}

/\*End exercise part 2, page 12\*/

/\*Begin exercise part 1, page 12\*/

static int maxargs(Stm s)

{

/\* you write this part \*/

if (s.getClass() == CompoundStm.class)

{

int cs1 = maxargs(((CompoundStm) s).stm1);

int cs2 = maxargs(((CompoundStm) s).stm2);

return cs1 > cs2 ? cs1 : cs2;

}

else if (s.getClass() == AssignStm.class)

{

return maxargs(((AssignStm) s).exp);

}

else if (s.getClass() == PrintStm.class)

{

return maxargs(((PrintStm) s).exps, 0);

}

return 0;

}

static int maxargs(Exp e)

{

if (e.getClass() == IdExp.class)

{

return 0;

}

else if (e.getClass() == NumExp.class)

{

return 0;

}

else if (e.getClass() == OpExp.class)

{

int oe1 = maxargs(((OpExp) e).left);

int oe2 = maxargs(((OpExp) e).right);

return oe1 > oe2 ? oe1 : oe2;

}

else if (e.getClass() == EseqExp.class)

{

int ee1 = maxargs(((EseqExp) e).stm);

int ee2 = maxargs(((EseqExp) e).exp);

return ee1 > ee2 ? ee1 : ee2;

}

return 0;

}

static int maxargs(ExpList l, int count)

{

if (l.getClass() == PairExpList.class)

{

return maxargs(((PairExpList) l).tail, count + 1);

}

else if (l.getClass() == LastExpList.class)

{

return count + 1;

}

return 0;

}

/\*End exercise part 1, page 12\*/

public static void main(String args[]) throws java.io.IOException

{

System.out.println("\t Anand Yadav Roll No.: 11");

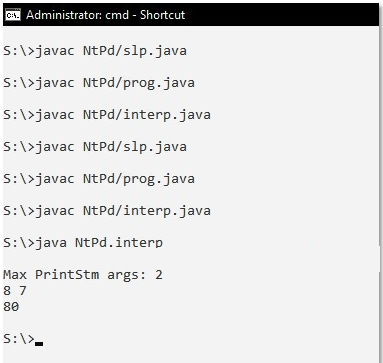
System.out.println("Max PrintStm args: " + maxargs(prog.prog));

interp(prog.prog);

}

}

Output:



**PRACTICAL 06**

**Aim:** Write a Code to generate the DAG for the Input arithmetic expression.

***DAG.java***

package NtPd;

import java.util.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class DAG

{

private String expression = "B\*-C+B\*-C";

private List<String> tacList = new ArrayList<String>();

public void printExpression()

{

System.out.println("\nInput Expression :");

System.out.println("===================\n");

System.out.println("Expression = " + expression + "\n\n");

}

public void convertToTAC()

{

int cnt = 1;

String tempStr = new String();

tempStr = "T" + cnt + "=" + expression.substring(2, 4);

cnt++;

tacList.add(tempStr);

tempStr = "T" + cnt + "=" + expression.substring(0, 2) + "T" + (cnt - 1);

cnt++;

tacList.add(tempStr);

tempStr = "T" + cnt + "=" + expression.substring(7, 9);

cnt++;

tacList.add(tempStr);

tempStr = "T" + cnt + "=" + expression.substring(5, 7) + "T" + (cnt - 1);

cnt++;

tacList.add(tempStr);

tempStr = "T5=T2+T4";

tacList.add(tempStr);

System.out.println("Three Address Code : ");

System.out.println("======================\n");

for (int i = 0; i < tacList.size(); i++)

{

System.out.println(" " + tacList.get(i));

}

System.out.println();

System.out.println(" Temporary variable count : " + tacList.size() + "\n\n");

System.out.println(" Temporary variables :");

System.out.println(" =====================\n");

for (int i = 0; i < tacList.size(); i++)

{

System.out.println(" T" + (i + 1));

}

}

public void convertToDAG()

{

Map<String, String[]> dagMap = new LinkedHashMap<String, String[]>();

String result;

String[] data;

String tempStr;

String oper;

String[] tempDAGStr = new String[3];

for (int i = 0; i < tacList.size(); i++)

{

String[] DAGtable = new String[3];

result = tacList.get(i).substring(0, tacList.get(i).indexOf("="));

tempStr = tacList.get(i).substring(tacList.get(i).indexOf("=") + 1, tacList.get(i).length());

data = tempStr.split("\\-|\\+|\\\*|\\/|\\[\\]|\\<|\\>");

Pattern pattern = Pattern.compile("\\-|\\+|\\\*|\\/|\\[\\]|\\<|\\>");

Matcher matcher = pattern.matcher(tempStr);

matcher.find();

oper = tempStr.substring(matcher.start(), matcher.start() + 1);

if (dagMap.containsKey(oper))

{

tempDAGStr = dagMap.get(oper);

if (data[0].length() == 0)

{

if (tempDAGStr[2] != null && !tempDAGStr[2].contains(data[1]))

{

tempDAGStr[2] += "," + data[1];

}

else

{

tempDAGStr[2] = data[1];

}

if (tempDAGStr[0] != null && !tempDAGStr[0].contains(data[1]))

{

tempDAGStr[0] += "," + result;

}

else

{

tempDAGStr[0] = result;

}

}

else

{

if (tempDAGStr[2] != null && !tempDAGStr[2].contains(data[1]))

{

tempDAGStr[2] += "," + data[1];

}

else

{

tempDAGStr[2] = data[1];

}

if (tempDAGStr[0] != null && !tempDAGStr[0].contains(result))

{

tempDAGStr[0] += "," + result;

}

else

{

tempDAGStr[0] = result;

}

if (tempDAGStr[1] != null && !tempDAGStr[1].contains(data[0]))

{

tempDAGStr[1] += "," + data[0];

}

else

{

tempDAGStr[1] = data[0];

}

}

dagMap.put(oper, tempDAGStr);

}

else

{

if (data[0].length() == 0)

{

DAGtable[2] = data[1];

DAGtable[0] = result;

DAGtable[1] = "";

}

else

{

DAGtable[2] = data[1];

DAGtable[0] = result;

DAGtable[1] = data[0];

}

dagMap.put(oper, DAGtable);

}

}

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

System.out.println("Label | Operator | Left Child | Right Child ");

System.out.println("\n========================================================\n");

Iterator it = dagMap.entrySet().iterator();

while (it.hasNext()) {

Map.Entry entry = (Map.Entry) it.next();

tempDAGStr = (String[]) entry.getValue();

System.out.println();

System.out.format(" %-10s%-15s%-15s%-15s", tempDAGStr[0], (String) entry.getKey(), tempDAGStr[1], tempDAGStr[2]);

}

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

}

public static void main(String[] args) {

System.out.println("\t Anand yadav Roll No.: 11");

DAG dag = new DAG();

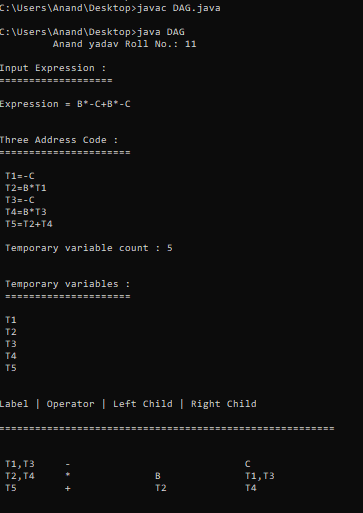
dag.printExpression();

dag.convertToTAC();

dag.convertToDAG();

}

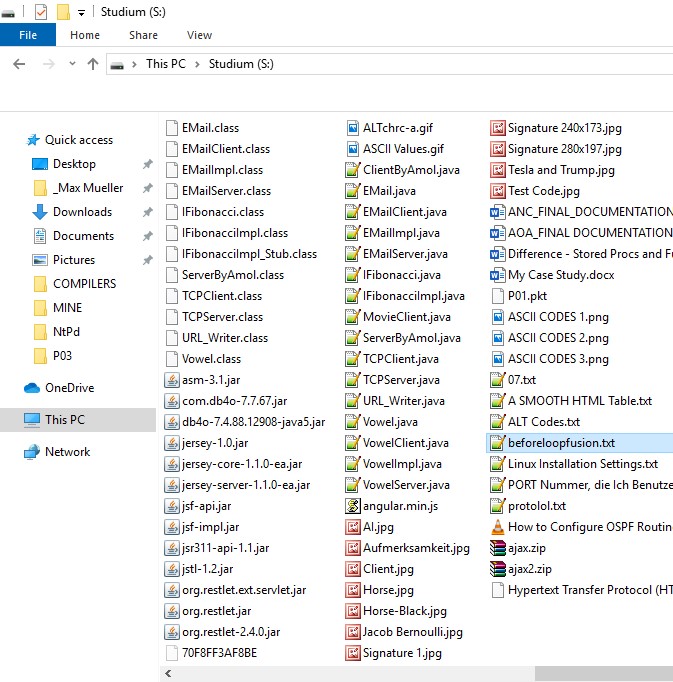
}Output:



**PRACTICAL 07**

**Aim:** Write a Program to demonstrate loop unrolling and loop jamming (fusion) for the given code sequence containing loop.

1. First save 'beforeloopfusion.txt' file in 'S' Drive



1. Beforeloopfusion.txt

int i, a[100], b[100];

for (i=0; i<100; i++) {

a[i] = 1;

}

for (i=0; i<100; i++) {

b[i] = 1;

}

***LoopUnrollingPgm.java***

package NtPd;

import java.io.\*;

public class **LoopUnrollingPgm**

{

String codeStrUnrolling = new String();

String codeStrFusion = new String();

String codeStrBeforeFusion = new String();

public void LoopUnrolling()

{

codeStrUnrolling = "int x; \n " + "for (x = 0; x < 100; x++)\n" + " {\n" + " delete(x);\n" + " }";

System.out.println("\n\n Code Before Loop Unrolling \n");

System.out.println("====================================\n");

System.out.println(codeStrUnrolling);

codeStrUnrolling = codeStrUnrolling.replace(codeStrUnrolling.substring(codeStrUnrolling.indexOf("x++"),

codeStrUnrolling.indexOf("x++") + 3), "x += 5");

String str = "\t delete(x); \r\n\t delete(x + 1); \r\n\t delete(x + 2); \r\n\t delete(x + 3); \r\n\t delete(x + 4); \r\n";

codeStrUnrolling = codeStrUnrolling.replace(codeStrUnrolling.substring(codeStrUnrolling.indexOf("delete(x);"),

codeStrUnrolling.indexOf("delete(x);") + "delete(x);".length()), str);

System.out.println("\n\n Code After Loop Unrolling \n");

System.out.println("====================================\n");

System.out.println(codeStrUnrolling);

}

public void LoopFusionCode()

{

int forCount = 0;

boolean forFound = false;

try (BufferedReader br = new BufferedReader(new FileReader("S:\\beforeloopfusion.txt")))

{

String currentLine;

while ((currentLine = br.readLine()) != null)

{

if (currentLine.contains("for"))

{

forCount++;

forFound = true;

}

if (forCount == 1 && forFound)

{

codeStrFusion += currentLine + "\r\n{\r\n";

forFound = false;

}

else if (forCount == 2 && forFound)

{

forFound = false;

}

else

{

codeStrFusion += currentLine + "\r\n";

}

codeStrBeforeFusion += currentLine + "\n";

}

codeStrFusion += "\r\n}";

}

catch (IOException e)

{

e.printStackTrace();

}

System.out.println("\n\n Code Before Loop Fusion \n");

System.out.println("==============================\n");

System.out.println(codeStrBeforeFusion);

System.out.println("\n\n Code After Loop Fusion \n");

System.out.println("=============================\n");

System.out.println(codeStrFusion);

}

public static void main(String[] args)

{

System.out.println("\t Anand yadav Roll No.: 11");

LoopUnrollingPgm loop = new LoopUnrollingPgm();

loop.LoopUnrolling();

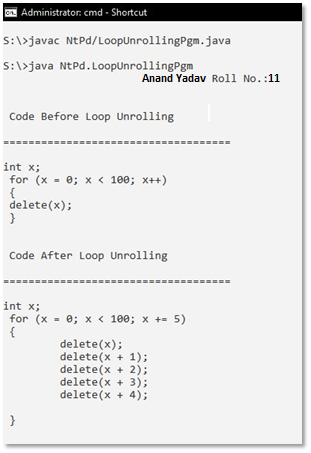
loop.LoopFusionCode();

}

}

Output:

1. Loop Unrolling



1. Loop Fusion:

