Practical No 4

Aim: Write a program to illustrate the generation on OPM for the input operator grammar

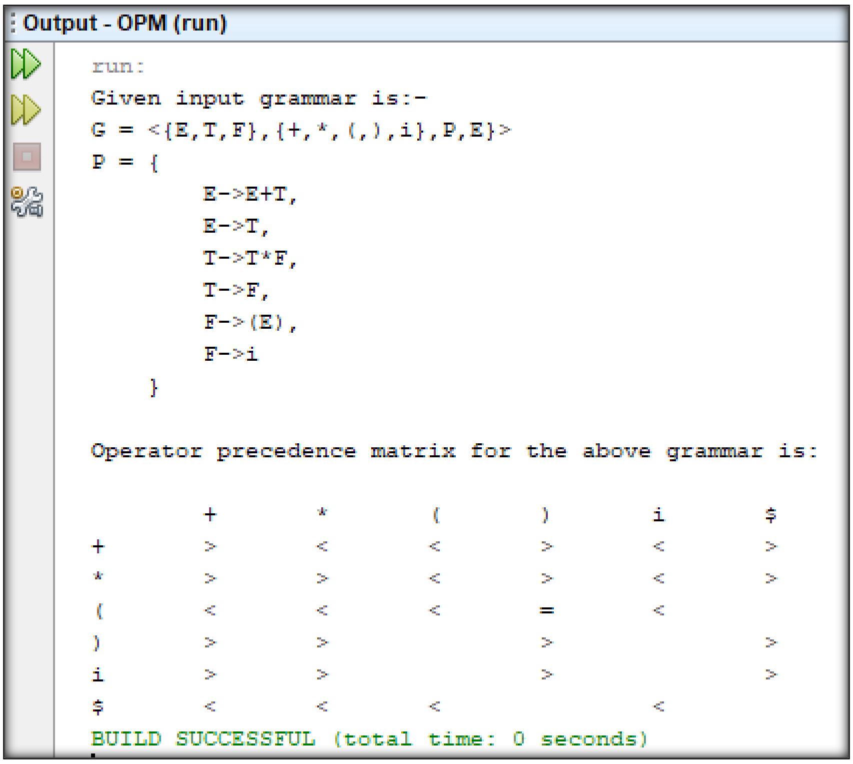
Program Code:

```
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)
{
    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();
        }

    }

}
```



```
run:
Given input grammar is:-
G = <{E,T,F},{+,*,(,),i},P,E}>
P = {
        E->E+T,
        E->T,
        T->T*F,
        T->F,
        F->(E),
        F->i
    }

Operator precedence matrix for the above grammar is:

        +       *       (       )       i       $
+       >       <       <       >       <       >
*       >       >       <       >       <       >
(       <       <       <       =       <
)       >       >               >               >
i       >       >               >               >
$       <       <       <               <
BUILD SUCCESSFUL (total time: 0 seconds)
```