```
**********************************************************************
**********************************************************************
Name :   Preksha Sheth
Roll no: 36
Subject: System Software
Class:   MCA - III
**********************************************************************
**********************************************************************
2 Pass Assembler
**********************************************************************
**********************************************************************
Pass 1:-
**********************************************************************
**********************************************************************
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.*;
import java.io.IOException;
import java.util.*;
import java.util.Map;


public class pass1
{
     public static void main(String args[])throws IOException
     {
          FileReader fr=null;
          FileWriter fw=null;
          BufferedReader br=null;
          BufferedWriter bw=null;
          try
          {
               String
inputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\Input.txt";
               fr=new FileReader(inputfilename);
               br=new BufferedReader(fr);

               String
outputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\IC.txt";
               fw=new FileWriter(outputfilename);
               bw=new BufferedWriter(fw);
               Hashtable<String,String> is=new
Hashtable<String,String>();
```

```
                is.put("STOP", "00");
                is.put("ADD", "01");
                is.put("SUB", "02");
                is.put("MULT", "03");
                is.put("MOVER", "04");
                is.put("MOVEM", "05");
                is.put("COMP", "06");
                is.put("BC", "07");
                is.put("DIV", "08");
                is.put("READ", "09");
                is.put("PRINT", "10");

                Hashtable<String,String> dl=new
Hashtable<String,String>();
                dl.put("DC","01");
                dl.put("DS","02");

                Hashtable<String,String> ad=new
Hashtable<String,String>();
                ad.put("START","01");
                ad.put("END", "02");
                ad.put("ORIGIN", "03");
                ad.put("EQU", "04");
                ad.put("LTORG", "05");

                Hashtable<String,String> cd=new
Hashtable<String,String>();
                cd.put("LT","1");
                cd.put("LE","2");
                cd.put("EQ","3");
                cd.put("GT","4");
                cd.put("GE","5");
                cd.put("ANY","6");

                Hashtable<String,String> symtab=new
Hashtable<String,String>();
                Hashtable<String,Integer> symtabi=new
Hashtable<String,Integer>();
                Hashtable<String,String> littab=new
Hashtable<String,String>();
                Hashtable<String,Integer> littabi=new
Hashtable<String,Integer>();
                ArrayList<Integer> pooltab=new
ArrayList<Integer>();

                String sCurrentLine;
                int locptr=0;
```

```
                int litin=0;
                int litptr=1;
                int symptr=1;
                int pooltabptr=1;

                sCurrentLine=br.readLine();

                String s1=sCurrentLine.split(" |   ")[1];
                if(s1.equals("START"))
                {
                     bw.write("     AD\t01 \t");
                     String s2=sCurrentLine.split(" |   ")[2];
                     bw.write("C \t"+s2+"\n");
                     locptr=Integer.parseInt(s2);
                }

                while((sCurrentLine=br.readLine())!=null)
                {
                     int LC=0,q=0;
                     String type=null;

                     int flag2=0;   //checks whether addr is
assigned to current symbol
                     String cs[]=sCurrentLine.split(" |\\,|  ");
                          String s=sCurrentLine.split(" |\\,|
     ")[0];    //consider the first word in the line
                          if(s.contains("="))
                          {
                               int in,l;
                               String sub;
                               in=s.indexOf(39);
                               in++;
                               l=s.length()-1;
                               sub=s.substring(in,l);

     bw.write("\r\n"+locptr+")\tL\t"+s);
                               locptr++;
                               continue;
                          }
                          for(Map.Entry m:symtab.entrySet())
                          {
                               if(s.equals(m.getKey()) &&
!s.equals(""))
                               {
                                    m.setValue(locptr);
                                    flag2=1;
                               }
```

```
                    }
                if (s.length() != 0 && flag2 == 0 &&
!s.equals(""))
                {         //if current string is not " " or
addr is not assigned,

            //then the current string must be a new symbol.

                    symtab.put(s, String.valueOf(locptr));
                    symtabi.put(s,symptr);
                    symptr++;
                }

                int isOpcode = 0;        //checks whether
current word is an opcode or not

                s = sCurrentLine.split(" |\\,")[1];
          //consider the second word in the line
                for (Map.Entry m : is.entrySet()) {
                    if (s.equals(m.getKey())) {
                        bw.write("\r\n"+locptr+")
     IS\t" + m.getValue() + "\t");      //if match found in
imperative stmt
                        type = "is";
                        isOpcode = 1;
                    }
                }

                int q1=0;
                for (Map.Entry m : ad.entrySet()) {
                    if (s.equals(m.getKey()) &&
!s.equals("END") && !s.equals("EQU") && !s.equals("ORIGIN")&&
!s.equals("LTORG")) {
                        bw.write("\r\n"+locptr+")
     AD\t" + m.getValue() + "\t");      //if match found in
Assembler Directive
                        type = "ad";
                        isOpcode = 1;
                    }
                    else if (s.equals(m.getKey()) &&
s.equals("END"))
                    {
                        bw.write("\r\n\tAD\t" +
m.getValue() + "\t");         //if match found in Assembler
Directive
                        type = "ad";
                        isOpcode = 1;
```

```
                    }
                    else if(s.equals(m.getKey()) &&
s.equals("EQU"))
                    {
                         bw.write("\r\n\tAD\t" +
m.getValue() + "\t");          //if match found in Assembler
Directive
                         type = "ad";
                         isOpcode = 1;
                         LC=1;
                    }
                    else if(s.equals(m.getKey()) &&
s.equals("ORIGIN"))
                    {
                         q1=1;
                         break;
                    }

               }
               for (Map.Entry m : dl.entrySet()) {
                    if (s.equals(m.getKey())) {
                         bw.write("\r\n"+locptr+")
     DL\t" + m.getValue() + "\t");      //if match found in
declarative stmt
                         type = "dl";
                         isOpcode = 1;
                    }
               }


               if (s.equals("LTORG"))
               {
                    pooltab.add(pooltabptr);
                    LC=1;
                    int llocp=locptr;
                    for (Map.Entry m : littab.entrySet()) {
                         if (m.getValue() == "") {
          //if addr is not assigned to the literal
                              m.setValue(locptr);
                              locptr++;
                              pooltabptr++;
                              LC = 1;
                              isOpcode = 1;
                         }
                    }
                    locptr=llocp;
                    bw.write("\r\n\tAD\t05\t");
```

```
                    continue;
                }


                if (s.equals("END")) {
                    pooltab.add(pooltabptr);
                    for (Map.Entry m : littab.entrySet()) {
                        if (m.getValue() == "") {
                            m.setValue(locptr);
                            locptr++;
                            LC = 1;
                        }
                    }
                }


                if(s.equals("EQU"))
                {
                    int sss=0;
                    for(Map.Entry mm:symtab.entrySet())
                    {
                        if(sCurrentLine.split("
|\\,")[3].equals(mm.getKey()) && sss==0)
                        {
                            for(Map.Entry
mmm:symtab.entrySet())
                            {
                                if(sCurrentLine.split("
|\\,")[0].equals(mmm.getKey()))
                                {

     mmm.setValue(mm.getValue());
                                    sss=1;
                                    break;
                                }
                            }
                        }
                        else if(sss==1)
                        {
                            break;
                        }
                    }
                    //symtab.put("equ",
String.valueOf(locptr));
                }
```

```
                    if (sCurrentLine.split(" |\\,").length > 2)
                    {         //if there are 3 words
                         s = sCurrentLine.split(" |\\,")[2];
                    //consider the 3rd word



                         //this is our first operand.

                         //it must be either a
Register/Declaration/Symbol
                         if (s.equals("AREG")) {
                              bw.write("1\t");
                              isOpcode = 1;
                         } else if (s.equals("BREG")) {
                              bw.write("2\t");
                              isOpcode = 1;
                         } else if (s.equals("CREG")) {
                              bw.write("3\t");
                              isOpcode = 1;
                         } else if (s.equals("DREG")) {
                              bw.write("4\t");
                              isOpcode = 1;
                         } else if (type == "dl") {
                              bw.write("C\t" + s + "\t");
                         }
                         else if(q1==1 && sCurrentLine.split("
|\\,")[1].equals("ORIGIN"))
                    {
                              LC=1;
                              String ns=null;
                              if(sCurrentLine.split("
|\\,")[2].contains("+"))
                                   {
                                        int
in=sCurrentLine.split(" |\\,")[2].indexOf('+');
                                        ns=sCurrentLine.split("
|\\,")[2].substring(0,in);
                                   }
                              for(Map.Entry m:symtab.entrySet())
                              {
                                   if(ns.equals(m.getKey()))
                                   {
                                        int nn=0,in=0;

     locptr=Integer.parseInt((String)m.getValue());
```

```
                                        in=sCurrentLine.split("
|\\,")[2].indexOf('+');
                                        in=in+1;
                                        ns=sCurrentLine.split("
|\\,")[2].substring(in);

     nn=Integer.parseInt((String)ns);
                                        locptr=locptr+nn;

     System.out.println("LOOP: "+locptr);
                                             break;
                                        }
                                   }
                              }
                              else
                              {
                                   int qqq=0;
                                   for(Map.Entry m:cd.entrySet())
                                   {
                                        if(s.equals(m.getKey()) &&
qqq==0)
                                        {

     bw.write(m.getValue()+"\t");
                                             qqq=1;
                                             break;
                                        }
                                   }
                                   if(qqq==0)
                                   {
                                        symtab.put(s, "");
                                   }
     //forward referenced symbol
                              }
                         }


                         if (sCurrentLine.split(" |\\,").length > 3)
                         {         //if there are 4 words

                              s = sCurrentLine.split(" |\\,")[3];
                    //consider 4th word.

                              //this is our 2nd operand

                              //it is either a literal, or a symbol
                              if (s.contains("=")) {
```

```
                                litin++;
                                littabi.put(s,litin);
                                littab.put(s, "");
                                bw.write("L\t" + litptr + "\t");
                                isOpcode = 1;
                                litptr++;
                           }
                           else
                           {
                                     int sq=0,qq=0;
                                     for(Map.Entry
m:symtab.entrySet())
                                     {
                                          sq++;
                                          if(s.equals(m.getKey())
&& qq!=1)
                                          {
                                               for (Map.Entry
mm:symtabi.entrySet())
                                               {

     if(s.equals(mm.getKey()))
                                                    {

     bw.write("S\t" +mm.getValue()+ "\t");
                                                         qq=1;
                                                         break;
                                                    }
                                               }

                                          }
                                          else if(qq==1)
                                          {
                                               break;
                                          }
                                     }
                                if(qq==0)
                                {
                                     symtab.put(s, "");
     //Doubt : what if the current symbol is already present in
SYMTAB?

     //Overwrite?
                                     bw.write("S\t" + symptr +
"\t");
                                     symtabi.put(s,symptr);
```

```
                                        symptr++;
                                   }
                              }
                         }

                         bw.write("\n");          //done with a line.

                         if (LC == 0)
                              locptr++;
                    }

                    String f1 = "C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\SYMTAB.txt";
                    FileWriter fw1 = new FileWriter(f1);
                    BufferedWriter bw1 = new BufferedWriter(fw1);
                    for (Map.Entry m : symtab.entrySet())
                    {
                         for(Map.Entry mm:symtabi.entrySet())
                         {
                              String str=(String)mm.getKey();
                              if(str.equals(m.getKey()))
                              {

     bw1.write(mm.getValue()+"\t"+m.getKey() + "\t" +
m.getValue()+"\r\n");
                                   System.out.println(m.getKey() + "
" + m.getValue());
                              }
                         }
                    }

                    String f2 = "C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\LITTAB.txt";
                    FileWriter fw2 = new FileWriter(f2);
                    BufferedWriter bw2 = new BufferedWriter(fw2);
                    for (Map.Entry m : littab.entrySet())
                    {
                         for(Map.Entry mm:littabi.entrySet())
                         {
                              System.out.println("MM:
"+mm.getValue());
                              String str=(String)mm.getKey();
                              if(str.equals(m.getKey()))
                              {

     bw2.write(mm.getValue()+"\t"+m.getKey() + "\t" +
m.getValue()+"\r\n");
```

```
                                    System.out.println(mm.getValue()+"
"+m.getKey() + " " + m.getValue());

                              }
                         }
                    }

                    String f3 = "C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\POOLTAB.txt";
                    FileWriter fw3 = new FileWriter(f3);
                    BufferedWriter bw3 = new BufferedWriter(fw3);
                    for (Integer item : pooltab) {
                         bw3.write(item+"\r\n");
                        System.out.println(item);
                    }

                    bw.close();
                    bw1.close();
                    bw2.close();
                    bw3.close();

               }
               catch (Exception e)
               {
                    e.printStackTrace();
               }
          }
}
```

****

****

OUTPUT

****

****

```
D:\MCA-III\SS File>javac pass1.java
D:\MCA-III\SS File>java pass1
TWO 213
A 214
ONE 212
INPUT 200
LOOP 201
BACK 201
B 215
MM: 1
1 ='2' 208
1
2
```

```
**********************************************************************
**********************************************************************
Input file:- Input.txt
**********************************************************************
**********************************************************************
 START 200
INPUT READ  A
LOOP MOVER AREG,A
 MOVEM BREG,A
 ADD AREG,ONE
 ADD BREG,TWO
 COMP BREG,='2'
 BC LT,LOOP
 BC GT,BACK
 LTORG
='2'
BACK EQU  LOOP
 MOVEM BREG,B
 PRINT  B
 STOP
ONE DC '1'
TWO DC '2'
A DS 1
B DS 1
 END
**********************************************************************
**********************************************************************
Intermediate Code:- IC.txt
**********************************************************************
**********************************************************************
     AD   01   C    200

200) IS   09   S    2

201) IS   04   1    S    2

202) IS   05   2    S    2

203) IS   01   1    S    4

204) IS   01   2    S    5

205) IS   06   2    L    1

206) IS   07   1    S    3

207) IS   07   4    S    6
```

```
     AD   05
208) L    ='2'
     AD   04   S    3

209) IS   05   2    S    7

210) IS   10   S    7

211) IS   00

212) DL   01   C    '1'

213) DL   01   C    '2'

214) DL   02   C    1

215) DL   02   C    1

     AD   02
```

****
****

LITTAB:- LITTAB.txt

****
****

```
1    ='2' 208
```

****
****

POOLTAB:- POOLTAB.txt

****
****

```
1
2
```

****
****

SYMTAB:- SYMTAB.txt

****
****

```
5    TWO  213
2    A    214
4    ONE  212
1    INPUT     200
3    LOOP 201
6    BACK 201
7    B    215
```

```
**********************************************************************
**********************************************************************
Pass 2:-
**********************************************************************
**********************************************************************
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.*;
import java.io.IOException;
import java.util.*;
import java.util.Map;


public class pass2
{
     public static void main(String args[])throws IOException
     {
          FileReader fr=null;
          FileWriter fw=null;

          BufferedReader br=null;
          BufferedWriter bw=null;

          try
          {
               String
inputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\IC.txt";

               String
outputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\MC.txt";

               fr=new FileReader(inputfilename);
               fw=new FileWriter(outputfilename);
               br=new BufferedReader(fr);
               bw=new BufferedWriter(fw);

               if(br==null)
               {
                    System.out.println("NULL!");
               }

               String sCurrentLine=br.readLine();
               while((sCurrentLine=br.readLine())!=null)
```

```
                {
            if(sCurrentLine.trim().isEmpty())
            {
                 continue;
            }
                        String lc=null;
                        if(sCurrentLine.split(" |
      ")[1].equals("AD"))
                        {
                             continue;
                        }
                        lc=sCurrentLine.split(" |     ")[0];
                        if(sCurrentLine.split(" |
      ")[1].equals("L"))
                        {
                                  String s=sCurrentLine.split(" |
      ")[2];
                                  if(s.contains("="))
                                  {
                                       int in,l;
                                       String sub;
                                       in=s.indexOf(39);
                                       in++;
                                       l=s.length()-1;
                                       sub=s.substring(in,l);

      bw.write("\r\n"+lc+"\t+00\t0\t");

      if(Integer.parseInt((String)sub)<10)
                                       {
                                            bw.write("00"+sub);
                                       }
                                       else
if(Integer.parseInt((String)sub)>=10 &&
Integer.parseInt((String)sub)<100)
                                       {
                                            bw.write("0"+sub);
                                       }
                                       else
                                       {
                                            bw.write(sub);
                                       }

      //bw.write("\r\n"+locptr+")\t"+sub);
                                       continue;
                                  }
                        }
```

```
                    if(sCurrentLine.split(" |
     ")[1].equals("DL"))
                    {
                         if(sCurrentLine.split(" |
     ")[2].equals("01"))
                         {

                              bw.write("\r\n"+lc+"\t+00\t0\t");
                              String s=sCurrentLine.split(" |
     ")[4];
                              int in=s.indexOf(39);
                              in++;
                              int l=s.length();
                              l--;
                              s=s.substring(in,l);
                              if(Integer.parseInt((String)s)<10)
                              {
                                   bw.write("00"+s);
                              }
                              else
if(Integer.parseInt((String)s)>=10 &&
(Integer.parseInt((String)s)<100))
                              {
                                   bw.write("0"+s);

                              }
                              else
                              {
                                   bw.write(s);

                              }
                         }
                         else
                         {

                         bw.write("\r\n"+lc+"\t+00\t0\t000");
                         }
                         continue;
                    }
                    if(sCurrentLine.split(" |
     ")[1].equals("IS") && sCurrentLine.split(" |
     ")[2].equals("00"))
                    {
                         bw.write("\r\n"+lc+"\t+00\t0\t000");
                         continue;
                    }
                    bw.write("\r\n"+lc);
```

```
                    String s=sCurrentLine.split(" |    ")[2];
                    bw.write("\t+"+s);
                    if(sCurrentLine.split(" |     ").length>5)
                    {
                         s=sCurrentLine.split(" | ")[3];
                         bw.write("\t"+s);
                         String s1=sCurrentLine.split(" |
     ")[4];
                         String s2=null;
                         s2=sCurrentLine.split(" |     ")[5];
                         if(s1.equals("S"))
                         {
                              int cc=0;
                              BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\SYMTAB.txt"));
                              String sline=null;
                              int qr=0;
                              while((sline=br1.readLine())!=null
&& qr==0)
                              {
                                   if(sline.trim().isEmpty())
                                   {
                                        continue;
                                   }
                                   if(s2.equals(sline.split(" |
     ")[0]))
                                   {
                                        s2=sline.split(" |
     ")[2];
                                        bw.write("\t"+s2);
                                        qr=1;
                                        break;
                                   }
                              }
                              br1.close();
                         }
                         else if(s1.equals("L"))
                         {
                              int cc=0;
                              BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\LITTAB.txt"));
                              String sline=null;
                              int qr=0;
                              while((sline=br1.readLine())!=null
&& qr==0)
```

```
                                {
                                     String str=sline.split(" |
     ")[0];
                                     if(s2.equals(str))
                                     {
                                          str=sline.split(" |
     ")[2];
                                          bw.write("\t"+str);
                                          qr=1;
                                          break;
                                     }
                                }
                                br1.close();
                           }
                      }
                      else if(sCurrentLine.split(" |
     ").length==5)
                      {
                           bw.write("\t0");
                           String s1=sCurrentLine.split(" |
     ")[3];
                           String s2=sCurrentLine.split(" |
     ")[4];
                           if(s1.equals("S"))
                           {
                                int cc=0;
                                BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\SYMTAB.txt"));
                                String sline=null;
                                int qr=0;
                                while((sline=br1.readLine())!=null
&& qr==0)
                                {
                                     if(s2.equals(sline.split(" |
     ")[0]))
                                     {
                                          s2=sline.split(" |
     ")[2];
                                          bw.write("\t"+s2);
                                          qr=1;
                                          break;
                                     }
                                }
                                br1.close();
                           }
                           else if(s1.equals("L"))
```

```
                         {
                              int cc=0;
                              BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\LITTAB.txt"));
                              String sline=null;
                              int qr=0;
                              while((sline=br1.readLine())!=null
&& qr==0)
                              {
                                   int in=sline.split(" |
     ")[0].indexOf(39);
                                   in=in+1;
                                   int l=sline.split(" |
     ")[0].length();
                                   l=l-1;
                                   String str=sline.split(" |
     ")[0].substring(in,l);
                                   if(s2.equals(str))
                                   {
                                        bw.write("\t00"+str);
                                        qr=1;
                                        break;
                                   }
                              }
                              br1.close();
                         }
                    }
               }
               br.close();
               bw.close();
          }
          catch(Exception e)
          {
               e.printStackTrace();
          }
     }
}
```

****

****

OUTPUT

****

****

```
D:\MCA-III\SS File>javac pass2.java

D:\MCA-III\SS File>java pass2
```

```
**********************************************************************
**********************************************************************
Machine Code:- MC.txt
**********************************************************************
**********************************************************************

200) +09  0    214
201) +04  1    214
202) +05  2    214
203) +01  1    212
204) +01  2    213
205) +06  2    208
206) +07  1    201
207) +07  4    201
208) +00  0    002
209) +05  2    215
210) +10  0    215
211) +00  0    000
212) +00  0    001
213) +00  0    002
214) +00  0    000
215) +00  0    000
**********************************************************************
**********************************************************************
Macro:-
**********************************************************************
**********************************************************************
Pass - 1:-
**********************************************************************
**********************************************************************
import java.util.*;
import java.io.*;

class Macro_Expansion{
     public static String MDT_check(String
word,ArrayList<String> SSNTAB,ArrayList<String>
EVT,ArrayList<String> PNT,String mdt_word,int
count,ArrayList<String> MODEL)
     {
         if(SSNTAB.contains(word))
               mdt_word=mdt_word+"(S ,"
+(SSNTAB.indexOf(word)+1)+")";
          else if(PNT.contains(word.substring(1,word.length())))
               mdt_word=mdt_word+"(P ,"
+(PNT.indexOf(word.substring(1,word.length()))+1)+")";
          else if(EVT.contains(word.substring(1,word.length())))
```

```
                mdt_word=mdt_word+"(E ,"
+(EVT.indexOf(word.substring(1,word.length()))+1)+")";
          else
          {
              if(count==1 && !MODEL.contains(word)){}
               else
               mdt_word=mdt_word+" "+word+" ";

          }
          return mdt_word;
     }
     public static void main(String args[])throws IOException{
          File f1=new File("Macro.txt");
          BufferedReader br_macro=new BufferedReader(new
FileReader(f1));

          String
line=null,word,mdt_word=null,prev_word=null,line1;

          int
count=0,pos_count=0,key_count=0,ev_count=0,n,count_token=0,ssnta
b_count=0,line_no=1,mdt_ptr=1;

          ArrayList<String> MNT=new ArrayList<String>();
          ArrayList<String> PNT=new ArrayList<String>();
          ArrayList<String> KPD=new ArrayList<String>();
          ArrayList<String> EVT=new ArrayList<String>();
          ArrayList<String> MDT=new ArrayList<String>();
          ArrayList<String> Model=new ArrayList<String>();
          ArrayList<String> SSNTAB=new ArrayList<String>();
          ArrayList<Integer> SSTAB=new ArrayList<Integer>();

          File f2=new File("Models.txt");
          BufferedReader br=new BufferedReader(new
FileReader(f2));

          while((line1=br.readLine())!=null)
          {
               Model.add(line1);
          }
          while((line=br_macro.readLine())!=null)
          {
               count_token=1;
               mdt_word=" ";
               StringTokenizer st1= new StringTokenizer(line);
               n=st1.countTokens();
               while(st1.hasMoreTokens())
```

```
                {
                     word=st1.nextToken();
                     if(word.equals("MACRO"))
                     {
                          count++;
                          continue;
                     }
                     if(count==1)
                     {
                          MNT.add(word);
                          while(count<n)
                          {
                               word=st1.nextToken();
                              if(word.contains("="))
                               {
                                  key_count++;
                                  String[] str=word.split("=",2);
                                  String key_par;

key_par=str[0].substring(1,str[0].length());
                                  PNT.add(key_par);
                                  KPD.add(key_par);
                                  KPD.add(str[1]);
                               }
                              else
                               {
                                    String pos_par;

     pos_par=word.substring(1,word.length()-1);
                                    PNT.add(pos_par);
                                    pos_count++;
                               }
                             count++;
                          }
                     }
                     if(word.equals("LCL"))
                     {
                          mdt_word=mdt_word+word+" ";
                          word=st1.nextToken();

     EVT.add(word.substring(1,word.length()));
                          ev_count++;
                          mdt_word=mdt_word+"(E ,"+ev_count+")";
                     }
                     if(count_token==2 && Model.contains(word))
                     {
                          ssntab_count++;
```

```
                         SSNTAB.add(prev_word);
                    }
                    if(count_token==2 &&
SSNTAB.contains(prev_word))
                    {
                         int q=SSNTAB.indexOf(prev_word);
                         SSTAB.add(mdt_ptr);
                    }
                    if(line_no>3 && !word.equals("LCL"))
                    {

     mdt_word=MDT_check(word,SSNTAB,EVT,PNT,mdt_word,count_token
,Model);
                    }
                    count_token++;
                    prev_word=word;
               }
               if(!mdt_word.equals(" "))
               {
               MDT.add(mdt_word);
               mdt_ptr++;
               }
               line_no++;
          }
          MNT.add(Integer.toString(pos_count));
          MNT.add(Integer.toString(key_count));
          MNT.add(Integer.toString(ev_count));
          MNT.add("1");
          MNT.add("1");
          MNT.add("1");

         System.out.println("\nMNT table: "+MNT);
          System.out.println("\nPNT table: "+PNT);
          System.out.println("\nKPD table: "+KPD);
          System.out.println("\nEV table: "+EVT);
          System.out.println("\nSSTAB table: "+SSTAB);
          System.out.println("\nSSNTAB table: "+SSNTAB);
          for(String s:MDT)
          {
               System.out.println(s);
          }
     }
}
```

****

****

OUTPUT

```
**********************************************************************
**********************************************************************
D:\MCA-III\SS File>javac Macro_Expansion.java

D:\MCA-III\SS File>java Macro_Expansion

MNT table: [CLEARMEM, 2, 1, 1, 1, 1, 1]

PNT table: [X, N, REG]

KPD table: [REG, AREG]

EV table: [M]

SSTAB table: [4]

SSNTAB table: [MORE]
 LCL (E ,1)
 (E ,1) SET  0
  MOVER (P ,3) ,  ='0'
  MOVEM (P ,3) , (P ,1) + (E ,1)
 (E ,1) SET (E ,1) +  1
  AIF  ( (E ,1) NE (P ,2) ) (S ,1)

D:\MCA-III\SS File>
**********************************************************************
**********************************************************************
Macro.txt
**********************************************************************
**********************************************************************
     MACRO
     CLEARMEM &X, &N, &REG=AREG
     LCL &M
&M   SET 0
     MOVER &REG , ='0'
MORE MOVEM &REG , &X + &M
&M   SET &M + 1
     AIF ( &M NE &N ) MORE
     MEND
**********************************************************************
**********************************************************************
Models.txt
**********************************************************************
**********************************************************************
ADD
SUB
MUL
```

```
MOVER
MOVEM
COMP
BC
DIV
READ
PRINT
AIF
AGO
**********************************************************************
**********************************************************************
Scanner:-
**********************************************************************
**********************************************************************
import java.util.*;
import java.io.*;
class DFA1{
    static String table[][];
    public static void main(String args[]) throws Exception{
        Scanner sc=new Scanner(System.in);
        System.out.println("Regular Expression :
aa+(c|d)*b(bd)*");
        System.out.println("Enter Expression : ");
        String expr=sc.nextLine();
        getRules();
        eval(expr);
    }
    static void eval(String expr){
        String curr_state=table[1][0],next,curr_char;
        for(int i=0;i<expr.length();i++){
            curr_char=expr.charAt(i)+"";
            next=getnext(curr_state,curr_char);
            if(next == null || next.equals("-")){
                System.out.println("Invalid Expression");
            }
            curr_state=next;
        }
        if(curr_state.equals("D")){
            System.out.println("Valid Expression");
        }
        else{
            System.out.println("Invalid Expression");
        }
    }
    static String getnext(String curr_state,String curr_char){
        int row=0,col=0;
        for(int i=0;i<table.length;i++){
```

```
            if(curr_state.equals(table[i][0])){
                row=i;
                break;
            }
        }
        for(int i=0;i<table[0].length;i++){
            if(curr_char.equals(table[0][i])){
                col=i;
                break;
            }
        }
        return table[row][col];
    }
    static void getRules() throws Exception{
        table=new String[6][5];
        BufferedReader br=new BufferedReader(new
FileReader("states.txt"));
        String line;
        int i=0,j;
        String token[];
        while((line=br.readLine())!=null){
            token=line.split("\t");
            for(j=0;j<token.length;j++){
                table[i][j]=token[j];
            }
            i++;
        }
        System.out.println("State Table : ");
        for(i=0;i<table.length;i++){
            for(j=0;j<table[i].length;j++){
                System.out.print(table[i][j]+"\t");
            }
            System.out.println("");
        }
    }
}
```

states.txt

```
s    a    b    c    d
A    B    -    -    -
B    C    -    -    -
C    C    D    C    C
```

```
D    -    E    -    -
E    -    -    -    D

**********************************************************************
**********************************************************************
 OUTPUT
**********************************************************************
**********************************************************************

D:\MCA-III\SS File>java DFA1
Regular Expression : aa+(c|d)*b(bd)*
Enter Expression :
aacbbd
State Table :
s       a       b       c       d
A       B       -       -       -
B       C       -       -       -
C       C       D       C       C
D       -       E       -       -
E       -       -       -       D
Valid Expression

**********************************************************************
**********************************************************************
RECURSIVE DESCENT PARSER:-
**********************************************************************
**********************************************************************
import java.io.*;
import java.util.*;

class TreeNode{
     String val;
     TreeNode leftChild,rightChild;

     TreeNode(){
          val="";
          leftChild=rightChild=null;
     }
     TreeNode(String val){
          this.val=val;
          leftChild=rightChild=null;
     }
     TreeNode(String val,TreeNode lchild,TreeNode rhild){
          this.val=val;
          leftChild=lchild;
          rightChild=rhild;
     }
```

```
     void dispPostOrder(){
          if(leftChild!=null)
               leftChild.dispPostOrder();
          if(rightChild!=null)
               rightChild.dispPostOrder();
          System.out.print(val);
     }
}

class RecDecParser{
     static int SSM=0;
     static String expression;

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

     System.out.print("_____\nEnter
Expression : ");
          expression=sc.next();

     System.out.println("\n_____");
          TreeNode root=new TreeNode();

          root=Proc_E(expression);
          if(SSM != expression.length() || root==null )
               System.out.println("!!! INVALID EXPRESSION !!!");
          else{
               System.out.println("\nPostfix Expression
:\n_____");
               root.dispPostOrder();
          }

     System.out.println("\n_____");
     }

     static TreeNode Proc_E(String expression){
          TreeNode left,right;
          left=Proc_T(expression);
          while(SSM < expression.length() &&
expression.charAt(SSM)=='+' ){
               SSM++;
               right=Proc_T(expression);
               if(right==null)
                    return null;
               else
                    left=new TreeNode("+",left,right);
          }
```

```
          return left;
     }

     static TreeNode Proc_T(String expression){
          TreeNode left,right;
          left=Proc_V(expression);
          while(SSM < expression.length() &&
expression.charAt(SSM)=='*'){
               SSM++;
               right=Proc_V(expression);
               if(right==null)
                    return null;
               else
                    left=new TreeNode("*",left,right);
          }
          return left;
     }

     static TreeNode Proc_V(String expression){
          TreeNode node;
          if(SSM < expression.length() &&
expression.charAt(SSM)!='*' && expression.charAt(SSM)!='+'){
               node=new
TreeNode(expression.charAt(SSM++)+"",null,null);
               return node;
          }
          System.out.println("!!! INVALID EXPRESSION !!!");
          return null;
     }
}
```

****

****

OUTPUT

****

****

```
D:\MCA-III\SS File>JAVAC RecDecParser.java

D:\MCA-III\SS File>JAVA RecDecParser
_____________________________
Enter Expression : a+b*c


_____________________________

Postfix Expression :
_____________________________
abc*+
_____________________________
```

```
D:\MCA-III\SS File>JAVAC RecDecParser.java

D:\MCA-III\SS File>JAVA RecDecParser
_____________________________
Enter Expression : ab+c


_____________________________
!!! INVALID EXPRESSION !!!


_____________________________

**********************************************************************
**********************************************************************
LL(1) PARSER:-
**********************************************************************
**********************************************************************
import java.io.*;
import java.util.*;

class myLL1{
     public static void main(String args[])throws Exception{
          LL1Parser obj=new LL1Parser();
          obj.initTable();
          obj.showTable();
          obj.getExpression();
          obj.parseExpression();
     }
}
class LL1Parser{
     String expression;
     String ruleArray[][]=new String[6][5];

     void getExpression(){
          Scanner sc=new Scanner(System.in);

     System.out.print("\n_____________________________________________
_____\nEnter Expression : ");
          expression=sc.next();

     System.out.println("\n_____________________________________________
_______");
          expression=expression+"|";
     }

     void initTable()throws Exception{
```

```
          BufferedReader br=new BufferedReader(new
FileReader("ruleTable.txt"));
          String arr[],line=br.readLine();
          int i=0;
          while(line!=null){
               arr=line.split("\t");
               for(int j=0;j<arr.length;j++)
                    ruleArray[i][j]=arr[j];
               i++;
               line=br.readLine();
          }
          br.close();
     }

     void showTable(){
          System.out.println("\n_____________RULE
TABLE_____________________ ");
          for(int i=0;i<ruleArray.length;i++){
               for(int j=0;j<ruleArray[i].length;j++)
                    System.out.print( ruleArray[i][j]+"\t" );
               System.out.print("\n");
          }
     }

     void parseExpression(){
          int SSM=0;
          String csf="",newRule;
          if(expression.charAt(SSM)=='a')
          {

     newRule=getNextRule(ruleArray[1][0],expression.charAt(0)+""
);
               csf=newRule+csf;
               System.out.println("\nCSF \t\t Symbol \t
Prediction
\n_____________________________________________\n");

               while(SSM<expression.length()){
                    System.out.println(csf+" \t\t
"+expression.charAt(SSM) +" \t\t "+newRule);

     newRule=getNextRule(csf.charAt(0)+"",expression.charAt(SSM)
+"");

                    if(newRule==null){
                         System.out.println("!!! INVALID
EXPRESSION !!!");
```

```
                    return;
                }

                csf=new
StringBuilder(csf).deleteCharAt(0).toString();
                csf=newRule+csf;


     if((csf.charAt(0)+"").equals(expression.charAt(SSM)+"")){
                    csf=new
StringBuilder(csf).deleteCharAt(0).toString();
                    SSM++;
                }

                if(newRule.equals("e"))
                    csf=new
StringBuilder(csf).deleteCharAt(0).toString();

                if(csf.equals("")){

     System.out.println("\n_____________________\n EXPRESSION
IS VALID \n_____________________");
                    return;
                }
            }
            System.out.println("!!! INVALID EXPRESSION !!!");
        }
        else{
            System.out.println("!!! INVALID EXPRESSION !!!");
        }
    }
    String getNextRule(String r,String c){
        boolean row=false,col=false;
        int i,j;
        for(i=1;i<ruleArray.length;i++){
            if(ruleArray[i][0].equals(r)){
                row=true;
                break;
            }
        }
        for(j=1;j<ruleArray[0].length;j++){
            if(ruleArray[0][j].equals(c)){
                col=true;
                break;
            }
        }
        if(row && col)
```

```
                return ruleArray[i][j];
           else
                return null;
      }
}
```

```
**********************************************************************
**********************************************************************
 OUTPUT
**********************************************************************
**********************************************************************
D:\MCA-III\SS File>javac myLL1.java

D:\MCA-III\SS File>java myLL1

_____________RULE TABLE____________________
NT      a       +       *       |
E       TF      -       -       -
F       -       +TF     -       e
T       VU      -       -       -
U       -       e       *VU     e
V       a       -       -       -

_____________________________________________
Enter Expression : a+a*a

_____________________________________________

CSF               Symbol          Prediction
_____________________________________________________

TF                a               TF
VUF               a               VU
UF                +               a
F                 +               e
TF                a               +TF
VUF               a               VU
UF                *               a
VUF               a               *VU
UF                |               a
F                 |               e

_____________________________
 EXPRESSION IS VALID
_____________________________

D:\MCA-III\SS File>java myLL1
```

```
_____________RULE TABLE_____________________
NT      a       +       *       |
E       TF      -       -       -
F       -       +TF     -       e
T       VU      -       -       -
U       -       e       *VU     e
V       a       -       -       -

_____________________________________________
Enter Expression : a+a*

_____________________________________________

CSF             Symbol          Prediction
_____________________________________________

TF              a               TF
VUF             a               VU
UF              +               a
F               +               e
TF              a               +TF
VUF             a               VU
UF              *               a
VUF             |               *VU
-UF             |               -
!!! INVALID EXPRESSION !!!

D:\MCA-III\SS File>
```

```
**************************************************************
**************************************************************
OPERATOR PRECEDENCE PARSER:-
**************************************************************
**************************************************************
import java.io.*;
import java.util.*;
class TreeNode{
     String value;
     TreeNode leftChild,rightChild;
     TreeNode(){
          value="";
          leftChild=null;
          rightChild=null;
     }
     TreeNode(String val){
          value=val;
          leftChild=null;
          rightChild=null;
```

```
	}
	TreeNode(String val,TreeNode lchild,TreeNode rchild){
		value=val;
		leftChild=lchild;
		rightChild=rchild;
	}
	void postOrder(){
		if(leftChild!=null)
			leftChild.postOrder();
		if(rightChild!=null)
			rightChild.postOrder();
		System.out.print(value);
	}
}
class Stack{
	String operator;
	TreeNode operand;
	static int TOS =-1;
	void push(String operator,TreeNode operand){
		this.operand=operand;
		this.operator=operator;
	}
}
class OpePreParser{
	static String precedenceTable[][];
	static TreeNode rootNode;

	public static void main(String args[]) throws Exception{
		fillTable();
		showPrecTable();
		Scanner scanner=new Scanner(System.in);

	System.out.println("_____________________________________");
		System.out.print("Enter Expression : ");
		String expression=scanner.next();

	System.out.println("_____________________________________");
		expression="|"+expression+"|";

		int ssm=0,row=0,col=0;
		Stack stack[]=new Stack[10];
		String operator=expression.charAt(ssm++)+"";
		String operand=expression.charAt(ssm)+"";

		TreeNode node=new TreeNode(operand);
		stack[++Stack.TOS]=new Stack();
		stack[Stack.TOS].push(operator,node);
```

```
          String str;
          boolean error=false;

          while(expression.charAt(ssm)!='|'){
               ssm++;
               row=getRow(stack[Stack.TOS].operator);
               col=getColumn(expression.charAt(ssm)+"");
               if(row != -1 && col != -1)
                    str=precedenceTable[row][col];
               else{

     System.out.println("_____________________________________");
                    System.out.println("!!! INVALID EXPRESSION
!!!");

     System.out.println("_____________________________________");
                    error=true;
                    break;
               }
               if(str.equals("<")){
                    operator=expression.charAt(ssm++)+"";
                    operand=expression.charAt(ssm)+"";
                    node=new TreeNode(operand);
                    stack[++Stack.TOS]=new Stack();
                    stack[Stack.TOS].push(operator,node);
               }
               else if(str.equals(">")){
                    pop(stack);
                    ssm--;
               }
          }
          if(!error){
               rootNode=stack[Stack.TOS].operand;

     System.out.println("_____________________________________");
               System.out.print(">>Post Order Expression : ");
               rootNode.postOrder();

     System.out.println("\n_____________________________________");
          }
     }
     static void pop(Stack stack[]){
          TreeNode node1=stack[Stack.TOS].operand;
          String op1=stack[Stack.TOS--].operator;
          TreeNode node2=stack[Stack.TOS].operand;
          String op2=stack[Stack.TOS--].operator;
```

```
          TreeNode node=new TreeNode(op1,node2,node1);
          stack[++Stack.TOS].push(op2,node);
     }
     static void fillTable() throws Exception{
          BufferedReader bufRead=new BufferedReader(new
FileReader("OpePreTable.txt"));
          precedenceTable=new String[6][6];
          String arr[],line=bufRead.readLine();
          int i=0;
          while(line!=null){
               arr=line.split("\t");
               for(int j=0;j<arr.length;j++)
                    precedenceTable[i][j]=arr[j];
               i++;
               line=bufRead.readLine();
          }
          bufRead.close();
     }
     static void showPrecTable(){

     System.out.println("\n_____________________Precedence
Table_____________________________");
          for(int i=0;i<precedenceTable.length;i++){
               for(int j=0;j<precedenceTable[i].length;j++)

     System.out.print("\t"+precedenceTable[i][j]);
               System.out.println("");
          }
     }
     static int getRow(String str){
          for(int i=0;i<precedenceTable.length;i++)
               if(precedenceTable[i][0].equals(str))
                    return i;
          return -1;
     }
     static int getColumn(String str){
          for(int i=0;i<precedenceTable[0].length;i++)
               if(precedenceTable[0][i].equals(str))
                    return i;
          return -1;
     }
}
**************************************************************
**************************************************************
OUTPUT
**************************************************************
**************************************************************
```

```
D:\MCA-III\SS File>javac OpePreParser.java

D:\MCA-III\SS File>java OpePreParser

_____________________Precedence
Table_____________________________
        op      +       *       (       )       |
        +       >       <       <       >       >
        *       >       >       <       >       >
        (       <       <       <       =       $
        )       >       >       $       >       >
        |       <       <       <       $       =

_____________________________________
Enter Expression : a+b*c

_____________________________________
_____________________________________
>>Post Order Expression : abc*+

_____________________________________

D:\MCA-III\SS File>java OpePreParser

_____________________Precedence Table_____________________________
       op      +       *       (       )       |
       +       >       <       <       >       >
       *       >       >       <       >       >
       (       <       <       <       =       $
       )       >       >       $       >       >
       |       <       <       <       $       =

_____________________________________
Enter Expression : a*bc

_____________________________________
_____________________________________
!!! INVALID EXPRESSION !!!

_____________________________________

D:\MCA-III\SS File>
**********************************************************************
**********************************************************************
```