```java
//Kevin Lee
//CS311 Summer 2014
//Note: fsa.txt file must have spaces between each state & parenthesis in the transitions
//example: ( 0 0 0 ) is accepted. (0 0 0) is not.
//Each of the 8 machines are separated by a "~" in fsa.txt
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

public class FSA
{
   static boolean exit, symbolAccepted, alternate;
   static int position = 0, currentFSA = 1, numStates;
   static String[] alphArray, stringToTest, transArray, alphabet, capAlphabet, numbers;
   static ArrayList<String>[][] states;
   static String[][] dfsa;
   static ArrayList<ArrayList<String>> subsetStates;
   static ArrayList<String> finalStates, subset, subsetFinalStates;
   static String result, symbol, dead_state;
   static String initial_state = "0";

//////////////////////////////////////////////////////////////////////////////
   public static void UFSA()
   {
      String state = initial_state; dead_state = "-";
      exit = false; position = 0;

      while(exit == false)
      {
         //special case: if input file contains "a-z" "A-Z" "0-9" or "_"
         for(int i = 0; i < alphArray.length; i++)
            if( alphArray[i].equals("a-z") || alphArray[i].equals("A-Z") ||
               alphArray[i].equals("_"))
            {
               symbolAccepted = true;
               alternate = true;
            }

         symbol = nextSymbol();

         //check if the incoming symbol is in the alphabet
         for(int i = 0; i < alphArray.length; i++)
            if( symbol.equalsIgnoreCase(alphArray[i]) ||
               symbol.equals(Integer.toString(i)) )
               symbolAccepted = true;

            if(symbolAccepted == true && position < stringToTest.length)
               state = nextState(state, symbol);
            else
            {
               exit = true;
               for(int i = 0; i < finalStates.size(); i++)
               {
                  if(state.equals(finalStates.get(i)))
                  {
                     //System.out.println(state + " compared to " + finalStates.get(i));
                     accept();
                     break;
                  }
                  else reject();
               }//end for
            }//end else
      }//end while
   }
///////////////////////////////////////////////////////////////////////////////
   public static String nextSymbol()
   {
      String nextSymbol = "";
      if(position < stringToTest.length)
      {
         if(alternate == true)
         {
               for(int j = 0; j < numbers.length; j++) //for 0-9
                  if(stringToTest[position].equals(numbers[j])) nextSymbol = "1";

               for(int j = 0; j < alphabet.length; j++) //for a-z, A-Z
                  if(stringToTest[position].equalsIgnoreCase(alphabet[j]))
                      nextSymbol = "0";

               if(stringToTest[position].equals("_")) nextSymbol = "0";
               else if(stringToTest[position].equals("=")) symbolAccepted = false;
               else if(stringToTest[position].equals("*")) symbolAccepted = false;
         }//end if

         else //alternate == false
         for(int i = 0; i < alphabet.length; i++)
         {
            //System.out.println("alphabet: " + alphabet[i] + " index: " + i);
            //System.out.println("using normal");
            if(stringToTest[position].equals(alphabet[i]))
            {
               nextSymbol = Integer.toString(i); //gets index of letter. ex: a = 0
               break;
            }
            else nextSymbol = stringToTest[position];
         }//end for
         return nextSymbol;
      }//end if
      return symbol;
   }
///////////////////////////////////////////////////////////////////////////////
   public static String nextState(String state, String Symbol)
   {
      String nextState = states[Integer.parseInt(state)][Integer.parseInt(Symbol)].get(0);
      //System.out.println("current state: " + state);
      //System.out.println("next state: " + nextState);
      position++;
      return nextState;
   }
//////////////////////////////////////////////////////////////////////////////
   public static void accept(){ result = "Accepted"; }
//////////////////////////////////////////////////////////////////////////////
   public static void reject(){ result = "Rejected"; }
//////////////////////////////////////////////////////////////////////////////
   public static void subsetRows()
   {
      subset = new ArrayList<String>();
      subsetStates = new ArrayList<ArrayList<String>>();

      //get the first row of the transition table
      for(int cols = 0; cols < alphArray.length; cols++)
      {
         subset = new ArrayList<String>();
         for(int i = 0; i < states[0][cols].size(); i++)
            subset.add(states[0][cols].get(i));
         subsetStates.add(subset);
      }

      int temp1 = 0;
      for(int row = 1; row < numStates*2; row++)
      {
         ArrayList<String> temp = subsetStates.get(temp1++);
         for(int col = 0; col < alphArray.length; col++)
         {
            subset = new ArrayList<String>();
            for(int j = 0; j < temp.size(); j++) //for the length of previous set
            {
               //for all the numbers in the previous set
               int newRow = Integer.parseInt(temp.get(j));
               //for all numbers at this location
               for(int i = 0; i < states[newRow][col].size(); i++)
                  if(!subset.contains(states[newRow][col].get(i)))
                     subset.add(states[newRow][col].get(i));
            }
            Collections.sort(subset);
            if(!subsetStates.contains(subset)) subsetStates.add(subset);
         }
      }
   }
//////////////////////////////////////////////////////////////////////////////
   public static void subsetFinalStates()
   {
      subsetFinalStates = new ArrayList<String>();

      for(int i = 0; i < subsetStates.size(); i++) //go through each subsetState
      //go through each set in the states
      for(int j = 0; j < subsetStates.get(i).size(); j++)
         for(int k = 0; k < finalStates.size(); k++)
            if(subsetStates.get(i).get(j).contains(finalStates.get(k)))
                  if(!subsetFinalStates.contains(Integer.toString(i)))
                     subsetFinalStates.add(Integer.toString(i));

      finalStates = new ArrayList();
      for(int i = 0; i < subsetFinalStates.size(); i++)
         finalStates.add(subsetFinalStates.get(i));
   }
///////////////////////////////////////////////////////////////////////////////
   public static void subsetTransitions()
   {
      dfsa = new String[numStates*2][alphArray.length];
      subsetStates = new ArrayList<ArrayList<String>>();

      //get the first row of the transition table
      for(int cols = 0; cols < alphArray.length; cols++)
      {
         subset = new ArrayList<String>();
         for(int i = 0; i < states[0][cols].size(); i++)
            subset.add(states[0][cols].get(i));
         subsetStates.add(subset);
         dfsa[0][cols] = Integer.toString(subsetStates.size()-1);
      }
      System.out.printf("%9s", "state 0:");
      for(int i = 0; i < subsetStates.size(); i++)
      System.out.printf("%15s", subsetStates.get(i));
      System.out.println();

      int temp1 = 1;
      for(int row = 1; row < numStates*2; row++)
      {
         System.out.printf("%9s", "state " + row + ":");
         ArrayList<String> temp = subsetStates.get(temp1);
         for(int col = 0; col < alphArray.length; col++)
         {
            subset = new ArrayList<String>();
            for(int j = 0; j < temp.size(); j++) //for the length of previous set
            {
               //for all the numbers in the previous set
               int newRow = Integer.parseInt(temp.get(j));
               for(int i = 0; i < states[newRow][col].size(); i++)
                  if(!subset.contains(states[newRow][col].get(i)))
                     subset.add(states[newRow][col].get(i));
            }
            System.out.printf("%15s", subset);
            subsetStates.add(subset);

            //fill the dfsa table respectively
            for(int i = 0; i < subsetStates.size(); i++)
            {
               if(subsetStates.get(i).equals(subset))
               {
                  dfsa[row][col] = Integer.toString(i);
                  break;
               }
            }//end dfsa for
         }//end column for
            System.out.println();
            temp1++;
      }//end top level for

      //put back into original array to run UFSA
      states = new ArrayList[numStates*2][alphArray.length];

      for(int i = 0; i < numStates*2; i++)
      {
         for(int j = 0; j < alphArray.length; j++)
         {
            states[i][j] = new ArrayList();
            states[i][j].add(dfsa[i][j]);
         }
      }
   }
//////////////////////////////////////////////////////////////////////////////
   public static void read() throws IOException
   {
      FileReader fileReader = new FileReader(new File("fsa.txt"));
      BufferedReader br = new BufferedReader(fileReader);
      String line;
      int row = 0, col = 0;

      try //reads input file
      {
         while( (line = br.readLine()) != null)
         {
            if(currentFSA <= 4) //for FSA #1-4 (DFSA)
            {
               System.out.println("-----------------------------------");
               System.out.println("Finite State Automaton #" + currentFSA++ + " (DFSA)");
               System.out.println("-----------------------------------");

               numStates = Integer.parseInt(line);
               System.out.println("Number of states: " + numStates);

               //add final states into ArrayList
               finalStates = new ArrayList<String>();
               String tempFinalStates = br.readLine();
               String[] temp3 = tempFinalStates.split("\\s+");
               for(int i = 0; i < temp3.length; i++) finalStates.add(temp3[i]);

               System.out.println("Final states: " + finalStates);

               //split alphabet by characters and put into array
```

```
String alph = br.readLine();
alphArray = alph.split("\\s+");
System.out.print("Alphabet: ");
for(int i = 0; i < alphArray.length; i++)
    System.out.print(alphArray[i] + " ");
System.out.println();

//2D array. rows = number of states, columns = length of alphabet array
states = new ArrayList[numStates][alphArray.length];

//transition lines
System.out.println("Transitions: ");
String transition;
while ((line = br.readLine()).contains("("))
{
   transition = line;
   transition = transition.replace("(", ""); //removes '(' and ')'
   transition = transition.replace(")", "");
   transArray = transition.split("\\s+"); //remove spaces

   //first digit in transition
   int temp = 1;
   row = Integer.parseInt(transArray[temp++]);
   //second digit in transition
   try { col = Integer.parseInt(transArray[temp]); }
   catch (NumberFormatException e)
   {
      for(int i = 0; i < alphabet.length; i++)
         if(transArray[temp].equals(alphabet[i])) { col = i; }
   }

   //3rd+ in transition
   states[row][col] = new ArrayList();
   while(temp < transArray.length-1)
       states[row][col].add(transArray[++temp]);
   System.out.println("\t" + transition);
}

//display current string to be tested and its result
String currentString = line;
stringToTest = currentString.split("(?<!^)");
System.out.println("Strings: ");
UFSA();
System.out.printf("%17s %10s \n", currentString, result);

while ( !(line = br.readLine()).equals("~") )
{
   currentString = line;
   stringToTest = currentString.split("(?<!^)");
   UFSA();
   System.out.printf("%17s %10s \n", currentString, result);
}
```

```
   if(line.equals("~")) continue; //restart loop
}// end currentFSA < 5

//for FSA #5-8 (NFSA)
System.out.println("-----------------------------------");
System.out.println("Finite State Automaton #" + currentFSA++ + " (NFSA)");
System.out.println("-----------------------------------");

int numStates = Integer.parseInt(line);
System.out.println("Number of states: " + numStates);

//add final states into ArrayList
finalStates = new ArrayList<String>();
String tempFinalStates = br.readLine();
String[] temp3 = tempFinalStates.split("\\s+");
for(int i = 0; i < temp3.length; i++) finalStates.add(temp3[i]);

System.out.println("Final states: " + finalStates);

//split alphabet by characters and put into array
String alph = br.readLine();
alphArray = alph.split("\\s+");
System.out.print("Alphabet: ");
for(int i = 0; i < alphArray.length; i++)
   System.out.print(alphArray[i] + " ");
System.out.println();

//2D array. rows = number of states, columns = length of alphabet array
states = new ArrayList[numStates][alphArray.length];

//transition lines
System.out.println("Transitions: ");
String transition;
while ((line = br.readLine()).contains("("))
{
   transition = line;
   transition = transition.replace("(", ""); //removes '(' and ')'
   transition = transition.replace(")", "");
   transArray = transition.split("\\s+"); //remove spaces and stores in array

   //first digit in transition
   int temp = 1;
   row = Integer.parseInt(transArray[temp++]); //first digit in the transition
   //second digit in transition
   try { col = Integer.parseInt(transArray[temp]); }
   catch (NumberFormatException e)
   {
      for(int i = 0; i < alphabet.length; i++)
         if(transArray[temp].equals(alphabet[i])) { col = i; }
   }

   //3rd+ in transition
   states[row][col] = new ArrayList();
```

```
               while(temp < transArray.length-1)
                   states[row][col].add(transArray[++temp]);

               System.out.println("\t" + transition);
            }//end while

               System.out.println("Equivalent DFSA by subset construction:");
               subsetRows();
               System.out.println("Number of states : " + subsetStates.size());
               int temp = 0;
               for(int i = 0; i < subsetStates.size(); i++)
                  System.out.println("\t" + "state " + temp++ + ": " +
                         subsetStates.get(i));

               subsetFinalStates();
               System.out.println("Final states: " + subsetFinalStates);

               System.out.println("Transitions: ");
               subsetTransitions();

               //display current string to be tested and its result
               String currentString = line;
               stringToTest = currentString.split("(?<!^)");
               System.out.println("Strings: ");
               UFSA();
               System.out.printf("%17s %10s \n", currentString, result);

               while ( !(line = br.readLine()).equals("~") )
               {
                  currentString = line;
                  stringToTest = currentString.split("(?<!^)");
                  UFSA();
                  System.out.printf("%17s %10s \n", currentString, result);
               }
               if(line.equals("~")) continue; //restart loop
      }//end while
      fileReader.close();
   }//end try
   catch (IOException e) {}
   }//end read
//////////////////////////////////////////////////////////////////////////////
   public static void main(String[] args) throws IOException
   {
      String alpha = "a b c d e f g h i j k l m n o p q r s t u v w x y z";
      String capAlpha = "A B C D E F G H I J K L M N O P Q R S T U V W X Y Z";
      String integers = "0 1 2 3 4 5 6 7 8 9";

      alphabet = alpha.split("\\s+");
      capAlphabet = capAlpha.split("\\s+");
      numbers = integers.split("\\s+");
      read();
   }
//////////////////////////////////////////////////////////////////////////////
```

```
}//end FSA
```

```
-----------------------------------
Finite State Automaton #1 (DFSA)
-----------------------------------
Number of states: 4
Final states: [2]
Alphabet: a b
Transitions:
       0 a 1
       0 b 2
       1 a 3
       1 b 2
       2 a 1
       2 b 2
       3 a 3
       3 b 3
Strings:
                a   Rejected
                b   Accepted
              bab   Accepted
              abb   Accepted
            bbaba   Rejected
           babaab   Rejected
        bbbababab   Accepted
            bbbbb   Accepted
-----------------------------------
Finite State Automaton #2 (DFSA)
-----------------------------------
Number of states: 4
Final states: [0, 1, 2]
Alphabet: a b
Transitions:
       0 a 0
       0 b 1
       1 a 0
       1 b 2
       2 a 0
       2 b 3
       3 a 3
       3 b 3
Strings:
               bb   Accepted
              aab   Accepted
            babbb   Rejected
           ababab   Accepted
          bbaabba   Accepted
        abababbba   Rejected
              bbb   Rejected
-----------------------------------
Finite State Automaton #3 (DFSA)
-----------------------------------
Number of states: 5
Final states: [4]
Alphabet: 0 1
```

```
Transitions:
       0 0 1
       0 1 0
       1 0 2
       1 1 3
       2 0 3
       2 1 3
       3 0 4
       3 1 3
       4 0 4
       4 1 3
Strings:
               11   Rejected
              000   Rejected
            10110   Accepted
            00110   Accepted
         01110101   Rejected
         11001110   Accepted
        010101010   Accepted
             0000   Accepted
-----------------------------------
Finite State Automaton #4 (DFSA)
-----------------------------------
Number of states: 4
Final states: [2]
Alphabet: a-z A-Z 0-9 _
Transitions:
       0 0 1
       0 1 3
       1 0 2
       1 1 2
       2 0 2
       2 1 2
       3 0 3
       3 1 3
Strings:
   1st_Assignment   Rejected
           Pascal   Accepted
_finite_automaton   Accepted
          program   Accepted
             X3Y7   Accepted
                _   Rejected
             X=90   Rejected
              X*Y   Rejected
-----------------------------------
Finite State Automaton #5 (NFSA)
-----------------------------------
Number of states: 4
Final states: [3]
Alphabet: 0 1
Transitions:
       0 0 0
       0 1 0 1
       1 0 2
       1 1 2
       2 0 3
       2 1 3
       3 0 3
       3 1 3
Equivalent DFSA by subset construction:
Number of states : 8
      state 0: [0]
      state 1: [0, 1]
      state 2: [0, 2]
      state 3: [0, 1, 2]
      state 4: [0, 3]
      state 5: [0, 1, 3]
      state 6: [0, 2, 3]
      state 7: [0, 1, 2, 3]
Final states: [4, 5, 6, 7]
Transitions:
 state 0:            [0]         [0, 1]
 state 1:         [0, 2]      [0, 1, 2]
 state 2:         [0, 3]      [0, 1, 3]
 state 3:      [0, 2, 3]   [0, 1, 2, 3]
 state 4:         [0, 3]      [0, 1, 3]
 state 5:      [0, 2, 3]   [0, 1, 2, 3]
 state 6:         [0, 3]      [0, 1, 3]
 state 7:      [0, 2, 3]   [0, 1, 2, 3]
Strings:
           010100   Accepted
              011   Accepted
          0000111   Accepted
          1111000   Accepted
                0   Rejected
                1   Rejected
          1010100   Accepted
-----------------------------------
Finite State Automaton #6 (NFSA)
-----------------------------------
Number of states: 5
Final states: [2, 4]
Alphabet: 0 1
Transitions:
       0 0 0 3
       0 1 0 1
       1 0 1
       1 1 2
       2 0 2
       2 1 2
       3 0 4
       3 1 3
       4 0 4
       4 1 4
Equivalent DFSA by subset construction:
Number of states : 8
```

```
      state 0: [0, 3]
      state 1: [0, 1]
      state 2: [0, 3, 4]
      state 3: [0, 1, 3]
      state 4: [0, 1, 2]
      state 5: [0, 1, 3, 4]
      state 6: [0, 1, 2, 3]
      state 7: [0, 1, 2, 3, 4]
Final states: [2, 4, 5, 6, 7]
Transitions:
 state 0:         [0, 3]         [0, 1]
 state 1:      [0, 3, 1]      [0, 1, 2]
 state 2:   [0, 3, 4, 1]   [0, 1, 3, 2]
 state 3:   [0, 3, 1, 2]      [0, 1, 2]
 state 4:   [0, 3, 4, 1][0, 1, 3, 4, 2]
 state 5:[0, 3, 1, 4, 2]   [0, 1, 2, 3]
 state 6:[0, 3, 4, 1, 2]   [0, 1, 3, 2]
 state 7:   [0, 3, 1, 2]      [0, 1, 2]
Strings:
             0101   Rejected
            10110   Rejected
          1010010   Rejected
           011000   Rejected
               00   Rejected
                1   Rejected
              111   Rejected
-----------------------------------
Finite State Automaton #7 (NFSA)
-----------------------------------
Number of states: 5
Final states: [4]
Alphabet: a b c
Transitions:
       0 a 0 1
       0 b 0 2
       0 c 0
       1 a 0
       1 b 3
       1 c 2
       2 a 3 4
       2 b 2
       2 c 2
       3 a 0
       3 b 0
       3 c 0 4
       4 a 4
       4 b 4
       4 c 0
Equivalent DFSA by subset construction:
Number of states : 8
      state 0: [0, 1]
      state 1: [0, 2]
      state 2: [0]
      state 3: [0, 2, 3]
      state 4: [0, 1, 3, 4]
      state 5: [0, 2, 4]
      state 6: [0, 1, 4]
      state 7: [0, 2, 3, 4]
Final states: [4, 5, 6, 7]
Transitions:
 state 0:         [0, 1]         [0, 2]            [0]
 state 1:   [0, 1, 3, 4]         [0, 2]         [0, 2]
 state 2:         [0, 1]         [0, 2]            [0]
 state 3:      [0, 1, 4]   [0, 2, 3, 4]      [0, 2, 4]
 state 4:   [0, 1, 3, 4]         [0, 2]         [0, 2]
 state 5:   [0, 1, 3, 4]         [0, 2]         [0, 2]
 state 6:         [0, 1]      [0, 2, 3]         [0, 2]
 state 7:   [0, 1, 3, 4]         [0, 2]         [0, 2]
Strings:
                a   Rejected
               ba   Rejected
               ac   Rejected
              abc   Rejected
            cabca   Rejected
            acbac   Rejected
            bacbc   Rejected
            aabcc   Rejected
-----------------------------------
Finite State Automaton #8 (NFSA)
-----------------------------------
Number of states: 6
Final states: [5]
Alphabet: 0 1
Transitions:
       0 0 0 1
       0 1 0
       1 0 2
       1 1 2
       2 0 3
       2 1 3
       3 0 4
       3 1 4
       4 0 4
       4 1 4
       5 0 5
       5 1 5
Equivalent DFSA by subset construction:
Number of states : 14
      state 0: [0, 1]
      state 1: [0]
      state 2: [0, 1, 2]
      state 3: [0, 2]
      state 4: [0, 1, 2, 3]
      state 5: [0, 2, 3]
      state 6: [0, 1, 3]
      state 7: [0, 3]
      state 8: [0, 1, 2, 3, 4]
      state 9: [0, 2, 3, 4]
      state 10: [0, 1, 3, 4]
      state 11: [0, 3, 4]
      state 12: [0, 1, 2, 4]
      state 13: [0, 2, 4]
Final states: []
Transitions:
 state 0:         [0, 1]            [0]
 state 1:         [0, 1]            [0]
 state 2:      [0, 1, 2]         [0, 2]
 state 3:         [0, 1]            [0]
 state 4:   [0, 1, 2, 3]      [0, 2, 3]
 state 5:      [0, 1, 3]         [0, 3]
 state 6:      [0, 1, 2]         [0, 2]
 state 7:         [0, 1]            [0]
Strings:
             1010   Rejected
         00000111   Rejected
         11111000   Rejected
            01110   Rejected
          0101010   Rejected
              000   Rejected
                1   Rejected
```