NFA to DFA program
Lab 4
Aiman Fatima
20BCS008

```
#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <map>
#include <algorithm>
#include <string>
using namespace std;

// global variables
int initial;
vector<int> final;
vector<vector<vector<int>>> NFA;
vector<vector<int>> DFA;

// string to integer conversion
int stringToInt(string line)
{
   int i = 0;
   int num = 0;
   while (line[i] != 0)
   {
      if (line[0] == '-')
      {
         return -1;
      }
      num *= 10;
      num += (line[i] - '0');
      i++;
   }
   return num;
}

// function converting NFA to DFA
void nfaToDfa()
{
   getchar();
   map<int, vector<int>> state_map;

   int inserted_size = NFA.size();
   int original_size = NFA.size();

   for (int i = 0; i < NFA.size(); i++)
   {
      vector<vector<int>> tempArray;
      vector<vector<int>> temp(NFA[i].size());
      for (int j = 0; j < NFA[i].size(); j++)
```

```
{
   if (NFA[i][j].size() > 1)
   {
      // mapping and changing state to new state
      for (auto it : state_map)
      {
         if (it.second == NFA[i][j])
         {
            NFA[i][j].clear();
            NFA[i][j].push_back(it.first);
         }
      }
      state_map.insert(pair<int, vector<int>>(inserted_size, NFA[i][j]));
      NFA[i][j].clear();
      NFA[i][j].push_back(inserted_size++);

      // merging states
      for (auto it : state_map[inserted_size - 1])
      {
         for (int x = 0; x < NFA[it].size(); x++)
         {
            vector<int> s;
            for (int y = 0; y < NFA[it][x].size(); y++)
            {
               s.push_back(NFA[it][x][y]);
            }
            for (int y = 0; y < s.size(); y++)
            {
               temp[x].push_back(s[y]);
            }
         }
      }

      // removing duplicates and -1
      for (int x = 0; x < temp.size(); x++)
      {
         vector<int> v;
         set<int> s;
         for (int y = 0; y < temp[x].size(); y++)
         {
            if (temp[x][y] != -1)
            {
               s.insert(temp[x][y]);
            }
         }
         if (s.empty())
         {
            s.insert(-1);
         }
         for (auto it : s)
         {
            v.push_back(it);
```

```
            }
            temp[x] = v;
        }

        // checking if state already exists in the map
        for (int x = 0; x < temp.size(); x++)
        {
            if (temp[x].size() > 1)
            {
                bool found = false;
                for (auto it : state_map)
                {
                    if (it.second == temp[x])
                    {
                        temp[x].clear();
                        temp[x].push_back(it.first);
                        tempArray.push_back(temp[x]);
                        found = true;
                    }
                }
                if (found)
                {
                    continue;
                }

                for (int y = temp[x].size() - 1; y >= 0; y--)
                {
                    // checking if the new create state exists in the temp array and if it does, then merge it with removing the duplicate
                    if (temp[x][y] >= original_size)
                    {
                        temp[x].erase(temp[x].begin() + y);
                        for (auto it : state_map[temp[x][y]])
                        {
                            temp[x].push_back(it);
                        }
                        vector<int> v;
                        set<int> s;
                        for (int y = 0; y < temp[x].size(); y++)
                        {
                            s.insert(temp[x][y]);
                        }
                        for (auto it : s)
                        {
                            v.push_back(it);
                        }
                        temp[x] = v;
                    }
                }

                bool newFound = false;
                for (auto it : state_map)
```

```
                {
                    if (it.second == temp[x])
                    {
                        temp[x].clear();
                        temp[x].push_back(it.first);
                        tempArray.push_back(temp[x]);
                        newFound = true;
                        break;
                    }
                }
                if (newFound)
                {
                    continue;
                }
                state_map.insert(pair<int, vector<int>>(inserted_size, temp[x]));
                temp[x].clear();
                temp[x].push_back(inserted_size);
                tempArray.push_back(temp[x]);
            }
            else
            {
                tempArray.push_back(temp[x]);
            }
        }
        NFA.push_back(tempArray);
        tempArray.clear();
    }
  }
}

  // making final states
  for (auto it : state_map)
  {
    for (auto it1 : it.second)
    {
      for (auto it2 : final)
      {
        if (it1 == it2)
        {
          final.push_back(it.first);
        }
      }
    }
  }
}

void printDFA()
{
  cout << "\n__Printing the DFA__\n\n";
  //  printing initial state
  cout << "Initial State: " << initial << endl;
```

```
    // printing final states
    cout << "Final States: ";
    for (auto it : final)
    {
        cout << it << " ";
    }
    cout << endl;

    // printing DFA
    cout << "State Transitions" << endl;
    for (int i = 0; i < NFA.size(); i++)
    {
        for (int j = 0; j < NFA[i].size(); j++)
        {
            for (int k = 0; k < NFA[i][j].size(); k++)
            {
                cout << NFA[i][j][k] << " ";
            }
        }
        cout << endl;
    }
}

void DFAfile()
{
    // writing to file "Converted_DFA.txt"
    ofstream file1;
    file1.open("Converted_DFA.txt");
    file1 << initial << endl;
    for (auto it : final)
    {
        file1 << it << " ";
    }
    file1 << endl;
    for (int i = 0; i < NFA.size(); i++)
    {
        for (int j = 0; j < NFA[i].size(); j++)
        {
            for (int k = 0; k < NFA[i][j].size(); k++)
            {
                file1 << NFA[i][j][k];
            }
            if (j != NFA[i].size() - 1)
            {
                file1 << " ";
            }
        }
        if (i != NFA.size() - 1)
        {
            file1 << endl;
        }
    }
```

```
    file1.close();
}

int main()
{
    ifstream file; // file's ptr that accepts
    string line;
    file.open("l4_inp.txt"); // opens the file
    int i = 0;

    while (getline(file, line)) // reads the file line by line
    {
        if (i == 0)
        {
            initial = stringToInt(line);
        }
        else if (i == 1)
        {
            int x = 0;
            string num = "";
            while (line[x] != 0)
            {
                if (line[x] == ' ')
                {
                    final.push_back(stringToInt(num));
                    num = "";
                    x++;
                    continue;
                }
                num += line[x++];
            }
            final.push_back(stringToInt(num));
        }
        else
        {
            vector<vector<int>> temp;
            vector<int> temp1;
            int x = 0;
            string num = "";
            while (line[x] != 0)
            {
                if (line[x] == ',')
                {
                    temp1.push_back(stringToInt(num));
                    num = "";
                    x++;
                    continue;
                }
                if (line[x] == ' ')
                {
                    temp1.push_back(stringToInt(num));
                    temp.push_back(temp1);
```

```
            temp1.clear();
            num = "";
            x++;
            continue;
        }
        num += line[x++];
    }
    temp1.push_back(stringToInt(num));
    temp.push_back(temp1);
    NFA.push_back(temp);
  }
  i++;
}

file.close();

// NFA to DFA
// nfaToDfa();

bool flag1(false);

while (1)
{
  cout << "\n----------------\n";
  cout << "1. Conversion NFA to DFA\n";
  cout << "2. Print the previous DFA\n";
  cout << "3. Output previous DFA to file\n";
  cout << "4. Exit\n";
  cout << "\nEnter choice : ";
  int choice;
  cin >> choice;
  switch (choice)
  {
  case 1:
    // final.clear();
    // NFA.clear();
    DFA.clear();
    nfaToDfa();
    printDFA();
    DFAfile();
    flag1 = true;
    break;
  case 2:
    if (flag1)
      printDFA();
    else
      cout << "\nNo previous NFA conversion done yet!\n";
    break;
  case 3:
    if (flag1)
      DFAfile();
    else
```

```
            cout << "\nNo previous NFA conversion done yet!\n";
            break;
        case 4:
            cout << "\n--The End--\n\n";
            return 0;
        default:
            cout << "\nWrong Choice\n";
            break;
        }
    }

    // Printing the DFA

    return 0;
}
```