COMPILER DESIGN LAB

MASEERA ALI
13 BCS 0032
BTECH COMPUTER ENGG.
2013 – 2017

1. Program to implement a finite automata. Program should read automata from a text file and check if the string input at the console is accepted or not.

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>

using namespace std;

void print_matrix(vector< vector<int> >);

void auto(int, vector<int>, vector< vector<int> >);

int main() {
	string line;
	int start;
	vector<int> final;
	vector< vector<int> >mat;
	ifstream fin("fa.txt");
	if(fin.is_open()) {
		getline(fin, line);
		start = line[0] - '0';
		getline(fin, line);
		for(int i=0;i<line.length();i++) {
			if(line[i]!=' ' && line[i] != '\n') {
				final.push_back(line[i] - '0');
			}
		}
		while(getline(fin, line)) {
			vector<int> row;
			for(int i=0;i<line.length();i++) {
				if(line[i] == '-') {
					i+=2;
					row.push_back(-1);
```

```
                }
                if(line[i]!=' ') {
                    row.push_back(line[i] - '0');
                }

            }
            mat.push_back(row);
        }
    }

    std::cout<<endl<<"Initial State: "<<start<<endl;
    std::cout<<endl<<"Final States: ";
    for(int i=0;i<final.size();i++) {
        std::cout<<final[i]<<" ";
    }
    std::cout<<endl;
    print_matrix(mat);

    auto(start, final, mat);

    return 0;
}

void print_matrix(vector< vector<int> > mat) {
    std::cout<<endl<<"Matrix: "<<endl;
    for(int i=0;i<mat.size();i++) {
        vector<int> v = mat[i];
        for(int j=0;j<v.size();j++) {
            std::cout<<v[j]<<" ";
        }
        std::cout<<endl;
    }
}

int find_final(vector<int> final, int next) {
    for(int i=0;i<final.size();i++) {
```

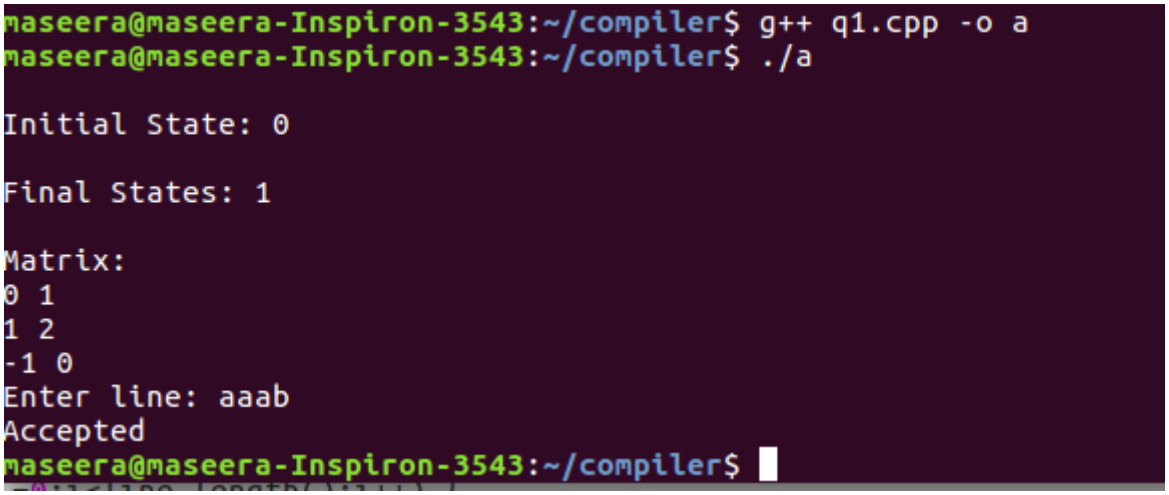
```
            if(final[i] == next) {
                return 1;
            }
        }
        return 0;
}

void auto(int start, vector<int> final, vector< vector<int> >
mat) {
        string line;
        int prev = 0;
        int next;
        std::cout<<"Enter line: ";
        getline(std::cin, line);
        for(int i=0;i<line.length();i++) {
            int cur = line[i] - 'a';
            next = mat[prev][cur];
            if(i == line.length() - 1) {
                if(find_final(final, next)) {
                    std::cout<<"Accepted"<<endl;
                    break;
                } else {
                    std::cout<<"Not accepted"<<endl;
                }
            }
            if(next == -1) {
                std::cout<<"Not accepted"<<endl;
            }
            prev = next;
        }
}
```

Input File:

```
0
1
0 1
1 2
-1 0
```

Output:



2. Write a program to implement a Mealy machine. The program should read the machine from a file and generate the corresponding output for a string given from a console.

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cstdlib>
using namespace std;

char mealy[100][100][10];

int main()
{

	int initial,state;
	int temp;
	FILE *f;
	char str[100];
	char outstr[100];
	int row=0,col=0,maxrow=0,maxcol=0;

	f = fopen("mealy_input.txt","r");

	//read initial state
	fscanf(f,"%s",str);
	initial = atoi(str);

	char c;
	int k=0;
	c = getc(f);

	while(!feof(f))
	{
		c = getc(f);
```

```
            if(c == ' ' || c == '\n')
            {
                  str[k] = '\0';
                  strcpy(mealy[row][col],str);
                  k=0;
                  if(c == ' ')
                  {
                        col++;
                        maxcol = max(col,maxcol);
                  }
                  else
                  {
                        col=0;
                        row++;
                        maxrow = max(row,maxrow);
                  }
            }
            else
                  str[k++] = c;
}

for(int i=0;i<maxrow+1;i++)
{
      for(int j=0;j<maxcol+1;j++)
            std::cout << mealy[i][j] << " ";
      std::cout << endl;
}


std::cout << "Enter the input string" << endl;
std::cin>>str;
int i=0,j;
int l=strlen(str);
char output[100];
state = initial;
while(i<l)
```

```
    {
          temp = str[i]-48;
        strcpy(outstr,mealy[state][2*temp+1]);
        state = atoi(mealy[state][2*temp]);
        j = atoi(outstr);
        if(j==-1)
        {
         strcpy(output,"Error");
         break;
        }
           strcat(output,outstr);
        i++;
    }

    std::cout << output << endl;

    fclose(f);
}
```

Input File:

```
0
0  1  1  0
-1  -1  1  1
```

Output:

3. Write a program to implement a Moore machine. The program should read the machine from a file and generate the corresponding output for a string given from a console.

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cstdlib>
using namespace std;

char moore[100][100][10];

int main()
{

	int initial,state;
	int temp;
	FILE *f;
	char str[100];
	char outstr[100];
	int row=0,col=0,maxrow=0,maxcol=0;

	f = fopen("moore_input.txt","r");

	//read initial state
	fscanf(f,"%s",str);
	initial = atoi(str);
	// std::cout<<initial;
	char c;
	int k=0;
	c = getc(f);

	while(!feof(f))
	{
		c = getc(f);
```

```
            if(c == ' ' || c == '\n')
            {
                  str[k] = '\0';
                  strcpy(moore[row][col],str);
                  k=0;
                  if(c == ' ')
                  {
                        col++;
                        maxcol = max(col,maxcol);
                  }
                  else
                  {
                        col=0;
                        row++;
                        maxrow = max(row,maxrow);
                  }
            }
            else
                  str[k++] = c;
}

for(int i=0;i<maxrow+1;i++)
{
      for(int j=0;j<maxcol+1;j++)
            std::cout << moore[i][j] << " ";
      std::cout << endl;
}


std::cout<< "Enter the input string" << endl;
std::cin>>str;
int i=0;
int l=strlen(str);
char output[100];
state = initial;
strcpy(output,moore[initial][maxcol]);
```

```
        while(i<l)
        {
              temp = str[i]-48;
             state = atoi(moore[state][temp]);
             if(state==-1)
             {
              strcpy(output,"Error");
              break;
             }
               strcat(output,moore[state][maxcol]);
             i++;
        }

        std::cout << output << endl;


}
```

Input File:

```
0
0 1 1
1 -1 0
```

Output:

```
maseera@maseera-Inspiron-3543:~/compiler$ g++ moore.cpp -o b
maseera@maseera-Inspiron-3543:~/compiler$ ./b
0 1 1
1 -1 0

Enter the input string
011
Error
maseera@maseera-Inspiron-3543:~/compiler$ ./b
0 1 1
1 -1 0

Enter the input string
010
1100
maseera@maseera-Inspiron-3543:~/compiler$
```

4. Write a program to convert NFA to DFA. Program should take the given NFA from a text file and output the table of the DFA.

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>

using namespace std;

void display(int, vector<int>, vector< vector< vector<int> > >);
void compute(int, vector<int>, vector< vector< vector<int> > >&);
void display_row(vector< vector<int> >);
void display_one(vector<int>);
int exists(vector< vector<int> >, vector<int>);

int main() {
    int start;
    string line;
    vector<int> final;
    vector< vector< vector<int> > > mat;
    ifstream fin("nfa.txt");
    if(fin.is_open()) {
        getline(fin, line);

        // Get initial state
        start = line[0] - 48;
        getline(fin, line);

        // Get final states
        for (int i=0; i<line.length(); i++) {
            if(line[i] != ' ') {
                final.push_back((int)(line[i] - 48));
```

```
            }
        }

        // Get rest of the inputs
        while(getline(fin, line)) {
            vector< vector<int> >row;
            vector<int> one;
            for (int i=0; i<line.length(); i++) {
                //std::cout<<"Line length:
"<<line.length()<<endl;
                if(line[i] != ' ') {
                    if(line[i] == '-') {
                        one.push_back(-1);
                        //std::cout<<"Pushing:
"<<one[one.size()-1]<<endl;
                        i++;
                    } else if(line[i] != ',') {
                        one.push_back((int)(line[i]-48));
                        //std::cout<<"Pushing:
"<<one[one.size()-1]<<endl;
                    }
                    if(i == line.length()-1) {
                        row.push_back(one);
                    }
                } else {
                    row.push_back(one);
                    one.clear();
                }
            }

            mat.push_back(row);

        }
    }

    display(start, final, mat);
```

```
    compute(start, final, mat);

    fin.close();
    return 0;
}

void display(int start, vector<int> final, vector< vector<
vector<int> > >mat) {
     // Print initial state
     std::cout<<"Inital state: "<<start<<endl;

     // Print final states
    std::cout<<"Final States: ";
    for (int i=0; i<final.size(); i++) {
        std::cout<<final[i]<<" ";
    }
    std::cout<<endl;

    // Print matrix
    std::cout<<"Matrix:"<<endl;
    for (int i=0; i<mat.size(); i++) {
        vector< vector<int> > row = mat[i];
        for (int j=0; j<row.size(); j++) {
            vector<int> one = row[j];
            for (int k=0; k<one.size(); k++) {
                std::cout<<one[k]<<",";
            }
            std::cout<<" ";
        }
        std::cout<<endl;
    }
}

void display_row(vector< vector<int> > row) {
     std::cout<<"Inputting row: ";
     for (int j=0; j<row.size(); j++) {
```

```
        vector<int> one = row[j];
        for (int k=0; k<one.size(); k++) {
            std::cout<<one[k]<<",";
        }
        std::cout<<" ";
    }
    std::cout<<endl;
}

void display_one(vector<int> one) {
     std::cout<<"One: ";
     for (int k=0; k<one.size(); k++) {
        std::cout<<one[k]<<",";
    }
    std::cout<<endl;
}

int exists(vector< vector<int> > pushing, vector<int> one) {
     int flag;
     for(int i=0;i<pushing.size();i++) {
          vector<int> temp = pushing[i];
          flag = 0;
          if(temp.size() != one.size()) {
               continue;
          }
          for (int j = 0; j < temp.size(); ++j) {
               if(temp[j] == one[j]) {
                    flag = 1;
               } else {
                    flag = 0;
                    break;
               }
          }
          if(flag == 1) {
               return 1;
          }
```

```
	}
	return 0;
}

int find_element(vector<int> one, int el) {
	for(int i=0;i<one.size();i++) {
		if(one[i] == el) {
			return 1;
		}
	}
	return 0;
}

void compute(int start, vector<int> final, vector< vector<
vector<int> > >&mat) {
	std::cout<<endl<<"Converting to DFA"<<endl;
	vector< vector<int> > pushing;
	for (int i=0; i<mat.size(); i++) {
	   vector< vector<int> > row = mat[i];
	   for (int j=0; j<row.size(); j++) {
	      vector<int> one = row[j];
	      if(one.size() > 1) {
	         // Check if this combination has been pushed
already or not
	         if(exists(pushing, one)) {
	            continue;
	         }
	         // display_one(one);
	         vector< vector<int> > temp_d;
	         int index;
	         for(int a=0;a<one.size();a++) {
	            int mat_i = one[a];
	            vector< vector<int> > temp_mat =
mat[mat_i];
	            for(int b=0;b<temp_mat.size();b++) {
	               vector<int> temp_s;
```

```
                        if(a != 0) {
                            temp_s = temp_d[b];
                        }
                        vector<int> temp_row = temp_mat[b];
                        // display_one(temp_row);
                        if(a == 0) {
                            vector<int> x;
                            for(int z=0;z<temp_row.size();z++) {
                                if(temp_row[z] != -1) {
                                    x.push_back(temp_row[z]);
                                }
                            }
                            temp_d.push_back(x);
                            index = temp_d.size() - 1;
                        } else {
                            for(int c=0;c<temp_row.size();c++) {
                                if(!find_element(temp_s,
temp_row[c])) {
                                    if(temp_row[c] != -1) {

    temp_s.push_back(temp_row[c]);
                                    }
                                }
                            }
                            int ti = index - 1 + b;
                            // display_one(temp_s);
                            temp_d.erase(temp_d.begin() + ti);
                            temp_d.insert(temp_d.begin() + ti,
temp_s);
                        }

                    }
                }
                // display_row(temp_d);
                mat.push_back(temp_d);
```

```
                //  std::cout<<"Pushing:  ";
                //  display_one(one);
                pushing.push_back(one);
                //one.clear();
                //one.push_back(state);
            }
        }
    }
    display(start,  final,  mat);
}
```

Input:

```
0
2
2  0,1
1  2
2  -1
```

Output:

```
maseera@maseera-Inspiron-3543:~/compiler$ g++ nfa_to_dfa.cpp -o b
maseera@maseera-Inspiron-3543:~/compiler$ ./b
Inital state: 0
Final States: 2
Matrix:
2, 0,1,
1, 2,
2, -1,

Converting to DFA
Inital state: 0
Final States: 2
Matrix:
2, 0,1,
1, 2,
2, -1,
2,1, 0,1,2,
2,1, 2,
2,1, 0,1,2,
maseera@maseera-Inspiron-3543:~/compiler$
```

Q5. Program to implement a NFA . Program should read automata from a text file and check if the string input at the console is accepted or not.

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>

using namespace std;

void display(int, vector<int>, vector< vector< vector<int> > >);
void compute(int, vector<int>, vector< vector< vector<int> > >&);
void display_row(vector< vector<int> >);
void display_one(vector<int>);
int exists(vector< vector<int> >, vector<int>);

int main() {
    int start;
    string line;
    vector<int> final;
    vector< vector< vector<int> > > mat;
    ifstream fin("nfa.txt");
    if(fin.is_open()) {
        getline(fin, line);

        // Get initial state
        start = line[0] - 48;
        getline(fin, line);

        // Get final states
        for (int i=0; i<line.length(); i++) {
            if(line[i] != ' ') {
```

```
                final.push_back((int)(line[i] - 48));
            }
        }

        // Get rest of the inputs
        while(getline(fin, line)) {
            vector< vector<int> >row;
            vector<int> one;
            for (int i=0; i<line.length(); i++) {
                //std::cout<<"Line length:
"<<line.length()<<endl;
                if(line[i] != ' ') {
                    if(line[i] == '-') {
                        one.push_back(-1);
                        //std::cout<<"Pushing:
"<<one[one.size()-1]<<endl;
                        i++;
                    } else if(line[i] != ',') {
                        one.push_back((int)(line[i]-48));
                        //std::cout<<"Pushing:
"<<one[one.size()-1]<<endl;
                    }
                    if(i == line.length()-1) {
                        row.push_back(one);
                    }
                } else {
                    row.push_back(one);
                    one.clear();
                }
            }

            mat.push_back(row);

        }
    }
```

```
    display(start, final, mat);
    compute(start, final, mat);

    fin.close();
    return 0;
}

void display(int start, vector<int> final, vector< vector<
vector<int> > >mat) {
     // Print initial state
     std::cout<<"Inital state: "<<start<<endl;

     // Print final states
    std::cout<<"Final States: ";
    for (int i=0; i<final.size(); i++) {
        std::cout<<final[i]<<" ";
    }
    std::cout<<endl;

    // Print matrix
    std::cout<<"Matrix:"<<endl;
    for (int i=0; i<mat.size(); i++) {
        vector< vector<int> > row = mat[i];
        for (int j=0; j<row.size(); j++) {
            vector<int> one = row[j];
            for (int k=0; k<one.size(); k++) {
                std::cout<<one[k]<<",";
            }
            std::cout<<" ";
        }
        std::cout<<endl;
    }
}

void display_row(vector< vector<int> > row) {
     std::cout<<"Inputting row: ";
```

```
        for (int j=0; j<row.size(); j++) {
            vector<int> one = row[j];
            for (int k=0; k<one.size(); k++) {
                std::cout<<one[k]<<",";
            }
            std::cout<<" ";
        }
        std::cout<<endl;
}

void display_one(vector<int> one) {
        std::cout<<"One: ";
        for (int k=0; k<one.size(); k++) {
            std::cout<<one[k]<<",";
        }
        std::cout<<endl;
}

int exists(vector< vector<int> > pushing, vector<int> one) {
        int flag;
        for(int i=0;i<pushing.size();i++) {
                vector<int> temp = pushing[i];
                flag = 0;
                if(temp.size() != one.size()) {
                        continue;
                }
                for (int j = 0; j < temp.size(); ++j) {
                        if(temp[j] == one[j]) {
                                flag = 1;
                        } else {
                                flag = 0;
                                break;
                        }
                }
                if(flag == 1) {
                        return 1;
```

```
            }
        }
        return 0;
}

int find_element(vector<int> one, int el) {
        for(int i=0;i<one.size();i++) {
            if(one[i] == el) {
                return 1;
            }
        }
        return 0;
}

void compute(int start, vector<int> final, vector< vector<
vector<int> > >&mat) {
        std::cout<<endl<<"Converting to DFA"<<endl;
        vector< vector<int> > pushing;
        for (int i=0; i<mat.size(); i++) {
            vector< vector<int> > row = mat[i];
            for (int j=0; j<row.size(); j++) {
                vector<int> one = row[j];
                if(one.size() > 1) {
                    // Check if this combination has been pushed
already or not
                    if(exists(pushing, one)) {
                        continue;
                    }
                    // display_one(one);
                    vector< vector<int> > temp_d;
                    int index;
                    for(int a=0;a<one.size();a++) {
                        int mat_i = one[a];
                        vector< vector<int> > temp_mat =
mat[mat_i];
                        for(int b=0;b<temp_mat.size();b++) {
```

```
                        vector<int> temp_s;

                        if(a != 0) {
                            temp_s = temp_d[b];
                        }
                        vector<int> temp_row = temp_mat[b];
                        // display_one(temp_row);
                        if(a == 0) {
                            vector<int> x;
                            for(int z=0;z<temp_row.size();z++) {
                                if(temp_row[z] != -1) {
                                    x.push_back(temp_row[z]);
                                }
                            }
                            temp_d.push_back(x);
                            index = temp_d.size() - 1;
                        } else {
                            for(int c=0;c<temp_row.size();c++) {
                                if(!find_element(temp_s,
temp_row[c])) {
                                    if(temp_row[c] != -1) {

    temp_s.push_back(temp_row[c]);
                                    }
                                }
                            }
                            int ti = index - 1 + b;
                            // display_one(temp_s);
                            temp_d.erase(temp_d.begin() + ti);
                            temp_d.insert(temp_d.begin() + ti,
temp_s);
                        }

                    }
                }
                // display_row(temp_d);
```

```
                mat.push_back(temp_d);
                // std::cout<<"Pushing: ";
                // display_one(one);
                pushing.push_back(one);
                //one.clear();
                //one.push_back(state);
            }
        }
    }
    display(start, final, mat);
}
```

INPUTS

```
0
2
2 0,1
1 2
2 -1
```