COMP 4400 Automata Theory
Project#1 – Implementing a DFA
Due Date: 03/06/24

Overview:

You will need to implement a general purpose DFA in C++ that can read the description of the DFA from an input file and construct the DFA. The input file will be assumed to store the DFA description in a standard format. The content of a sample file is shown next:

```
<states>
q0
q1
<alphabet>
a
b
<transitions>
q0 a q0
q0 b q1
q1 a q0
q1 b q1
<initial state>
q0
<final states>
q1
```

You will implement the DFA as a class having the following data:

- a vector of states where each state will be represented as a string
- a vector of alphabet symbols where each symbol will be represented as a character
- a map (key-value) of transitions ($\delta: Q \times \Sigma \rightarrow Q$), where the "key" will be a pair of a string (representing the current state of the automata) and a character (representing the input symbol being processed), and the "value" will be a string representing the next state
- a string representing the initial state
- a vector of final states where each final state will be represented as a string
- a string representing the current state of the automata

The DFA class will have the following member functions:

- a function for adding a given state to the automata
- a function for adding a given alphabet symbol
- a function to add a transition (<state,symbol> to <state>)
- a function to set the initial state
- a function to add a final state
- a function to display the DFA description (as shown in the sample execution)
- a function to set the current state to the initial state
- a function to make a transition from the current state to the next state for a given input symbol
- a function that will return the current state of the automata
- a function to check if a symbol is valid
- a function to check if a given state is a final state

The driver code should read a given input file (whose name will be provided from command line) and instantiate the DFA. Then the program should display the DFA description in the format shown in the sample execution. Then the user will be continually prompted to enter an input string (until Ctrl^C is pressed to terminate the program) and the program will determine, using the DFA functions, whether this string is accepted or rejected by the DFA. The program should also display the sequence of all the moves in the format shown in the sample execution. In case the input string contains an invalid symbol, the program should stop processing the rest of the input. Three input files have been provided so that you can test your program.

Deliverables:

- You will need to submit the C++ source file(s) in Blackboard by 03/06/24.
- You will need to demonstrate the project by 03/22/24.

Instructions:

- You MUST develop your program in the Linux environment using the g++ compiler.
- Write your driver code in *program1.cpp* and the DFA class in *dfa,h* and *dfa.cpp*.
- Each program file should have the appropriate comment block at the top:

```
// Name: Your Name
// File Name:
// Date: Day Month, Year
// Description: Brief description of the code written in the file
```

- Use meaningful identifiers, sufficient and helpful comments, and a consistent coding style.
- This will be an individual project. However, feel free to contact the instructor (and ONLY the instructor) if you need help. You may discuss general aspects of the project with your classmates, but you may not collaborate in any way in producing code. Failure to follow these requirements will result in a grade of F in the project.

Grading:

This project will be graded as follows:

Program Correctness	90 pts
Program Style	10 pts

Sample Executions:

```
$ ./prog1
usage: <prog_name> <file_name>

$ ./prog1 foo
Automata file could not be opened!

$ ./prog1 test1.txt
----------D F A----------
<states>
q0 q1 q2
<alphabet>
a b
<transitions>
(q0,a)->q1
(q0,b)->q0
(q1,a)->q0
(q1,b)->q2
(q2,a)->q2
(q2,b)->q2
<initial state>
q0
<final states>
q1 q2
-------------------------

Enter a string to process (Ctrl^C to end): aa

[q0]-a->[q1]-a->[q0] : Rejected

Enter a string to process (Ctrl^C to end): aaa

[q0]-a->[q1]-a->[q0]-a->[q1] : Accepted

Enter a string (Ctrl^C to end):

[q0] : Rejected

Enter a string to process (Ctrl^C to end): aaxbb

[q0]-a->[q1]-a->[q0] (Invalid symbol x) : Rejected

Enter a string to process (Ctrl^C to end):
```

```
$ ./prog1 test2.txt
----------D F A----------
<states>
q0 q1
<alphabet>
a b
<transitions>
(q0,a)->q0
(q0,b)->q1
(q1,a)->q0
(q1,b)->q1
<initial state>
q0
<final states>
q1
-------------------------

Enter a string (Ctrl^C to end): aba

[q0]-a->[q0]-b->[q1]-a->[q0] : Rejected

Enter a string (Ctrl^C to end): aab

[q0]-a->[q0]-a->[q0]-b->[q1] : Accepted

Enter a string (Ctrl^C to end):

[q0] : Rejected

Enter a string (Ctrl^C to end): abbb

[q0]-a->[q0]-b->[q1]-b->[q1]-b->[q1] : Accepted

Enter a string (Ctrl^C to end):
```

```
$ ./prog1 test3.txt
----------D F A----------
<states>
q0 q1 q2 q3
<alphabet>
a b
<transitions>
(q0,a)->q1
(q0,b)->q2
(q1,a)->q0
(q1,b)->q3
(q2,a)->q3
(q2,b)->q0
(q3,a)->q3
(q3,b)->q3
<initial state>
q0
<final states>
q0
-------------------------

Enter a string (Ctrl^C to end):

[q0] : Accepted

Enter a string (Ctrl^C to end): a

[q0]-a->[q1] : Rejected

Enter a string (Ctrl^C to end): aa

[q0]-a->[q1]-a->[q0] : Accepted

Enter a string (Ctrl^C to end): aaaa

[q0]-a->[q1]-a->[q0]-a->[q1]-a->[q0] : Accepted

Enter a string (Ctrl^C to end): aab

[q0]-a->[q1]-a->[q0]-b->[q2] : Rejected

Enter a string (Ctrl^C to end): aabb

[q0]-a->[q1]-a->[q0]-b->[q2]-b->[q0] : Accepted

Enter a string (Ctrl^C to end): bb

[q0]-b->[q2]-b->[q0] : Accepted

Enter a string (Ctrl^C to end):
```