Simulate DFA

Objective

The objective of this lab is to write a program that simulates a *deterministic finite automaton* (DFA) using C++ `switch` statement.

Prerequisites

- Make sure **g++** is installed. `$g++ --version$`
- Latest version of the course repository.

```
$cd <path>/<to>/CSC355_Student
$git pull origin
```

- Lab3 directory in your repository (e.g., `$CSC355_telim/Labs/Lab3$`).

Detail

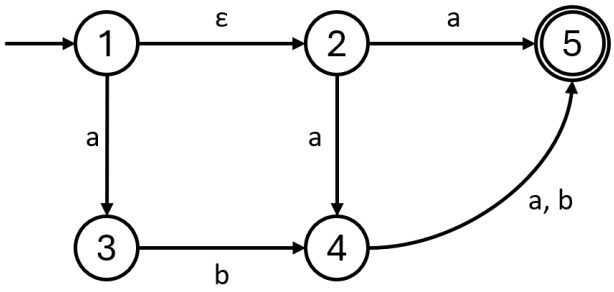

Figure 1: Nondeterministic finite automaton

Figure 1 is a *nondeterministic finite automaton*[1] (NFA) that accepts strings over the alphabet {a, b} with a maximum length of 3 and always starts with the symbol 'a'. Your task for this lab is to convert this NFA to equivalent DFA by hand, then complete the provided skeleton file, **dfa.cpp**, which you can find in **CSC355_Student/Labs/Lab3** directory.

You will use the *subsetConstruction* algorithm (Figure 2) we discussed in class to convert the NFA to DFA. In the text file (**steps.txt**), you will write all your steps in converting NFA to DFA. For the reference, check **CSC355_Student/Examples/DFA/steps.txt**. You do not have to draw transition diagrams like in the example file.

[1]This NFA was referenced from Dr. Glenn A. Lancaster

```
procedure SubsetConstruction(NFA N)
    Dstates := {e-closure(q0)}
    Dtrans := {}
    repeat
        T := an unexplored state in Dstates
            for each input symbol on s do
                X := e-closure(move(T, s))
                if X is not in Dstates then
                    Dstates := Dstates U X
                Dtrans := Dstrans U (T-s->X)
    until all states have been explored
    return (Dstates, Dtrans)
```

Figure 2: Subsetconstruction Pseudocode

In the **dfa.cpp** file, there are four places you have to complete, which are specified with `COMPLETE ME` in inline comments. You will specify all the DFA states in the following `enum`. Note that `ERROR` is a unique state to indicate the invalid transition. The DFA transition will halt if `ERROR` is returned from the `transition` function. See lines 88 - 94 in **dfa.cpp**.

```
enum State {
    // COMPLETE ME
    ERROR
};
```

You will specify the accepting states in the `isAcceptingState()` function. The function returns true if the passed state is an accepting state. Otherwise, false.

```
bool isAcceptingState(State currentState) {
    // COMPLETE ME
    return false;
}
```

You will complete the `toString()` function, which converts `enum` state value (i.e., a number) to string. For example, given the state `A`, you will return `"A"`. The function will return `"ERROR"` for `ERROR`. In the case of trying to convert a non-defined state, you will halt the program using `assert(false);`.

```
string toString(State currentState) {
    string state;
    // COMPLETE ME
    return state;
}
```

You will complete the `transition()` function, which is the core function of this program that simulates DFA state transitions. Given the current state of the DFA and the input symbol, the function returns the transited stated, if available. Otherwise, it will return `ERROR`. You need to use `switch` statement to simulate the transitions. The default of the statement is returning `ERROR`, i.e., either or both invalid `curretnState` or `symbol` is received.

```
State transition(State currentState, char symbol) {
    // COMPLETE ME
    return ERROR;
}
```

How to Compile and Test Your Code

```
$g++ -o dfa dfa.cpp      // Compile dfa.cpp.
$./dfa                   // Run the dfa executable binary file.
```

How to Submit Your Code

You will submit two files:

- Completed **dfa.cpp** file.
- **steps.txt** file that holds steps to convert NFA to DFA.

Submit both files to your GitHub repository under the `Lab3` directory.