OOP Spring 2024 : Assignment 4 (FSM)

Due: Tuesday May 21, 2024 11:59 PM. – You can work in a group of 2.

You are asked to write a C++ program to implement a finite state machine (FSM) simulator. The objective is to enable system designers to model and simulate their systems' behavior while still in the design phase. The simulator reads an input file which contains the FSM description. The FSM description file contains four main sections: (1) machine name; (2) variables section; (3) states section; and (4) a transitions section (see example below).

The sections are defined as follows:

1. The machine name section contain the machine name (which must be the same as the file name that contains the machine (a machine called x would be stored in a file called **x.fsm**).

2. The variables section which starts with the keyword **VAR** followed by a list of machine variables.

3. The states section starts with the keyword **STATES**: (on a separate line) followed with state descriptions each on a separate line. Each state description starts with a state name (e.g., [a]) followed by a list of actions to be performed while the machine is in this particular state. The **state actions** can be one of the following:

a) **<variable> = <op> (<expression>, <expression>, ...)**. Where **<op>** can be **+** or ***** and **<expression>** can either be a constant or a machine variable name. ***<op> can take two or more operands.***

b) **PRINT (<expression>)** which prints out the **<expression>** on the display. **<expression>** can either be a string or a machine variable name.

c) **JMP (<state>)** where **<state>** is a valid state name; this action should transfer the execution to the given state without carrying out any of the actions until reaching to the target state. There must be valid transitions from the current state to the target state, otherwise an error is reported on the display and skip this action.

d) **JMP0 (<m>)** where **<m>** is a valid machine name in the same directory; this action is like **JMP** but should transfer the execution to the start state of the target machine. The start state is supposed to be the first state in the states section.

e) **SLEEP (<amount>)** which causes the simulator to pause for **<amount>** seconds.

f) **WAIT** which causes the simulator to wait for the next transition to take place depending on the user input. You should validate the input and allow the user to re-enter in case of no corresponding transition found. *You must allow for user to input **END** for this action to terminate the execution of the state machine at this point.*

g) **END** which ends the execution of the state machine.

4. The transitions section starts with the keyword **TRANSITIONS**: (on a separate line) followed with transition descriptions each on a separate line. Each transition is a list of three elements: the input value that causes this transition to take place, a source state, and a destination state.

- ***You must follow the described syntax (****two sample input files are given below****).***
- ***You must handle ERRORs and validate the input as much as*** possible (e.g., error occurred while parsing the input file for example action name or number of operands is not valid as specified and validating the input of a specific function). You have to provide *a clear error reporting* (use exceptions and try to use classes like 'invalid_argument' and 'out_of_range' in <stdexcept>).
- You must employ **inheritance** in this assignment in a reasonable and useful way. ***A big part of the points will be deducted if there's no use for the inheritance.***
- You must employ other OOP tools as much as possible such as ***polymorphism, abstract class, pure virtual functions***, … etc.

- You should make use of template (everywhere possible), smart pointers (everywhere possible) design patterns (State + 1 more pattern of your choice), and lambda expression once where appropriate.
- ***Some points will be awarded for a professional evaluation.***
- Effective Modern C++ Design is expected.
- Recommended to print the name of active state and currently executed action preceded by the active machine name on the display.
- You can use template for handling different types of variables.

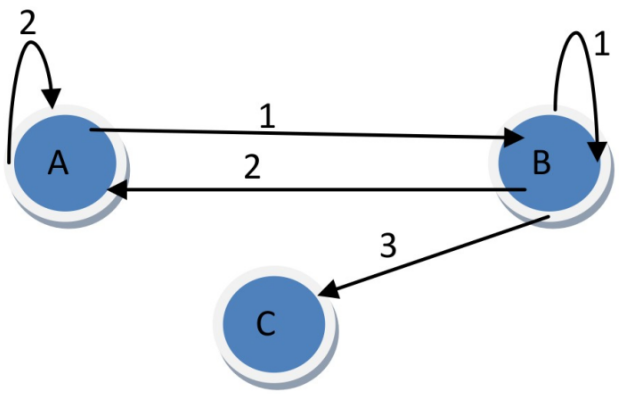
You should submit the following items:
(a) Simple class diagram showing the structure of the classes in your simulator.
(b) The documented C++ source code
(c) Sample test FSMs to be run on your simulator
(d) A short user guide:

- describes the general flow of your project and how it works.
- a short description for each class in your project and basic functionality of each part of the code.
- mentions any reasonable restrictions.
- ***You need to mention how and where you applied (OOP tools mentioned above, template, smart pointers, design patterns, and lambda expression) .***

Example ***fsm1.fsm***:

```
FSM fsm1
VAR X, Y
STATES:
[a] PRINT (State A); X= +(X, 1); SLEEP (10); WAIT;
[b] PRINT (State B); Y= +(Y, 1, 5); SLEEP (10); WAIT;
[c] PRINT (Thank you for using fsm1); PRINT (X); PRINT (Y); END;
TRANSITIONS:
[1] a >> b
[2] b >> a
[2] a >> a
[2] b >> b
[3] b >> c
```

visualized in figure below:



Example fsm2.fsm:

```
FSM fsm2
VAR W, X
STATES:
[A] PRINT (State A); X = *(X, 2); SLEEP 10; WAIT;
[B] PRINT (State B); W = +(3, W, 4); SLEEP (10); WAIT;
[C] PRINT (Thank you for using fsm2); PRINT (X); PRINT (Y); END;
[D] PRINT (State D); JMP (C);
[E] PRINT (Jumping to another FSM); JMP0 (fsm1.fsm);
TRANSITIONS:
[7] D >> A
[1] A >> B
[2] B >> A
[2] A >> A
[1] B >> B
[3] B >> C
[4] B >> E
```

JMP (C) //This action will JUMP to the state C as there are valid transitions from D to C (from D to A, then from A to B, and finally from B to C)