Family Name: . . . . . **Other Names:** . . . . . .

Student ID: . . . . . **Signature** . . . . . .

ENGR 101: Final Exam

2019, June 14 ** WITH SOLUTIONS **

Instructions

- Time allowed: **2 hours**
- Attempt **all** the questions. There are **40 marks** in total.
- Write your answers in this exam paper and hand in all sheets.
- If you think some question is unclear, ask for clarification.
- This test contributes **25%** of your final grade
- You may use unmarked paper Chinese-English translation dictionaries.
- You may write notes and working on this paper, but make sure your answers are clear.

Questions	**Marks**	
1. Finite State machines	[20]	
2. Arduino	[20]	
	TOTAL:	

SPARE PAGE FOR EXTRA ANSWERS

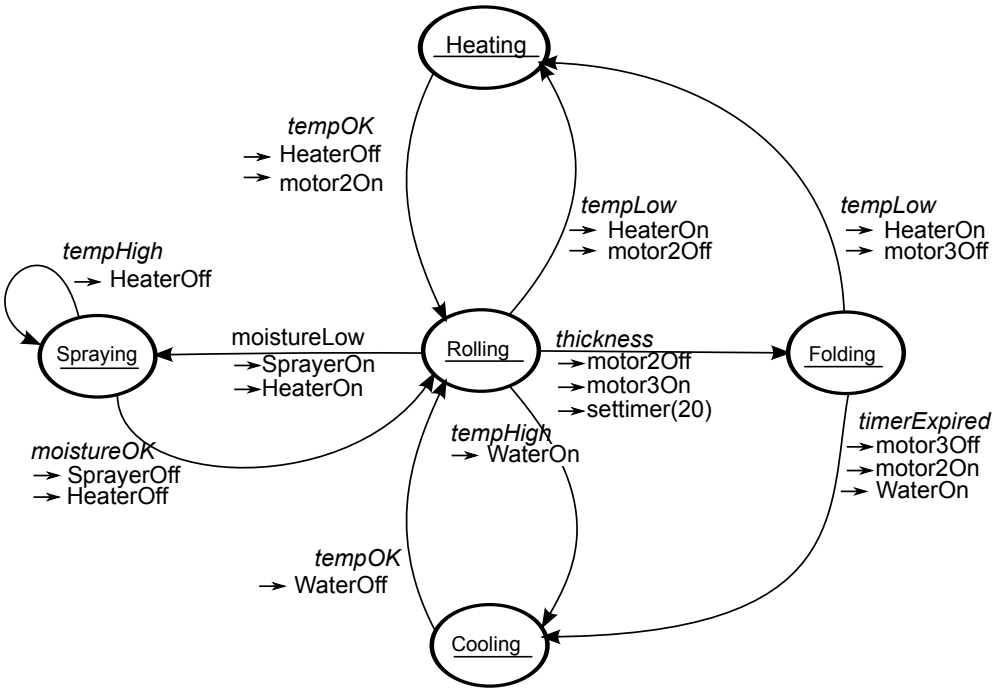
Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 1. FSM controllers **[20 marks]**

The diagram below shows a FSM controller for a machine that processes plastic.

It has sensors for the temperature, moisture, thickness, and timer.

It has actions to turn the heater, sprayer, water, and motors on and off.



(a) **[5 marks]** If the controller starts in the state **Rolling** and gets this sequence of sensor values, what sequence of actions will the controller do, and what state will it end in?

Sensor sequence: *moistureLow*, *tempHigh*, *moistureOK*, *tempLow*, *tempLow*

Actions: sprayerOn, heaterOn, heaterOff, sprayerOff, heaterOff, heaterOn, motor2Off

Final State: Heating

(b) **[5 marks]** Suppose the machine is in the state **Folding**. Give two different sequences of sensors that would make the machine start **Spraying**.

Sequence 1: tempLow, tempOK, moistureLow

Sequence 2: timerExpired, tempOK, moistureLow

(Question 1 continued on next page)

(Question 1 continued)

(c) **[10 marks]**

Complete the design for an event-driven Finite State Machine controller for a food steamer on the next page.

The steamer has four sensors:

- **foodHot**: senses when the food has reached 100°
- **foodAdded**: senses when more food is added to the steamer,
- **foodRemoved**: senses when all the food is removed from the steamer.
- **timerExpired**: senses when the timer has finished.

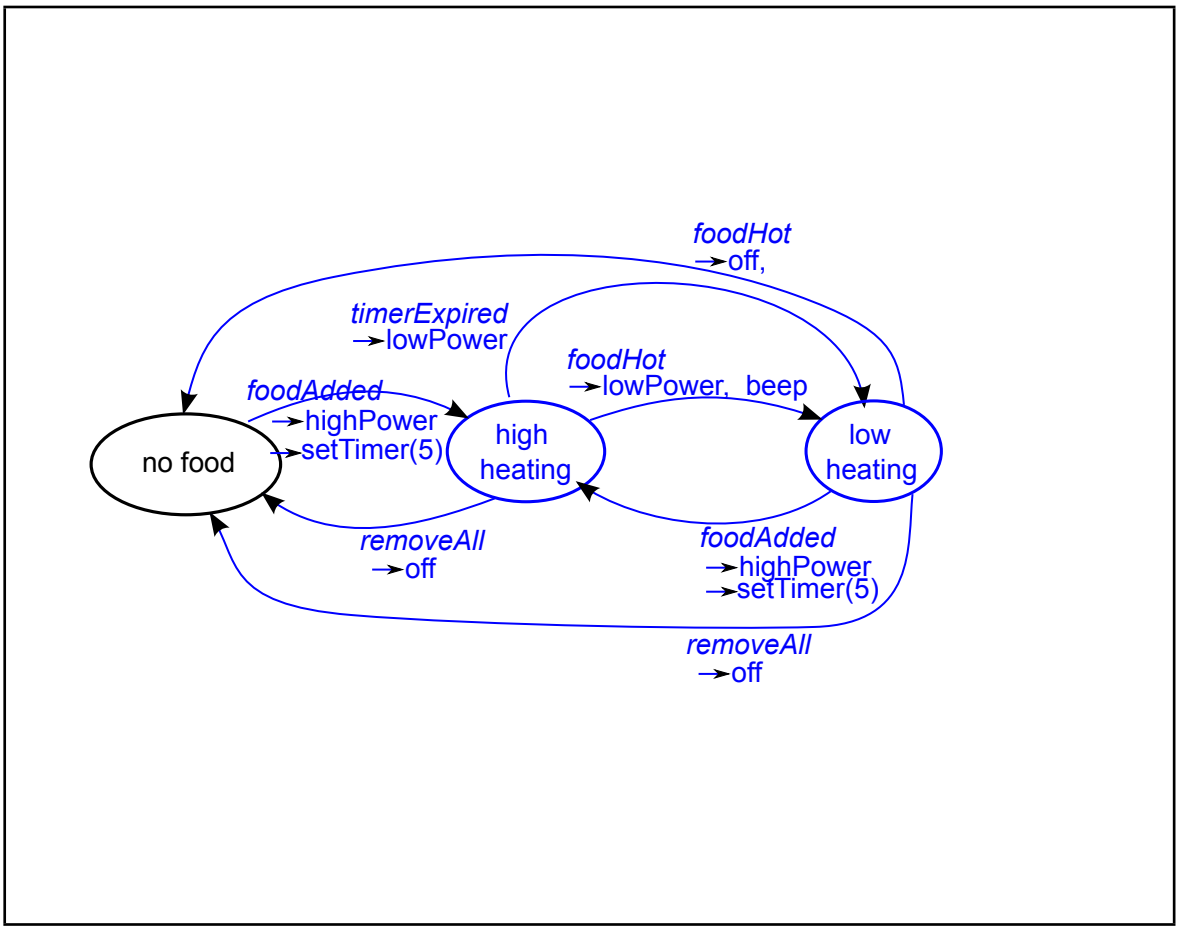
The steamer has five actions:

- → **highPower**: turns the heating element to high power.
- → **lowPower**: turns the heating element to low power.
- → **off**: turns the heating element off.
- → **setTimer(5)**: sets the timer for 5 minutes
- → **beep**: makes a warning sound

How it works:

- The steamer will turn on high power whenever food is added to the steamer, but will never stay on high power for more than 5 minutes—after 5 minutes it will change to low power.
- If all food is removed from the steamer, it will turn off.
- If the steamer is on high power and the temperature sensor senses that the food has become hot, the steamer will make a warning sound and change to low power.
- If the steamer is on low power and the temperature sensor senses that the food has become hot, the steamer will turn off with no warning sound.

Hint: Every transition between states needs exactly one sensor.
Hint: A transition can have zero, one, or more actions on it.


foodHot
→off,
timerExpired
→lowPower
foodHot
→lowPower, beep
foodAdded
→highPower
→setTimer(5)
no food
high
heating
low
heating
removeAll
→off
foodAdded
→highPower
→setTimer(5)
removeAll
→off

Question 2. Arduino **[20 marks]**

(a) **[3 marks]** What will be the output of the following sketch?

```
void setup() {
    Serial.begin(9600);
    int x = 0;
    while (x < 4) {
        Serial.println(x);
        x = x + foo(x);
    }
}

void loop() {
}

int foo (int q) {
    int x = 1;
    return (q + x);
}
```

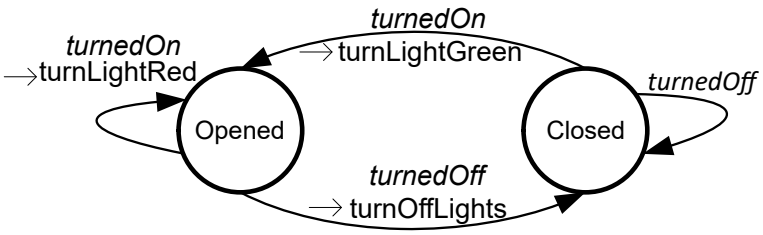
```
0

1

3
```

(b) **[17 marks]** Consider the following event-driven Finite State Machine controller for a simple device.

- The controller has two states and four transitions.
- The controller receives two signals from the device: *turnedOn* and *turnedOff*
- It can perform three different actions on the device: turnLightGreen, turnLightRed, and turnOffLights




```
short state = 1;

// The states
const short Opened =1;
const short Closed =2;
// Sensors
const short turnedOn =3;
const short turnedOff =4;
// the number of the pushbutton pins
const int On_Button = 11;
const int Off_Button = 12;
// the number of the LED pins
const int green = 2;
const int red = 3;

void setup(){
   pinMode(green, OUTPUT);
   pinMode(red, OUTPUT);
   pinMode(On_Button, INPUT);
   pinMode(Off_Button, INPUT);
}

void loop(){
   if (digitalRead(On_Button) == HIGH){
      delay(15); // software debounce
      if (digitalRead(On_Button) == HIGH) {
         inputSignal(turnedOn); // TurnedOn signal will be passed to inputSignal() function
      }
   }
   if (digitalRead(Off_Button) == HIGH){
      delay(15); // software debounce
      if (digitalRead(Off_Button) == HIGH) {
         inputSignal(turnedOff); // TurnedOff signal will be passed to inputSignal() function
      }
   }
}
```

On the facing page, implement the controller in C by completing the inputSignal, trunLightRed and trunLightGreen functions below. inputSignal is passed an signal as an integer number. The state field contains the state of the controller.

```
void inputSignal(short sensor){
    if(state==Opened){
        if(sensor==turnedOn){
            trunLightRed();
            state=Opened;
        }
        else if(sensor==turnedOff){
            trunOffLights();
            state=Closed;
        }
    }
    else if(state==Closed){
        if(sensor==turnedOn){
            trunLightGreen();
            state=Opened;
        }
        else if(sensor==turnedOff){
            state=Closed;
        }
    }
}

/** Make the lights be red */
void trunLightRed(){
    digitalWrite(red, HIGH);
    digitalWrite(green, LOW);
}

/** Make the lights be Green */
void trunLightGreen(){
    digitalWrite(red, LOW);
    digitalWrite(green, HIGH);
}

/** Turn off lights */
void trunOffLights(){
 digitalWrite(red, LOW);
 digitalWrite(green, LOW);
}
```

* * * * * * * * * * * * * * *