EEET2250 Lab Exercise 3 - Classes and State Machines with the OUSB board
These Lab Tasks are due in your Week 11 lab session - no extensions!

Total possible marks: 40 (6% of final mark)

Aim:
These Lab Tasks are aimed at helping you to build up code for the 3rd Lab Test for EEET2250. These Lab Tasks must be demonstrated to your lab tutor by the ***end*** of your **Week 11** lab class, as during your **Week 12** lab class, you will be sitting Lab Test 3 and there will be no time to mark lab tasks – No extensions will be provided! These Lab Tasks are directly related to your Lab Test 3.

The Lab Tasks and Lab Test require you to write an object-oriented (class-based) program to implement a traffic light state machine using the OUSB board. Two classes are required:

- Class to interface to the OUSB board
- Class to implement a traffic light state machine

It is recommended that a generic OUSB access method be implemented for the Lab Tasks (and Lab Tests) as part of the OUSB class. Your code from earlier Lab Tasks can be modified to do this or you can refer to page 22 of the OUSB board reference manual for an example of a generic function.

The tasks specified below are aimed at helping you to break down the program into smaller functional 'modules'. You should attempt to code in modules, checking the functionality of each module before continuing to code (i.e., do not write the whole program at once and expect it all to work!). Once you have finished all the tasks, notify your tutor who will assess your work. The marking for these tasks is binary – it either works or does not. Remember to use the lecture notes and OUSB board manual as references as well as the prescribed textbook.

By completing the Lab Tasks below along with the tasks from Lab exercise 1 and 2, you will have effectively written most of your Lab Test 3 code, except for the required autotester output for the program and input error checking. The Lab Test 3 proforma cpp file is available on EEET2250 Canvas Shell, and you may wish to code these Lab tasks inside the provided Lab Test 3 proforma to gain familiarity with the Lab Test 3 requirements and provided Class structure.

Tasks

1. Write a class to implement access to the OUSB board. Ensure that you have private class data members to store the current values of PORTA, PORTB, PORTC and PORTD, with the appropriate accessor functions and class methods to read from each port and write to PORTB and PORTD (as PORTA and PORTC are read-only) in the class.

    1.1. Identify the attributes/states (i.e., data) and behaviours/actions (i.e., functions) for the OUSB class.

    [2 marks]

    1.2. Write the class definition (i.e., you do not need to write the code to implement the class functions yet!). At this stage, just identify the class data variables and functions that you will need to act on the data.

    [2 marks]

    1.3. Now implement the class methods (outside of the class if more than one line of code is required). Note that the scope operator `::` will be required for the class functions written outside of the class definition.

    [6 marks]

2. Write a class to implement a single traffic light as a state machine, where the traffic light is composed of three lamps: Red, Yellow, and Green. The only lamp sequence possible is **Green**→**Yellow**→**Red** (then back to **Green** and repeat). Only one lamp can be on at a time.

   Ensure that you have private class variables to store the current state of the traffic light (either as a single state to represent all three lamps, or three separate lamp states). Include appropriate accessor functions and functions to change the traffic light state in the class.

   2.1. Identify the possible states (i.e., lamp combinations) of the traffic light and draw the *software* state machine diagram.

   2.2. Identify the attributes/states (i.e., data) and behaviours/actions (i.e., functions) for the traffic light state machine class.

   [2 marks]

   2.3. Write the class definition (i.e., you do not need to write the code to implement the class functions yet!). At this stage, just identify the class data variables and functions that you will need to act on the data.

   [2 marks]

   2.4. Now implement the class functions (outside of the class if more than one line of code is required). Note that the scope operator `::` will be required for the class functions written outside of the class definition. At this stage you only need to print the current state to the console.

   [6 marks]

3. Compare your OUSB class from Task 1 and traffic light class from Task 2 to the two class definitions provided in the Lab Test 3 proforma. Are your class designs similar to those provided? You may wish to modify your class definitions from Tasks 1 and 2 to more closely match the class definitions provided in the Lab Test 3 proforma, as you will have access to these class definitions during Lab Test 3, but this is optional and up to you!

4. Using your OUSB and traffic light classes, now write a program to that instantiates both classes so that the current state of the traffic light is also displayed on PORTB of the OUSB Board. Assume that the red, yellow and green states are represented by the following PORTB combinations:
   - **Red ON**: turn on bit 0 (bits 1, 2 and 3 are off)
   - **Yellow ON**: turn on both bits 1 and 2 (bits 0 and 3 are off)
   - **Green ON**: turn on bit 3 (bits 0, 1 and 2 are off)

   Note: Only the lower 4 bits of PORTB will be used as the upper for bits are reserved for another function, so you might want to use bit setting routines to only modify the bits of interest.

   4.2. Remember the six (6) steps of engineering design: draw flowcharts and write pseudocode to represent the logical flow of your program.

   [2 marks]

   4.3. Write your program to take in two command line parameters:
   - 'R', 'Y' or 'G' (one letter only): to indicate which state to start the traffic light on
   - Number to indicate the number of state transitions to run the traffic light for

e.g., `myProgram.exe R 5` // Starts the traffic light state to red first, then changes the traffic light state 5 times to end on the yellow (R→G→Y→R→G→Y)

[4 marks]

4.4. Perform the following error checks in your program:

- If more than two parameters are entered by the user onto the command line
- If a letter other than R, Y or G is entered onto the command line for the first parameter
- If a number less than zero or greater than 50 (i.e., 0 and 50 are accepted numbers) is entered onto the command line for the second parameter

[2 marks]

4.5. Write two operator overloaded functions for the OUSB class found in the Lab Test 3 proforma that work on PORTB of the OUSB board. One function should be for writing a value (from a variable of type unsigned short) to PORTB using the insertion operator (<<), and the other function for reading the value stored at PORTB and saving it to a variable of type unsigned short by using the extraction operator (>>).

To test your solution you can use the follow code written in `main()` which will read and write to the OUSB board:

```
OUSB myOUSBObj;
unsigned short value = 3;
myOUSBObj << value;        // write value to portb

myOUSBObj.writePORTB(8);   // write 8 to portb

myOUSBObj >> value;        // read portb and save to
                           // variable value

cout << "value =" << value << endl; // value = 8
```

You might find it easier to start by adding the following prototypes in OUSB class definition as public methods:

```
void operator<<(const unsigned short &val);
void operator>>(unsigned short &val);
```

Note: the operator overload functions can also be written as friend functions however, as the operators' only need to insert and extract values to a variable of type unsigned short, they can be made as methods of the OUSB class.

[6 marks]

5. In the Lab Test 3, you will be using both PORTB and PORTD to represent 1 set of traffic lights. To put the problem into perceptive, the three LEDs (Red, Yellow and Green) that are wired into the OUSB board prototype area will be used to represent the actual traffic light

lamps, whereas the standard bank of 8 LEDs connected to PORTB will be used to represents a simplified troubleshooting panel inside the traffic light control box (e.g. the box that is normally on the side of the road at an intersection). The troubleshooting (PORTB) lights will have the light states as well as other information encoded in the 8-bit. Both PORTB and PORTD are used to control the Red, Yellow and Green LEDs on the prototype area using the following configuration:

Traffic Light State	**OUSB connections**	**Meaning**
Red	PORTB, bit 0 PORTD, bit 4	Troubleshoot bit Red LED
Yellow	PORTB, bit 1 and bit 2 PORTD, bit 5	Troubleshoot bits Yellow LED
Green	PORTB, bit 3	Troubleshoot bit and Green LED (combined)

5.1. Re-write task 4.3 so that your state machine now switches on the Red, Yellow and Green LEDs as well as the new PORTB trouble shooting bits as specified in the table above.

[3 marks]

5.2. When writing to PORTB and PORTD you should not be modifying any of the other bits in the PORT except bits 0, 1, 2 and 3 for PORTB and bits 4 and 5 of PORTD. To demonstrate this, load PORTB with the value 240 prior to running your code. Similarly, you can load the value 123 to PORTD and the query the value after your code has completed.

Note: Selected bits in PORTD are used for USB and RS-232 support, therefore bits: PD7, PD2 will not stick and appear to always read logic zero. See OUSB board schematic to see how they are wired.

[3 marks]

Once you have finished these tasks, you should not leave the laboratory after completing the tasks on this sheet - ensure that you are prepared for the Lab Test 3.