Assessed Lab: Traffic Lights control in C

This assessed lab tests your practical knowledge of controlling LEDs, wired up through a breadboard to a Raspberry Pi 2, using a C program.

The tasks should be performed on a Raspberry Pi 2, which you can get on loan from the department, hooked up via a KVM to a monitor and keyboard of one of the machines in the Linux lab (EM 2.50). You can try the same exercise in your own time with the Pi2 directly connected to a keyboard and monitor, e.g. at home. In each case, you need to wire up the external devices (LED and Button) using a breadboard and jumper cables as described below.

Wiring up external devices

3 **LEDs**, as output devices, should be connected to the RPi2 using **GPIO pins 10, 11 and 13**, using a breadboard. **Optionally,** use a button, wired to **pin 26**. You will need resistors to control the input to the LEDs and to the button.

Note: If you don't have a third LED in your Raspberry Pi kit, use **GPIO pin 47** to control the on-chip green activity LED (as in Tutorial 2) and use it for the green traffic light.

Task: Developing a C program to simulate traffic lights (Week 5)

The task of this assessed lab is to write a C program to simulate a traffic light protocol with the 3 LEDs that are wired to the Raspberry Pi2. This needs to implement one sequence of blinking LEDs as follows:

```
RED ON                 indicating stop
RED ON, YELLOW on      indicating attention
GREEN ON               indicating go
PAUSE or BUTTON PRESS
GREEN ON               indicating go
YELLOW blinking        indicating attention
RED ON                 indicating stop
```

You should configure the mode for all LEDs, turn on or off the relevant LEDs in each phase of the pattern by using C one-liners each, directly using **literals** (rather than variables), to write to the relevant registers. This means to set, for example, the 2nd bit in register 1 use `*(gpio+1) = 0b100` but **not** `*(gpio+1) = 1 << shift` because `1 << shift` is an *expression* that needs to be evaluated, whereas `0b100` directly stands for the value, thus a literal.

What is a literal?
A literal value is a constant, which doesn't need further evaluation in the program. This can be a value in any number system, e.g. `7` for a decimal number, `0b111` for a binary number or `0x07` for a decimal number. Values that defined using a `#define` are also literals. On the other hand, any expression containing a variable is not a literal: `7 << pin` is not a literal, because it needs evaluation of the left-shift operation (<<).

Start from the sample code for controlling an LED (in C) discussed in class: tut_led.c, which implements this control for the green, on-chip activity LED (pin 47). You will have to change the code to use several different pins. See the detailed discussion in the Tutorial 2 slides.

Optionally, connect a button to **pin 26** and use it as a pedestrian's request to turn the traffic lights to red again. This means, instead of the `PAUSE` phase, probe for a button input, and if it is spotted initiate the second phase in the sequence above.

contributes 10% to CW2 (5% overall); tick-off of this lab will be in the labs of Week 5 (Edinburgh)

For guidance on how to test the pins in your wiring and about pin numberings, refer to the Labsheet on LED control in C (Labsheet 2). For this exercise, the pin numbers (always using **BCM numbering**) are:

red LED	pin 10 (BCM)
yellow LED	pin 11 (BCM)
green LED	pin 13 (BCM)
button	pin 26 (BCM) (optional)

Demos of the completed implementations should be run in the **Lab slot of Week 5** (Edinburgh). We will tick off completed solution, by testing the functionality and by examining key parts of the code. Only the LED implementation is required, the button functionality is optional and may earn extra points. The program should be run from the command-line, and it should be possible to compile the program as a stand-alone binary from the command-line.