Event groups

Debug printing

Implement a debug print task for printing debug/log message. Debug print task must the only task that uses standard output. Priority of debug task must idle task priority + 1. Other tasks must run on a higher priority than debug print task. Tasks communicate the message to print by sending data to a queue.

Debug print task waits on a queue. Queue items contain **a constant string** and three numbers. The string must be a literal or other type of string that does not change at run time. The string is a format string that is accepted by printf/sprinf. When debug() is called (see examples below) it constructs a debug event and sends it to queue. For example we could call:

```
debug("Button: %d pressed. Count: %d\n", 2, count, 0);
```

Function must always send a timestamp to the queue with the message to print. The print task prints the timestamp at the beginning of the line followed by the message

The queue maintains ordering of the items so the debug messages will print out in the same order as they were sent to the queue.

Note that the implementation below is far from complete. It is provided as an example of partial solution. Feel free to use this or create your own implementation.

```
void debug(const char *format, uint32_t d1, uint32_t d2, uint32_t d3);


struct debugEvent {
    const char *format;
    uint32_t data[3];
};



void debugTask(void *pvParameters)
{
    char buffer[64];
    debugEvent e;

    // this is not complete! how do we know which queue to wait on?

    while (1) {
        // read queue
        xQueueReceive(syslog_q, &e, portMAX_DELAY);
        snprintf(buffer, 64, e.format, e.data[0], e.data[1], e.data[2]);
        std::cout << buffer;
    }
}
```

Exercise 1

Implement a program with four tasks and an event group. Task 1 waits for user to press a button. When the button is pressed task 1 sets bit 0 of the event group. Tasks 2 and 3 wait on the event bit with an infinite timeout. When the bit is set the tasks start running their main loops. In the main loop each task prints task number and number of elapsed ticks since the last print at random interval that is between 1 – 2 seconds. Task 4 is debug print task. Tasks 1 – 3 must run at higher priority than the debug task. The tasks must use the debug function described above for printing.

Exercise 2

Implement a program with five tasks and an event group. Task 4 is a watchdog task to monitor that tasks 1 – 3 run at least once every 30 seconds. Tasks 1 – 3 implement a loop that waits for button presses. When a button is **pressed and released** the task sets a bit in the event group, prints a debug message, and goes back to wait for another button press. Holding the button without releasing must block the main loop from running. Each task monitors one button.

Task 4 prints “OK” and number of elapsed ticks from last “OK” when all (other) tasks have notified that they have run the loop. If some of the tasks does not run within 30 seconds Task 4 prints “Fail” and the number of the task(s) not meeting the deadline and then Task 4 suspends itself.

Task 5 is the debug print task. It must run at a lower priority than tasks 1 – 4.