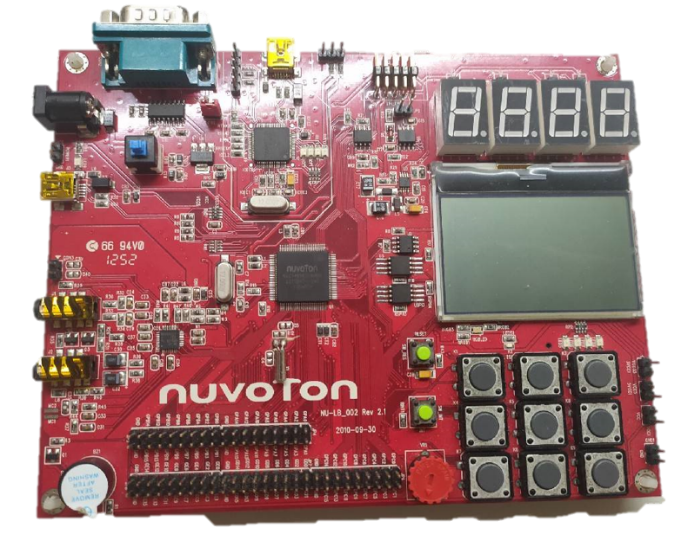
**嵌入式作業系統 LAB 2**

系所：通訊四 學號 :409430030 姓名:翁佳煌

**<實驗器材及環境>**

NUC 140 開發板 FreeRTOSv10.4.1



**<實驗過程與方法>**

# Basic:

首先根據下圖1，

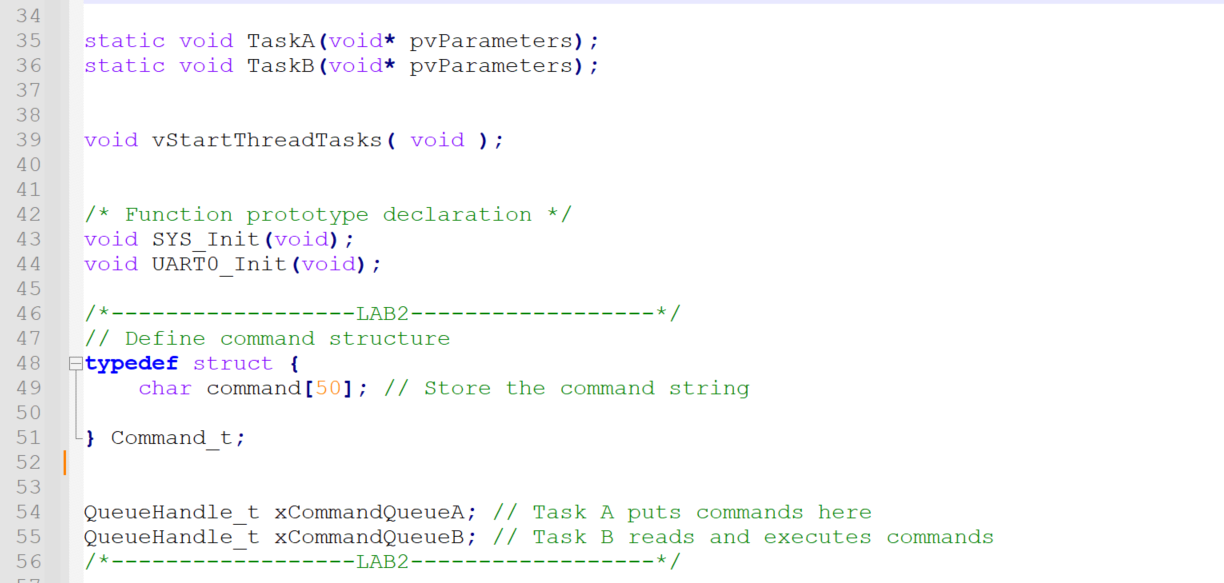
35~36行:定義了兩個任務：TaskA 和 TaskB。

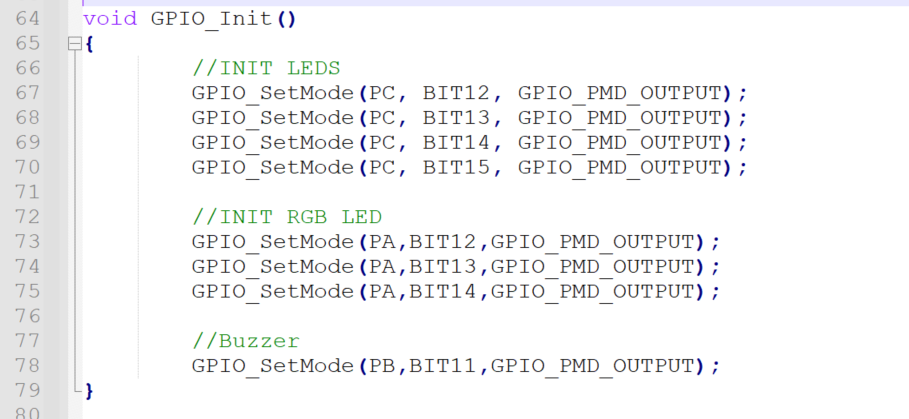
43~44行:宣告了 SYS\_Init 和 UART0\_Init 的函數，負責初始化系統和 UART。

48~51行:定義了一個名為 Command\_t 的結構，其中包含一個大小為50的字元陣列 (command[50])。這個結構用在存儲命令字串。

54~55行:宣告了兩個queue，分別為xCommandQueueA 和 xCommandQueueB。這些隊列的類型是 QueueHandle\_t，這是一個指向FreeRTOS隊列的句柄。但在basic中，只會用到xCommandQueueA，其中TaskA負責發送命令到這個queue裡面，而TaskB負責讀取這個queue並做出相應的動作。

64~79行:負責初始化對應的GPIO，有LEDs、RGB LED、Buzzer。





▲圖1

再來看到下圖2中的main函數。

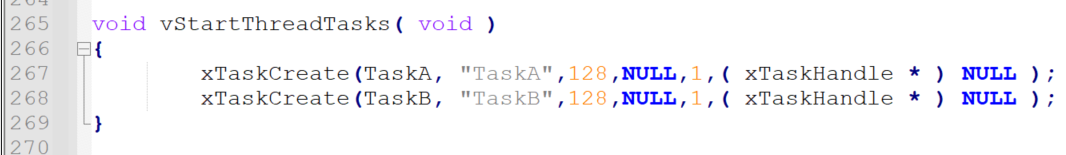
xCommandQueueA 和 xCommandQueueB，使用xQueueCreate 函數建立兩個隊列。這些隊列用於任務之間的溝通，可以用於在不同的任務中傳遞命令。

vStartThreadTasks()如下圖3，啟動任務的函數。包括創建和啟動在前文提到的 TaskA 和 TaskB。

vTaskStartScheduler()啟動FreeRTOS任務調度器，開始執行定義的任務。

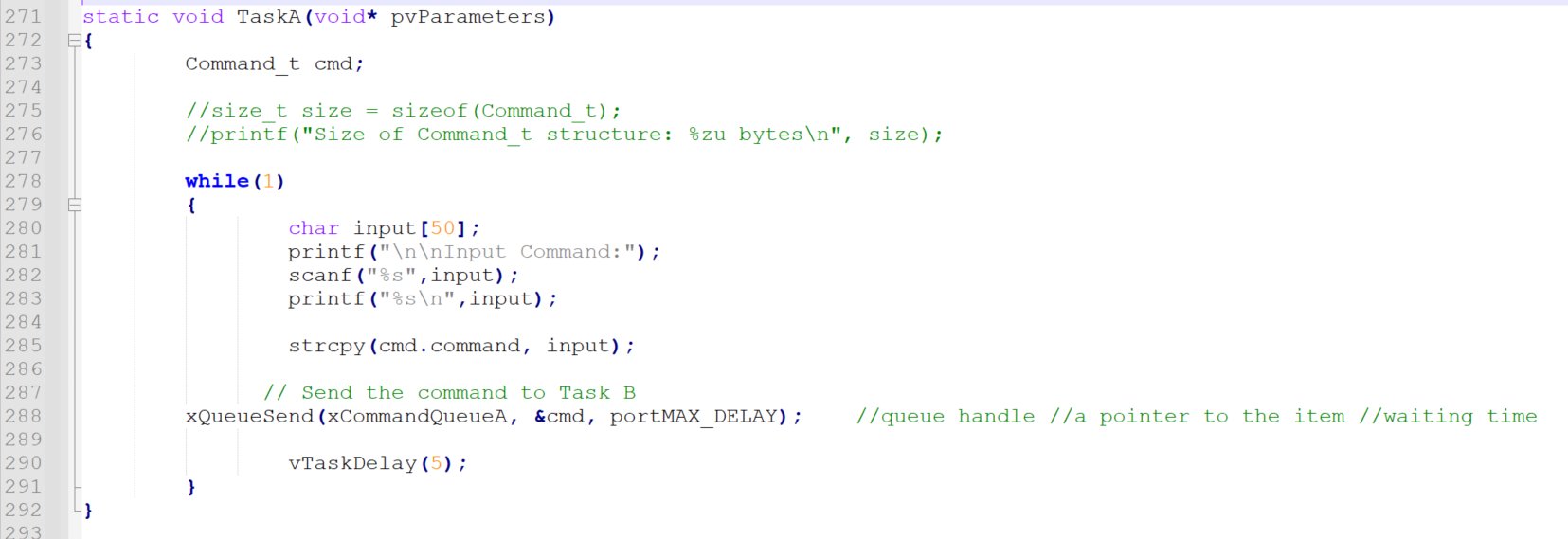


▲圖2



▲圖3

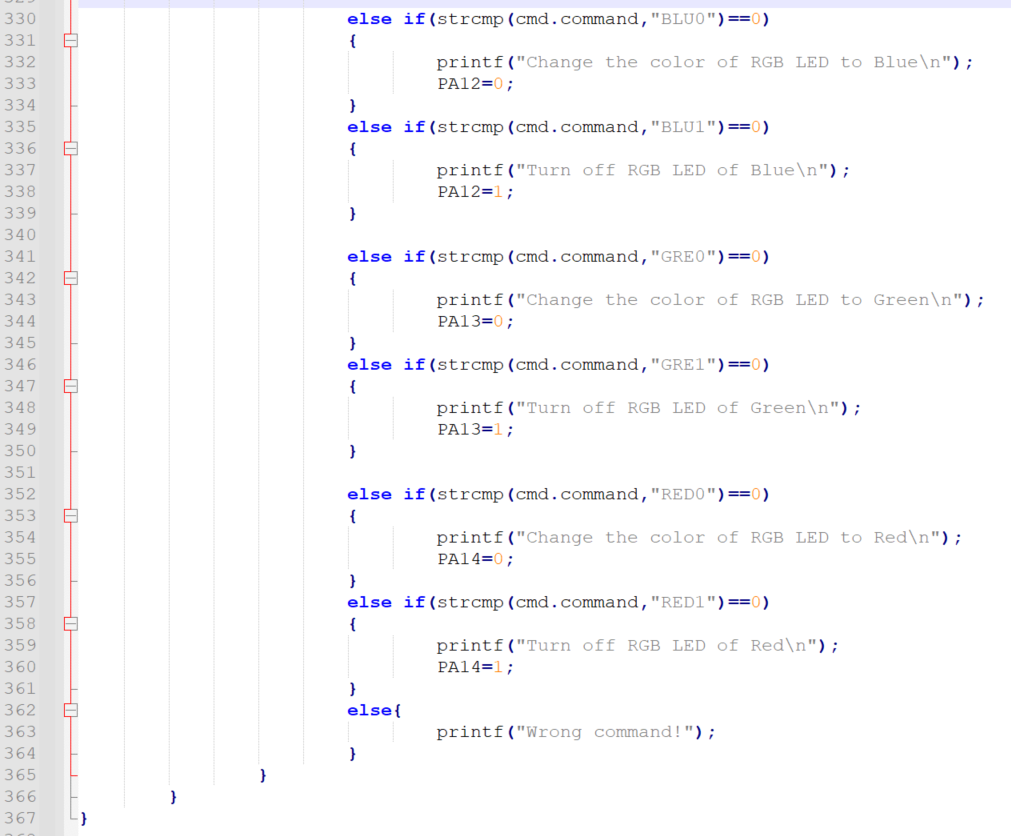
接下來看到下圖4，TaskA的部分，在 TaskA 函數中，程式進入一個無窮迴圈，等待使用者輸入指令。使用 scanf 函數讀取輸入的指令，將其複製到 cmd 結構的 command 成員中，然後透過 xQueueSend 函數將命令發送到 xCommandQueueA 佇列中。portMAX\_DELAY 參數表示如果佇列已滿，則任務將等待直到有空間可用。



▲圖4

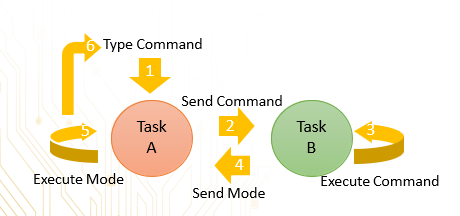
再來是下圖5，TaskB的部分，在 TaskB 函數中，程式進入一個無窮迴圈，等待從 xCommandQueueA 佇列接收命令。使用 xQueueReceive 函數，如果成功接收到命令（pdPASS），則進行命令處理。在這裡，根據不同的命令，控制不同的外設，例如打開或關閉LED、啟動或停止蜂鳴器、改變RGB LED的顏色等。如果接收到未知的命令，則輸出錯誤訊息。





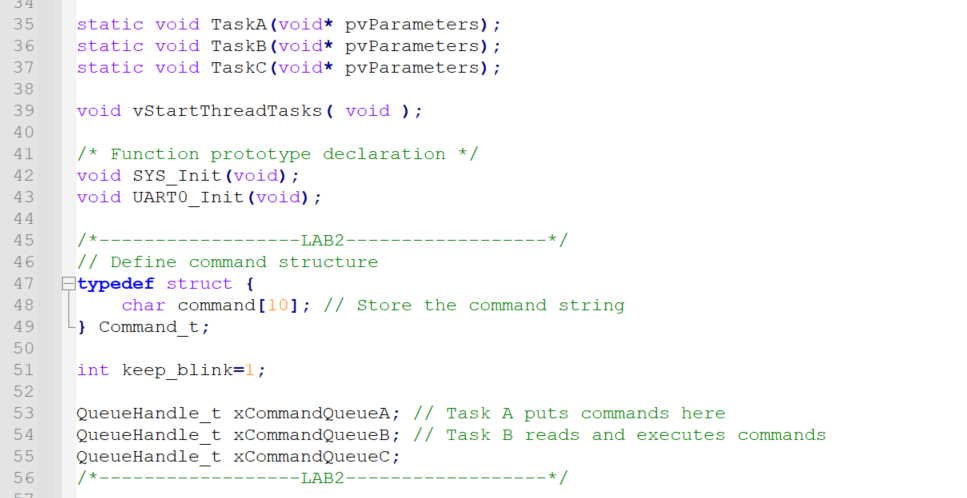
▲圖5

# Bonus:



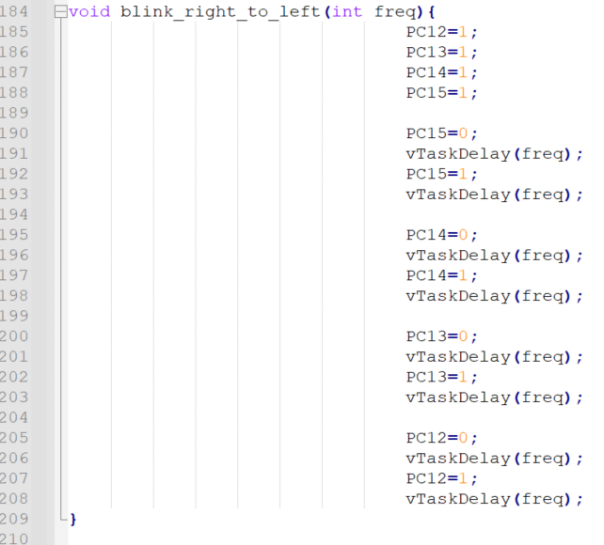
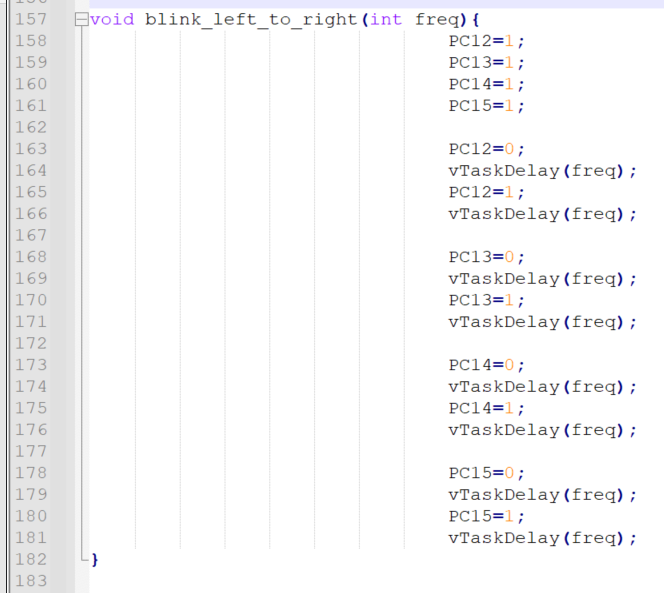
在來是LAB2 Bonus的部分，35~55行絕大部分都與Basic大同小異，唯一差別是為了達到讓LED不斷閃爍的功能，只靠兩個Task是無法達成的，因為TaskA接收到TaskB後，如果要不斷閃爍LED的話，會導致整個Input Command卡死無法順利運行，因此只能另外建一個TaskC以及queueC來負責執行LED不斷閃爍的功能。

此外，後續初始化的部分都與Basic相同，就不再贅述。



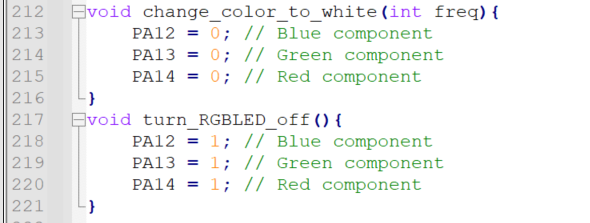
▲圖6

下圖7的部分是控制LED從左到右或是從右到左閃爍的函式，應用簡單的開關和延遲來達成，等等後面會使用到。



▲圖7

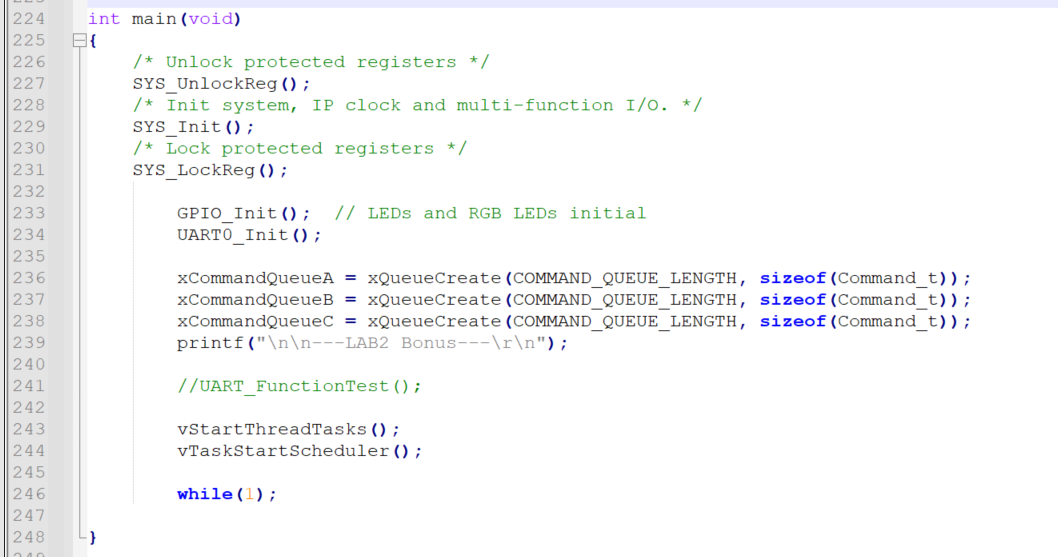
下圖8，212~216行負責把LED顏色改成白色，217~221行負責關掉所有RGB\_LED。



▲圖8

接下來看到下圖9的main函數，236~239行創建三個不同的佇列 xCommandQueueA、xCommandQueueB 和 xCommandQueueC，用於在不同的任務間傳遞 Command\_t 類型的命令。vStartThreadTasks()啟動任務。這可能是用來初始化並啟動FreeRTOS中的各種任務的函數。

vTaskStartScheduler()啟動FreeRTOS調度器，開始任務的執行。



▲圖9

看到下圖10，負責接收使用者輸入的指令，然後通過佇列將該指令發送到xCommandQueueA。根據不同的指令，它還會等待 Task B 發送的相關模式或參數到xCommandQueueB，TaskA在藉由xQueueReceiveB來做相對應的動作。另外，若是LED0的命令，則會進入到376行的判斷式，此時，會接收queueB裡面的mode，在把mode傳到QueueC裡面，讓TaskC去讀取queueC來完成不斷閃爍LED，這樣才不會導致TaskA無法進入下一輪Input Command的步驟。



▲圖10

看到下圖11，TaskB的部分，負責處理從 xCommandQueueA 佇列接收到的命令。它根據不同的命令執行相應的操作，並且對於某些命令，它會等待從 TaskA 來的額外信息，並將回應發送到 xCommandQueueB 佇列。

**變數宣告：**

Command\_t cmd：保存接收到的命令的結構。

int mode;：用於保存 LED 模式的整數變數。

char white：用於保存是否將 RGB LED 顏色更改為白色的字符變數。

**接收命令：**

if (xQueueReceive(xCommandQueueA, &cmd, portMAX\_DELAY) == pdPASS)：從 xCommandQueueA 佇列接收命令。

**處理不同的指令：**

對於 "LED0"，它打開LED，然後等待來自 TaskA 的模式信息（0 或 1），並將模式發送到 xCommandQueueB 佇列。

對於 "LED1"，它關閉LED。

對於 "BUZ0" 和 "BUZ1"，它開啟或關閉蜂鳴器。

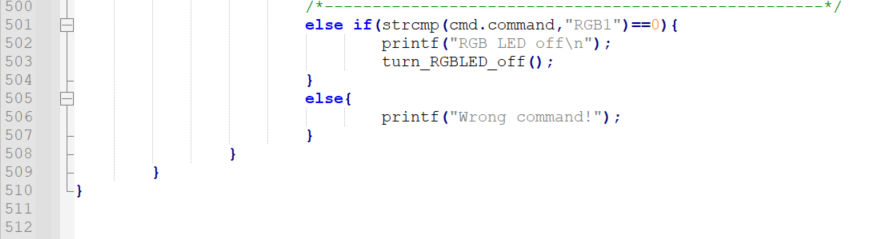
對於 "BLU0"、"GRE0"、"RED0"，它改變 RGB LED 的顏色並等待是否轉換為白色的信息，然後將信息發送到 xCommandQueueB 佇列。

對於 "BLU1"、"GRE1"、"RED1"，它關閉相應顏色的 RGB LED。

對於 "RGB1"，它將 RGB LED 關閉。

對於未知的指令，它輸出錯誤信息。





▲圖11

最後是下圖12，TaskC的部分，它負責處理從 xCommandQueueC 佇列接收到的模式信息。根據接收到的模式，該任務會啟動不同的 LED 閃爍模式。

**變數宣告：**

int mode：保存接收到的模式信息的整數變數。

int freq = 30：保存閃爍頻率的整數變數，初始值我設為 30。

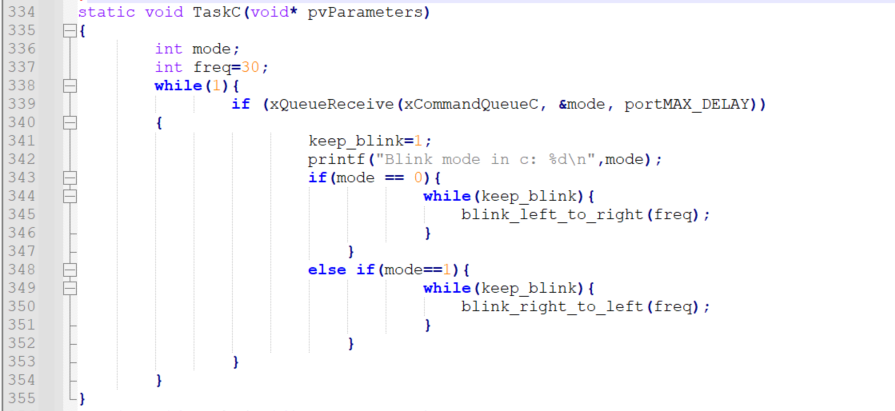
**接收模式信息：**

if (xQueueReceive(xCommandQueueC, &mode, portMAX\_DELAY))：從 xCommandQueueC 佇列接收模式信息。

**處理不同的模式：**

如果模式是 0，表示左到右的閃爍模式，它將 keep\_blink 設置為 1，然後在 keep\_blink 為1的情況下執行 blink\_left\_to\_right 函數，該函數負責左到右的 LED 閃爍。這個循環會一直執行，直到 keep\_blink 變為 0。如果模式是 1，表示右到左的閃爍模式，同樣地，它將 keep\_blink 設置為 1，然後在 keep\_blink 為1的情況下執行 blink\_right\_to\_left 函數，該函數負責右到左的 LED 閃爍。

閃爍將持續到TaskA中下一個命令輸入後，會將keep\_blink設為0，從而停止迴圈，讓LED燈熄滅。



▲圖12

**<心得與收穫>**

這次LAB2在Bonus的部分花了比較多時間在處理連續閃爍LED的部分，由於不確定到底要不要連續，因此兩個版本都有做，這個過程讓我更深入地了解了嵌入式系統的開發和FreeRTOS的使用，這些經驗將對我未來非常有幫助。