Mbed Lab 6 Report
Interrupts, timers, tasks and RTOS

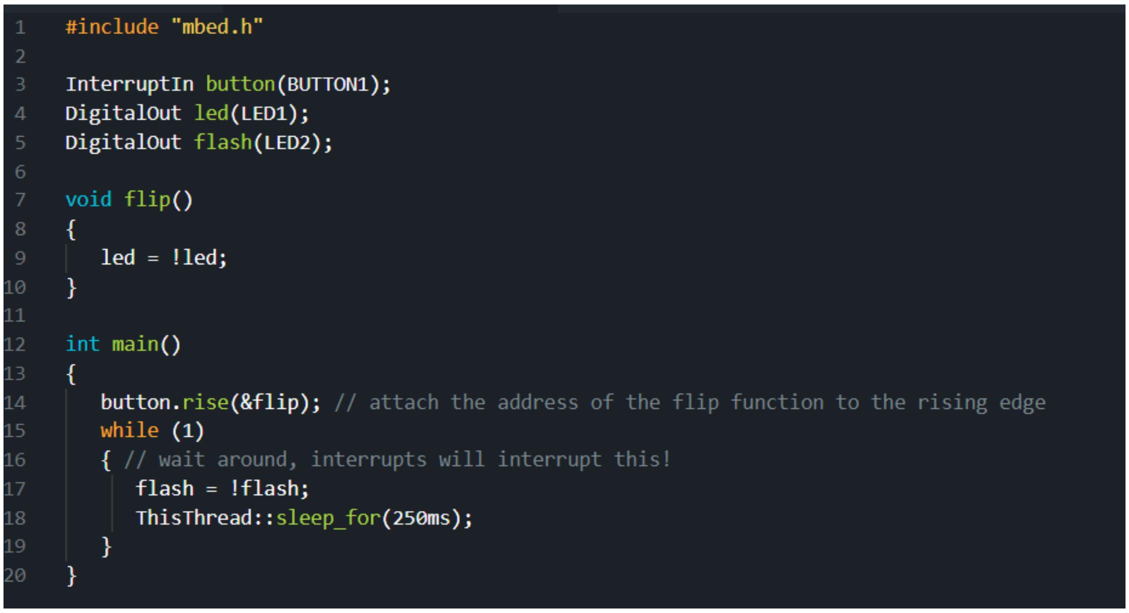
109033130 唐振家

一、Lab Description

1、Interrupt

說明：

利用Interrupt物件，使得當按鈕按下時，能夠中斷，同時做到兩個loop



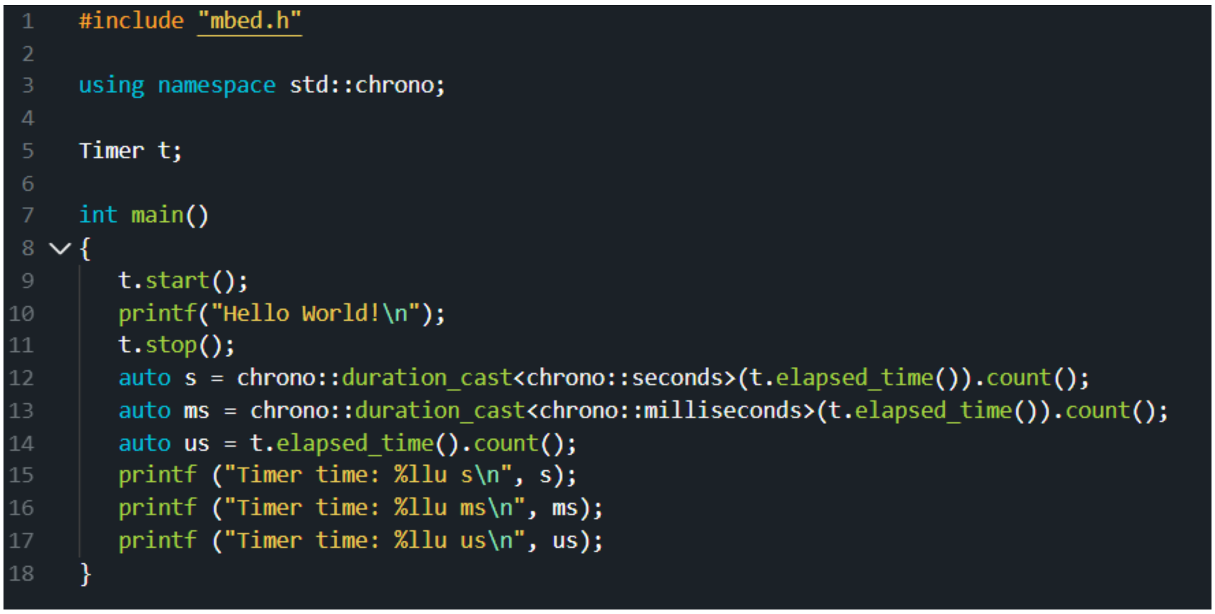
```cpp
#include "mbed.h"

InterruptIn button(BUTTON1);
DigitalOut led(LED1);
DigitalOut flash(LED2);

void flip()
{
    led = !led;
}

int main()
{
    button.rise(&flip); // attach the address of the flip function to the rising edge
    while (1)
    { // wait around, interrupts will interrupt this!
        flash = !flash;
        ThisThread::sleep_for(250ms);
    }
}
```

一、Lab Description

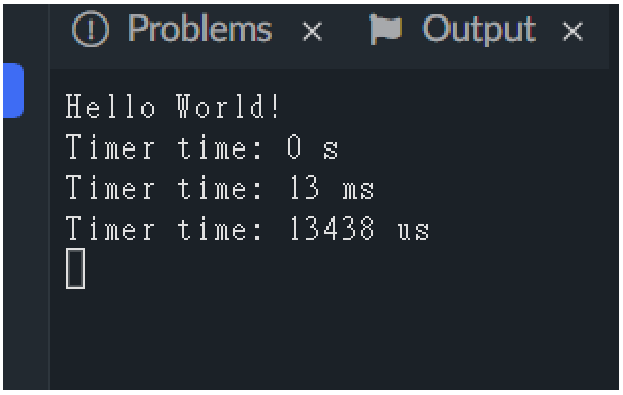
2、Simple Timer

說明：

使用Timer物件計時，並用chrono底下的duration_cast去更改顯示秒數的單位



```cpp
#include "mbed.h"

using namespace std::chrono;

Timer t;

int main()
{
    t.start();
    printf("Hello World!\n");
    t.stop();
    auto s = chrono::duration_cast<chrono::seconds>(t.elapsed_time()).count();
    auto ms = chrono::duration_cast<chrono::milliseconds>(t.elapsed_time()).count();
    auto us = t.elapsed_time().count();
    printf ("Timer time: %llu s\n", s);
    printf ("Timer time: %llu ms\n", ms);
    printf ("Timer time: %llu us\n", us);
}
```



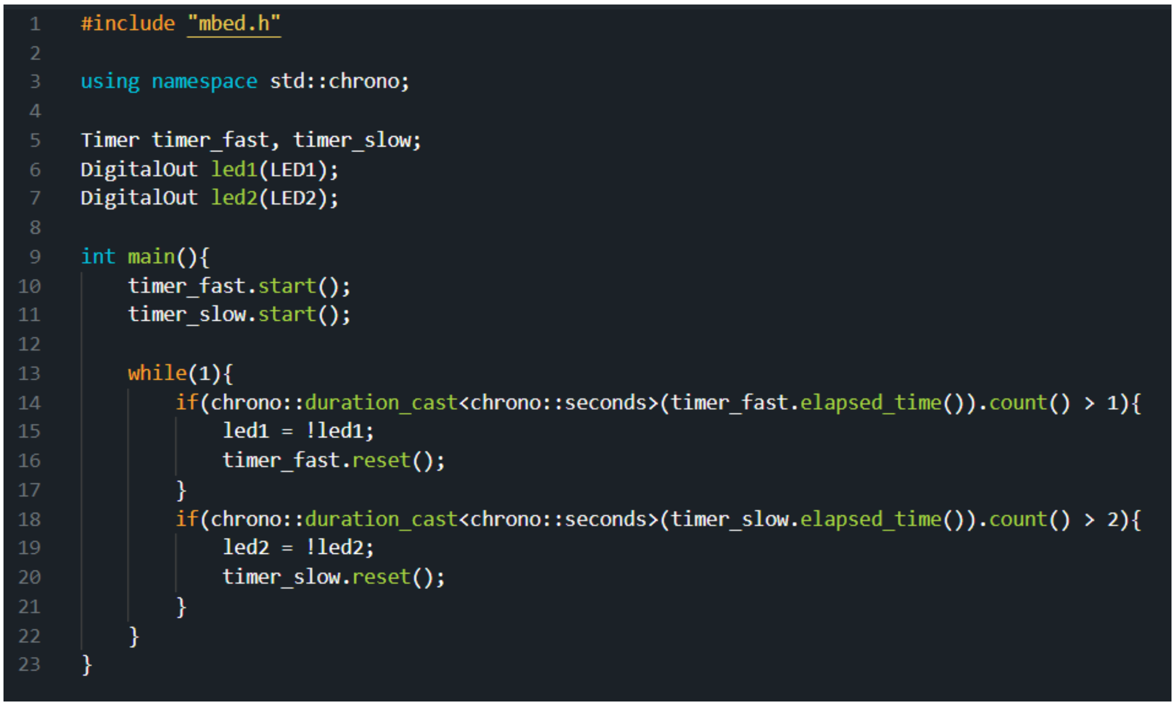
```
Hello World!
Timer time: 0 s
Timer time: 13 ms
Timer time: 13438 us
```

一、Lab Description

3、Multiple Timer

說明：

　　使用多個Timer，使用Timer物件計時，使Led燈可以分別以1、2秒閃爍，每亮或暗就執行一次reset，後重新計時。



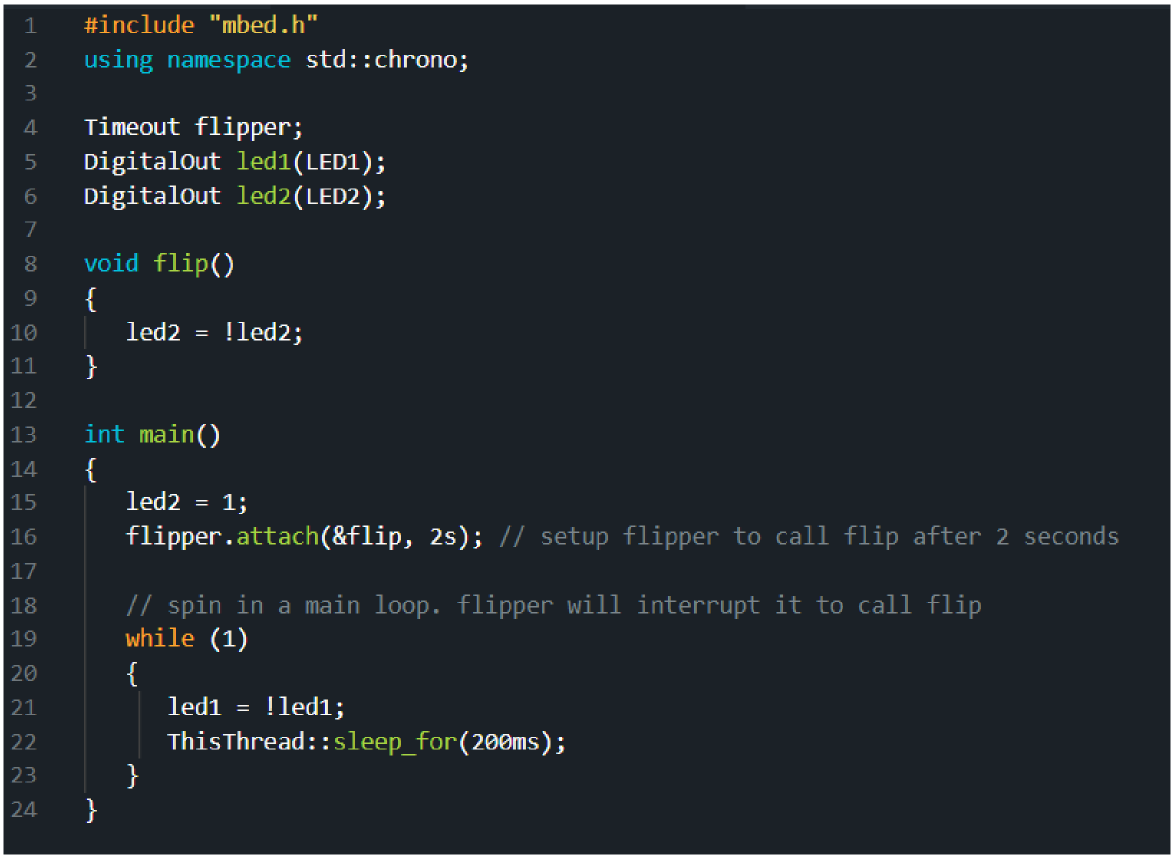
```cpp
#include "mbed.h"

using namespace std::chrono;

Timer timer_fast, timer_slow;
DigitalOut led1(LED1);
DigitalOut led2(LED2);

int main(){
    timer_fast.start();
    timer_slow.start();

    while(1){
        if(chrono::duration_cast<chrono::seconds>(timer_fast.elapsed_time()).count() > 1){
            led1 = !led1;
            timer_fast.reset();
        }
        if(chrono::duration_cast<chrono::seconds>(timer_slow.elapsed_time()).count() > 2){
            led2 = !led2;
            timer_slow.reset();
        }
    }
}
```

一、Lab Description

4、Simple Timeout

說明：

使用Timeout物件，使Led燈在亮2秒之後變暗，而在while loop中，讓另一Led燈以200ms閃爍。



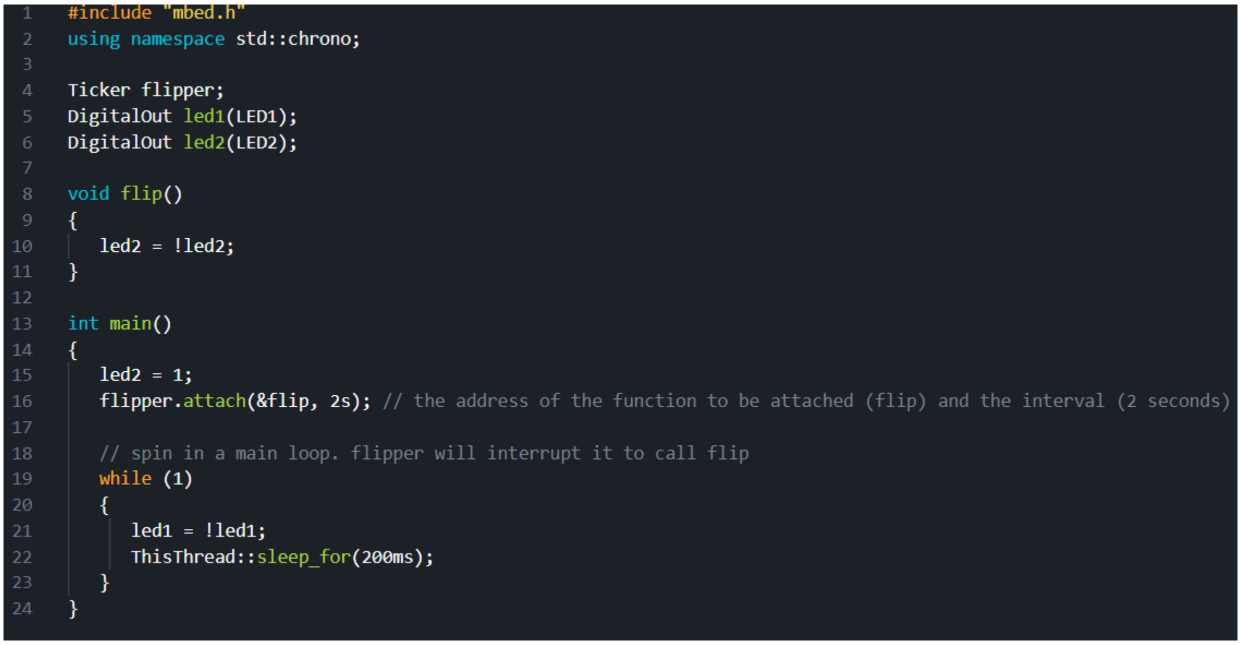
```
#include "mbed.h"
using namespace std::chrono;

Timeout flipper;
DigitalOut led1(LED1);
DigitalOut led2(LED2);

void flip()
{
    led2 = !led2;
}

int main()
{
    led2 = 1;
    flipper.attach(&flip, 2s); // setup flipper to call flip after 2 seconds

    // spin in a main loop. flipper will interrupt it to call flip
    while (1)
    {
        led1 = !led1;
        ThisThread::sleep_for(200ms);
    }
}
```

一、Lab Description

5、Simple Ticker

說明：

使用Ticker物件，使Led燈可以以2秒為循環作明暗變換，而在while loop中，讓另一Led燈以200ms閃爍。



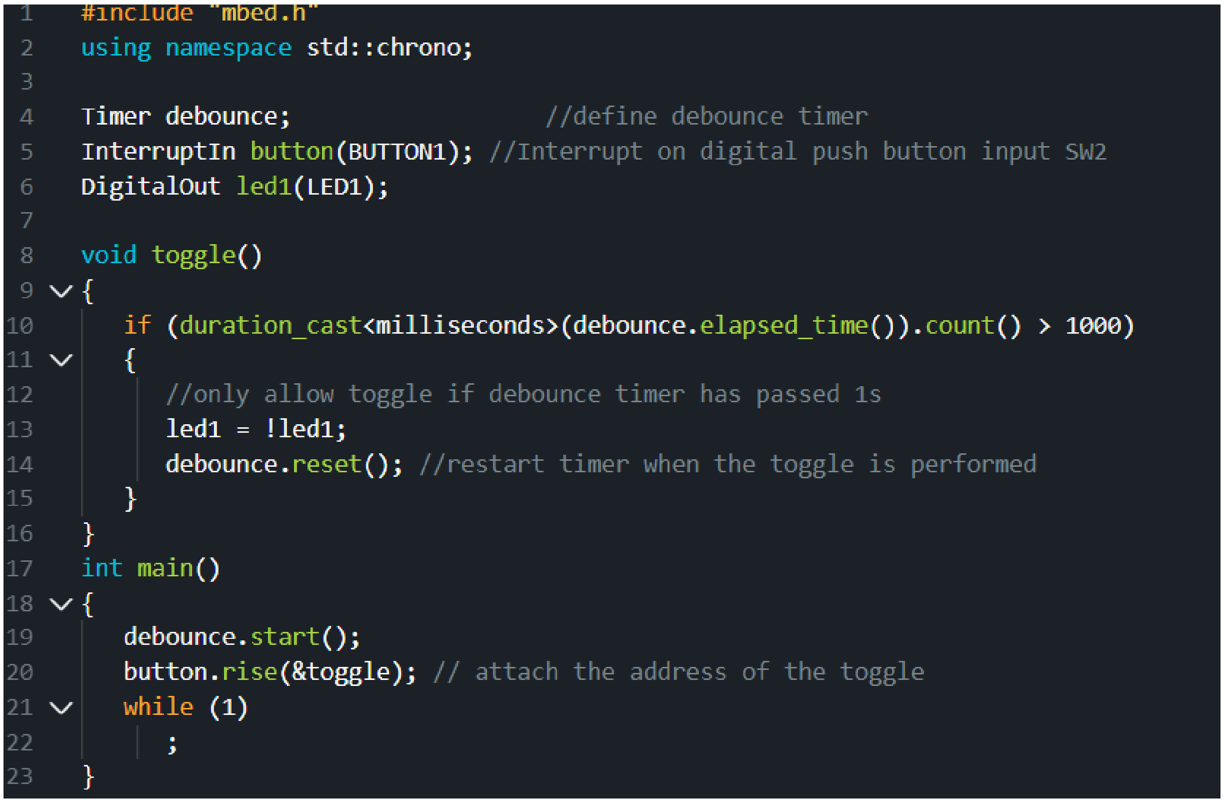
```
#include "mbed.h"
using namespace std::chrono;

Ticker flipper;
DigitalOut led1(LED1);
DigitalOut led2(LED2);

void flip()
{
    led2 = !led2;
}

int main()
{
    led2 = 1;
    flipper.attach(&flip, 2s); // the address of the function to be attached (flip) and the interval (2 seconds)

    // spin in a main loop. flipper will interrupt it to call flip
    while (1)
    {
        led1 = !led1;
        ThisThread::sleep_for(200ms);
    }
}
```

一、Lab Description

6、Application -- Debounce

說明：

使用Timer、Interrupt物件做到按鈕以及防彈跳，當按鍵按下去之後，進入function toggle中，確保距離上次觸發經過1秒之後，改變Led燈狀態，然後重新計時。



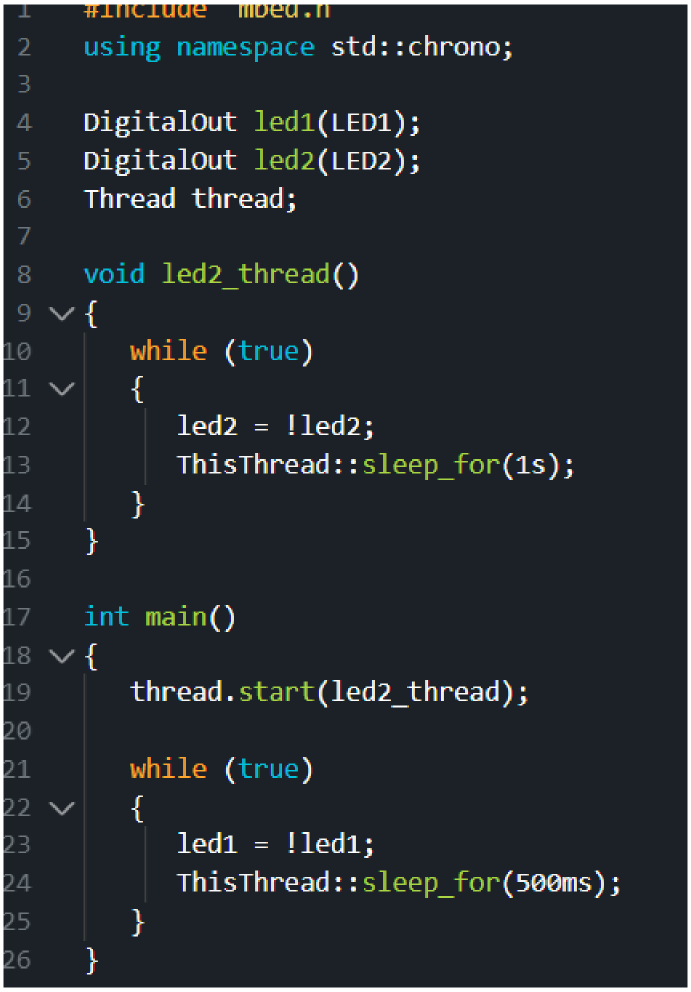
```
#include "mbed.h"
using namespace std::chrono;

Timer debounce;                    //define debounce timer
InterruptIn button(BUTTON1); //Interrupt on digital push button input SW2
DigitalOut led1(LED1);

void toggle()
{
    if (duration_cast<milliseconds>(debounce.elapsed_time()).count() > 1000)
    {
        //only allow toggle if debounce timer has passed 1s
        led1 = !led1;
        debounce.reset(); //restart timer when the toggle is performed
    }
}
int main()
{
    debounce.start();
    button.rise(&toggle); // attach the address of the toggle
    while (1)
        ;
}
```

一、Lab Description

7、Multi-Thread Example

說明：

使用Thread物件，開始成後除了while loop可以執行Led燈以1s亮暗變化外，也可以另外執行另一顆Led燈以500ms亮暗



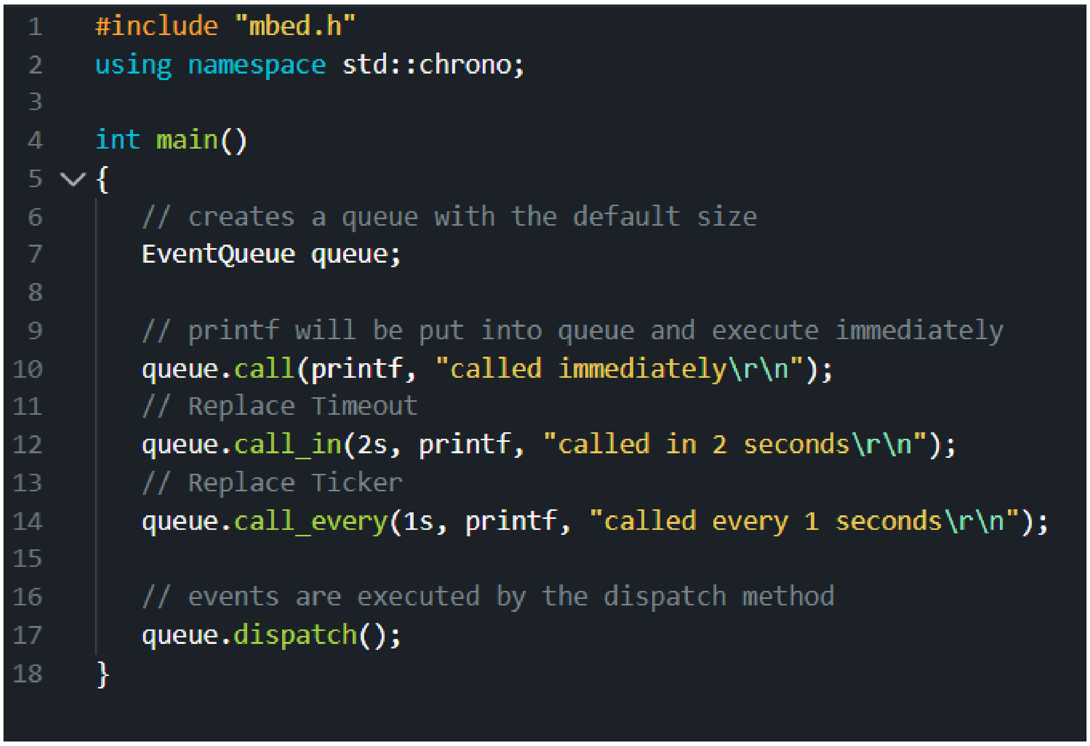
```cpp
#include "mbed.h"
using namespace std::chrono;

DigitalOut led1(LED1);
DigitalOut led2(LED2);
Thread thread;

void led2_thread()
{
    while (true)
    {
        led2 = !led2;
        ThisThread::sleep_for(1s);
    }
}

int main()
{
    thread.start(led2_thread);

    while (true)
    {
        led1 = !led1;
        ThisThread::sleep_for(500ms);
    }
}
```

一、Lab Description

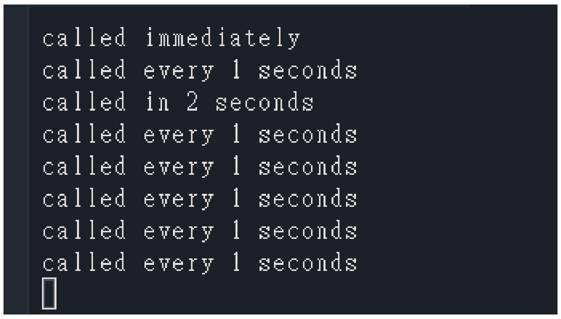
8、EventQueue Example

說明：

　　使用EventQueue物件，利用call馬上執行指令，call_in取代Timeout，在開始2秒後執行，call_every取代Ticker，在開始後每1秒執行一次



```cpp
#include "mbed.h"
using namespace std::chrono;

int main()
{
    // creates a queue with the default size
    EventQueue queue;

    // printf will be put into queue and execute immediately
    queue.call(printf, "called immediately\r\n");
    // Replace Timeout
    queue.call_in(2s, printf, "called in 2 seconds\r\n");
    // Replace Ticker
    queue.call_every(1s, printf, "called every 1 seconds\r\n");

    // events are executed by the dispatch method
    queue.dispatch();
}
```



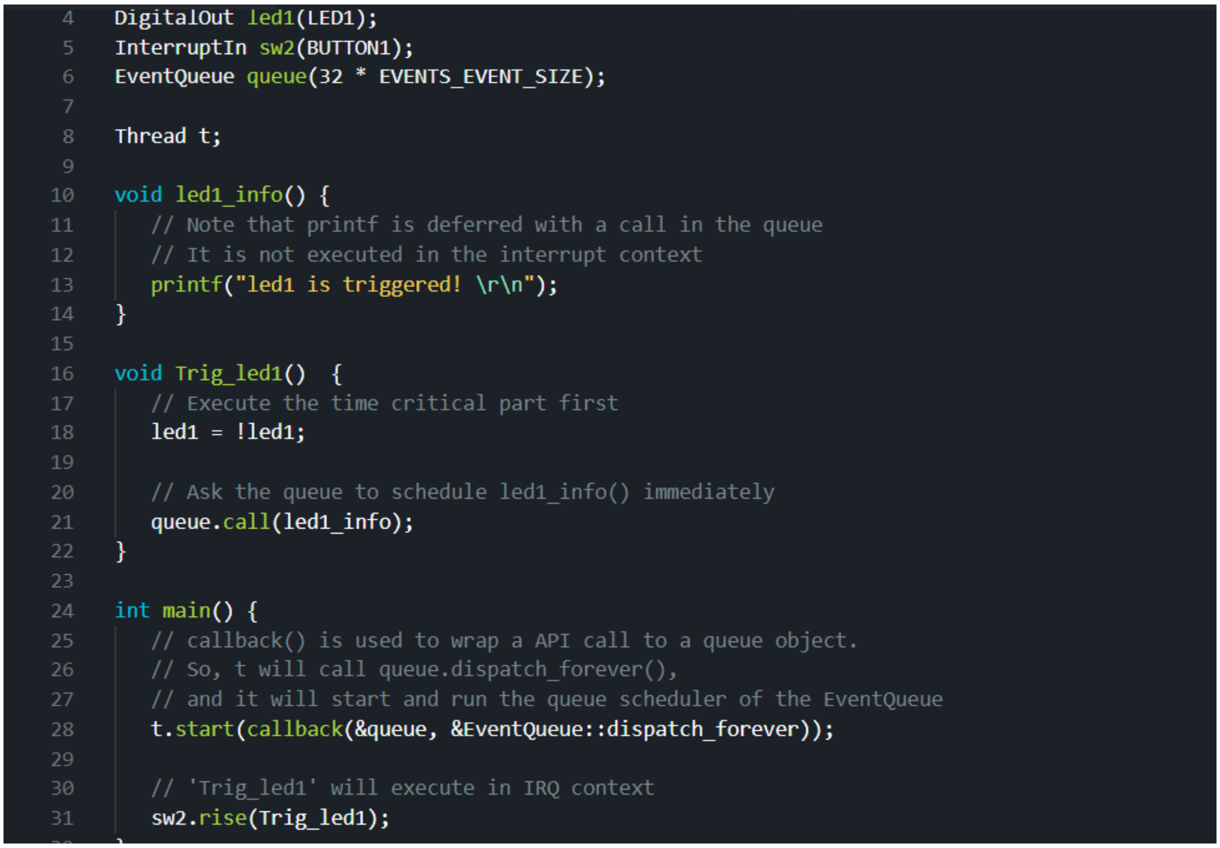
```
called immediately
called every 1 seconds
called in 2 seconds
called every 1 seconds
called every 1 seconds
called every 1 seconds
called every 1 seconds
called every 1 seconds
```

一、Lab Description

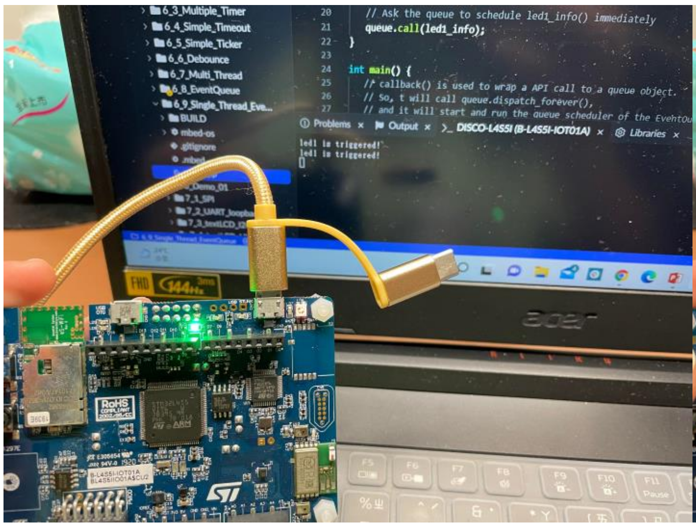
9、EventQueue in a Thread

說明：

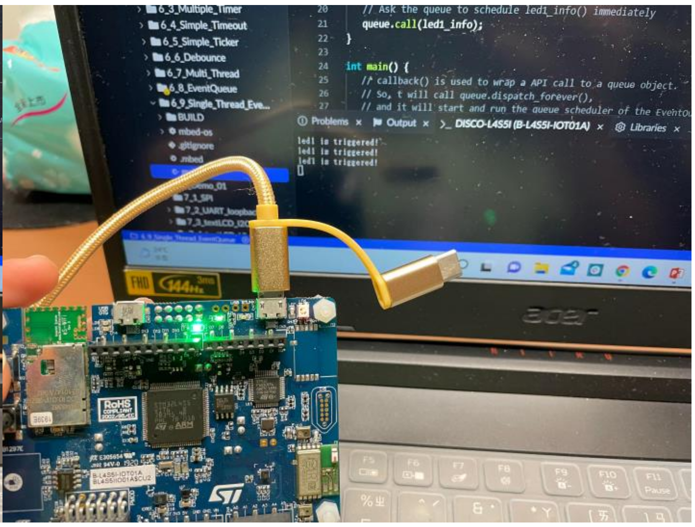
　　使用EventQueue、 Interrupt、Thread物件，利用Thread呼叫callback function，使EventQueue開始啟用，當按鍵被按下時進到Trig_led1，led1變換後， EventQueue因為已被啟動所以進入led1_info，print出字樣。



```
DigitalOut led1(LED1);
InterruptIn sw2(BUTTON1);
EventQueue queue(32 * EVENTS_EVENT_SIZE);

Thread t;

void led1_info() {
    // Note that printf is deferred with a call in the queue
    // It is not executed in the interrupt context
    printf("led1 is triggered! \r\n");
}

void Trig_led1()  {
    // Execute the time critical part first
    led1 = !led1;

    // Ask the queue to schedule led1_info() immediately
    queue.call(led1_info);
}

int main() {
    // callback() is used to wrap a API call to a queue object.
    // So, t will call queue.dispatch_forever(),
    // and it will start and run the queue scheduler of the EventQueue
    t.start(callback(&queue, &EventQueue::dispatch_forever));

    // 'Trig_led1' will execute in IRQ context
    sw2.rise(Trig_led1);
```



```
led1 is triggered!
led1 is triggered!
```



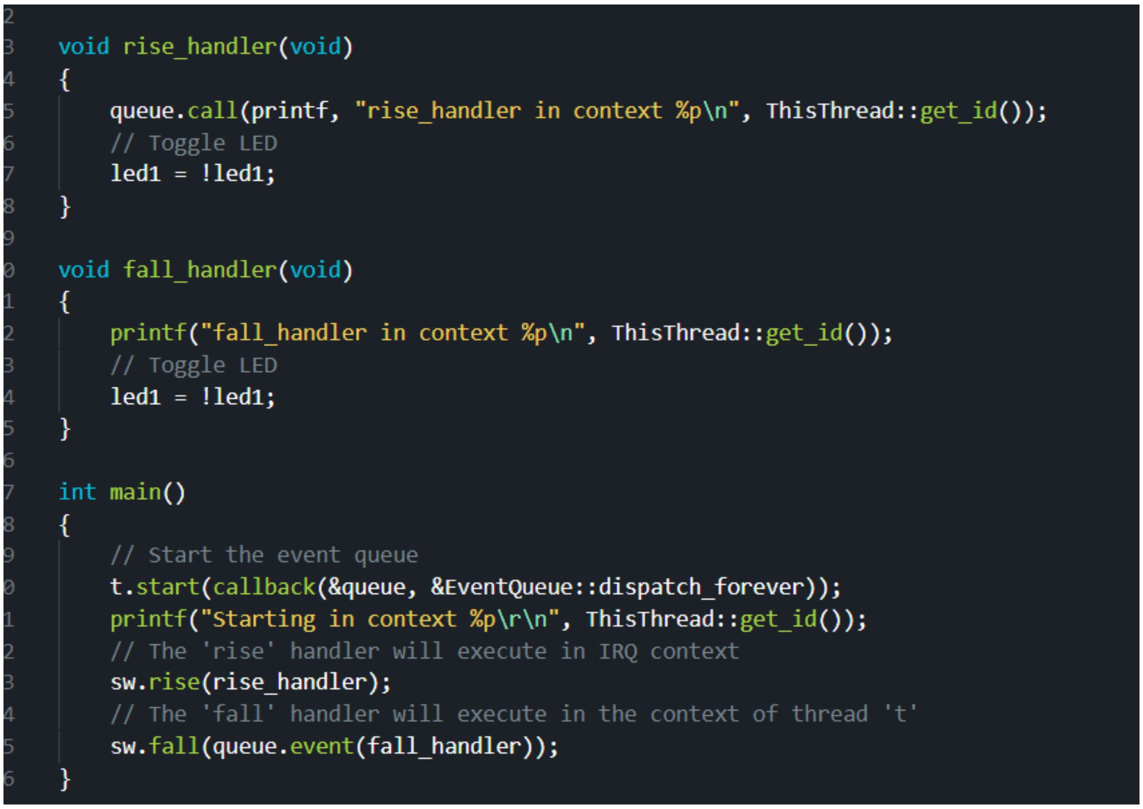
```
led1 is triggered!
led1 is triggered!
led1 is triggered!
```

一、Lab Description

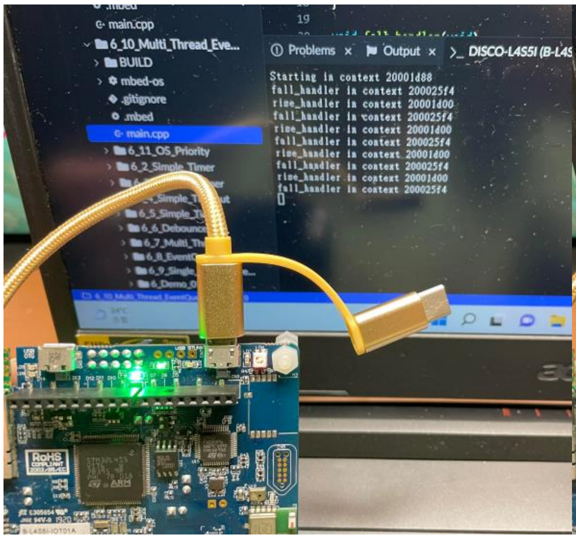
10、Scheduling of EventQueue Calls

說明：

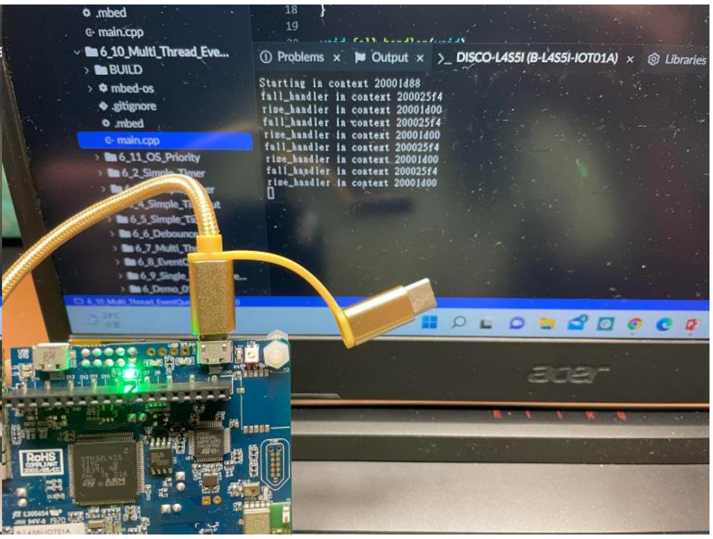
使用EventQueue、 Interrupt、Thread物件，利用Thread呼叫callback function，使EventQueue開始啟用，先印出字體，當按鍵被按下時進到rise_handler，使用EventQueue 的call印出資料，且led1變換，當按鍵被芳開時進到fall_handler，印出資訊後led1變換



```
void rise_handler(void)
{
    queue.call(printf, "rise_handler in context %p\n", ThisThread::get_id());
    // Toggle LED
    led1 = !led1;
}

void fall_handler(void)
{
    printf("fall_handler in context %p\n", ThisThread::get_id());
    // Toggle LED
    led1 = !led1;
}

int main()
{
    // Start the event queue
    t.start(callback(&queue, &EventQueue::dispatch_forever));
    printf("Starting in context %p\r\n", ThisThread::get_id());
    // The 'rise' handler will execute in IRQ context
    sw.rise(rise_handler);
    // The 'fall' handler will execute in the context of thread 't'
    sw.fall(queue.event(fall_handler));
}
```



```
Starting in context 20001d88
fall_handler in context 200025f4
rise_handler in context 20001d00
fall_handler in context 200025f4
rise_handler in context 20001d00
fall_handler in context 200025f4
rise_handler in context 20001d00
fall_handler in context 200025f4
rise_handler in context 20001d00
fall_handler in context 200025f4
```



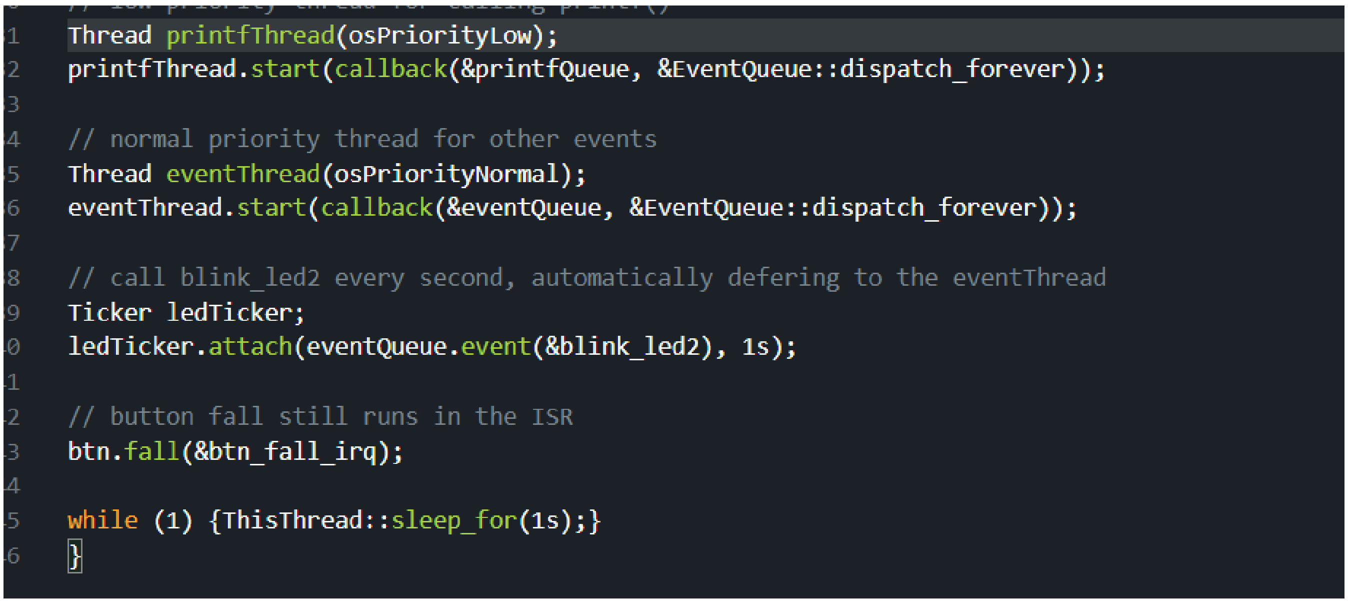
```
Starting in context 20001d88
fall_handler in context 200025f4
rise_handler in context 20001d00
fall_handler in context 200025f4
rise_handler in context 20001d00
fall_handler in context 200025f4
rise_handler in context 20001d00
fall_handler in context 200025f4
rise_handler in context 20001d00
```

一、Lab Description

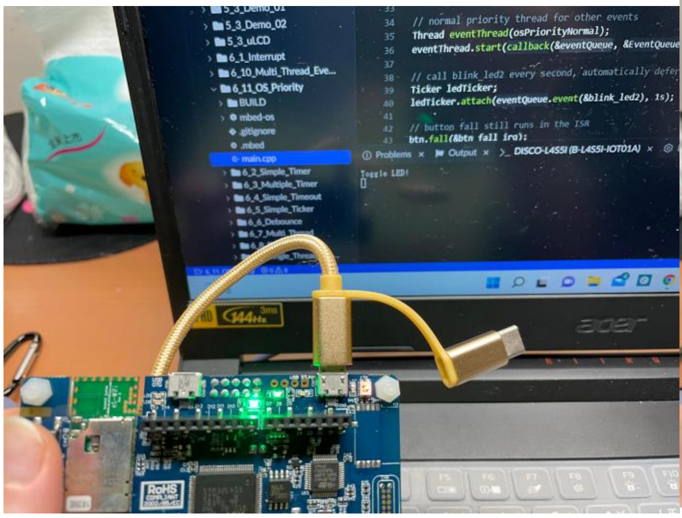
11、Two Threads of EventQueue with Different Priorities

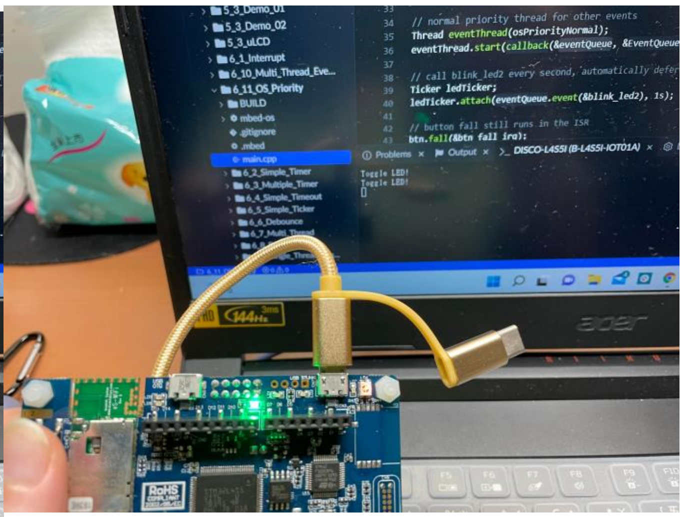
說明：

將Thread設置Priority分別喚醒printfQueue、eventQueue，Ticker透過eventQueue使led燈在1秒後變換，當按鍵被放開時，印出資料。



```
// low priority thread for calling printf()
Thread printfThread(osPriorityLow);
printfThread.start(callback(&printfQueue, &EventQueue::dispatch_forever));

// normal priority thread for other events
Thread eventThread(osPriorityNormal);
eventThread.start(callback(&eventQueue, &EventQueue::dispatch_forever));

// call blink_led2 every second, automatically defering to the eventThread
Ticker ledTicker;
ledTicker.attach(eventQueue.event(&blink_led2), 1s);

// button fall still runs in the ISR
btn.fall(&btn_fall_irq);

while (1) {ThisThread::sleep_for(1s);}
}
```



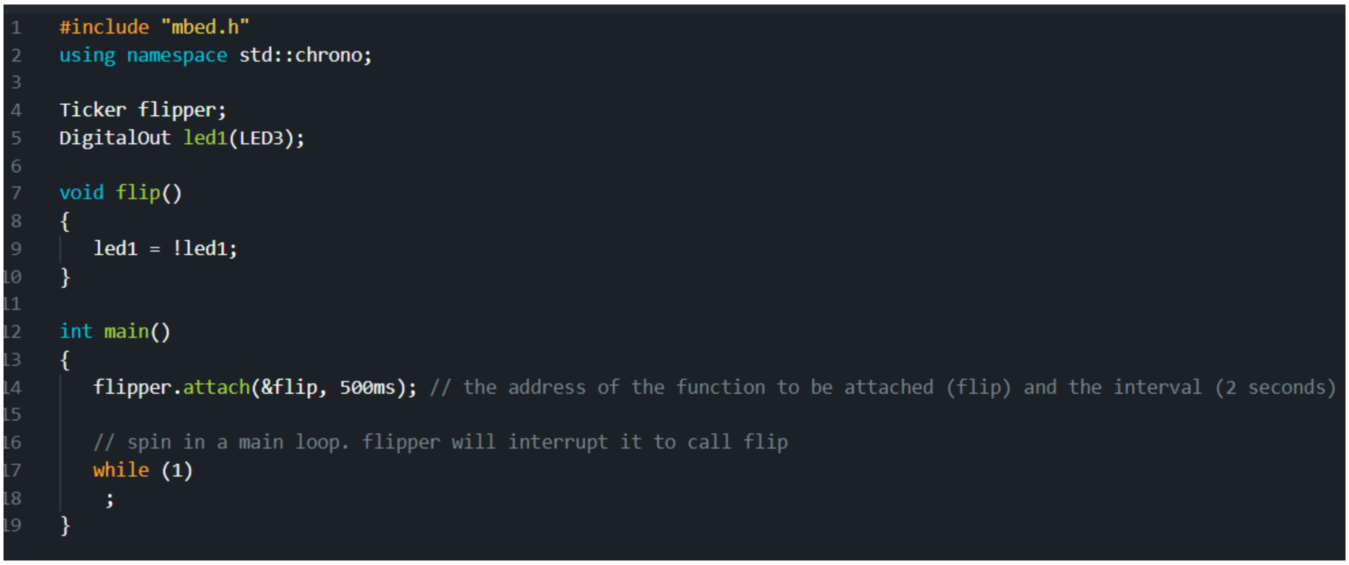


二、Demo and Checkpoints

git remote repository

說明：

Switch the blue LED every 500ms without using the sleep for function



```cpp
#include "mbed.h"
using namespace std::chrono;

Ticker flipper;
DigitalOut led1(LED3);

void flip()
{
    led1 = !led1;
}

int main()
{
    flipper.attach(&flip, 500ms); // the address of the function to be attached (flip) and the interval (2 seconds)

    // spin in a main loop. flipper will interrupt it to call flip
    while (1)
     ;
}
```