Lancaster University

SCC.131: Digital Systems
Introduction to C/C++ CODAL (Part 2)

Ioannis Chatzigeorgiou
(i.chatzigeorgiou@lancaster.ac.uk)

Summary of the last lecture

Lancaster University

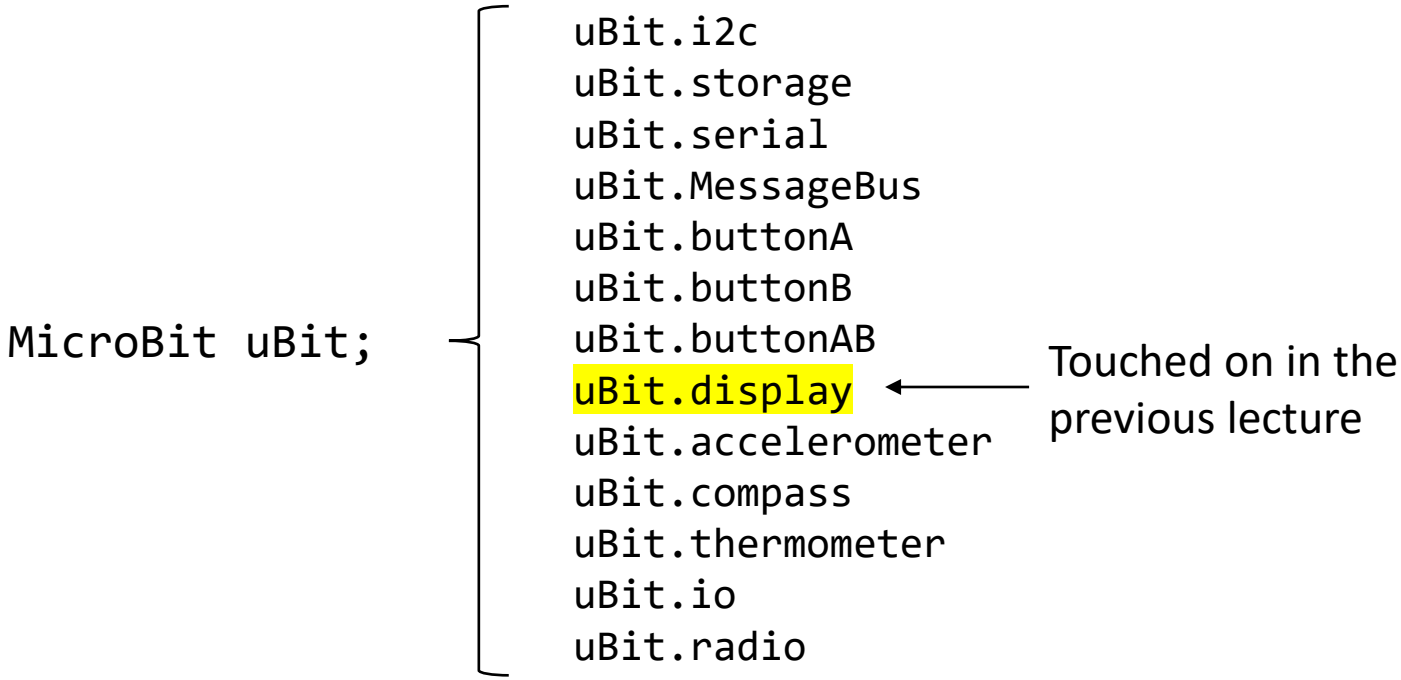
The following points were covered in the last lecture:

- The necessary **tools** to create C/C++ programs for micro:bit.
- The sequence of **steps**, from editing main.cpp and building MICROBIT.hex to 'flashing' it using Windows, Linux or the browser-based WebUSB approach.
- The general **MicroBit** class (`init`, `sleep`).
- The **MicroBitDisplay** class (`scroll`, `print`, `clear`, `setDisplayMode`, `setBrightness`, `image.setPixelValue`, `image.getPixelValue`).
- The **MicroBitImage** class (`setPixelValue`, `getPixelValue`, `paste`).

The MicroBit class

Lancaster University

- The MicroBit class uses the following classes to control features of the micro:bit



Detecting if buttons have been pressed

Lancaster University

- The micro:bit has two buttons, either side of the display: **A** and **B**.
- These are exposed on the MicroBit object as `uBit.buttonA` and `uBit.buttonB`. They are instances of the MicroBitButton class.
- A third button, uBit.buttonAB is used to detect the combined input of buttonA and button. This is an instance of the class MicroBitMultiButton.
- The method isPressed() returns 1 (i.e., true) if the corresponding button has been pressed; otherwise, it returns 0 (i.e., false).
- In **synchronous** (sequential) **programming**, detection of pressed buttons and subsequent actions take place in the `main()` function.

Example of syncrhonous button detection

Lancaster University

```
while (1)
{
    if (uBit.buttonA.isPressed())
        uBit.display.print("A");
    if (uBit.buttonB.isPressed())
        uBit.display.print("B");
    if (uBit.buttonAB.isPressed())
    {
        uBit.display.print("C");
        uBit.sleep(100);
    }
}
```

The inclusion of `MicroBit.h`, the declaration `MicroBit uBit;` the header `int main()`, and the initialization of `uBit` are omitted.

Why do you think `uBit.sleep(100)` has been added?

Comment out `uBit.sleep(100)` and flash the code to see what happens.

Asynchronous programming

Lancaster University

- Although computer programs execute sequentially, we often want to be able to determine **when** something has happened, as opposed to **if** something has happened.
- Components have been designed to raise **events** when they sense a change. For example:
  - The MicroBitAccelerometer class will raise events when the micro:bit has been shaken.
  - The MicroBitButton class will send events if a button has been pressed.
- A key aim of the MicroBitMessageBus class is to **listen to events** that our program is interested in, and to deliver MicroBitEvents to our program as they occur.
- When an event of interest is detected, the MicroBitMessageBus class calls a function linked to that event, known as an **event handler**.

Example of asyncrhonous button detection

Lancaster University

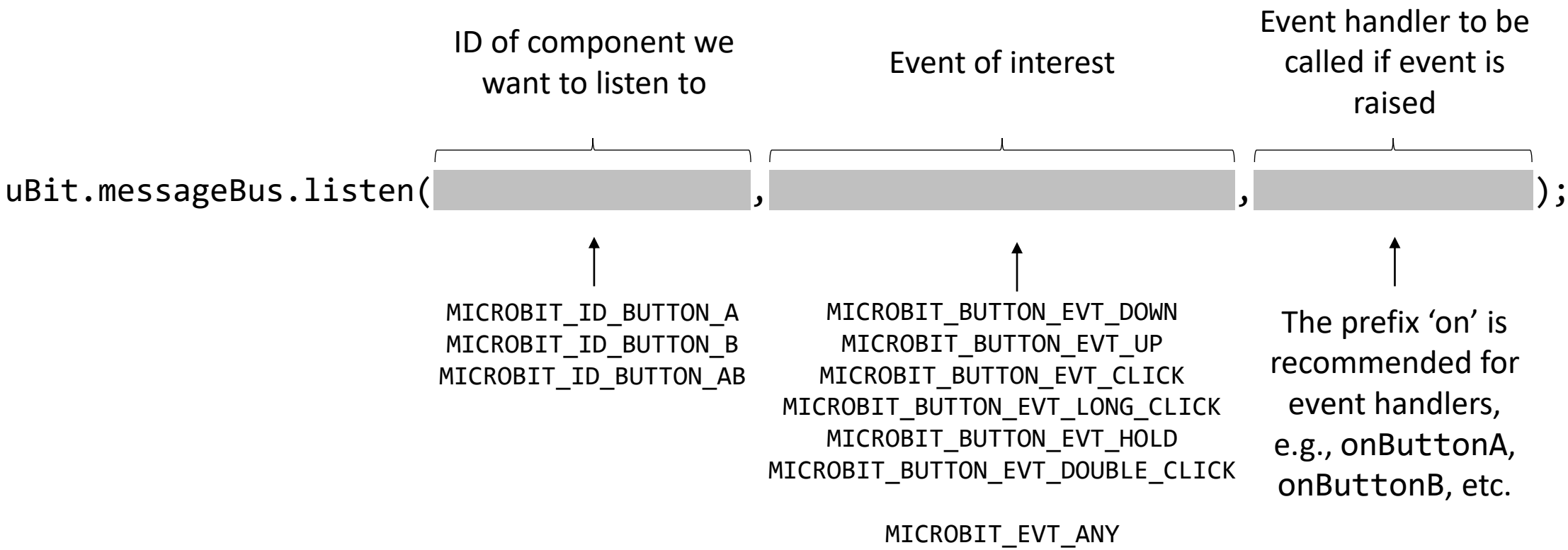
```
void onButtonAB(MicroBitEvent e)
{
    uBit.display.print("C");
}

int main()
{
    uBit.init();
    uBit.messageBus.listen(MICROBIT_ID_BUTTON_AB, MICROBIT_BUTTON_EVT_CLICK, onButtonAB);

    while(1)
        uBit.sleep(100);
}
```

Event handler

Notice that now the micro:bit does not need to go to sleep for a few milliseconds when both buttons are pressed.

We do not want to exit `main()`.
Alternatively, replace with `release_fiber();`

Input arguments of listeners for buttons



Same component, different events

Lancaster University

```
void onButtonA(MicroBitEvent e)
{
    if (e.value == MICROBIT_BUTTON_EVT_UP)
        uBit.display.print("U");
    if (e.value == MICROBIT_BUTTON_EVT_CLICK)
        uBit.display.print("C");
    if (e.value == MICROBIT_BUTTON_EVT_DOWN)
        uBit.display.print("D");
}

int main()
{
    uBit.init();
    uBit.messageBus.listen(MICROBIT_ID_BUTTON_A, MICROBIT_EVT_ANY, onButtonA);
    release_fiber();
}
```

Push (D will appear), hold and then release (U will appear).

Push (D will appear) and then release quickly.

Push (down).

Known as a ‘wildcard’

The MicroBitThermometer class

Lancaster University

- The MicroBitThermometer class provides access to the surface temperature of the application MCU.
- The temperature reading is not representative of the ambient temperature, but of the surface temperature of the application MCU:

```
readTemp = uBit.thermometer.getTemperature(); // Uncalibrated reading
```

- We can make the reading representative of the ambient temperature through "calibrating" the thermometer if we know what the real temperature is:

```
uBit.thermometer.setCalibration(readTemp-20); // The temperature is 20C
readTemp = uBit.thermometer.getTemperature(); // Calibrated reading
```

Synchronous implementation of thermometer

Lancaster University

```
int readTemp;
int ambientTemp = 20;
uBit.init();
readTemp = uBit.thermometer.getTemperature();
uBit.thermometer.setCalibration(readTemp - ambientTemp);
while(1) {
    readTemp = uBit.thermometer.getTemperature();
    uBit.display.scroll(readTemp);
    uBit.sleep(3000);
}
```

Compute offset and calibrate.

Obtain calibrated reading, display it, wait for 3 seconds and repeat.

Asynchronous temperature reading

Lancaster University

- As in the case of asynchronous detection of pressed buttons, a **listener** can be set up and trigger an event whenever the thermometer has an update, i.e., a new reading:

```
uBit.messageBus.listen(MICROBIT_ID_THERMOMETER,
                       MICROBIT_THERMOMETER_EVT_UPDATE,
                       name of event handler);
```

- The **sampling period**, that is, the time between temperature readings, can be defined using:

```
uBit.thermometer.setPeriod(time in ms);
```

Asynchronous implementation of thermometer

Lancaster University

```
void onTempUpdate(MicroBitEvent e)
{
    uBit.display.scroll(uBit.thermometer.getTemperature());
}

int main()
{
    uBit.init();
    uBit.thermometer.setCalibration(uBit.thermometer.getTemperature() - 20);
    uBit.thermometer.setPeriod(3000);
    uBit.messageBus.listen(MICROBIT_ID_THERMOMETER, MICROBIT_THERMOMETER_EVT_UPDATE,
onTempUpdate);
    release_fiber();
}
```

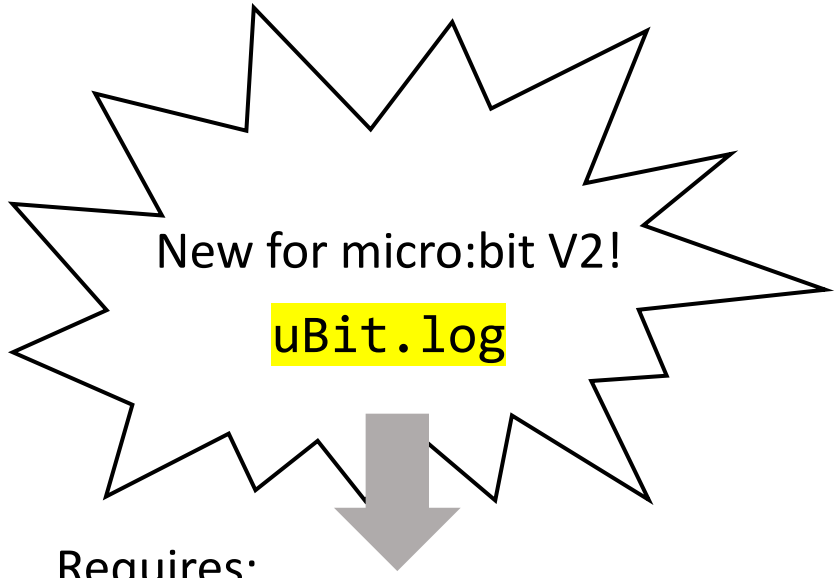
The MicroBit class

Lancaster University

- Classes of MicroBit that we have covered this week:

```
MicroBit uBit;
```

- uBit.i2c
- uBit.storage
- uBit.serial
- uBit.MessageBus
- uBit.buttonA
- uBit.buttonB
- uBit.buttonAB
- uBit.display
- uBit.accelerometer
- uBit.compass
- uBit.thermometer
- uBit.io
- uBit.radio



The MicroBitLog class

Lancaster University

- This class enables us to store data in a **table-like format**, containing **rows** of readings or other types of data.
- Use `beginRow()` to open the file and create a new row.
- Use `logData(“label of column”, value to log)` to identify the label of the column where a value will be entered in the new row.
- Use `endRow()` to complete logging and close the file. For example:

```
uBit.log.beginRow();
uBit.log.logData("temperature", uBit.thermometer.getTemperature());
uBit.log.endRow();
```

The MicroBitLog class

Lancaster University

- Before we start collecting and logging data, we often add the following line to initiate the logger:

```
uBit.log.setTimeStamp(TimeStampFormat::Seconds);
```

- Although adding a **time stamp** is not required for logging to work, it allows us to create a **plot** of the recorded values.
- To find the **log file** (named MY_DATA.HTM) in the micro:bit folder and access it using a browser, we need to include the following line to make it **visible**:

```
uBit.log.setVisibility(true);
```

- We can carry on logging for as long as `uBit.log.isFull()` returns 0.
- To clear the contents of the log file, we use `uBit.log.clear()`.

The log file (MY_DATA.HTM)

Lancaster University

DETAILS.TXT	22/03/2016 16:30	Text Document	1 KB
MICROBIT.HTM	22/03/2016 16:30	Chrome HTML Document	1 KB
MY_DATA.HTM	22/03/2016 16:30	Chrome HTML Document	124 KB

Open in browser

micro:bit

micro:bit data log

Download | Copy | Update data... | Clear log... | Visual preview

This is the data on your micro:bit. To analyse it and create your own graphs, transfer it to your computer. You can copy and paste your data, or download it as a CSV file which you can import into a spreadsheet or graphing tool. Learn more about micro:bit data logging.

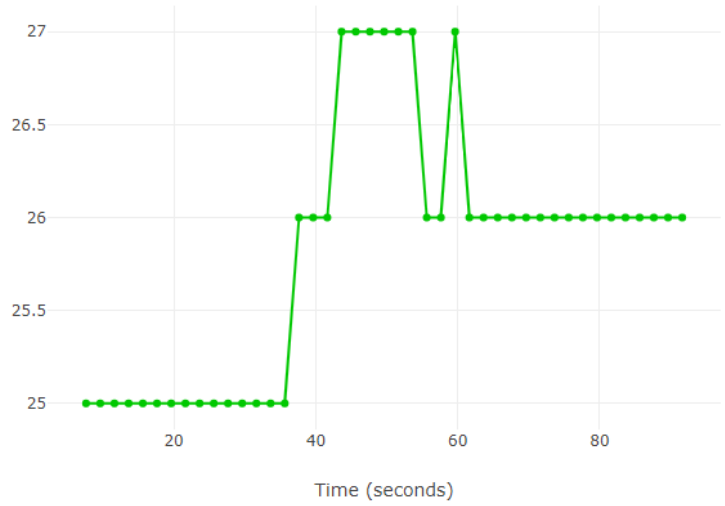
Time (seconds)	temperature
7.63	25
9.63	25
11.63	25
13.63	25
15.63	25
17.63	25
19.63	25
21.63	25
23.63	25
25.63	25
27.63	25
29.63	25
31.63	25
33.63	25

micro:bit

micro:bit data log

Download | Copy | Update data... | Clear log... | Close visual preview

This is a visual preview of the data on your micro:bit. To analyse it in more detail or create your own graphs, transfer it to your computer. You can copy and paste your data, or download it as a CSV file which you can import into a spreadsheet or graphing tool. Learn more about micro:bit data logging.



Summary

Lancaster University

Today we learnt:

- How to detect and react to events **synchronously** and **asynchronously**.
- How to set up **event listeners**, which call **event handlers** when a **MicroBitEvent** is detected (in the case of asynchronous programming).
- How to use **wildcards** that enable us to listen to multiple events triggered by the same component (e.g., a button) and associate a different response to each event.
- How to use the **MicroBitThermometer** class to measure temperature and the **MicroBitLog** class to log data to a file that can be accessed by a web browser.

Resources

Lancaster University

- The **MicroBitButton** class: https://lancaster-university.github.io/microbit-docs/ubit/button/
- The **MicroBitMultiButton** class: https://lancaster-university.github.io/microbit-docs/ubit/multibutton/
- **Events**: https://lancaster-university.github.io/microbit-docs/concepts/#events
- The **MicroBitMessageBus** class: https://lancaster-university.github.io/microbit-docs/ubit/messageBus/
- The **MicroBitEvent** class: https://lancaster-university.github.io/microbit-docs/data-types/event/
- The **MicroBitThermometer** class: https://lancaster-university.github.io/microbit-docs/ubit/thermometer/
- Logging with **MicroBitLog**: https://microbit.c272.org/guides/logging/