SCC.131: Digital Systems
Introduction to C/C++ CODAL (Part 1)

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

Summary of the last lecture

The following points were covered in the last lecture:

- What the **key hardware components** of micro:bit are and how they interact.
- What the functions of the **target MCU** and the **interface MCU** are.
- What the role of the **interface firmware** (DAPLink) is.
- How the **runtime environment** (CODAL) facilitates code development.
- What the pros and cons of compiled code and interpreted code are when it is flashed to the micro:bit.
- How micro:bit can be programmed using **MakeCode**.

Latest version of CODAL

- The repository that provides the necessary tooling to compile a **C/C++ CODAL** program for the micro:bit V2 and generate a HEX file that can be downloaded to the device is:

  https://github.com/lancaster-university/microbit-v2-samples

- Pre-requisite tools to build the repo include the **GNU Arm Embedded Toolchain**, **Git**, **Cmake** and **Python 3**. Details on how to install them are provided in README.md in the repository above.
- Ubuntu **Linux** is as assumed. Windows users should install the **Windows Subsystem for Linux** (WSL), install the pre-requisite tools and then clone the repository.
- Given that the repository is constantly updated / upgraded, we have created a stable **copy** of the repository, which can be found in scc-source.lancs.ac.uk.

How to build MICROBIT.hex

- Clone the repository https://scc-source.lancs.ac.uk/scc.Y1/scc.131/microbit-v2-samples

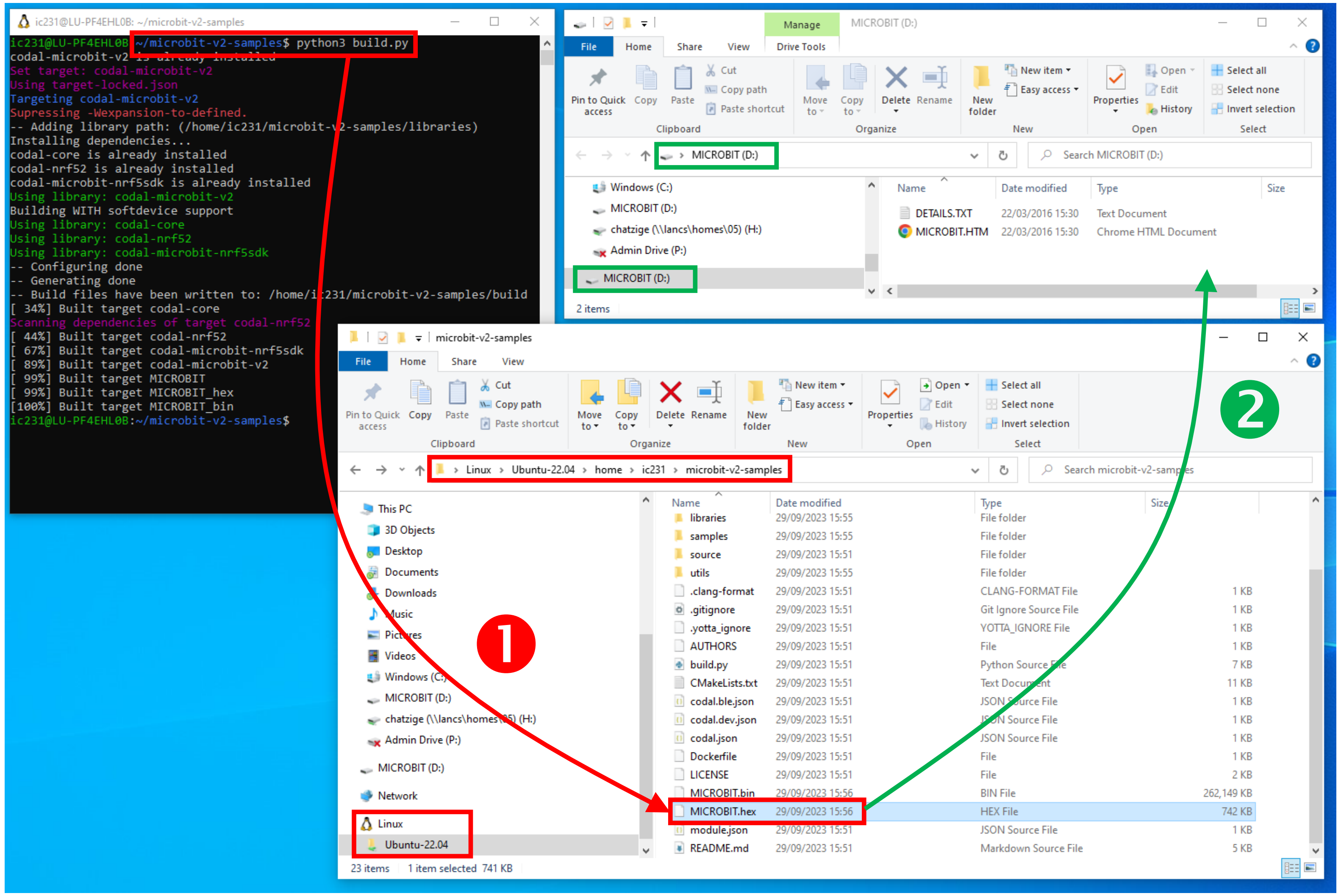
```
git clone https://scc-source.lancs.ac.uk/scc.Y1/scc.131/microbit-v2-samples.git
```

- In the root of the folder, where you cloned the repository, type:

  `python3 build.py` (or `python build.py`, depending on the installed Python version)

- The hex file `MICROBIT.hex` will be created in the root.
- Open File Explorer (Windows), find the location of the Windows Subsystem for Linux and go to the folder **microbit-v2-samples**, where the file `MICROBIT.hex` is located.
- Connect micro:bit and transfer the file MICROBIT.hex across (i.e., drag and drop).

How to build MICROBIT.hex



```
ic231@LU-PF4EHL0B:~/microbit-v2-samples$ python3 build.py
codal-microbit-v2 is already installed
Set target: codal-microbit-v2
Using target-locked.json
Targeting codal-microbit-v2
Supressing -Wexpansion-to-defined.
-- Adding library path: (/home/ic231/microbit-v2-samples/libraries)
Installing dependencies...
codal-core is already installed
codal-nrf52 is already installed
codal-microbit-nrf5sdk is already installed
Using library: codal-microbit-v2
Building WITH softdevice support
Using library: codal-core
Using library: codal-nrf52
Using library: codal-microbit-nrf5sdk
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ic231/microbit-v2-samples/build
[ 34%] Built target codal-core
Scanning dependencies of target codal-nrf52
[ 44%] Built target codal-nrf52
[ 67%] Built target codal-microbit-nrf5sdk
[ 89%] Built target codal-microbit-v2
[ 99%] Built target MICROBIT
[ 99%] Built target MICROBIT_hex
[100%] Built target MICROBIT_bin
ic231@LU-PF4EHL0B:~/microbit-v2-samples$
```

MICROBIT (D:)

Name	Date modified	Type	Size
DETAILS.TXT	22/03/2016 15:30	Text Document	
MICROBIT.HTM	22/03/2016 15:30	Chrome HTML Document	

2 items

Linux > Ubuntu-22.04 > home > ic231 > microbit-v2-samples

Name	Date modified	Type	Size
libraries	29/09/2023 15:55	File folder	
samples	29/09/2023 15:55	File folder	
source	29/09/2023 15:51	File folder	
utils	29/09/2023 15:55	File folder	
.clang-format	29/09/2023 15:51	CLANG-FORMAT File	1 KB
.gitignore	29/09/2023 15:51	Git Ignore Source File	1 KB
.yotta_ignore	29/09/2023 15:51	YOTTA_IGNORE File	1 KB
AUTHORS	29/09/2023 15:51	File	1 KB
build.py	29/09/2023 15:51	Python Source File	7 KB
CMakeLists.txt	29/09/2023 15:51	Text Document	11 KB
codal.ble.json	29/09/2023 15:51	JSON Source File	1 KB
codal.dev.json	29/09/2023 15:51	JSON Source File	1 KB
codal.json	29/09/2023 15:51	JSON Source File	1 KB
Dockerfile	29/09/2023 15:51	File	1 KB
LICENSE	29/09/2023 15:51	File	2 KB
MICROBIT.bin	29/09/2023 15:56	BIN File	262,149 KB
MICROBIT.hex	29/09/2023 15:56	HEX File	742 KB
module.json	29/09/2023 15:51	JSON Source File	1 KB
README.md	29/09/2023 15:51	Markdown Source File	5 KB

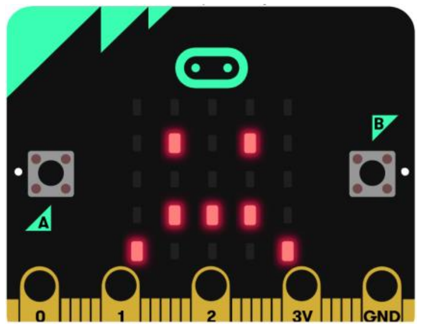
23 items 1 item selected 741 KB

❶ Type:
`python3 build.py`
to create
`MICROBIT.hex`

❷ Drag and drop
`MICROBIT.hex`
to the micro:bit
'USB disk'

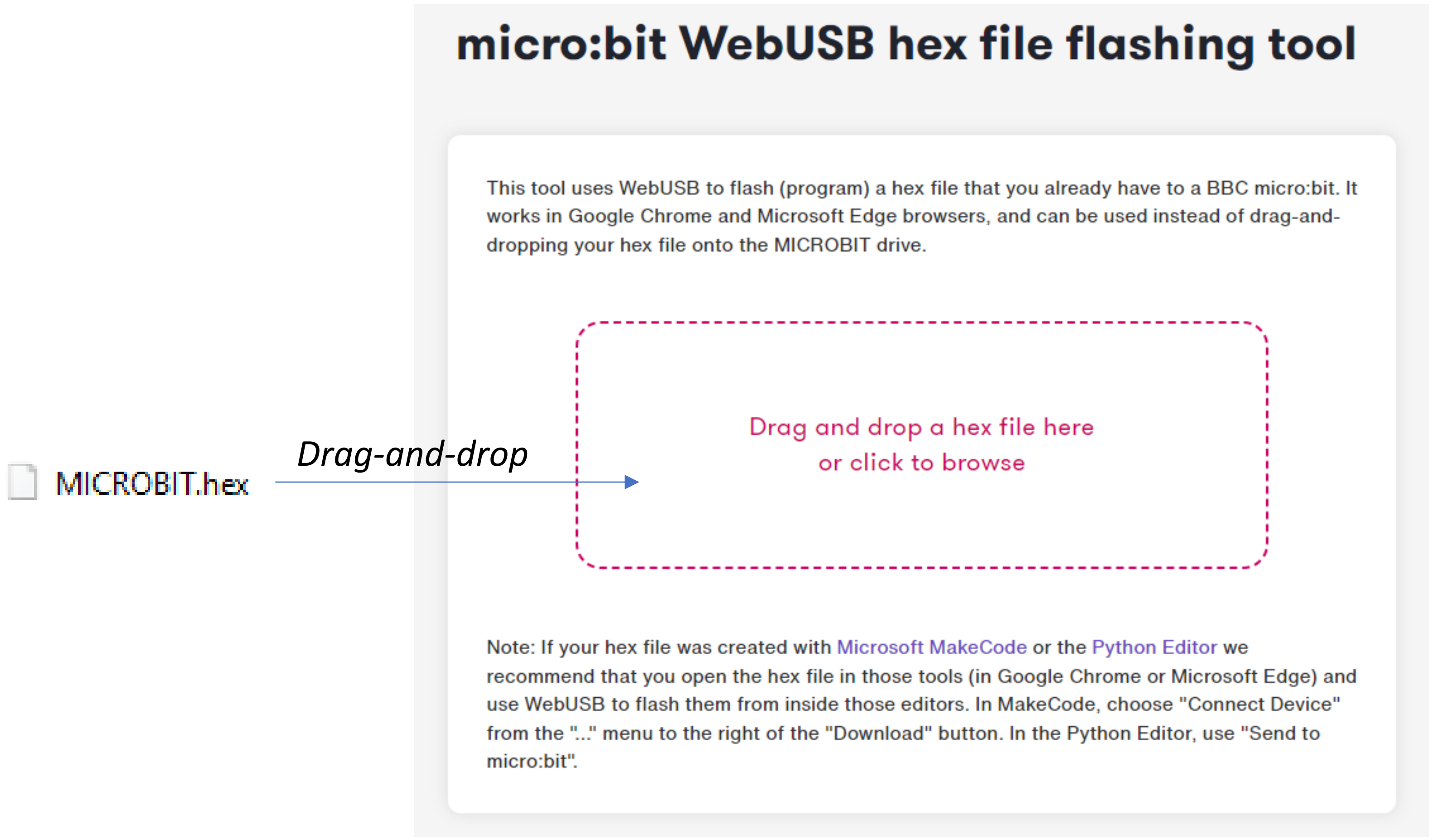
Problems?

- Since an update to **Windows** in February 2023, an increase in the prevalence of errors 504, 506, 521 and 537 has been observed (FAIL.TXT ⇨ error: The transfer timed out).
- The issue is with **DAPLink** (software for Arm Cortex CPUs that makes micro:bit appear as a USB drive when connected to a Windows PC).



- Programmers experience longer flashing times when using the drag-and-drop method to copy files to the micro:bit drive.
- Preferred workaround: **WebUSB** (https://microbit.org/tools/webusb-hex-flashing/)
  - WebUSB is a way for your **browser** to connect directly to the micro:bit.
  - WebUSB works in **Google Chrome** (version 79 or newer) and **Microsoft Edge** (recommended version 79 or newer).

WebUSB (https://microbit.org/tools/webusb-hex-flashing/)



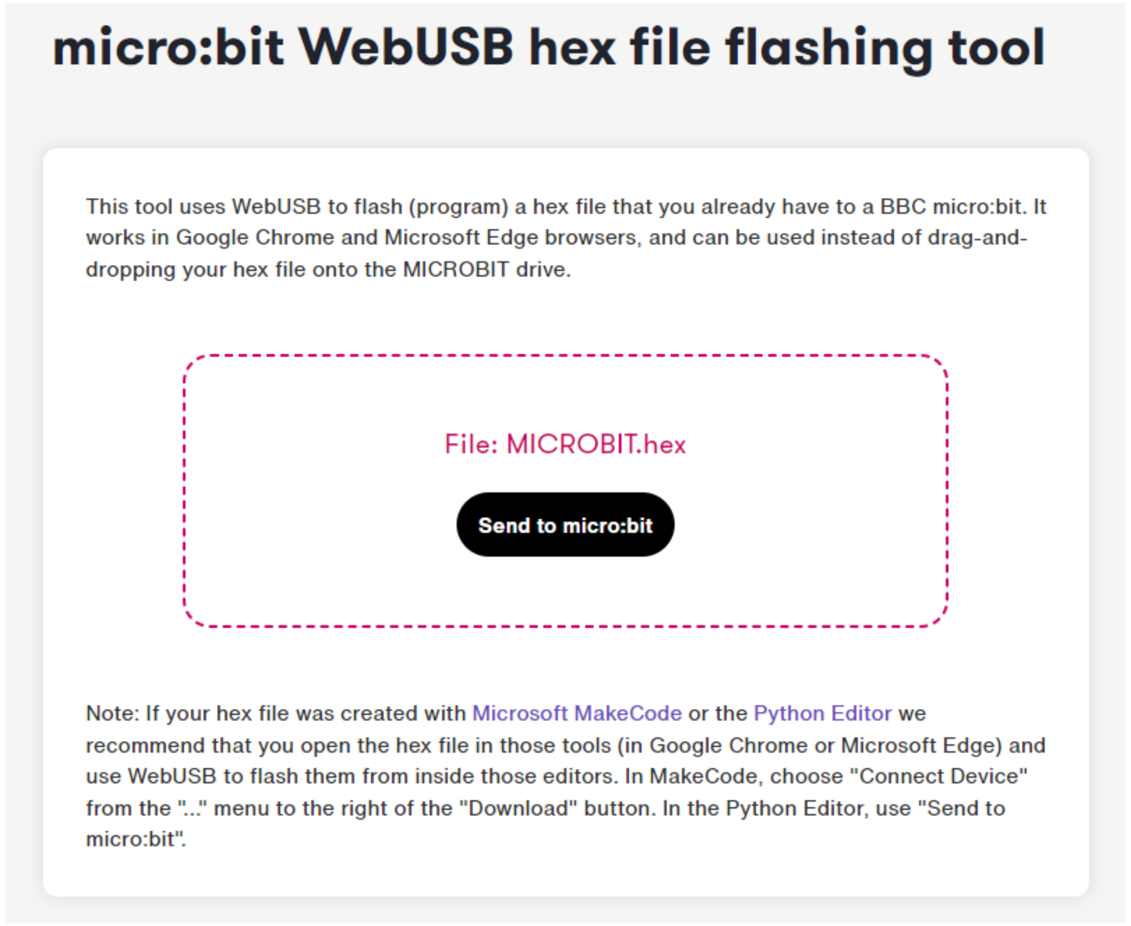
micro:bit WebUSB hex file flashing tool

This tool uses WebUSB to flash (program) a hex file that you already have to a BBC micro:bit. It works in Google Chrome and Microsoft Edge browsers, and can be used instead of drag-and-dropping your hex file onto the MICROBIT drive.

Drag and drop a hex file here
or click to browse

Note: If your hex file was created with Microsoft MakeCode or the Python Editor we recommend that you open the hex file in those tools (in Google Chrome or Microsoft Edge) and use WebUSB to flash them from inside those editors. In MakeCode, choose "Connect Device" from the "..." menu to the right of the "Download" button. In the Python Editor, use "Send to micro:bit".

WebUSB (https://microbit.org/tools/webusb-hex-flashing/)

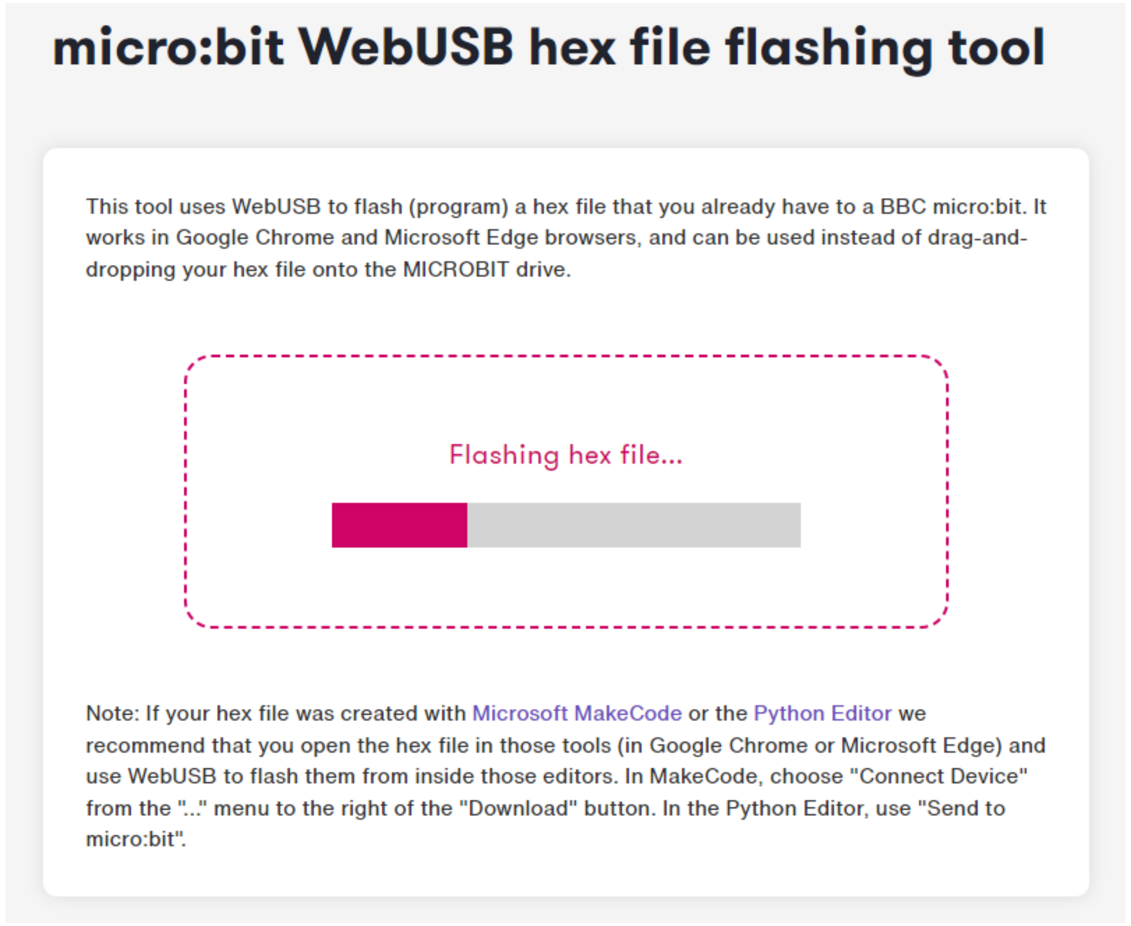


micro:bit WebUSB hex file flashing tool

This tool uses WebUSB to flash (program) a hex file that you already have to a BBC micro:bit. It works in Google Chrome and Microsoft Edge browsers, and can be used instead of drag-and-dropping your hex file onto the MICROBIT drive.

Note: If your hex file was created with Microsoft MakeCode or the Python Editor we recommend that you open the hex file in those tools (in Google Chrome or Microsoft Edge) and use WebUSB to flash them from inside those editors. In MakeCode, choose "Connect Device" from the "..." menu to the right of the "Download" button. In the Python Editor, use "Send to micro:bit".

WebUSB (https://microbit.org/tools/webusb-hex-flashing/)



micro:bit WebUSB hex file flashing tool

This tool uses WebUSB to flash (program) a hex file that you already have to a BBC micro:bit. It works in Google Chrome and Microsoft Edge browsers, and can be used instead of drag-and-dropping your hex file onto the MICROBIT drive.

Note: If your hex file was created with Microsoft MakeCode or the Python Editor we recommend that you open the hex file in those tools (in Google Chrome or Microsoft Edge) and use WebUSB to flash them from inside those editors. In MakeCode, choose "Connect Device" from the "..." menu to the right of the "Download" button. In the Python Editor, use "Send to micro:bit".

Summary of methods to upload HEX file

1. Drag the icon of the HEX file and drop it into the MICROBIT USB drive.
2. … or use WebUSB to upload the HEX file onto the micro:bit device.
3. … or type the following command in the terminal of a lab machine:

```
cp MICROBIT.hex /media/$USER/MICROBIT
```

The MicroBit class

- The MicroBit class consists of **variables** and **methods** (i.e., resources) that operate as drivers to control commonly used features of the micro:bit.

```
                    uBit.i2c
                    uBit.storage
                    uBit.serial
                    uBit.MessageBus
                    uBit.buttonA
                    uBit.buttonB
MicroBit uBit;  {   uBit.buttonAB
                    uBit.display
                    uBit.accelerometer
                    uBit.compass
                    uBit.thermometer
                    uBit.io
                    uBit.radio
```

"Hello World!" for micro:bit

Include the library that contains key functions for micro:bit.

Declare uBit as a 'variable' of type MicroBit. (Create object uBit, which is an instance of class MicroBit).

Initialise uBit (i.e., initialise the scheduler, memory allocator and Bluetooth stack).

Scroll "HELLO WORLD!" across the 5×5 display of uBit.

```
#include "MicroBit.h"

MicroBit uBit;

int main()
{
  uBit.init();
  while (1)
    uBit.display.scroll("HELLO WORLD!");
}
```

uBit.display (the MicroBitDisplay class)

Scrolling and printing text or numbers

```
uBit.display.scroll("HELLO!");
uBit.display.scroll("HELLO!", 100);
uBit.display.scroll(42);
uBit.display.scroll(7);
```

```
uBit.display.print("HELLO!");
uBit.display.print("HELLO!", 100);
uBit.display.print(42);
uBit.display.print(7);
```

- The **scroll** function scrolls a string (or number), pixel by pixel, across the display, from right to left.
- The **print** function shows the letters of a string (or numbers) in turn on the screen.
- The **delay** can also be specified: i.e., the lower the value, the faster the message will be scrolled/printed.
- If you **print** a string/number with **two or more** characters/digits, each will appear in turn, then disappear.
- If you **print** a **single** character/numeric digit, it will stay on screen until you explicitly change the screen.

uBit.display (the MicroBitDisplay class)

Changing Display Mode

```
uBit.display.setDisplayMode(DISPLAY_MODE_BLACK_AND_WHITE);
uBit.display.setDisplayMode(DISPLAY_MODE_BLACK_AND_WHITE_LIGHT_SENSE);
uBit.display.setDisplayMode(DISPLAY_MODE_GREYSCALE);
```

`DISPLAY_MODE_BLACK_AND_WHITE`: Each pixel can be **on** or **off**. The brightness of all pixels is controlled by the `setBrightness` function, e.g., `uBit.display.setBrightness(`value between 1 and 255`)`. This mode is the most efficient, in terms of processing time and battery power.

`DISPLAY_MODE_BLACK_AND_WHITE_LIGHT_SENSE`: Each pixel can be **on** or **off**, and the display driver will also sense the ambient brightness from the LEDs.

`DISPLAY_MODE_GREYSCALE`: Each pixel can independently have **256 levels of brightness** (0-255).

uBit.display (the MicroBitDisplay class)

Setting and getting pixel values

```
uBit.display.image.setPixelValue(2,2,255);
int i = uBit.display.image.getPixelValue(2,2);
```

- The function `setPixelValue` sets the pixel at the given (x, y) co-ordinates to a given brightness value. The coordinates are of type `int16_t`. The brightness is of type `uint8_t`.
- The function `getPixelValue` retrieves the brightness value of the pixel at the given (x, y) co-ordinates.

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)

(x, y) co-ordinates of pixels in the 5×5 display

- When the display mode is set to DISPLAY_MODE_BLACK_AND_WHITE, setting the brightness value to 1, 2, …, 255, will not make a difference.

The MicroBitImage class

Creating and showing images

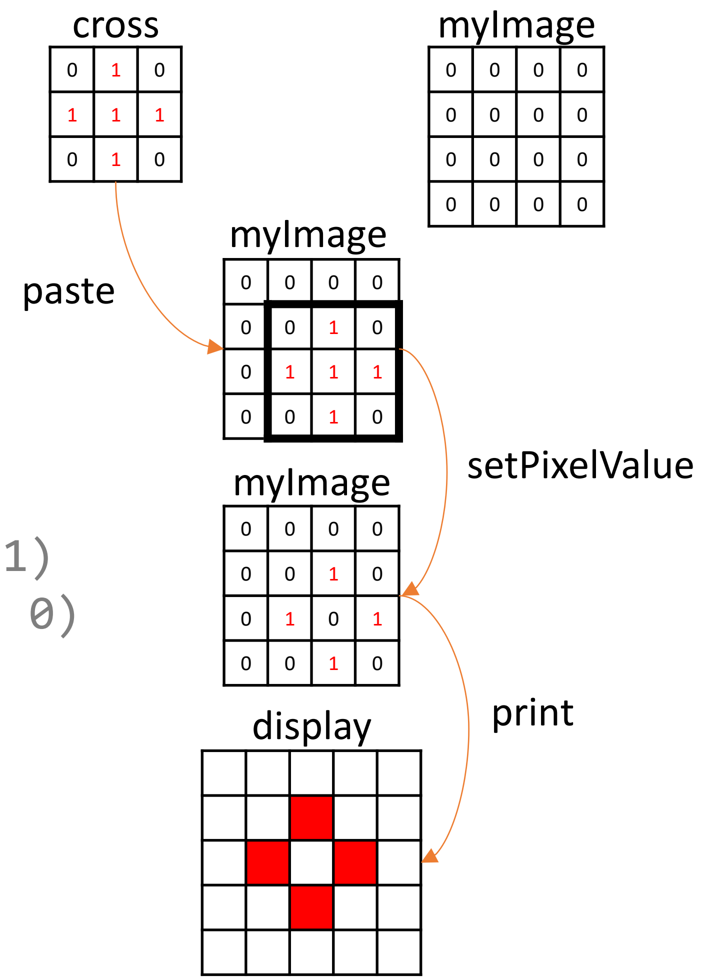
```
MicroBitImage smiley("0,255,0,255,0\n 0,255,0,255,0\n 0,0,0,0,0\n 255,0,0,0,255\n 0,255,255,255,0\n");
uBit.display.print(smiley);
```

- The `MicroBitImage` class represents a **bitmap** picture. It is a **managed type** (memory will be used and release automatically, as needed).
- The **string** constructor takes the form of a series of comma separated values. Each value is the brightness of a pixel value in the range 0 - 255, starting at the top left of your image and working to the right.
- Whenever you put a newline character \n in the string, this moves onto a new line of pixels.
- The class also provides functions to undertake graphical operations on an image (e.g., setting pixels, clearing the image, pasting one image onto another at a given position, etc.).

The MicroBitImage class

Creating and showing images (examples)

```
// Create a 3x3 picture of a cross
MicroBitImage cross("0,1,0\n1,0,1\n0,1,0\n");

// Create a blank 4x4 image (4 columns, 4 rows)
MicroBitImage myImage(4, 4);

// Paste the content of cross onto myImage at pixel (1, 1)
myImage.paste(cross, 1, 1); // Omit co-ordinates for (0, 0)

// Set brightness of pixel (2, 2) to 0
myImage.setPixelValue(2, 2, 0);

// Print myImage
uBit.display.print(myImage);
```



The MicroBitImage class

Storing and printing read-only images

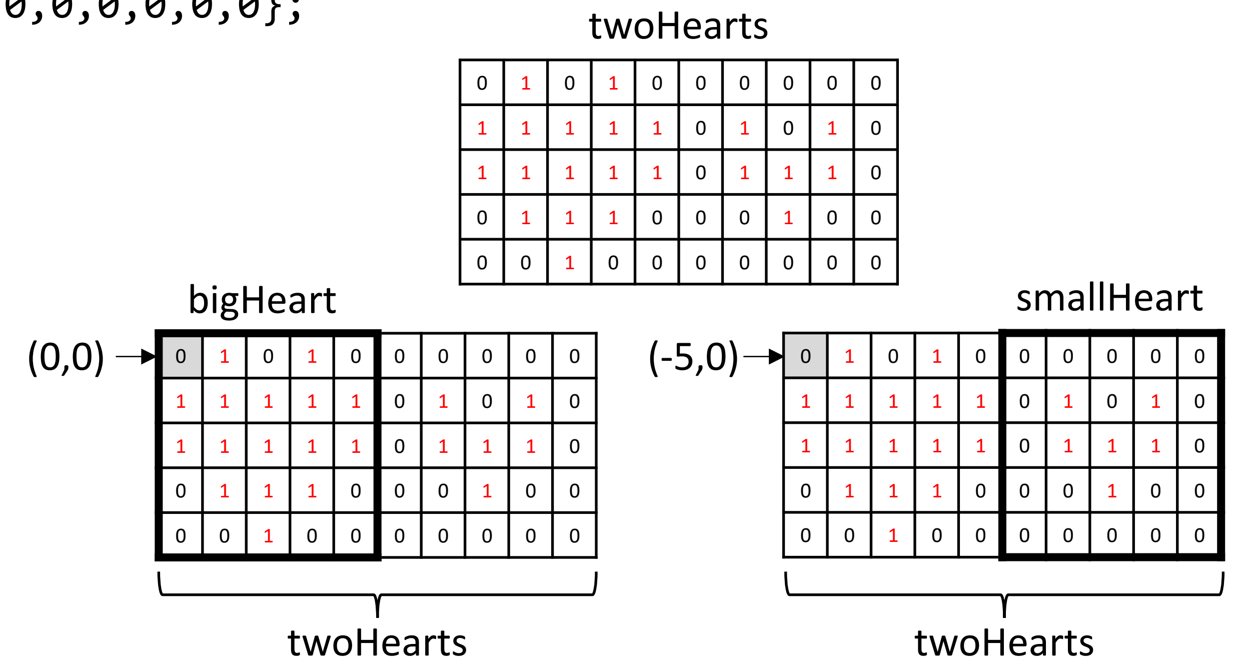
```
const uint8_t smiley[] = {0,1,0,1,0,0,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1,1,1,0};
MicroBitImage myImage(5,5,smiley);
uBit.display.print(myImage);
```

- A constant array of unsigned integers can be used to store a read-only picture. Values between 0 and 255 can be used. In this example, we have assumed that micro:bit operates in the black and white mode.
- The `MicroBitImage` class offers a constructor that creates a bitmap representation of a given size (i.e., 5×5 in this example), based on a given buffer (i.e., the array `smiley` in this example).
- The dimensions of `myImage` do not have to be 5×5; any size can be used, provided that the product matches the length of the array.

The MicroBitImage class

Storing and printing read-only images (example)

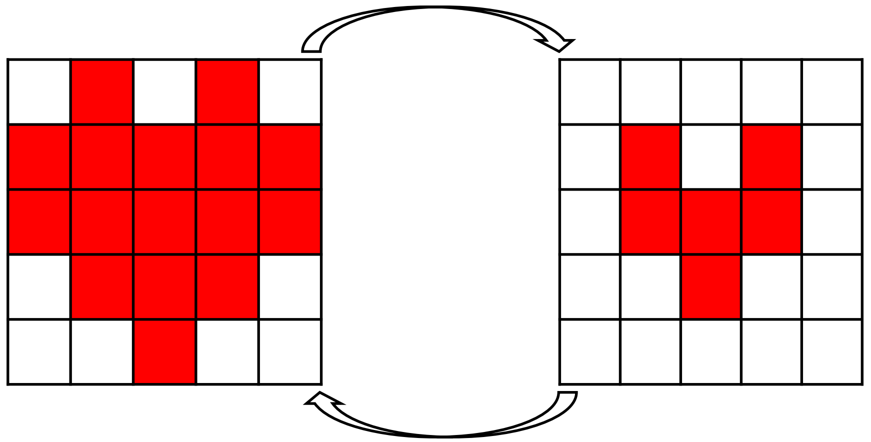
```
const uint8_t hearts[] = {0,1,0,1,0,0,0,0,0,0,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,
0,1,1,1,0,0,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0};
MicroBitImage twoHearts(10,5,hearts);
MicroBitImage bigHeart(5,5);
MicroBitImage smallHeart(5,5);
bigHeart.paste(twoHearts,0,0);
smallHeart.paste(twoHearts,-5,0);
while(1) {
   uBit.display.print(bigHeart);
   uBit.sleep(100); // time in ms
   uBit.display.print(smallHeart);
   uBit.sleep(100); // time in ms
}
```



C/C++ examples

Demonstration of code development using C/C++

Objective: Develop code for micro:bit to display a ‘flashing heart’.

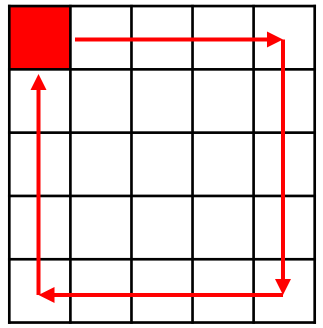


The script
`flashing_heart_v1.cpp`
is available on the
[SCC.130 Moodle page]()
(see ‘Code’ in Week 8).

flashing_heart_v1.cpp (1 of 2)

```cpp
#include "MicroBit.h"

MicroBit uBit;

int main()
{
    MicroBitImage largeHeart("0,1,0,1,0\n \
                             1,1,1,1,1\n \
                             1,1,1,1,1\n \
                             0,1,1,1,0\n \
                             0,0,1,0,0\n");

    MicroBitImage smallHeart("0,0,0,0,0\n \
                             0,1,0,1,0\n \
                             0,1,1,1,0\n \
                             0,0,1,0,0\n \
                             0,0,0,0,0\n");
    uBit.init();
    uBit.display.setDisplayMode(DISPLAY_MODE_BLACK_AND_WHITE);

    while (1)
    {
        uBit.display.print(largeHeart); // a large heart
        uBit.sleep(500);
        uBit.display.print(smallHeart); // a small heart
        uBit.sleep(500);
    }
}
```

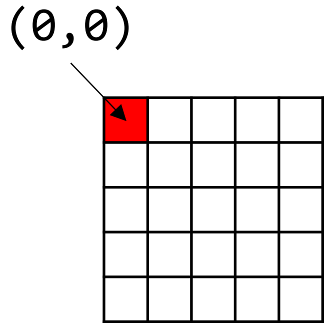
Objective: Develop code for micro:bit to display a ‘trapped dot’.



The script
`trapped_dot.cpp`
is available on the
SCC.130 Moodle page
(see ‘Code’ in Week 8).

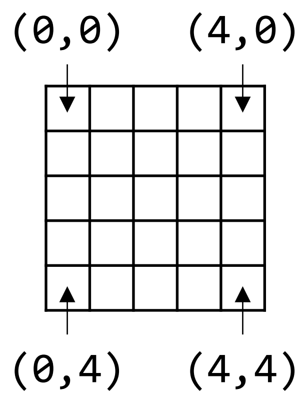
```
#include "MicroBit.h"

MicroBit uBit; // The MicroBit object

int main()
{
    int16_t x = 0; // The x co-ordinate
    int16_t y = 0; // The y co-ordinate

    // Initialise the micro:bit
    uBit.init();

    // Turn on the pixel in location (x, y)
    uBit.display.image.setPixelValue(x, y, 255);

    // Wait for 100 ms
    uBit.sleep(100)
```

```
#include "MicroBit.h"

MicroBit uBit; // The MicroBit object

int main()
{
    int16_t x = 0; // The x co-ordinate
    int16_t y = 0; // The y co-ordinate

    // Initialise the micro:bit
    uBit.init();

    // Turn on the pixel in location (x, y)
    uBit.display.image.setPixelValue(x, y, 255);

    // Wait for 100 ms
    uBit.sleep(100)
```

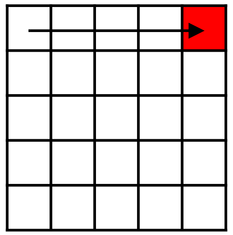


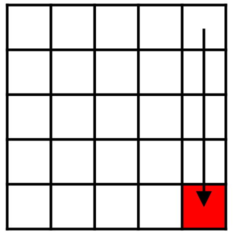
```
while (1)
{
    // Turn off the pixel in location (x, y)
    uBit.display.image.setPixelValue(x, y, 0);

    // Consider the edge conditions, i.e., (4,0), (4,4), (0,4), (0,0)
    // and update the x and y co-ordinates accordingly.
    if (x < 4 && y == 0) // left -> right of the top row
            x++;
        else
        {
            if (x == 4 && y < 4) // top -> bottom of right-most column
                y++;
            else
            {
                if (x > 0 && y == 4) // right -> left of the bottom row
                    x--;
```

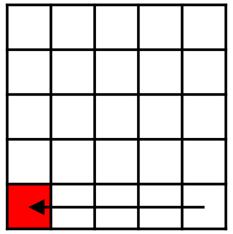
```cpp
while (1)
{
    // Turn off the pixel in location (x, y)
    uBit.display.image.setPixelValue(x, y, 0);

    // Consider the edge conditions, i.e., (4,0), (4,4), (0,4), (0,0)
    // and update the x and y co-ordinates accordingly.
    if (x < 4 && y == 0) // left -> right of the top row
            x++;
        else
        {
            if (x == 4 && y < 4) // top -> bottom of right-most column
                y++;
            else
            {
                if (x > 0 && y == 4) // right -> left of the bottom row
                    x--;
```



```
while (1)
{
    // Turn off the pixel in location (x, y)
    uBit.display.image.setPixelValue(x, y, 0);

    // Consider the edge conditions, i.e., (4,0), (4,4), (0,4), (0,0)
    // and update the x and y co-ordinates accordingly.
    if (x < 4 && y == 0) // left -> right of the top row
            x++;
        else
        {
            if (x == 4 && y < 4) // top -> bottom of right-most column
                y++;
            else
            {
                if (x > 0 && y == 4) // right -> left of the bottom row
                    x--;
```

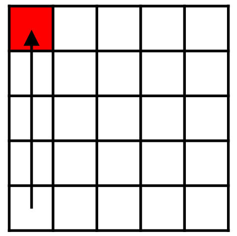






```
                else
                {
                    if (x == 0 && y > 0) // bottom -> top of left-most column
                        y--;
                }
            }
        }

        // Turn on the pixel in location (x, y)
        uBit.display.image.setPixelValue(x, y, 255);
        // Wait for 100 ms
        uBit.sleep(100);
    }
}
```

```
                else
                {
                    if (x == 0 && y > 0) // bottom -> top of left-most column
                        y--;
                }
            }
        }

        // Turn on the pixel in location (x, y)
        uBit.display.image.setPixelValue(x, y, 255);
        // Wait for 100 ms
        uBit.sleep(100);
    }
}
```



Summary

Today we learnt:

- How to copy the necessary **tools** to create C/C++ for micro:bit.
- How to **edit** main.cpp, **build** MICROBIT.hex and ‘**flash**’ it using Windows, Linux or the browser-based WebUSB approach.
- About the general **MicroBitClass** (`init`, `sleep`).
- About the **MicroBitDisplay** class (`scroll`, `print`, `setDisplayMode`, `setBrightness`, `image.setPixelValue`, `image.getPixelValue`).
- About the **MicroBitImage** class (`setPixelValue, getPixelValue, paste`).

`uBit.display.clear()` clears the screen!

Resources

- Tools to compile a C/C++ CODAL program for the micro:bit V2: https://github.com/lancaster-university/microbit-v2-samples
- Tools to compile a C/C++ CODAL program for the micro:bit V2 (stable version for SCC.131): https://scc-source.lancs.ac.uk/scc.Y1/scc.131/microbit-v2-samples
- WebUSB: https://microbit.org/tools/webusb-hex-flashing/
- Microbit documents: https://lancaster-university.github.io/microbit-docs/
    - The MicroBit class: https://lancaster-university.github.io/microbit-docs/ubit/
    - The MicroBitDisplay class: https://lancaster-university.github.io/microbit-docs/ubit/display/
    - The MicroBitImage class:

      https://lancaster-university.github.io/microbit-docs/data-types/image/