Lancaster University

SCC.131: Digital Systems

Revision and discussion of lab tasks

Ioannis Chatzigeorgiou

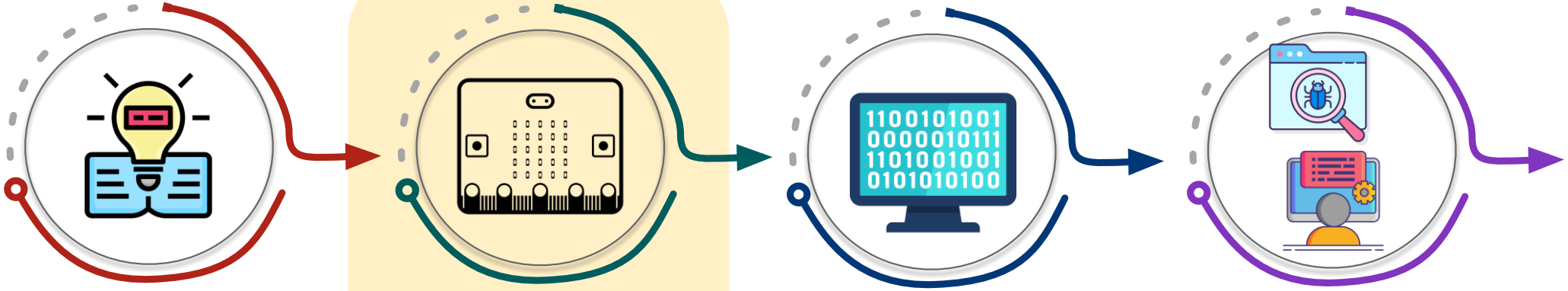
(i.chatzigeorgiou@lancaster.ac.uk)

Please remember to complete the

SCC.131-Term 1 Module Evaluation Survey.

Closing date: **20/12/2024**

Reminder: SCC.131 organization

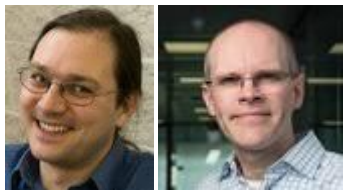


1. *Computer Architecture*
2. *Micro:bit Architecture*
3. ARM Assembly
4. Systems Architecture

Weeks 7 to 12

Lecture 1 – Week 7

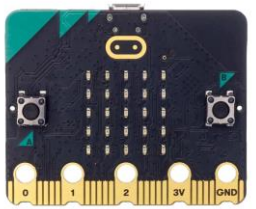
Lancaster University



- You met Joe and Steve, two of the inventors of micro:bit!
- Learnt about the **initial requirements**, e.g.:
  - A more tactile/tangible approach for teaching/learning about computing.
  - Easy to use, easy to engage with.
  - No installation, no setup, no internet.
- Found out **differences** between micro:bit and arduino / raspberry pi.
- Followed a demo in **MakeCode** using block coding.
- Given a very high-level **overview of the architecture** of micro:bit.

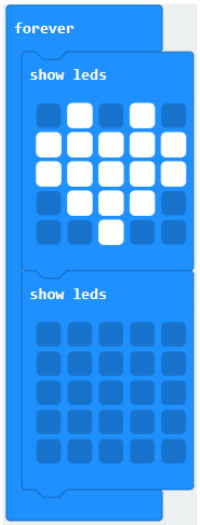
Lecture 2 – Week 7

Lancaster University



We talked about:

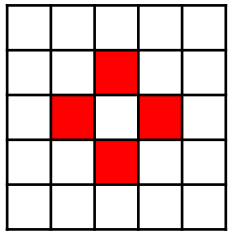
- **Key hardware components** of micro:bit are and how they interact.
- Functions of the **target MCU** and the **interface MCU**.
- The role of the **interface firmware** (DAPLink).
- The **runtime environment** (CODAL) and how it facilitates code development.
- The pros and cons of compiled code and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



Lecture 1 – Week 8

Lancaster University

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



Lecture 2 – Week 8

Lancaster University

```
MICROBIT_ID_BUTTON_A
MICROBIT_ID_BUTTON_B
MICROBIT_ID_BUTTON_AB
```

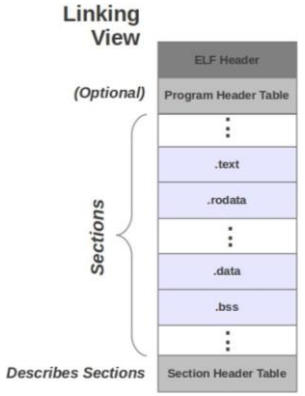
Explained:

- 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.

Lecture 1 – Week 9

Lancaster University

- Explained the stages of the **preprocessor**, **compiler** and **assembler**, which translate a C **source** file into an **object** file.
- Described sections of the object file and, in particular, the **symbol table** and the **relocation information**.
- Differentiated between dynamic linking against **shared objects** (not supported for micro:bit) and static linking against **archives** of objects.
- Highlighted the importance of **separating** the linking stage from preprocessing, compilation and translation into assembly.



Lecture 2 – Week 9

Lancaster University

- Initial processing: merge continued lines, break them, remove comments.
- Tokenization: each line is broken down into ‘tokens’; the preprocessor looks at **tokens** that contain directives and macros.
- The flexibility that preprocessing offers in:
  - Replacing **object-like** and **function-like** identifiers (names of macros) with their definitions (bodies of macros).
  - Including **header files** and **controlling compilation** (using **conditionals** and **computed include** directives).
  - Using **predefined macros** to diagnose problems.

```
#include "myLib.h"
#ifndef MINISET_H
#define MINISET_H
```

Lecture 1 – Week 10

Lancaster University

- Types of **bugs** and debugging **strategies**.
- Debugging using **logging** (printf) and reading the serial port to display messages sent by micro:bit.
- Potential **issues** that could arise when printf is removed.
- Debugging using **GDB**.
- Big-endian and little-endian systems.
- On-Chip Debugging (OCD).

Topics to discuss

Lancaster University

- Raw performance results in SCC.131 quiz questions of Week 10.
- Solution to the “Repeating Pattern” task (Week 8 Labs).
- Solution to the or “Trapped Snake” task (Week 9 Labs).

Quiz in Week 10 (raw results)

Lancaster University

Statistics of the cohort performance,

based on the raw results of the SCC.131 section of the quiz in Week 10,

will be presented live in the lecture on Friday 13th December

but will not be included in the PDF file,

as they are not final.

Week 8 - Hacker Edition: Repeating Pattern

Lancaster University

Our task is to write code that will generate the following repeating pattern on the display of the micro:bit:

To reverse-engineer the pattern, use these hints and tips:

- An image having dimensions **larger than 5x5** is shifted along the x-axis and y-axis of the display. Try to determine the shape that is being shifted and define it is as a **MicroBitImage** using the **String constructor**.
- You will need to **invoke only three functions** of MicroBit.h. If `uBit` is the declared micro:bit object, the three functions are:
  - `uBit.init()` to initialise micro:bit.
  - `uBit.display.image.paste()` to paste the MicroBitImage onto the 5x5 display at the specified coordinates.
  - `uBit.sleep()` to pause execution for the specified input time in milliseconds.

The use of **five for-loops** are required, **four of which will be in the while-loop** that repeats the pattern indefinitely.

```
#include "MicroBit.h"

MicroBit uBit;

int main()
{
    // 9x9 image
    MicroBitImage diamond("0,0,0,0,1,0,0,0,0\n\
                           0,0,0,1,0,1,0,0,0\n\
                           0,0,1,0,0,0,1,0,0\n\
                           0,1,0,0,0,0,0,1,0\n\
                           1,0,0,0,0,0,0,0,1\n\
                           0,1,0,0,0,0,0,1,0\n\
                           0,0,1,0,0,0,1,0,0\n\
                           0,0,0,1,0,1,0,0,0\n\
                           0,0,0,0,1,0,0,0,0\n");
    int x, y; // coordinates
    int waiting_time = 200; // waiting time in msec between animations

    uBit.init(); // initialise the micro:bit
    uBit.display.setDisplayMode(DISPLAY_MODE_BLACK_AND_WHITE);
```

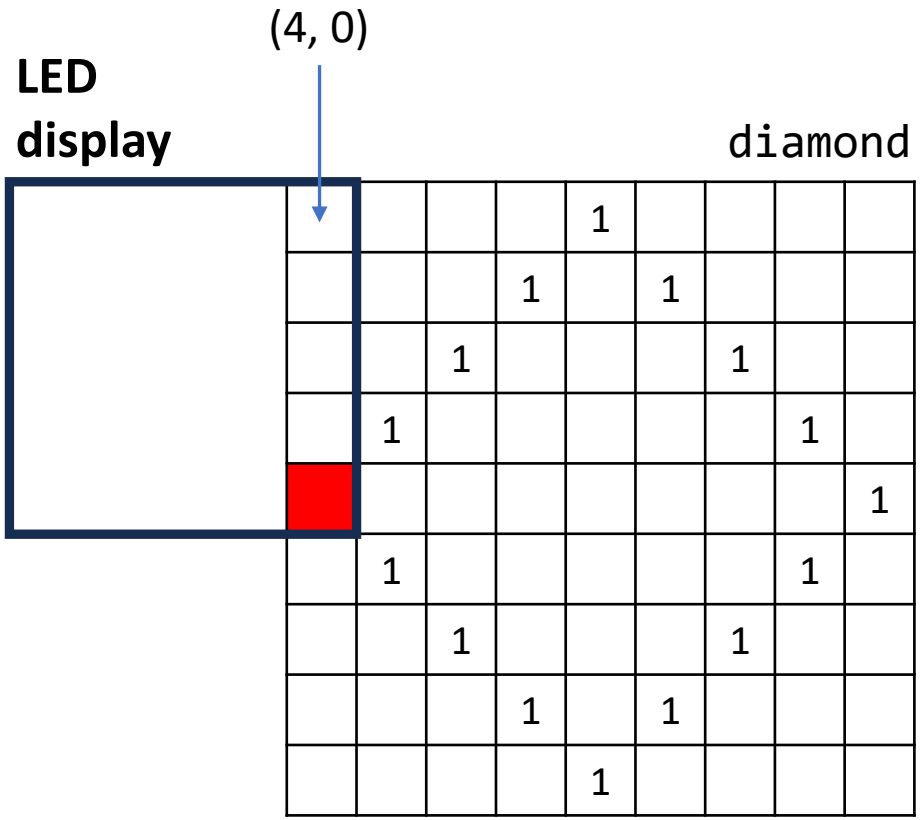
For the purpose of the lecture, `setDisplayMode()` has been used to express `diamond` in compact format. This function is not required if 1s are replaced by 255s in `diamond`.

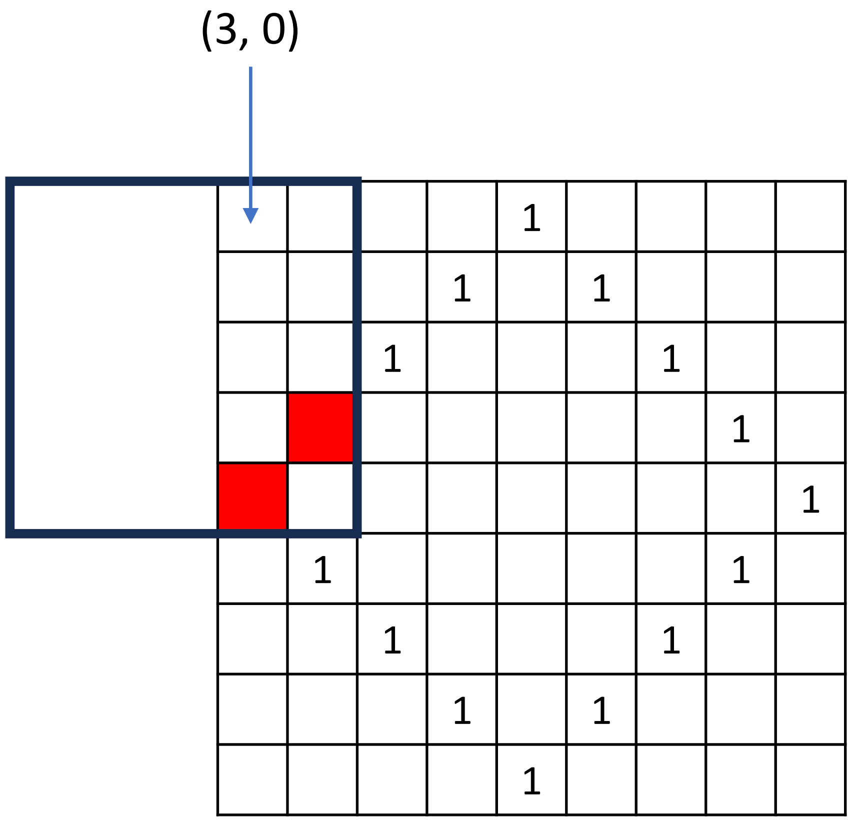
```
#include "MicroBit.h"

MicroBit uBit;

int main()
{
    // 9x9 image
    MicroBitImage diamond("0,0,0,0,1,0,0,0,0\n\
                           0,0,0,1,0,1,0,0,0\n\
                           0,0,1,0,0,0,1,0,0\n\
                           0,1,0,0,0,0,0,1,0\n\
                           1,0,0,0,0,0,0,0,1\n\
                           0,1,0,0,0,0,0,1,0\n\
                           0,0,1,0,0,0,1,0,0\n\
                           0,0,0,1,0,1,0,0,0\n\
                           0,0,0,0,1,0,0,0,0\n");
    int x, y; // coordinates
    int waiting_time = 200; // waiting time in msec between animations

    uBit.init(); // initialise the micro:bit
    uBit.display.setDisplayMode(DISPLAY_MODE_BLACK_AND_WHITE);
```

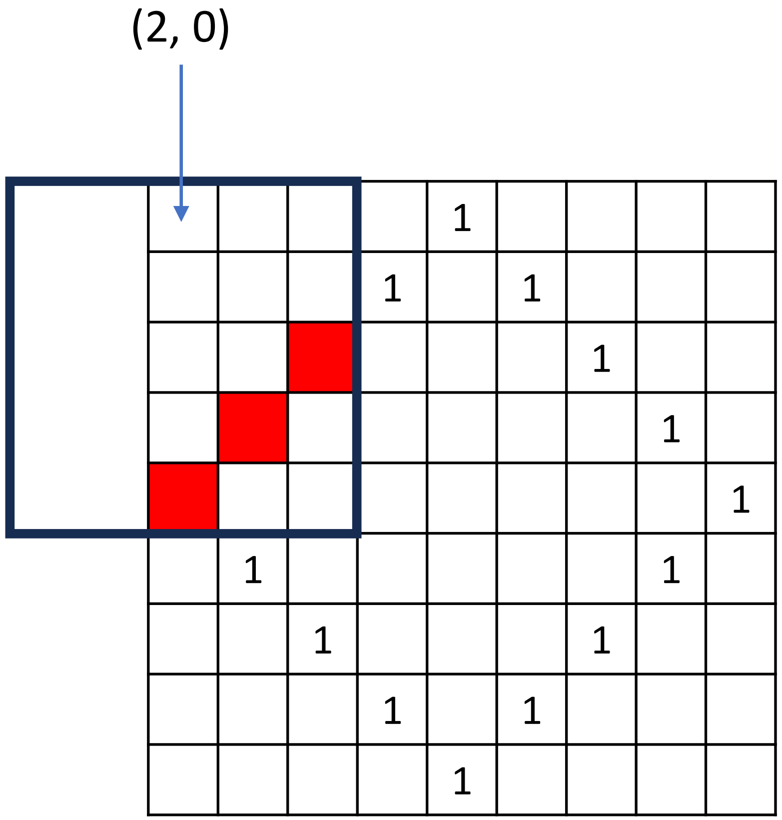
diamond

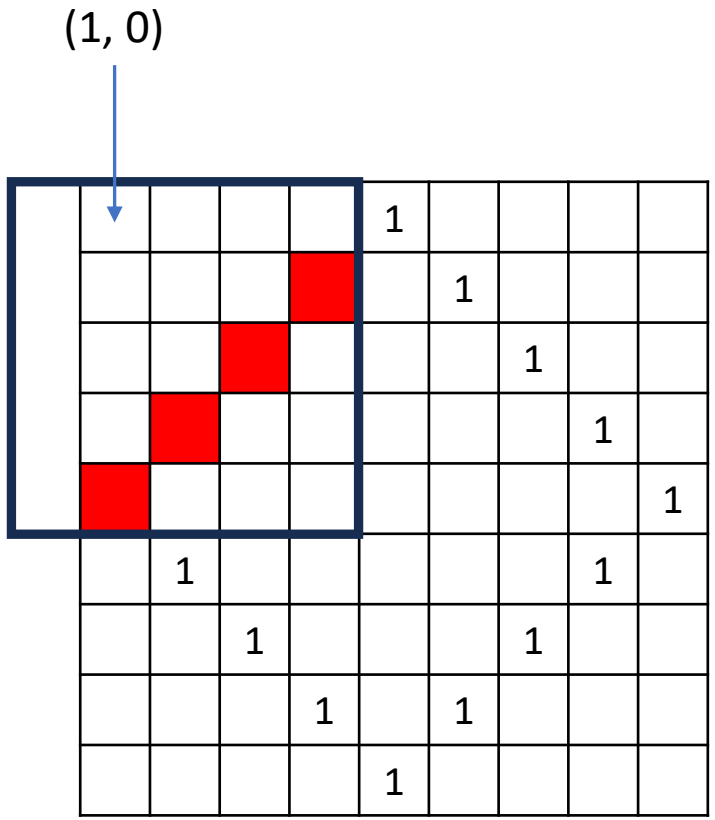
				1				
			1		1			
		1				1		
	1						1	
1								1
	1						1	
		1				1		
			1		1			
				1				

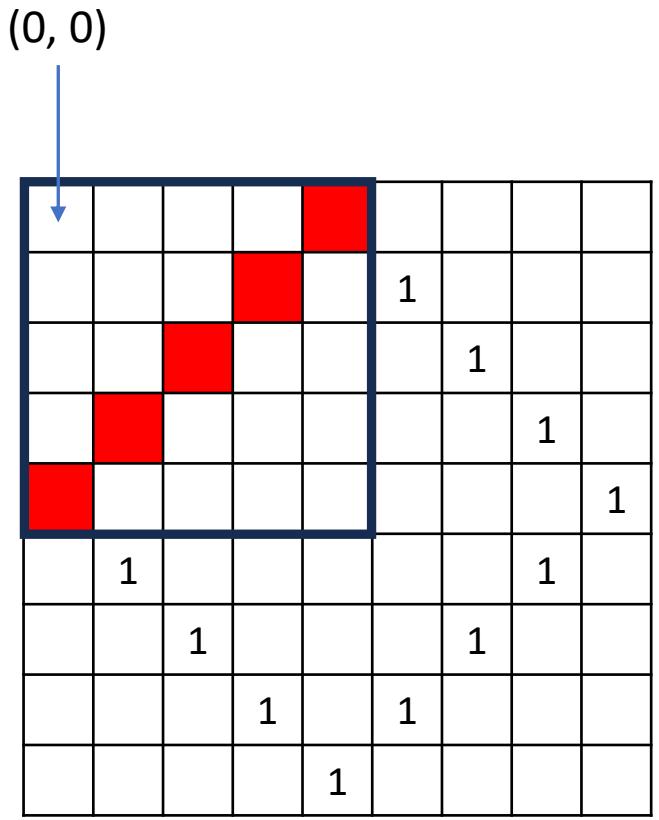
```
// introduce the top-left part of the image
// and shift it from left to right
for (x = 4; x >= 0; x--)
{
    uBit.display.image.paste(diamond, x, 0);
    uBit.sleep(waiting_time);
}
```

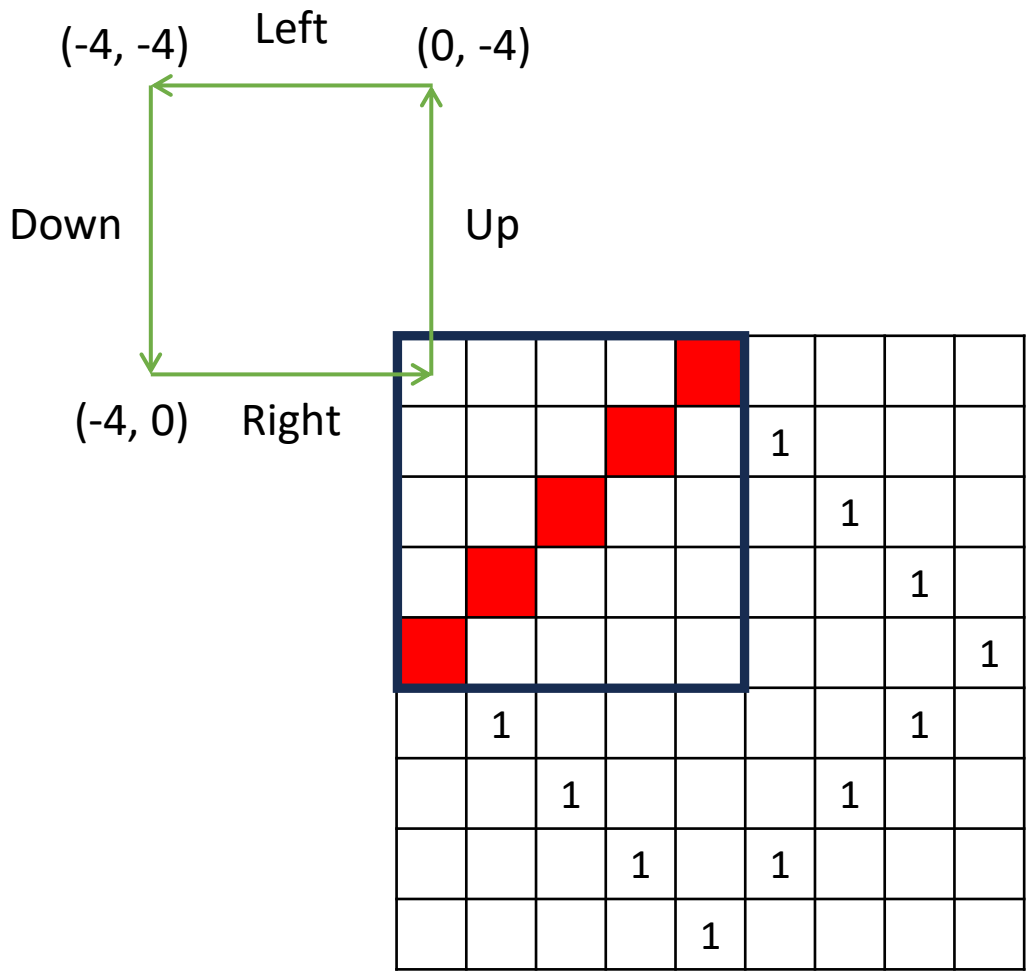

(4, 0)
LED
display
diamond


(3, 0)

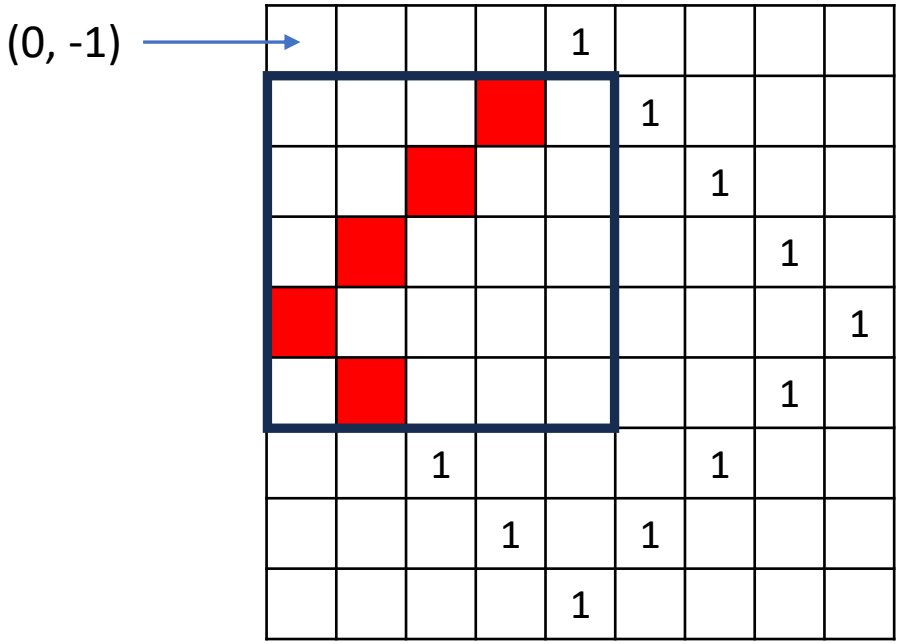

(2, 0)

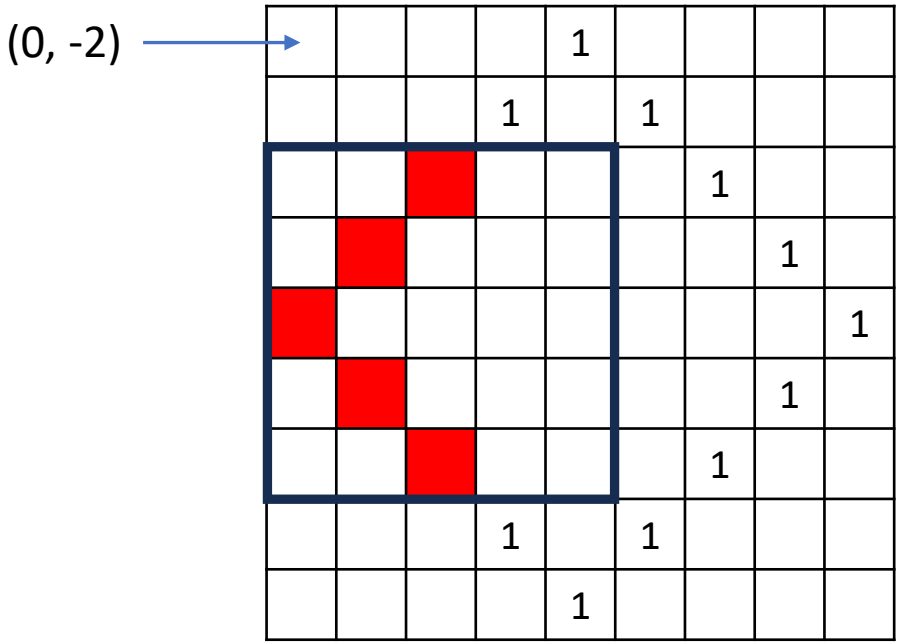

(1, 0)
1
1
1
1
1
1
1
1
1
1
1
1
1
1

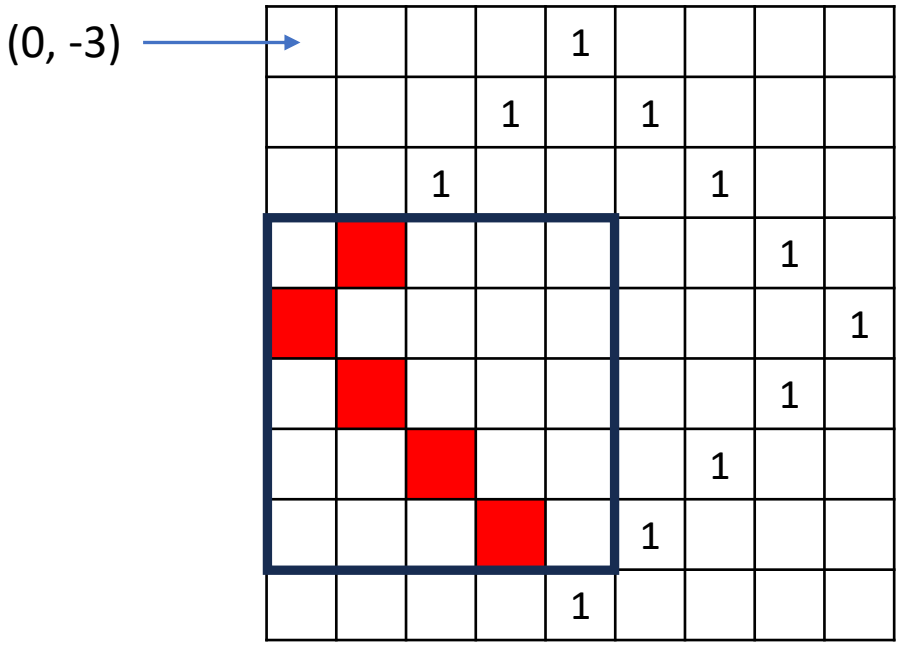

(0, 0)
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

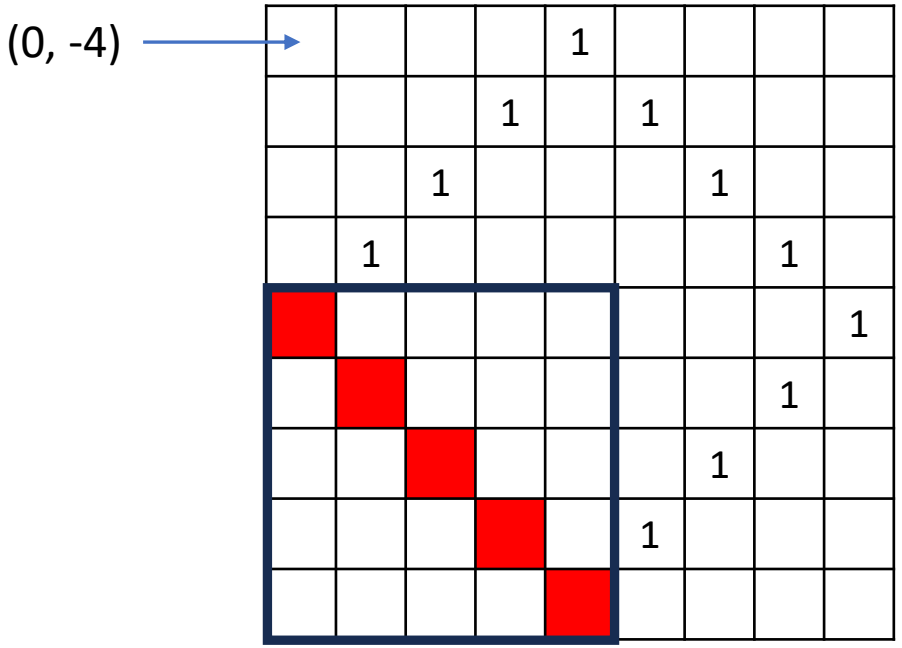

(-4, -4)
Left
(0, -4)
Down
Up
(-4, 0)
Right
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

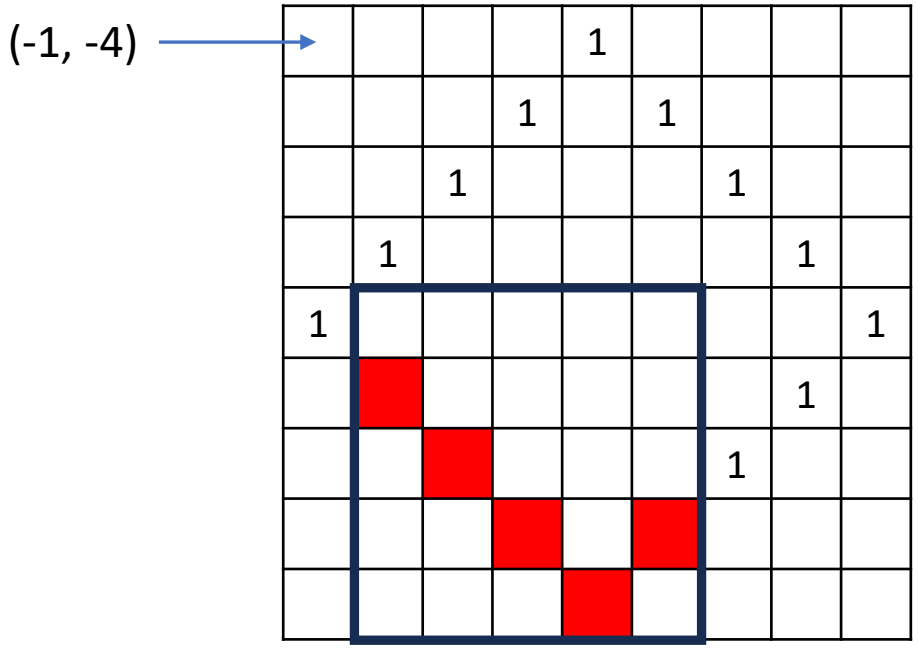
```
// keep animating indefinitely
while (1)
{

    // shift from top to bottom
    for (y = -1; y >= -4; y--)
    {
        uBit.display.image.paste(diamond, 0, y);
        uBit.sleep(waiting_time);
    }

    // shift from right to left
    for (x = -1; x >= -4; x--)
    {
        uBit.display.image.paste(diamond, x, -4);
        uBit.sleep(waiting_time);
    }
```

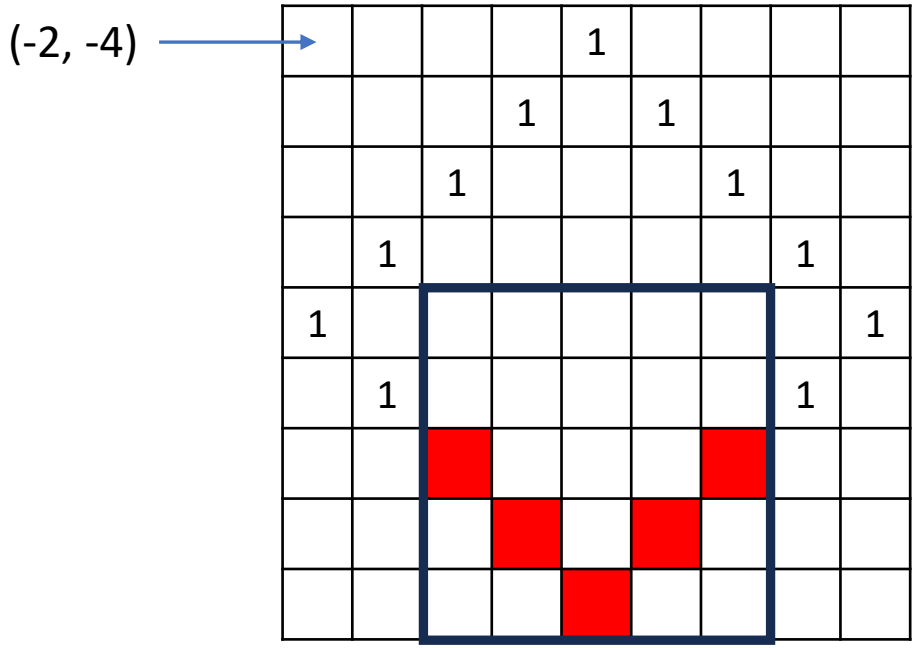

(0, -1)
1
1
1
1
1
1
1
1
1
1
1

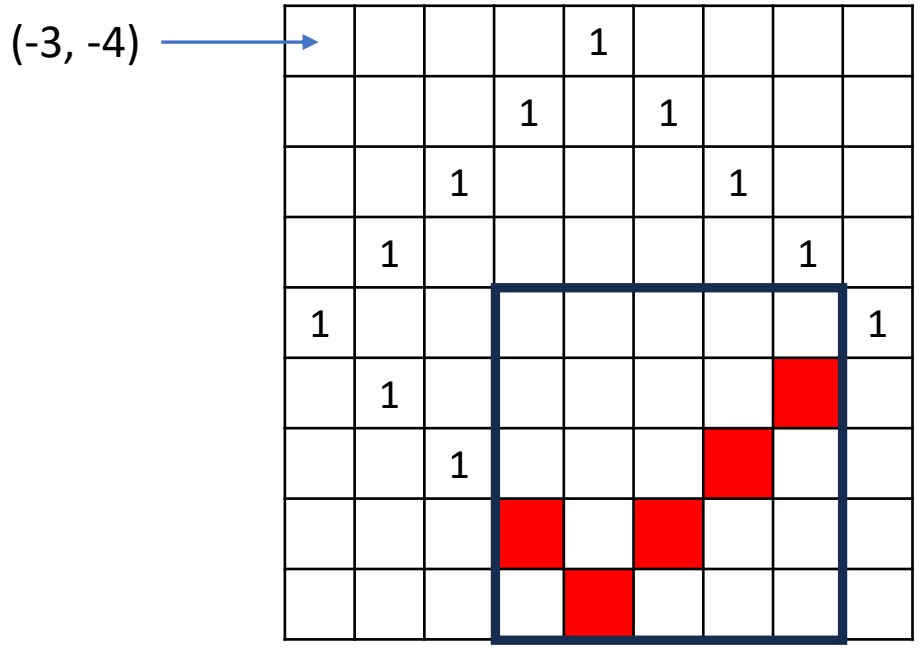

(0, -2)
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

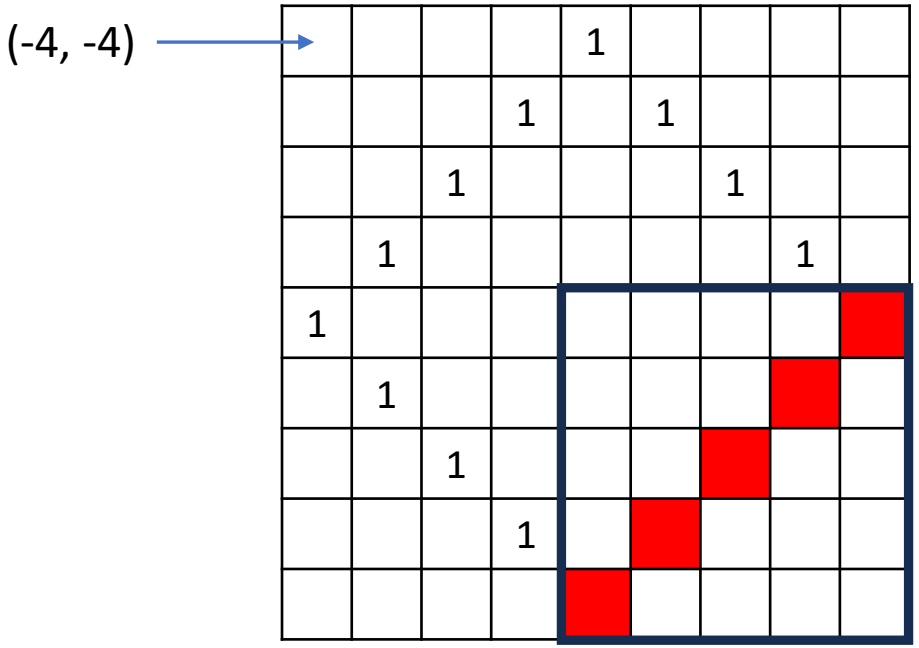

(0, -3)
1
1
1
1
1
1
1
1
1
1
1
1
1


(0, -4)
1
1
1
1
1
1
1
1
1
1
1
1
1
1


(-1, -4)


(-2, -4)
1
1
1
1
1
1
1
1
1
1
1
1
1
1


(-3, -4)
1
1
1
1
1
1
1
1
1
1
1
1


(-4, -4)
1
1
1
1
1
1
1
1
1
1
1
1

```
        // shift from bottom to top
        for (y = -3; y <= 0; y++)
        {
            uBit.display.image.paste(diamond, -4, y);
            uBit.sleep(waiting_time);
        }

        // shift from left to right
        for (x = -3; x <= 0; x++)
        {
            uBit.display.image.paste(diamond, x, 0);
            uBit.sleep(waiting_time);
        }

    } // end of while-loop

} // end of main()
```

Week 9 - Trapped Snake

Lancaster University

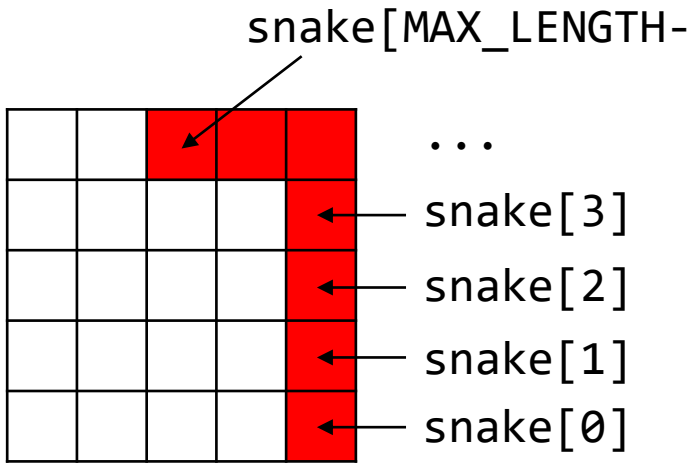
Your objective is to edit main.cpp, such that the display animates a snake instead of a dot. Your final code should produce the result shown in this short video.

For the development of the code, consider the following points:

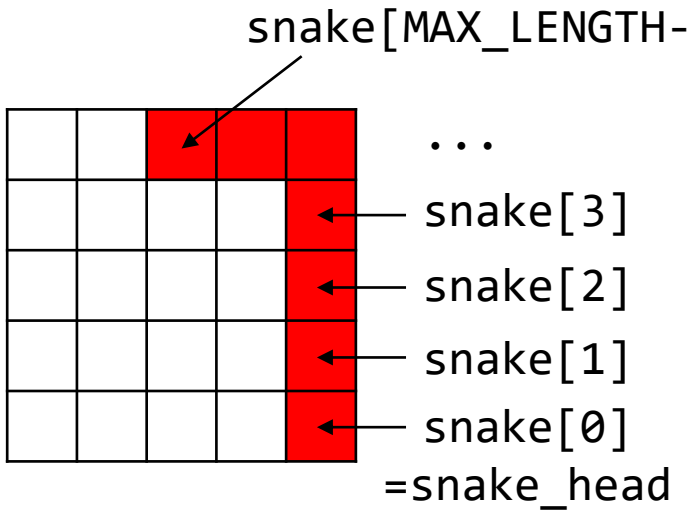
- Define a **constant** for the length of snake in pixels, e.g., `MAX_LENGTH`. Changing the value of the constant should change the length of the snake in the animation. Note that the length of the snake in the video is 7 pixels.
- Use knowledge acquired from SCC.111 to create a **structure** called `bodypart` with integer members `x` and `y` to store the coordinates of a part of the body of the snake. Essentially, the snake will be an array of type `struct bodypart`.
- Create **functions** that initialize, shift and display the snake on the screen of micro:bit.

```
#include "MicroBit.h"

MicroBit uBit;

const int16_t MAX_LENGTH = 7; // The length of the snake

// Create a structure for the coordinates of each
// 'body part' of the snake
struct bodypart {
  int16_t x, y;
};

// Create a snake that consists of MAX_LENGTH + 1 body parts.
// The actual (visible) length of the snake is MAX_LENGTH,
// i.e., from [0] to [MAX_LENGTH-1]. The last position
// (i.e., [MAX_LENGTH]) holds the previous position of the
// tail in order to erase it.
struct bodypart snake[MAX_LENGTH + 1];
...
```

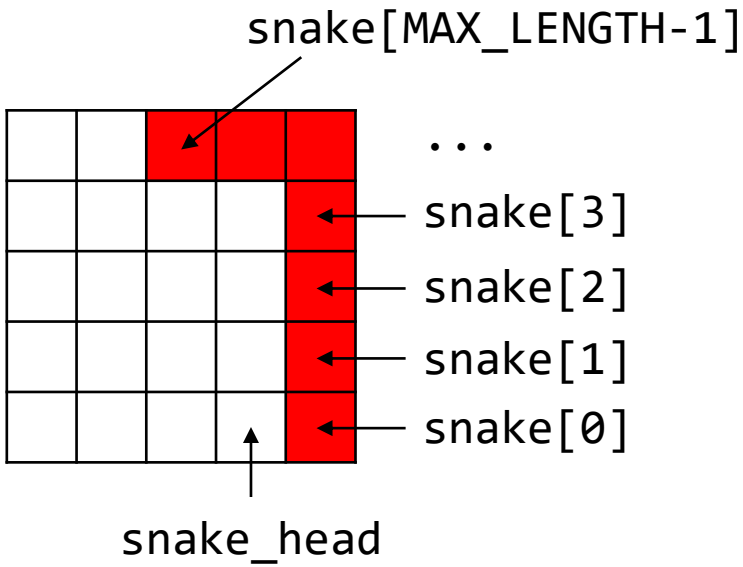
❶



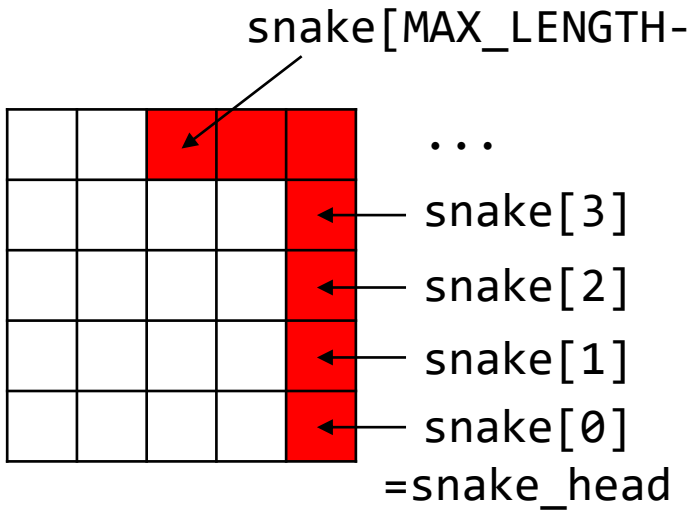
❶


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
=snake_head

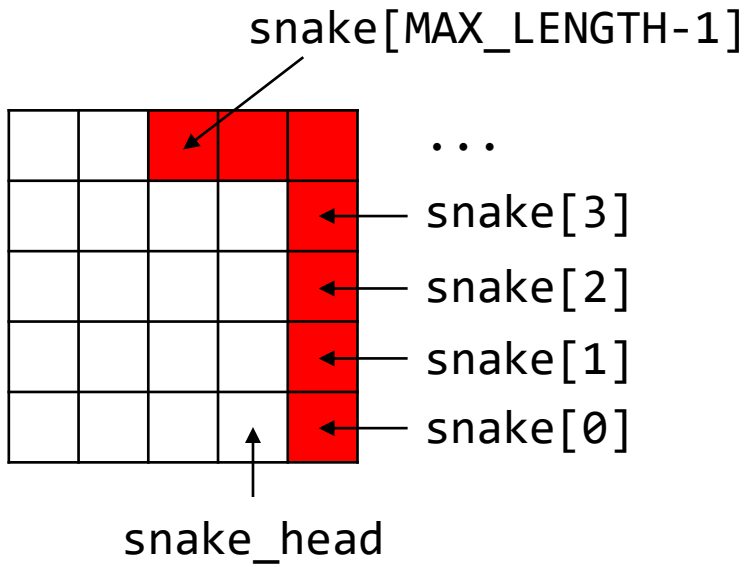
❷


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
snake_head

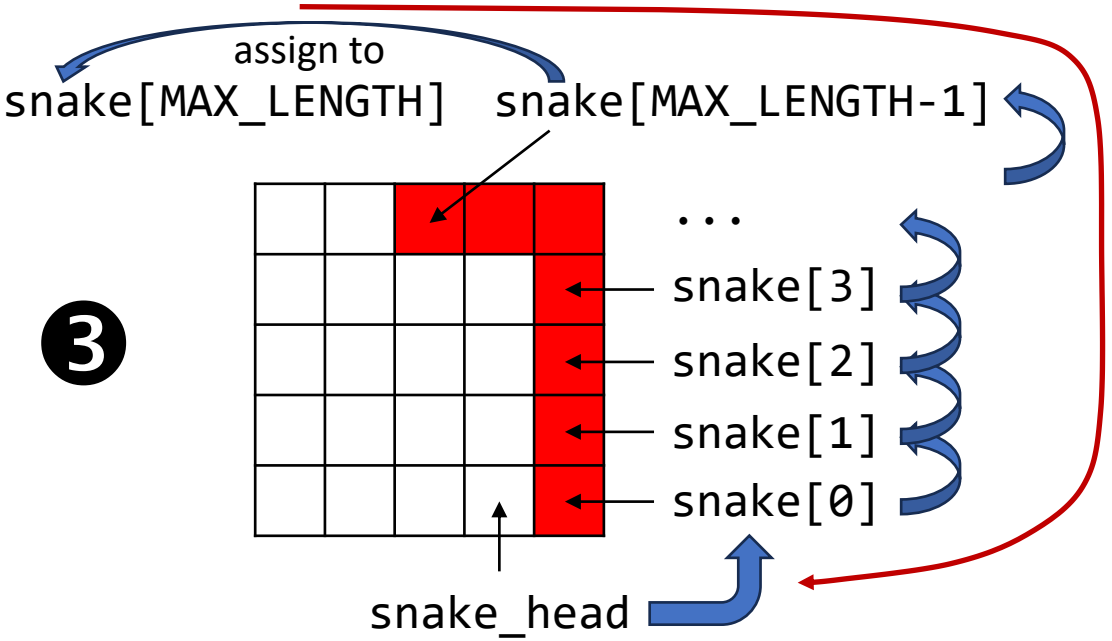
❶


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
=snake_head

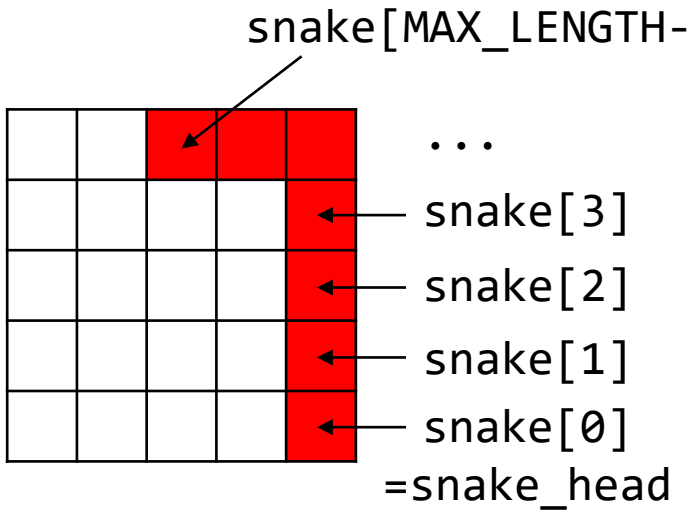
❷


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
snake_head

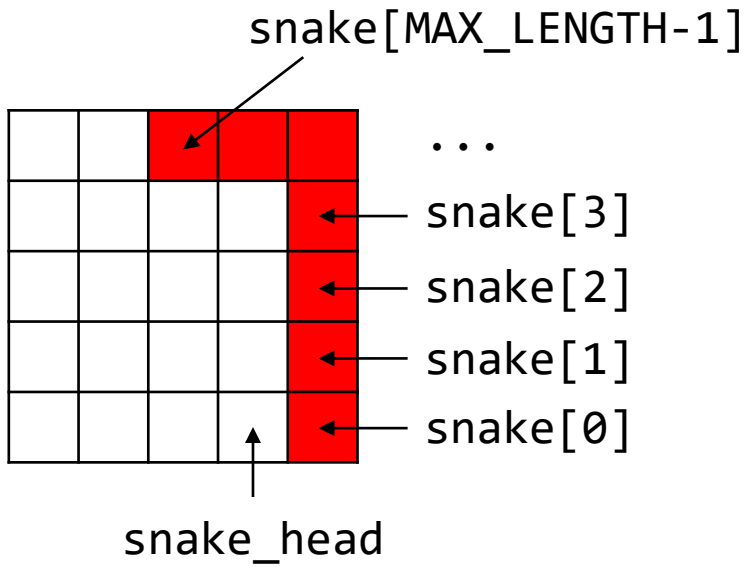
❸


assign to
snake[MAX_LENGTH]
snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
snake_head

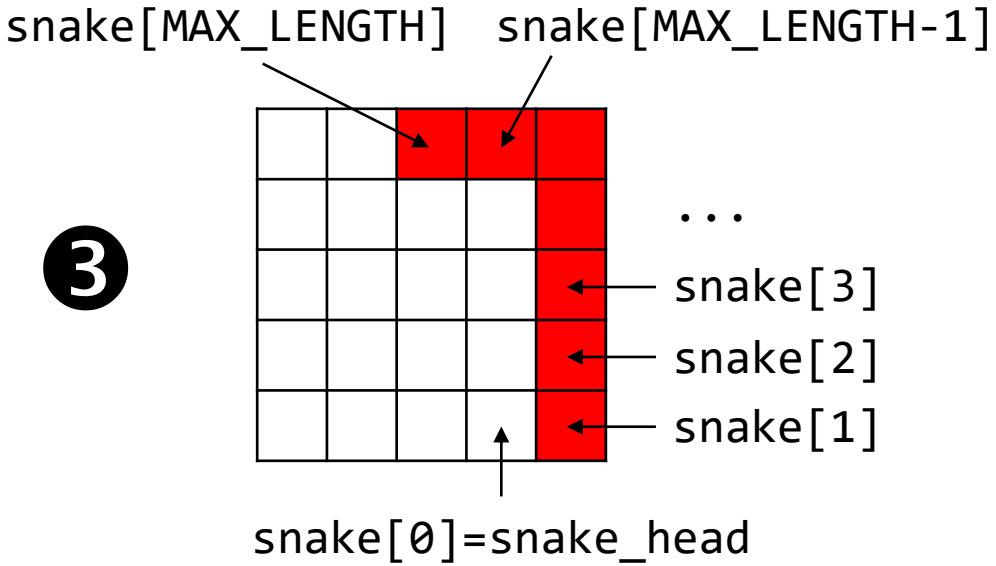
❶


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
=snake_head

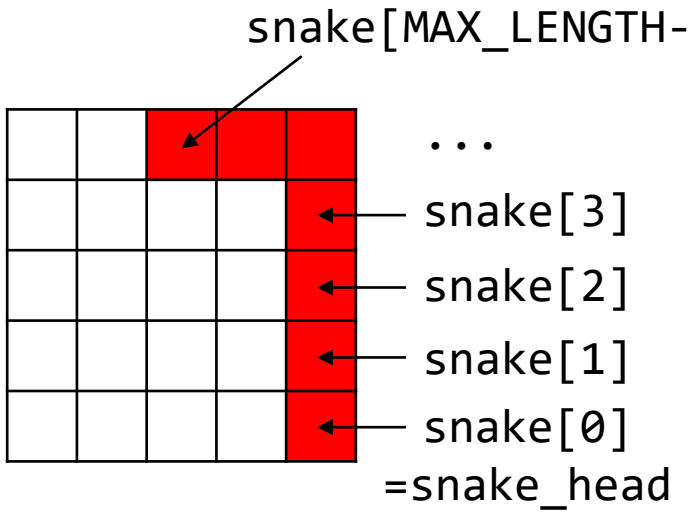
❷


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
snake_head

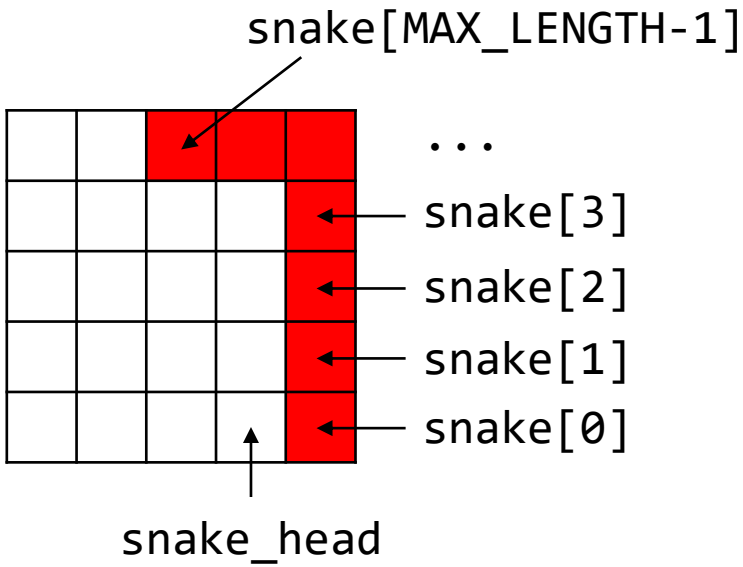
❸


snake[MAX_LENGTH]
snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]=snake_head

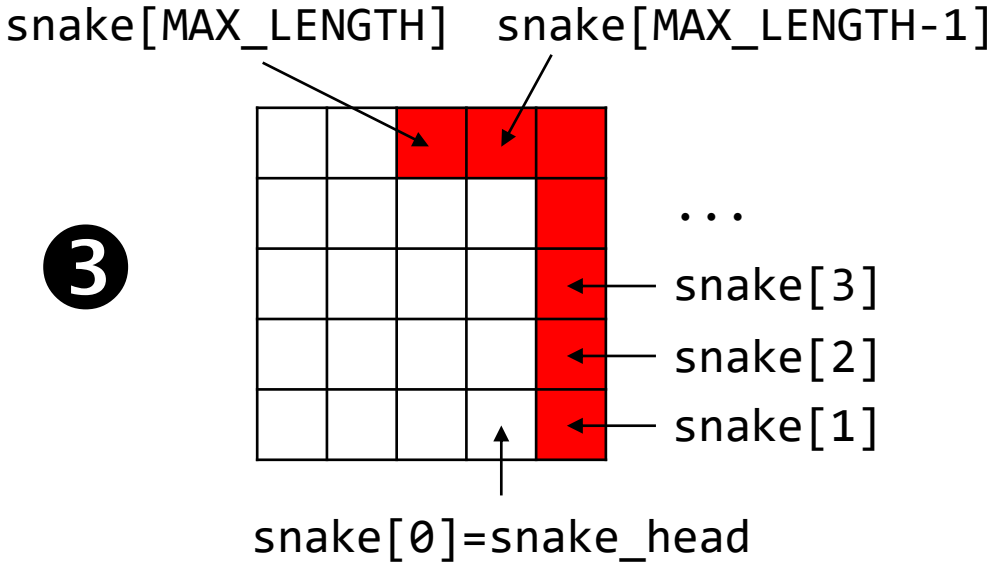
❶


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
=snake_head

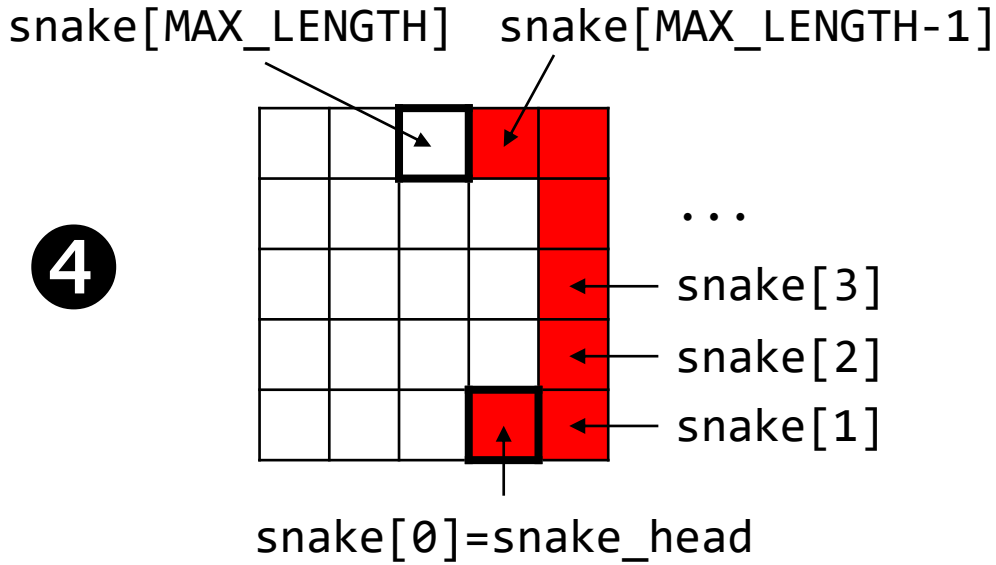
❷


snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]
snake_head

❸


snake[MAX_LENGTH]
snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]=snake_head

❹


snake[MAX_LENGTH]
snake[MAX_LENGTH-1]
...
snake[3]
snake[2]
snake[1]
snake[0]=snake_head

```
int main()
{
    // Initialise the coordinates of snake head
    struct bodypart snake_head = {0, 0};

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

    // Reveal the head of the snake
    // at the top-left corner of
    // the 5x5 display
    initialize_snake(snake_head);
```

```
int main()
{
    // Initialise the coordinates of snake head
    struct bodypart snake_head = {0, 0};

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

    // Reveal the head of the snake
    // at the top-left corner of
    // the 5x5 display
    initialize_snake(snake_head);
```

```
// Initialise the coordinates of
// each body part of the snake,
// so that the head of the snake is in
// position (new_head.x, new_head.y)
// and the remaining co-ordinates are
// set to -1.
void initialize_snake(struct bodypart new_head)
{
    int16_t i;
    snake[0] = new_head;

    for (i=1; i<MAX_LENGTH+1; i++)
        snake[i] = {-1, -1};
}
```

```
int main()
{
    // Initialise the coordinates of snake head
    struct bodypart snake_head = {0, 0};

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

    // Reveal the head of the snake
    // at the top-left corner of
    // the 5x5 display
    initialize_snake(snake_head);

    while (1)
    {
        // Print the snake
        display_snake();
        // Wait for 100 ms
        uBit.sleep(100);
```

```
int main()
{
    //
    str

    //
    uBi


    //
    //
    //
    ini

    while (1)
    {
        // Print the snake
        display_snake();
        // Wait for 100 ms
        uBit.sleep(100);
```

```
// Turn on body part [0] (the head) and turn off body part [MAX_LENGTH],
// where the tail (the last body part of the snake) used to be in the previous step.
void display_snake()
{
    // Move the head forward
    if (snake[0].x >=0 && snake[0].y >=0)
        uBit.display.image.setPixelValue(snake[0].x, snake[0].y, 255);

    // Erase the tail
    if (snake[MAX_LENGTH].x >=0 && snake[MAX_LENGTH].y >=0 )
        uBit.display.image.setPixelValue(snake[MAX_LENGTH].x, snake[MAX_LENGTH].y, 0);
}
```

```
if (snake_head.x < 4 && snake_head.y == 0)
    snake_head.x++;
else
{
    if (snake_head.x == 4 && snake_head.y < 4)
        snake_head.y++;
    else
    {
        if (snake_head.x > 0 && snake_head.y == 4)
            snake_head.x--;
        else
        {
            if (snake_head.x == 0 && snake_head.y > 0)
                snake_head.y--;
        }
    }
}
```

This part of the code follows the exact same logic of the ‘trapped dot’, which was explained in detail in Lecture 1 of Week 8 (slides 24-31)

```
        // Calculate the position of each body part of the snake,
        // now that the position of its head is known.
        shift_snake(snake_head);

    } // end of while-loop

} // end of main()
```

```
        // Calculate the position of each body part of the snake,
        // now that the position of its head is known.
        shift_snake(snake_head);

    } // end of while-loop

} // end of main()
```

```
// Move the snake. Its head (body part [0]) moves to the
// new coordinates (new_head.x, new_head.y)
// while body part i moves to the co-ordinates that
// body part i-1 used to occupy.
void shift_snake(struct bodypart new_head)
{
    int16_t i;
    for (i=MAX_LENGTH; i>0; i--)
        snake[i] = snake[i-1];

    snake[0] = new_head;
}
```


HAPPY
Holidays