Week 14 Lab Assignment

SCC.111: Object-Oriented Art

Goals

You have seen how programs can be separated different files (`Car.h`, `Car.cpp`, and `CarExample.cpp`), how libraries work, how to create your own library. Today you will apply this knowledge in a fun exercise: *Your job is to create and visualize a fun/creative drawing by using circles only*.

This lab aims to get you familiar with using external libraries written by other developers, and tests your creativity while creating your objects.

- ☐ Use and edit provided classes for your program
- ☐ Create an artistic drawing using the provided Circle and Canvas classes

Task 1: Environment Setup

Files Creation

To setup the work environment perform the following:

1. Open your terminal and navigate to a folder of your choice within your `h-drive`.
2. Run the following command:

```
git clone http://scc-source.lancs.ac.uk/scc.Y1/scc.111/week14.git
```

This will download a new folder named `week14` in the current directory that has the following files

```
1. `Circle.cpp`

2. `Circle.h`

3. `Canvas.cpp`

4. `Canvas.h`

5. `index.html`
```

These *cpp* files have no *main* function.

3. Create a new `.cpp` file that will be your main program (and will contain a `main` function). To do this:
    1. navigate to `cd week14`
    2. create the new file e.g., `touch art.cpp`
4. Open *VSCode* in the current directory using `code .`
5. Launch a Server: Open a **new terminal**, navigate to your folder (`week14`) and run the following command:

    ```
    python3 -m http.server
    ```

    - Don't close this terminal to keep the server running.
6. Open your browser and type `localhost:8000` in the address bar then *enter*, **don't close the browser**. The browser is your canvas, keep an eye on it when creating your drawings and updating the canvas as it should display the Canvas content as long as you don't *clear* it.

Note: you don't need to refresh the browser page.

Task 2: Creativity: Draw your Circles

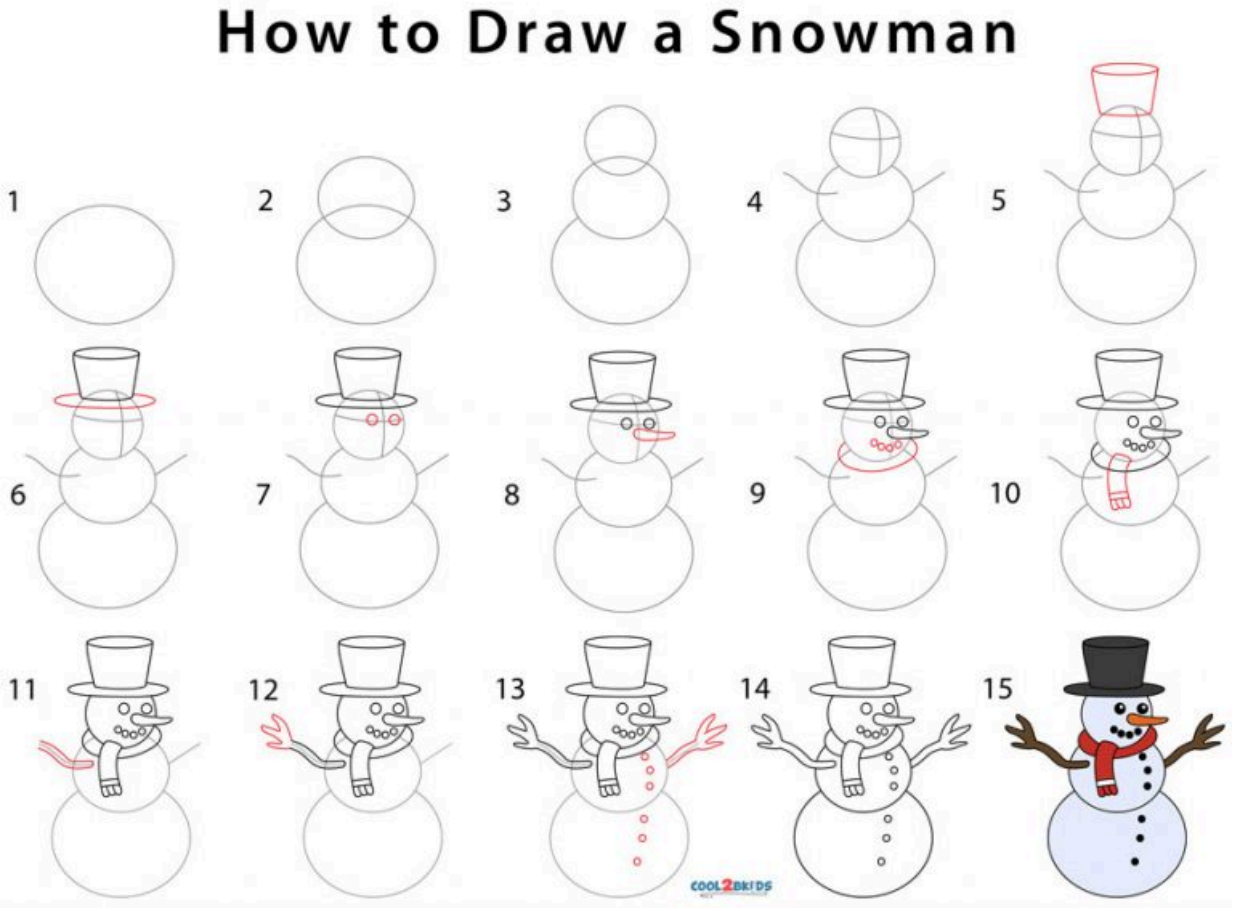
- Read and try to understand the code of *Circle* and *Canvas* files. Note particularly the libraries we rely upon, we'd recommend looking up the documentation for e.g. `std:string`, `std:vector` to find out how they work
- In your main program file, create new *Circle* objects and a new *Canvas* object. Add the *Circles* to the *Canvas* and use the `update` function to update your "drawing" in the browser page.
- Go back to the webpage and, if your code is working, you should be able to see your circle(s).

Here is a sample code to start with:

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-1)#include "Canvas.h"
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-2)#include "Circle.h"
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-3)#include <iostream>
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-4)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-5)int main() {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-6)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-7)    Canvas canvas(800, 600); // This create an 800x600 canvas
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-8)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-9)    Circle circle1(100, 100, 30, "white"); // a new white circle with a 30 radius in position (100,100)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-10)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-11)    canvas.addCircle(circle1);
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-12)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-13)    canvas.update();
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-14)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-15)    std::cout << "Canvas Updated!" << std::endl;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-16)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-17)    return 0;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-18)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb4-19)}
```

Sample output:

- **Simple Drawing**: Before diving into artistic creativity, let's try to draw a simple "*Snowman*". Try to follow a similar logic in the following image using circles only (we can skip other no-circle parts).



- Don't forget to compile both `Canvas.cpp` and `Circle.cpp` files when compiling your main program (which is `art.cpp` in our example):
  - `gcc -o art art.cpp Canvas.cpp Circle.cpp -lstdc++`
  - `./art`
- `Circle.colour` is a `string` variable, it accepts colour names supported by browsers e.g., `blue`, `red`, `green`, etc. Here is a list (https://www.w3schools.com/colors/colors_names.asp) of 140 values you can use.

> TIP: you can design your drawing online using https://editor.method.ac.
>
> - Select the Circle/Ellipse shape from the sidebar and draw your design
> - Click View->Source to see the coordinates of your circles as `cx` and `cy` and the radius as `rx` or `ry`. Note that `rx` and `ry` will have the same value if you draw perfect circles, otherwise for ellipses they will have different values, choose only one value as we are only drawing circles.

Share with us your designs

Best designs will be featured during next week lectures (And probably will receive a prize ;))

Task 3: Move it!

Now let's try to animate your drawing using the `move` function provided in the `Circle` class.

Create a loop and provide adequate moving commands for your drawing. Use `Canvas.update()` to update the movement, and `Canvas.pause()` for a brief pause for the new movement command, otherwise it will be too fast. You can call the `pause()` function multiple times for prolonged pauses.

Example:

```
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-1)    while(1)
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-2)    {
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-3)        circle1.move(1,0);
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-4)        circle2.move(1,0);
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-5)
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-6)        canvas.update();
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-7)        canvas.pause();
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb5-8)    }
```

Make sure that the circle coordinates stay within the Canvas borders (compare `X` and `Y` to the Canvas `Height` and `Width`).

Note: The web canvas can display up to 10 frames/updates per second.

SCC 121: Time Complexity of Algorithms

In this week's lab activities will continue to consider the time complexity of algorithms in terms of the number of operations executed and the complexity class (e.g. linear, logarithmic, quadratic, etc). We will build on the activities from last weeks labs to consider more examples of time complexity of different algorithms as well as practice some key mathematics. This week's lab exercises can all be done using pen and paper (or you can write your algorithms using c-code if you prefer). Where you are asked to write an algorithm, you can choose how you do this – remember there are different ways that an algorithm can be represented, however, you must ensure that you provide a clear, unambiguous instruction of how the algorithms input is transformed into an output.

Today we will cover:

- ☐ Practice math's questions on logarithms and sigma summation notation. Both are useful tools for evaluating the time complexity of algorithms.
- ☐ More examples of operation counting to determine the time complexity (in terms of the number of steps relative to the input size) of some simple algorithms
- ☐ Determining the complexity class of a given algorithm - in later weeks we will formalise this to use the Big-O, Big- and Big- notations
- Writing algorithms with a given complexity class.

Maths Practice

1. Compute the following (Please **DO NOT** use a calculator or computer):

-
-
-
-
-

2. Evaluate:

-
-
-
-

3. Recall that given a for loop of the following form,

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb6-1)for (i = a; i <= b; i++) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb6-2)    // instructions
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb6-3)}
```

The number of times the instruction within a for loop is executed can be given using the general rule

Apply this general rule to determine the final value of , expressed in terms of input size , given the following code fragment:

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb7-1)x = 0;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb7-2)for (int j = 1; j < n/2; j++) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb7-3)    for (int i = 1; i <= j; i++) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb7-4)        x = x + 1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb7-5)    }
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb7-6)}
```

Time Complexity

4. Count the number of operations (in the **best case** and the **worst case** for large) that the piece of code executes below. Assume is a positive integer number and assume that is counted as one operation.

```
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-1)int func_1(int n, int k){
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-2)    int count = 0;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-3)    while (n > 1) {
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-4)        count++;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-5)            if(k>5){
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-6)                n = n/3;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-7)            }
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-8)            else{
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-9)                n = n-1;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-10)            };
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-11)    }
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-12)    return count
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb8-13)}
```

5. Count the number of operations that the below piece of code executes. Assume is a positive integer number and that i=i*3 is counted as one operation.

```
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-1)int func2(int M){
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-2)    int count;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-3)    int i = 1;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-4)    while (i < M){
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-5)        count++;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-6)        i = i*3;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-7)        }
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-8)    return count;
(https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb9-9)}
```

6. The following algorithm describes a simple approach to computing the product of two n × n matrices A and B. What is its complexity class (e.g. constant, linear, quadratic, cubic etc.)?

```
input: A and B, both n by n matrices
initialize C to be an n by n matrix of all zeros
for i from 1 to n:
    for j from 1 to n:
        for k from 1 to n:
            C[i][j] = C[i][j] + A[i][k]*B[k][j]
**output** C
```

7. What is the complexity class (e.g. constant, linear, logarithmic, quadratic, cubic, etc) of the following functions? Which would you expect to complete the fastest for a large input size n?

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb11-1)exampleOne(int n){
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb11-2)int count = 0;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb11-3)for (int i = 0; i < n; i++)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb11-4)    for (int j = 0; j < n; j++)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb11-5)        for (int k = 0; k < n; k++)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb11-6)            count++;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb11-7)}
```

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb12-1)exampleTwo(int n){
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb12-2)int count = 0;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb12-3)for (int i=0; i<n; i++)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb12-4)    for (int j=n-5; j<n; j++)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb12-5)        for (int k=j; k<n; k++)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb12-6)            count++;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?
        time=1738593537571#cb12-7)}
```

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-1)exampleThree(int n){
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-2)int count = 0;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-3)    for (int i = 0; i < n; i*= 2) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-4)        for (int j = 0; j < n; j++) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-5)                count++;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-6)                break;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-7)        }
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-8)    }
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb13-9)}
```

8. You have been asked to provide a search algorithm to check if a given element is within a large unsorted array of integers of size n. Your algorithm must optimize the time complexity in the worst case. You have two options:

- First sort the data using `insertionSort`, then run a `binarySearch`.
- Run a `linearSearch`.

Assume that `insertionSort`, `binarySearch` and `linearSearch` are as given below. Which would you choose and why? What about in the case when your array was already sorted?

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-1)void insertionSort(int arr[], int n){
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-2)    int i, key, j;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-3)    for (i = 1; i < n; i++) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-4)        key = arr[i];
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-5)        j = i - 1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-6)        while (j >= 0 && arr[j] > key){
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-7)            arr[j + 1] = arr[j];
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-8)            j = j - 1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-9)        }
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-10)        arr[j + 1] = key;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-11)    }
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb14-12)}
```

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-1)int binarySearch(int arr[], int n, int s) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-2)    int lo = 0;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-3)    int hi = n - 1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-4)    int mid;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-5)    while (hi >=lo) {
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-6)        mid = (lo + hi)/2; //round to higher integer
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-7)        if (A[mid] == s)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-8)            return  1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-9)        elseif (A[mid]<s)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-10)            lo = mid + 1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-11)        else
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-12)            hi = mid – 1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-13)    }
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-14)    return 0
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb15-15)}
```

```
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb16-1)int linearSearch(int arr[], int n, int s)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb16-2){
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb16-3)    for (int i = 0; i < n; i++)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb16-4)        if (arr[i] == s)
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb16-5)            return 1;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb16-6)    return 0;
 (https://modules.lancaster.ac.uk/pluginfile.php/3929563/course/section/471755/week14-A.html?time=1738593537571#cb16-7)}
```

9. Let’s assume you are asked to implement an algorithm to break a password of length N by testing every possible combination (assuming numerical password, so each element could have a value 0-9). What is the time complexity class of the algorithm?

SCC.131: Intro to assembly

Goals

ARM assembly is a low-level programming language used for writing software for devices that use ARM processors. The language is characterized by its direct control over the hardware and its efficiency, making it suitable for applications where performance and resource usage are critical, such as in embedded systems, *i.e.* the MicroBit. The programming model is low-level and aligns with the capabilities of the ALU of the processor.

This lab aims to get you familiar with coding ARM assembly for the MicroBit. By the end of this lab, you should be able:

- ☐ Write simple Assembly instructions.
- ☐ Inspect the state of the CPU registers.
- ☐ Compile an Assembly program.
- ☐ Debug an Assembly program.

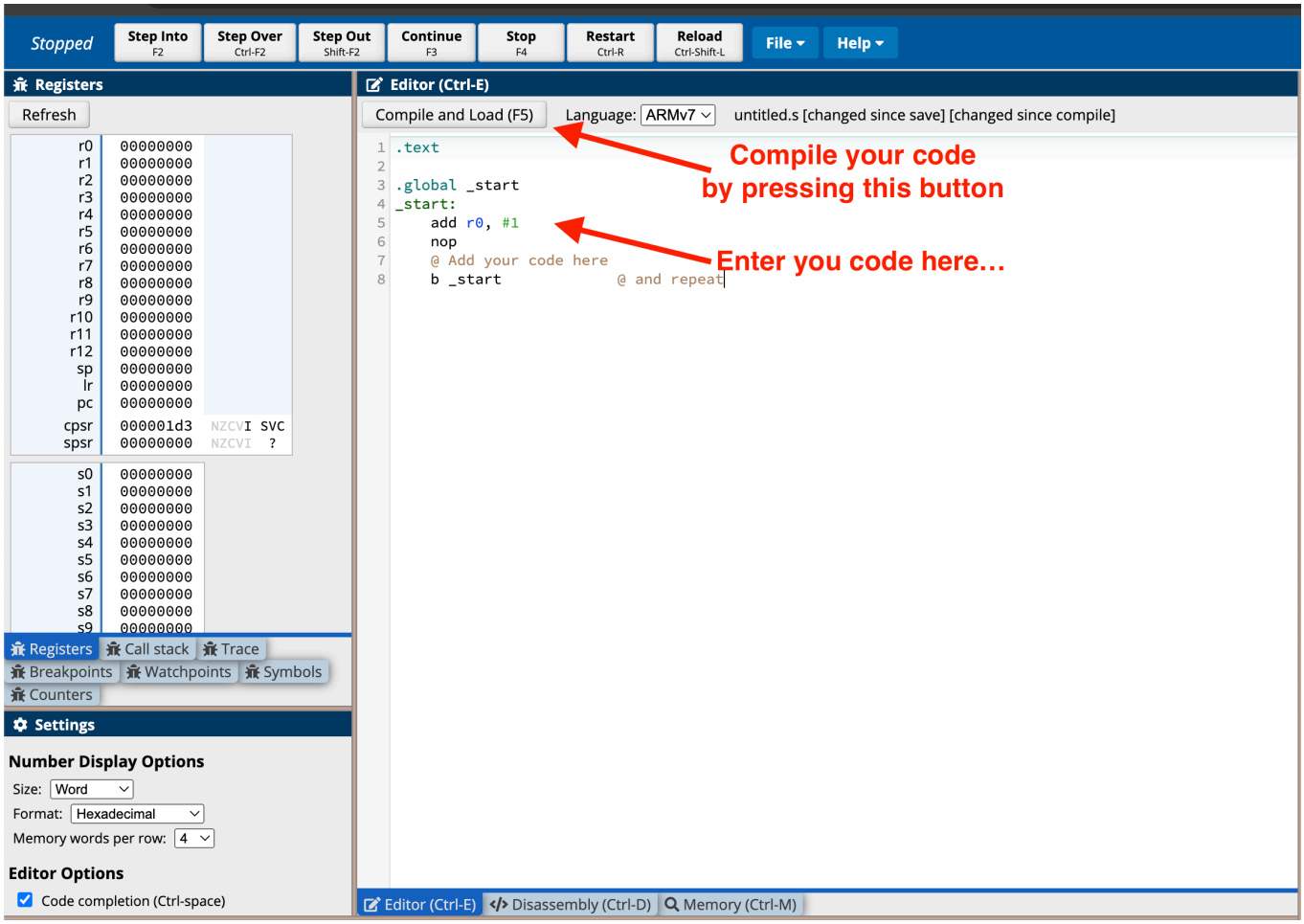
Task 1: ARM CPU simulator and micro:bit

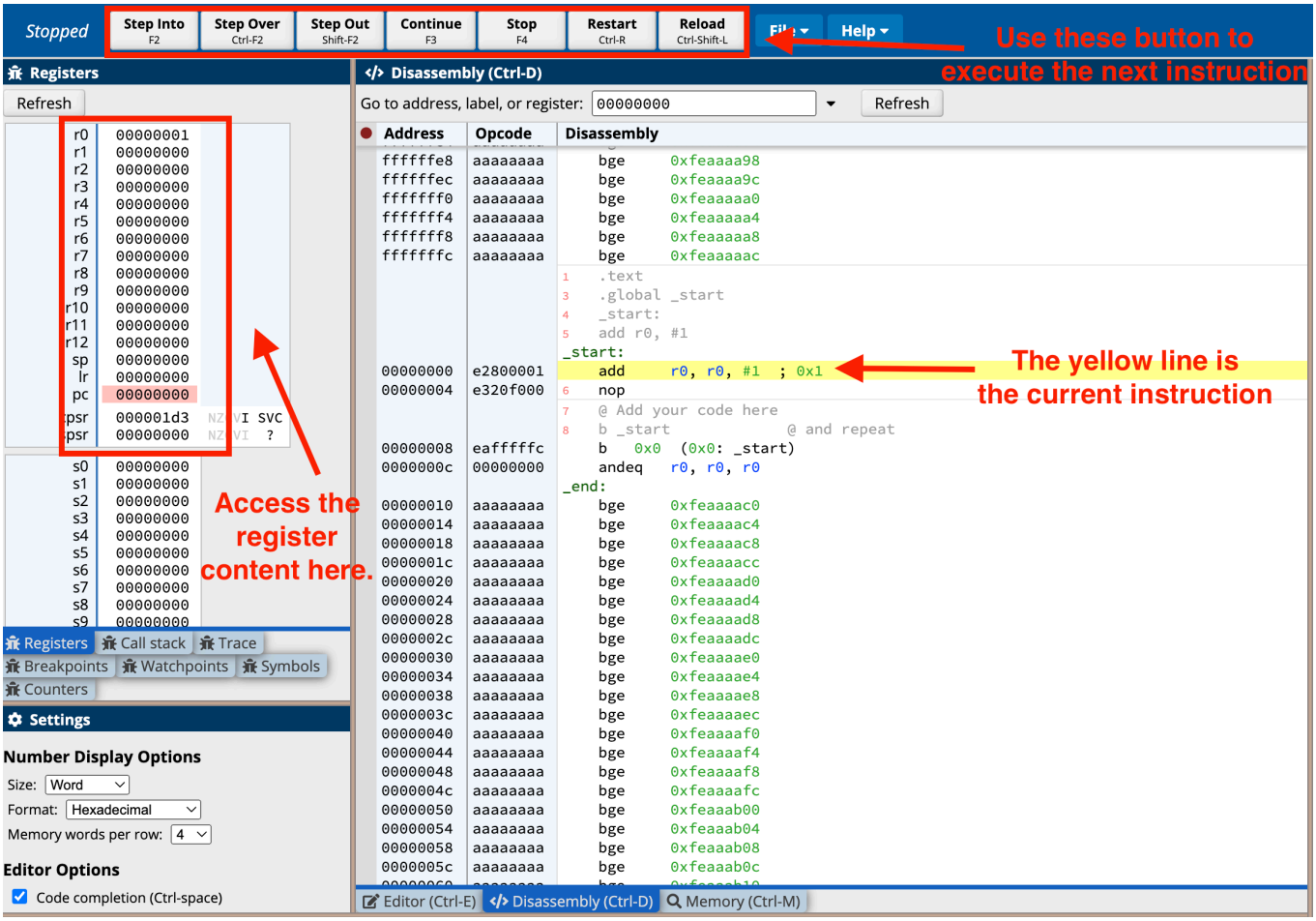
In order to allow greater flexibility in testing and writing code, we will be using two platforms to write and execute assembly code. The first one is called *CPUlator* and it emulates an ARM CPU in software. The platform is available online and it can be found at (https://cpulator.01xz.net/?sys=arm-de1soc)[https://cpulator.01xz.net/?sys=arm-de1soc]. The other platform will be based on the micro:bit and the debugger on board the chipset. To kick-off things, we will start off testing code on the CPUlator platform. Enter the following sample code:

```
.text

.global _start
_start:
    add r0, #1
    nop
    @ Add your code here
    b _start             @ and repeat
```

You should be able to compile the program above and execute it step by step, by using the key Ctrl+F2. Let’s start to code in assembly.


Stopped
Step Into F2
Step Over Ctrl-F2
Step Out Shift-F2
Continue F3
Stop F4
Restart Ctrl-R
Reload Ctrl-Shift-L
File
Help
Registers
Refresh
r0 00000000
r1 00000000
r2 00000000
r3 00000000
r4 00000000
r5 00000000
r6 00000000
r7 00000000
r8 00000000
r9 00000000
r10 00000000
r11 00000000
r12 00000000
sp 00000000
lr 00000000
pc 00000000
cpsr 000001d3 NZCVI SVC
spsr 00000000 NZCVI ?
s0 00000000
s1 00000000
s2 00000000
s3 00000000
s4 00000000
s5 00000000
s6 00000000
s7 00000000
s8 00000000
s9 00000000
Registers
Call stack
Trace
Breakpoints
Watchpoints
Symbols
Counters
Settings
Number Display Options
Size: Word
Format: Hexadecimal
Memory words per row: 4
Editor Options
Code completion (Ctrl-space)
Editor (Ctrl-E)
Compile and Load (F5)
Language: ARMv7
untitled.s [changed since save] [changed since compile]
1 .text
2
3 .global _start
4 _start:
5 add r0, #1
6 nop
7 @ Add your code here
8 b _start @ and repeat
Compile your code
by pressing this button
Enter you code here...
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)



Task 2: Register manipulation

For this first task, you should use a single instruction in your CPUlator program to set the value of a register. You can use the `MOV` (https://developer.arm.com/documentation/dui0473/j/writing-arm-assembly-language/load-immediate-values-using-mov-and-mvn) instruction to load immediate values into a register.

- **Goal**: Write a set of instructions that load the value 5 on register r0 and r1.*

Task 3: Basic arithmetic operations

Let's now try to execute an instruction the performs the operation and store the result in register r2. As discussed in lectures, in order to perform an operation on a CPU, you should first load the values into register. You can use the instructions in the previous step to initialize registers r0 and r1.

Note: there are more than ways to perform the add operation and your idea is correct as long as it computes the right outcome. You can also follow the link (https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/The-Instruction-Sets/Data-processing-instructions/Standard-data-processing-instructions?lang=en) to read more details for the ADD and

other arithmetic instructions.

Task 4: Large Constant Values

ARM assembly instruction can use immediate to hardcode integer values in an instruction, e.g., `add r0, r1, #2` will add the value 2 and the value stored in register r1 and store the result in register r0. In general, the maximum immediate values can vary across instructions, as the thumb format will try to immediate value, your assembler will generate and error and stop generating the compress the number of bits to 16 for most instructions. If you use a very big final binary. By compiling the assembly code, you vary the immediate values in an `add` and `mov` instruction to test what is the biggest immediate value that they support?

> Note: try different constant values for aßn add instruction, *e.g.* powers of 2, i.e. 2, 4, 8, …, 256, .

In order to control the full range of the 32-bit in a register the ARM assembly offers the instruction `MOVT register, constvalue`, which sets the upper 16 bits of the register to the constvalue (lower 16 bits remain the same).

```
movt r1, 0xFFFF @ r1 = 0xFFFF0000
```

For this task we want you to write a sequence of instructions to:

- Set r1 to hold a value of .
- Set r2 to hold a value of (note: thinking how to represent the value in hex or binary, will make your life easier as to how you should implement this).
- Compute and store the sum in r0.

Task 4: Bit Shifting

In this task we will explore what is the impact of the different shift instructions on register values. Let's load the value of 0x000000ff on register r0. Write an instruction for each operation:

- Shifting logically left by one bit.
- Shifting logically left by two bits.
- Shifting logically left by three bits.

Based on the results, can you think of a possible application of `lsl` in numeric calculation? Why does this work?

Let's now reset register r0 to the value , to explore the behaviour of the different right shift operations:

- Logically shift the right (`lsr`) the register r0 by 2 positions.
- Arithmetically shift to the right (`asr`) the register r0 by 2 positions.
- Perform a right rotation (`ror`) to the register r0 by 2 positions.

Are the results what you expected on the register value?

- Can you think of a shift operation that can transform register r0 to the value 0x7fe000000?

Task 5: Overflow

In this task, we will explore the behavior of the CPSR register for different arithmetic operations. Load and perform the following operators:

- 5 + 5
- 1 – 5
- 2147483647 + 1 (*note:*)

> Hint: in order to create the number you can use a combination of the `mov` and `movt` instructions (`mov r0, 0x0; movt r0, 0x8000`) or the `lsl` instruction (`mov r0, 0x1; lsl r0, r0, #31`) and then sub the value 1.

After each add instruction pause the program execution with a breakpoint and check the content of the register CPSR. For the last addition, run the code twice and use in one case the `add` instruction, and in the second run use the `adds` instruction. Do the values match what you expected? Did you get a different result on the register that you store the result of the addition? What is the impact of the `adds` instruction on the CPSR register?

Task 6: Compiling and debugging assembly for the MicroBit

For this final stage of this activity, we will use a simple assembly program, that runs bare-metal (no OS or runtime will be used to start your project) on the micro:bit, without the use of the CoDAL runtime. In future labs, we will explore how to execute custom assembly code from the main.cpp file, while using the CoDAL runtime.

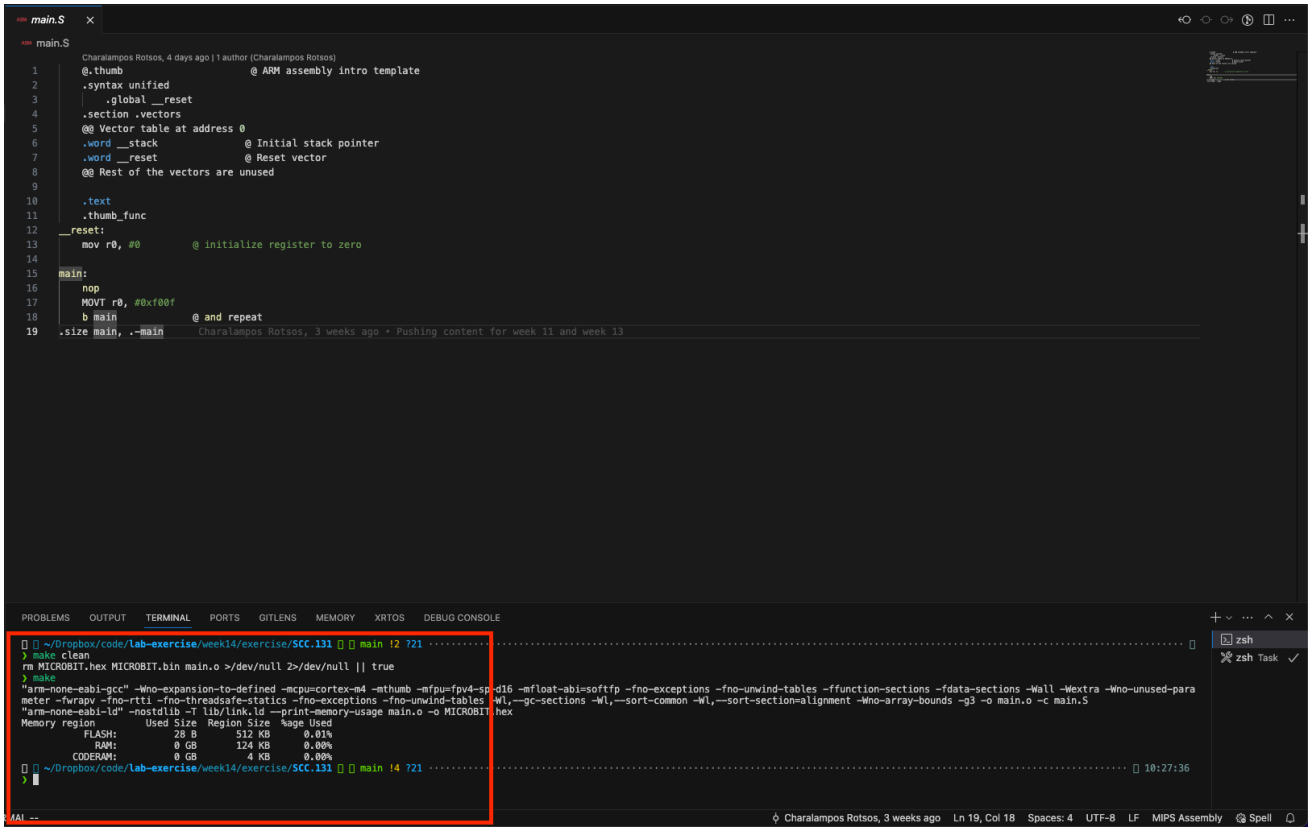
To start off, you must use the assembly project which you can download from the lab Moodle page (https://modules.lancaster.ac.uk/course/view.php?id=44651#section-3:~:text=microbit%20project%20code). It is worth having a look on the project files and try to understand the purpose of main files. In order to load the VS Code configuration of the project you must open from VS Code the root folder of the project using the command “Open Folder…” from the File menu or by pressing “Ctrl + O”.

In summary, the project contains the following files:

- **.vscode/*** This folder contains a set of configuration files for VS Code and allows your program to debug the project and build binaries.
- **src/main.S**: You should use this file to implement you coding task. This is the main file of the assembly project.
- **Makefile**: This is a configuration file for the make build tool. The file allows you to execute two actions: i) `make` will build the binary that can be flashed on the micro:bit and ii) `make upload` will flash the binary code on the micro:bit device.

In order to build your code, you have two options:

- Run the command `make` within the project folder. You can access a terminal using the bottom pane of your editor.



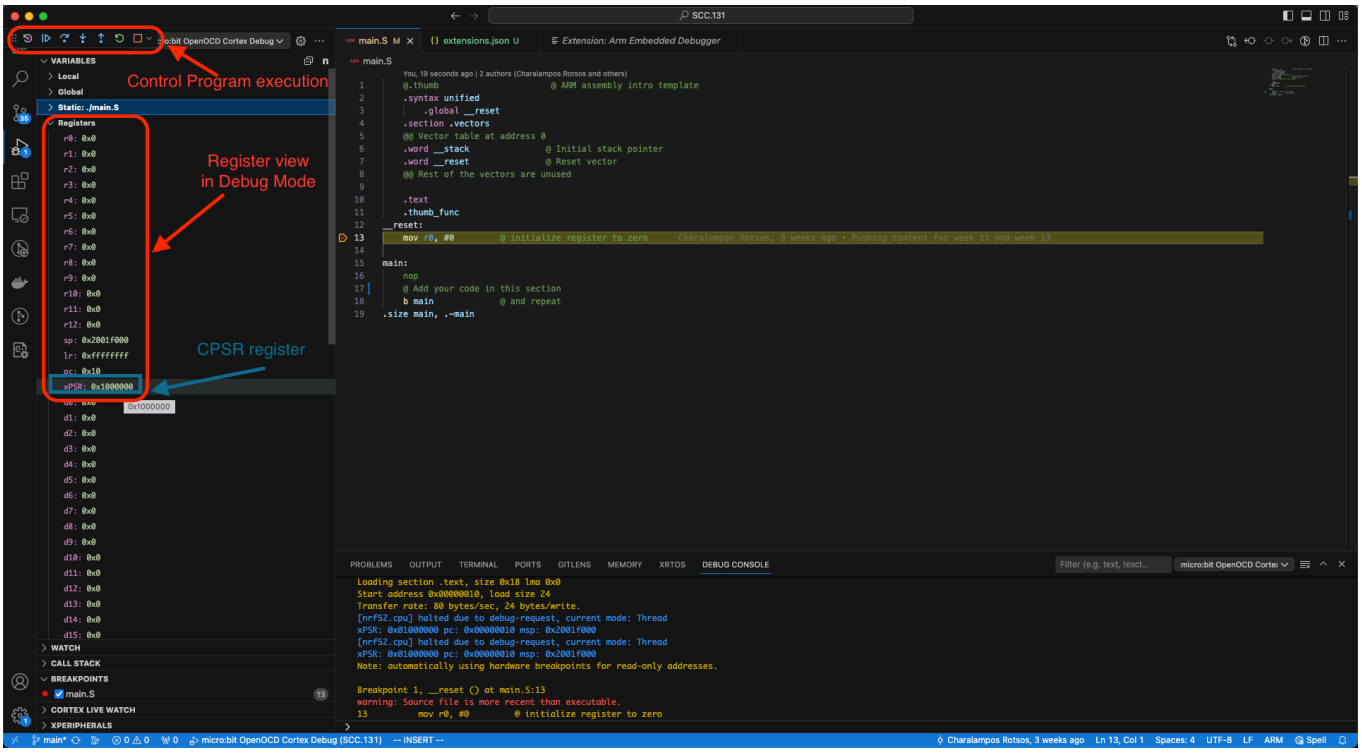
- You can run a build task from within VS Code, using the shortcut Shift+Ctrl+B.

Unfortunately, due to the low level nature of the ARM assembly, you will **not** have access to basic IO methods, like `printf`. The best way to test your code and verify that everything works well is by using the debugger. To start a debug session, you should first open the debug view on the side panel of VS Code. In order to run a debug session, you should select the *micro:bit OpenOCD Cortex Debug* option and press the play button. This step will flash the latest build code version on the micro:bit and start the debugger. Make sure that the micro:bit is connected and that you use a data USB cable. (Note: You do not need to move files on the MICROBIT device folder to flash the code.)

The debug process is similar to the work you did during week 13 (i.e., adding breakpoints, inspecting register content). If you want to learn more about debugging using the VS Code, please visit the following article (https://code.visualstudio.com/docs/editor/debugging).

The debugger offers on the side pane of the debug view direct access to all the CPU registers. You should use breakpoints and step through critical code regions to understand the purpose of different instructions. You can view the register values both in hex and decimal representations.

> The debugger functionality is only available in the lab machines, and it will not work on the mylab service.



Copy the code from the CPUlator window and test if the code still produces the same result, as in your simulation setup.

If you want to setup a build environment on your own machine, you can follow the guide released here (https://modules.lancaster.ac.uk/mod/folder/view.php?id=2589127).

Hacker Edition

SCC.111: Create a Drawing Class

Now, let's take your art project to the next level by creating a class to represent your drawing. This class will encapsulate the drawing logic and provide a clean structure for your artistic creations. Follow the steps below to enhance your project:

1. Create a *Drawing* Class Header and Implementation Files:
    - In the *week14* directory, create two new files named `Drawing.h` and `Drawing.cpp`.
    - Open `Drawing.h` and declare a class named *Drawing*. Include necessary headers, such as `Canvas.h` and `Circle.h`.
2. Implement the *Drawing* Class:
    - Open `Drawing.cpp` and implement the functions declared in the *Drawing* class.
3. Update Your Main Program:
    - In your `art.cpp` file, include the `Drawing.h` header and use the *Drawing* class to create and manage your artistic drawing.
4. Compile and Run:
    - Compile your program with the new *Drawing* class

Now, your artistic drawing is represented by a *Drawing* class, providing a more organized and extensible structure for your creative projects. Feel free to add more functionality to the *Drawing* class to enhance your drawing experience.

SCC.121: Time Complexity of Algorithms

1. A prime number is any natural number greater than 1 that cannot be exactly divided by any whole number other than itself and
    1. Write an algorithm with **better than linear** time complexity (in all cases) that checks if a number is prime or not.
2. A palindrome number is a number that reads the same forwards and backwards. In other words, if you have a palindrome number and reverse the digits you would get the same number. An example is 12321, if you read this backwards it is still 12321. Given a positive integer number, n, write an algorithm that checks if a number is a palindrome number or not. What is the time complexity class of your algorithm?

SCC.131: Reverse engineering assembly code

You are an ethical hacker, and you stumble upon the following assembly code when you reversed engineered a C binary. Do you understand the purpose of this code snippet? Initialize the value of register r0 with some example values (e.g., small and large integers, negative number etc.) and step through each instruction to inspect how the register value change.

```
mvn r0, r0
add r0, #1
```