Introduction to IoT

School Year 2023-2024

Valsalice

Course Structure

	Topic	Date
1	Introduction and Basics	
2	Basic Data Types and Operators	
3	Control Structures Pt. 1	
4	Control Structures Pt. 2	
5	Functions and Scope	
6	Arrays	NOV 21
10	Preprocessor and Macros	DEC 19
11	Custom Data Types	JAN 9
7	Introduction to Contiki-NG and nRF52840	NOV 28
8	Sensing and Actuating with Contiki-NG	DEC 5
9	Basic Communication and Networking	DEC 12
12	Introduction to RPL and Network Routing	JAN 16
13	Challenges in Wireless Communication	JAN 23
14	Advanced Protocols: TSCH and 6TiSCH	JAN 30
15	Advanced Topics in Wireless Communication	FEB 6
16	Reliable Data Transfer Challenge	FEB 20, FEB 27, MAR 5

🟥 = Core Topics ▨ = Optional Topics

Valsalice

Open your Virtual Machines

1. Turn on your Laptops
2. Login to Windows using “User”
3. Open the **Virtual Box** program
4. Add a new Virtual Machine (**Ctrl + A**)
5. Open the **VirtualBox** folder ⚠(**NOT** the .VirtualBox)
6. Select the **nRF52840LAB** file
7. Click **Start**

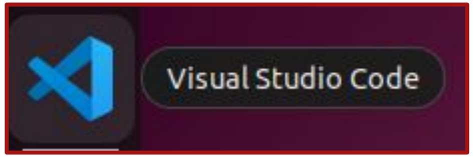
Prepare the Coding Environment

1. Start the Virtual Machine **nRF52840LAB**
2. Log-in using credentials:

Username: **ubuntu**

Password: **ubuntu**

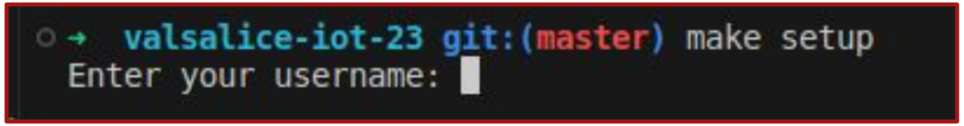
3. Open **Visual Studio Code** (use the App bar on the left)



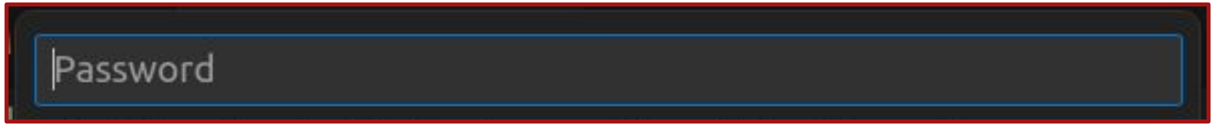
Prepare the Coding Environment

4 From the Terminal:

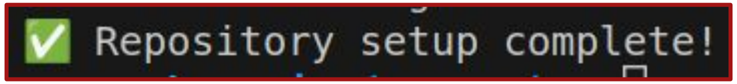
```
make setup
```

5


```
○ → valsalice-iot-23 git:(master) make setup
  Enter your username: █
```

6


```
Password
```

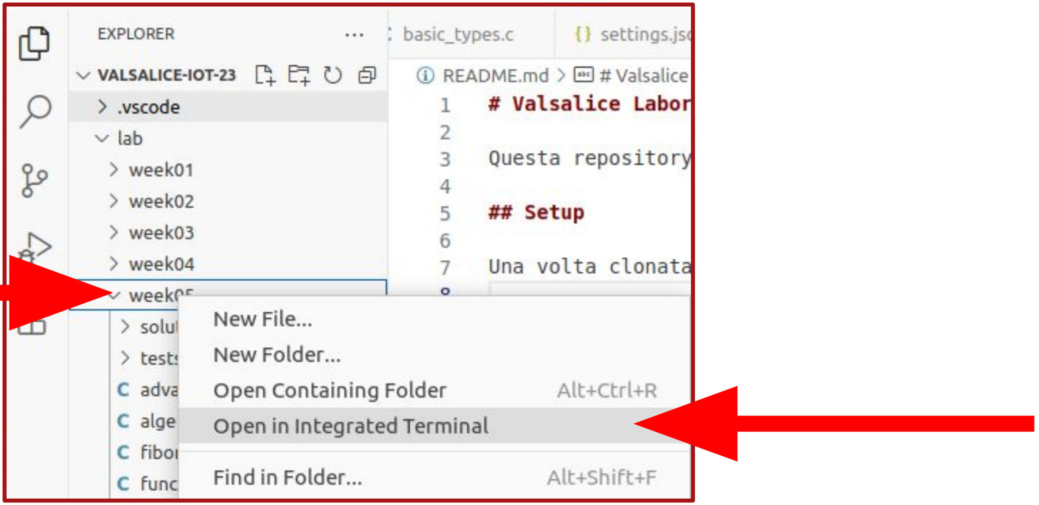
7


```
✅ Repository setup complete!
```

⚠ If you see **any (yellow) errors** input the credentials again

Prepare the Coding Environment

7 Open the **`week06`** folder in the terminal



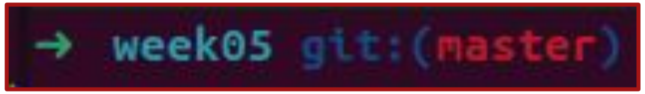
```
# Valsalice Labor

Questa repository

## Setup

Una volta clonata
```

8 You should see the following in the terminal:



```
➜  week05 git:(master)
```

Recap: Data Types

C has a number of primitive data types:

int	`42`	`1200`	`1_200`	`-3`
float	`3.14`	`0.00001`	`-2.1`	
char	`‘A’`	`‘@’`	`‘\n’`	
bool	`true`	`false`		

Strings are *NOT* a primitive data type, and have special syntax.

strings	`"Hello"`	`"A"`	`"I am a full sentence!"`

Recap: Variables

A variable is a named container that stores data or values.

```
int x = 42;
float y = -0.12;
char w = 'A';
char z[50] = "Full sentence";
```

Booleans require a custom include statement:

```
#include <stdbool.h>
bool hello = true;
```

Recap: Boolean Operators

Greater than	>
Greater or equal than	>=
Less than	<
Less or equal than	<=
Equals	==
Not equals	!=
Not	!

Recap: Chaining Comparisons

- **and** (both must be true)

```
true && false
```

```
(5 < 6) && (5 < 10)
```

- **or** (either must be true)

```
true || false
```

```
(5 < 3) || (5 < 10)
```

- **not** (negation)

```
!true
```

```
!(5 < 3)
```

Recap: If-Statement chaining

You can chain multiple conditions with **`else if`**.

What is the difference between these two snippets of code?

```c
int num;
scanf("%d", &num);

if (num < 3) {
    printf("Small number\n");
} else if (num < 10) {
    printf("Medium number\n");
}
```

```c
int num;
scanf("%d", &num);

if (num < 3) {
    printf("Small number\n");
}
if (num < 10) {
    printf("Medium number\n");
}
```

Recap: While-Loops

Repeat parts of your code!

```
int num;
printf("Input a number greater than 100: ");
scanf("%d", &num);

while (num <= 100) {
    printf("Wrong number, try again: ");
    scanf("%d", &num);
}

printf("Well done!\n");
```

Recap: For-Loops

Repeat a **specific** amount of times!

```
int x;


for (x = 1; x <= 5; x++) {
    printf("Hello %d\n", x);
}
```

```
int x = 0;


while (x < 5) {
    x += 1;
    printf("Hello %d\n", x);
}
```

Recap Exercise

Fill in all functions in (**`algebra.c`**):

- `calculate_expression(float x, float y, float z)`
  Returns the value of: 3x + 2y - z

- `double_or_negate(int x)`
  Returns 2x if x is even, otherwise -x

- `sum_or_min(int x, int y)`
  Return sum if it's less than 100, otherwise the smallest number

To execute:

```
make algebra.run
```

To test:

```
make algebra.test
```

Exercise - Solution

```
// Function to calculate 3x + 2y - z
float calculate_expression(float x, float y, float z)
{
    return 3 * x + 2 * y - z;
}
```

To execute: `make algebra.run` **To test:** `make algebra.test`

Exercise - Solution

```
// Function to return 2x if x is even, otherwise -x
int double_or_negate(int x)
{
   if (x % 2 == 0)
       return 2 * x;
   else
       return -x;
}
```

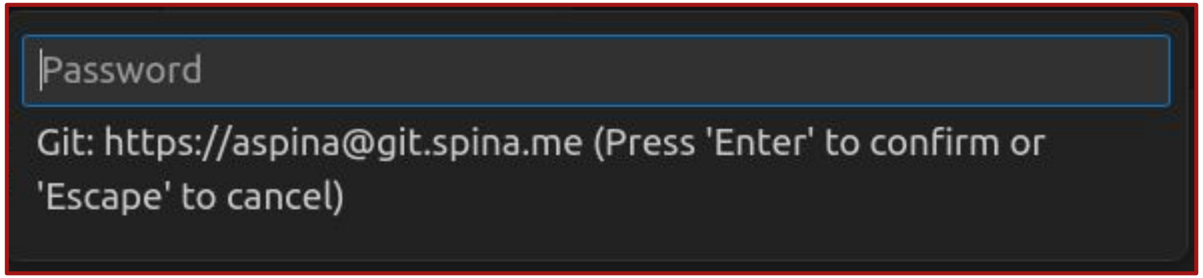
To execute: `make algebra.run` **To test:** `make algebra.test`

Exercise - Solution

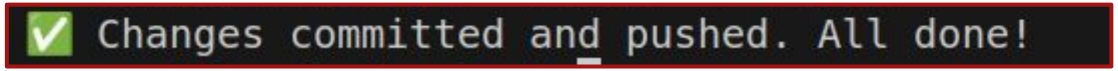
```
// Function to return sum if it's less than 100, otherwise the smallest number
int sum_or_min(int x, int y)
{
    int sum = x + y;
    if (sum < 100) {
        return sum;
    } else if (x < y) {
        return x;
    } else {
        return y;
    }
}
```

Save remotely your Changes

1
```
make save
```

2


```
Password
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or 'Escape' to cancel)
```

3


```
✅ Changes committed and pushed. All done!
```

Arrays

Modifiable containers for data.

With **variables**:

```
int num1 = 42;
int num2 = 100;
int num3 = 10;

printf("%d\n", num1);
printf("%d\n", num2);
printf("%d\n", num3);
```

With a **list**:

```
int array[] = {42, 100,
10};

for(int i = 0; i < 3; i++)
{
    printf("%d\n",
array[i]);
}
```

Arrays

Anatomy of an array:

1. Uses square brackets in the type declaration **[ ]**
2. Uses curly brackets for initialization **{ }**
3. Elements separated by comma **,**

```
int array[] = {42, 100, 10};
```

```
float array2[] = {0.23, 4.1};
```

```
char array3[] = {'a', 'Z'};
```

Accessing Array Elements

To access array elements you can use the **[index]** operator.

NOTE: List indices start from **0**

index:	**0**	**1**	**2**	**3**	**4**
`int array[] =`	`{17,`	`28,`	`33,`	`56,`	`6};`
index:	**-5**	**-4**	**-3**	**-2**	**-1**

```
printf("%d\n", array[0]);
```

```
printf("%d\n", array[3]);
```

Assigning Array Elements

To assign array elements you can use the **[index]** operator on the left-hand-side of a statement (like a variable)

```
int array[] = {17, 28, 33, 56, 6};
array[3] = 100;
array[2] = -7;
```

```
printf("%d\n", array[0]);
```

```
printf("%d\n", array[3]);
```

Array Simple Exercises

Fill in these functions in (**`array.c`**):

- `int getSecondElement(int arr[])`
  Returns the second element in arr

- `int getLastElement(int arr[], int size)`
  Returns the last element in arr

To execute:

```
make array.run
```

To test:

```
make array.test
```

Exercise - Solution

```c
// Function to get the second element of an array
int getSecondElement(int arr[])
{
    return arr[1];
}
```

To execute: `make array.run` **To test:** `make array.test`

Exercise - Solution

```
// Function to get the last element of an array
int getLastElement(int arr[], int size)
{
    return arr[size - 1];
}
```

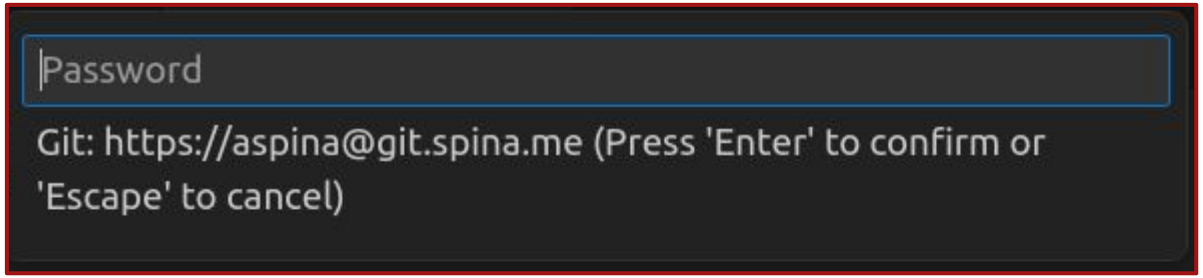
To execute: `make array.run` **To test:** `make array.test`

Save remotely your Changes

1

```
make save
```

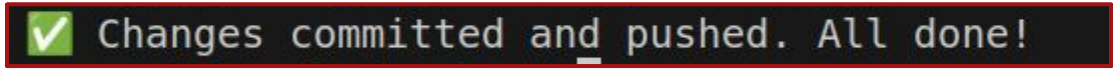
2



Password

Git: https://aspina@git.spina.me (Press 'Enter' to confirm or 'Escape' to cancel)

3



```
✅ Changes committed and pushed. All done!
```

Array Additional Exercises

Fill in these functions in (**`array.c`**):

- `void createArrayAllEights(int arr[])`
  Function to create an array of all 8s

- `void createArrayOneToFive(int arr[])`
  Function to create an array from 1 to 5

To execute:

```
make array.run
```

To test:

```
make array.test
```

Exercise - Solution

```
// Function to create an array of all 8s
void createArrayAllEights(int arr[])
{
    for (int i = 0; i < 5; i++)
    {
        arr[i] = 8;
    }
}
```

To execute: `make array.run` **To test:** `make array.test`

Exercise - Solution

```
// Function to create an array from 1 to 5
void createArrayOneToFive(int arr[])
{
    for (int i = 0; i < 5; i++)
    {
        arr[i] = i + 1;
    }
}
```

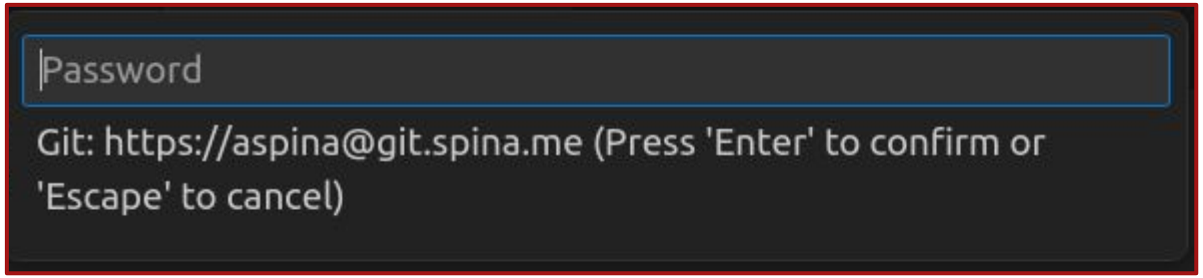
To execute: `make array.run` **To test:** `make array.test`

Save remotely your Changes

1

```
make save
```

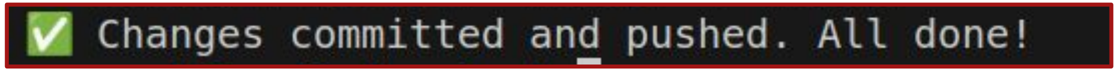
2



Password

Git: https://aspina@git.spina.me (Press 'Enter' to confirm or 'Escape' to cancel)

3



```
✅ Changes committed and pushed. All done!
```

Array Advanced Exercises

Fill in these functions in (**`array.c`**):

- `void doubleArrayValues(int arr[], int size)`
  Function to double the values in an array

- `bool containsNumber(int arr[], int size, int number)`
  Function to check if an array contains a number

- `int sumArray(int arr[], int size)`
  Function to sum the elements of an array

To execute:

```
make array.run
```

To test:

```
make array.test
```

Exercise - Solution

```c
// Function to double the values in an array
void doubleArrayValues(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        arr[i] *= 2;
    }
}
```

To execute: `make array.run` **To test:** `make array.test`

Exercise - Solution

```
// Function to check if an array contains a number
bool containsNumber(int arr[], int size, int number) {
   for (int i = 0; i < size; i++) {
       if (arr[i] == number) {
           return true;
       }
   }
   return false;
}
```

To execute: `make array.run` **To test:** `make array.test`

Exercise - Solution

```
// Function to sum the elements of an array
int sumArray(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}
```

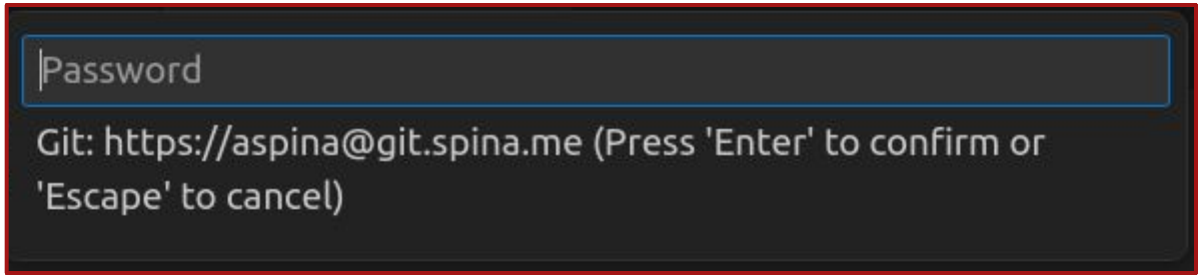
To execute: `make array.run` **To test:** `make array.test`

Save remotely your Changes

1

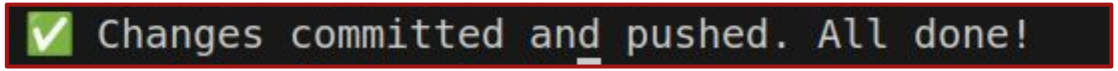
```
make save
```

2



```
Password
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or
'Escape' to cancel)
```

3



```
✅ Changes committed and pushed. All done!
```

End of Class

See you all next week!