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	NOV 14
6	Arrays and Strings	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**

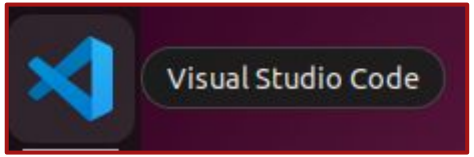
Prepare the Coding Environment

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

   Username: **ubuntu**

   Password: **ubuntu**

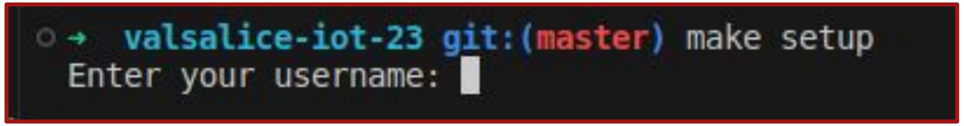
3. Open **Visual Studio Code** (use the App bar on the left)



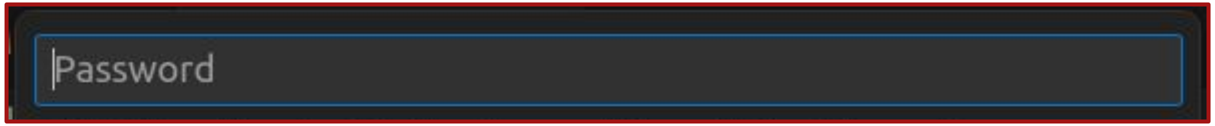
Prepare the Coding Environment

4 From the Terminal:

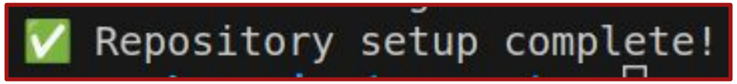
```
make setup
```

5 

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

6 

Password

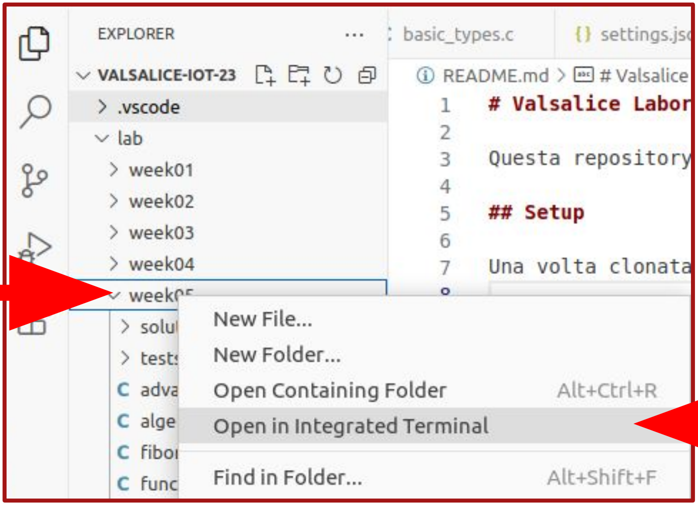
7 

```
✅ Repository setup complete!
```

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

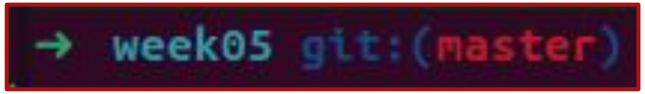
Prepare the Coding Environment

7 Open the **week05** 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: Basic Input/Output Functions

- **`printf`**: C function for formatted output

```
printf("Hello, World!\n");
```

- **`scanf`**: C function for formatted input

```
char name[50];
scanf("%s", name);
```

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.

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

Booleans require a custom include statement:

```c
#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-Statements

Allow for branches in your code!

```
int x = 5;


if (x < 10) {
    printf("X is small \n");
} else {
    printf("X is large \n");
}
```

```
int x = 20;


if (x < 10) {
    printf("X is small \n");
} else {
    printf("X is large \n");
}
```

NOTE: You **do not** need an else block, it's optional.

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!

```c
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

Write a program (**`squares.c`**) that, given a variable **num** prints out all the squares from **1** to **num**. (Use loops!)

```
________ (________)
{
    printf(________);
}
```

Example output:

```
Insert a number: 4
1
4
9
16
```

To execute:

```
make squares.run
```

Exercise - Solution

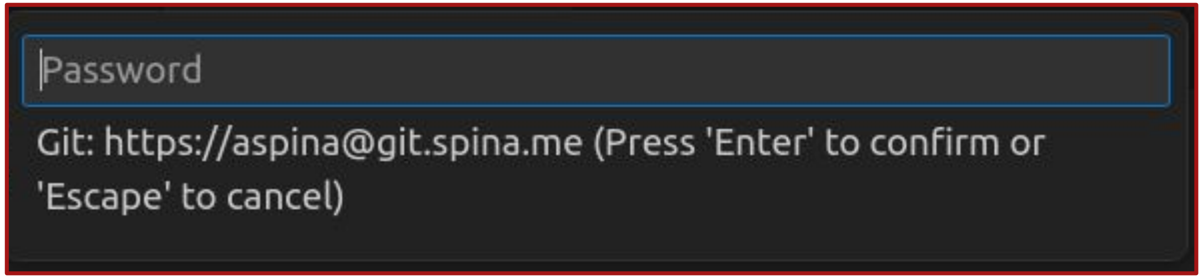
```c
for (int x = 0; x <= num; x++)
{
    printf("%d\n", x * x);
}
```

Save remotely your Changes

1

```
make save
```

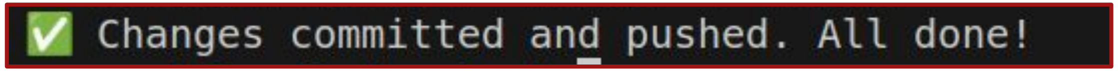
2



Password

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

3



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

Functions

You can define functions: custom snippets of reusable code

```
void print_square(int num)
{
    for (int x = 0; x <= num; x++)
    {
        printf("%d\n", x * x);
    }
}
```

Functions

Anatomy of a function:

1. Must start with the **return type**
2. **Parameters** between brackets:

   ( )

3. The **body** is between curly brackets:

   { }

```
void print_num(int num)
{
    printf("%d\n", num);
}
```

Exercise

Write a program (**`function.c`**) that, given a variable **num** uses calls the function **print_squares** three times.

```
void print_square(int num) {
    for (int x = 0; x <= num; x++)
    {
        printf("%d\n", x * x);
    }
}
```

To execute:

```
make function.run
```

Exercise - Solution

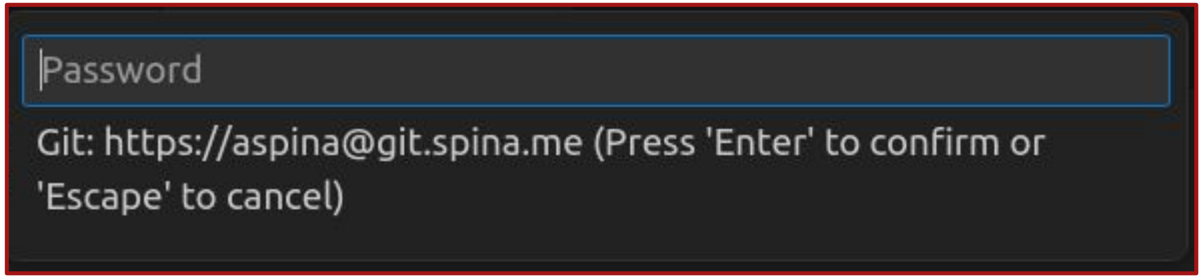
```
print_squares(num);


print_squares(num);


print_squares(num);
```

Save remotely your Changes

1

```
make save
```

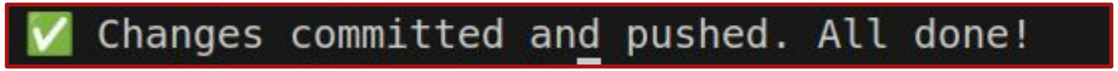
2



Password

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

3



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

Functions

Functions can return data!

The **return type** is no longer **void**, and it must use **return**.

```
// Function to add two numbers
int add(int num1, int num2)
{
    return num1 + num2;
}
```

```
// Function to divide two numbers
float divide(int num1, int num2)
{
    return (float)num1 / num2;
}
```

Exercise

Implement all functions inside (**`algebra.c`**).

Function descriptions are the specification

To execute: `make algebra.run`

To run automated tests: `make algebra.test`

If you finish early: **advanced.c**

Exercise - Solutions

```
// Function to add two numbers
int add(int num1, int num2)
{
    return num1 + num2;
}
```

Exercise - Solutions

```
// Function to subtract two numbers
int subtract(int num1, int num2)
{
   return num1 - num2;
}
```

Exercise - Solutions

```
// Function to multiply two numbers
int multiply(int num1, int num2)
{
    return num1 * num2;
}
```

Exercise - Solutions

```
// Function to divide two numbers
float divide(int num1, int num2)
{
    return (float)num1 / num2;
}
```

Exercise - Solutions

```
// Function to compare two numbers and return true
// if num1 is greater than num2
bool greater_than(int num1, int num2)
{
    return num1 > num2;
}
```

Exercise - Solutions

```
// Function to compare two numbers and return true
// if num1 is less than num2
bool less_than(int num1, int num2)
{
    return num1 < num2;
}
```

Exercise - Solutions

```
// Function to compare two numbers and return true
// if num1 equals num2
bool equals(int num1, int num2)
{
    return num1 == num2;
}
```

Advanced Exercise - Solutions

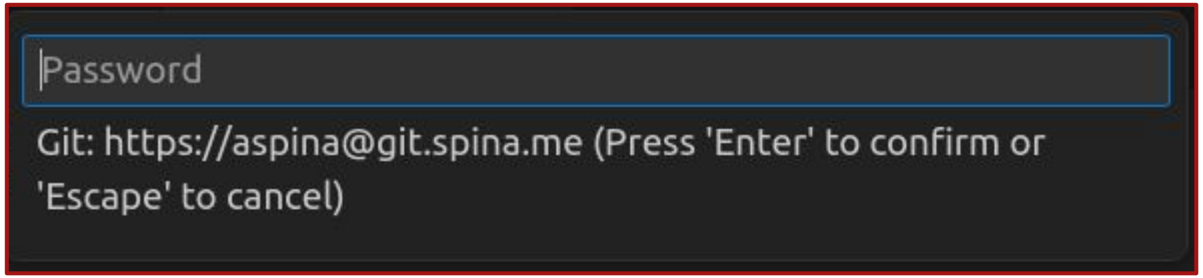
```c
// Function to calculate the next even number
int next_even(int num) {
    if (num % 2 == 0) {
        // If the input is even, return the next even number.
        return num + 2;
    } else {
        // If the input is odd, return the next even number.
        return num + 1;
    }
}
```

Save remotely your Changes

1

```
make save
```

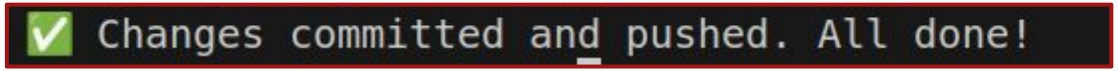
2



Password

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

3



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

Recursion

You can call functions from other functions.

Including the **same function**. Edit **fibonacci.c** as follows:

```c
int fibonacci(int n)
{
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

To execute & test: `make fibonacci.run` `make fibonacci.test`

Advanced Exercise - Solutions

```
// Function to calculate the factorial of a non-negative integer
int factorial(int num)
{
    // This exercise can also be solved with recursion
    int result = 1;
    for (int i = 1; i <= num; i++)
    {
        result *= i;
    }
    return result;
}
```

Advanced Exercise - Solutions

```
// Calculate the sum of all natural numbers from 1 to n
int sum_of_natural_numbers(int num)
{
   // This exercise cal also be solved with a loop
   if (num <= 0) {
       return num;
   } else {
       return num + sum_of_natural_numbers(num - 1);
   }
}
```

Advanced Exercise - Solutions

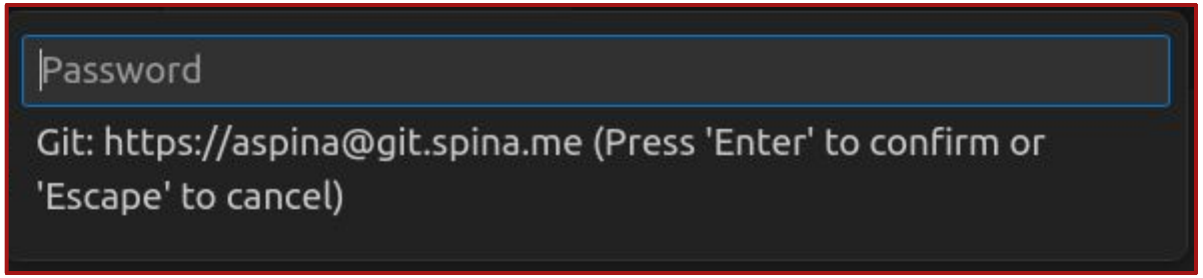
```
// Function to check if a number is a prime number
bool is_prime(int num) {
    if (num <= 1) {
        return false;
    }
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}
```

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!