Introduction to IoT

School Year 2023-2024

Valsalice

Course Structure

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
7. Introduction to Contiki-NG and nRF52840
8. Sensing and Actuating with Contiki-NG
9. Timers and Concurrency
10. Advanced String Usage
11. Custom Data Types — JAN 9
12. Basic Communication and Networking — JAN 16
13. Introduction to RPL and Network Routing — 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

Open your Virtual Machines

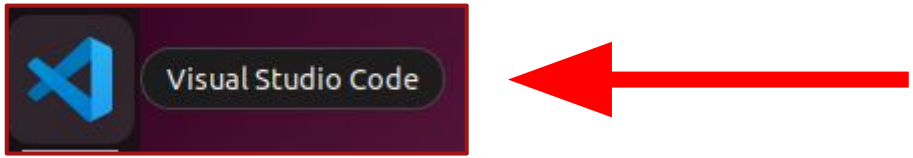
1. Turn on your Laptops
2. Login to Windows using “User”
3. Open the **Virtual Box** program
4. Add a new Virtual Machine (**Ctrl + A**)
5. Open the **VirtualBox** folder ⚠ (**NOT** the .VirtualBox)
6. Select the **nRF52840LAB** file
7. Click **Start**

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. 

6. 

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

7. Open the **week11** folder in the terminal

   Right click on the left + "**Open in Integrated terminal**"

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!

```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);
}
```

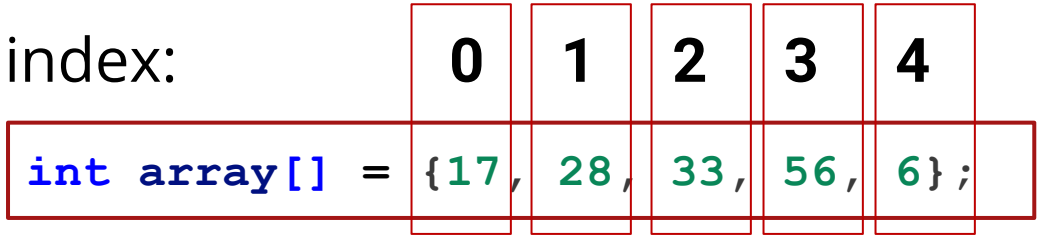
```
int x = 0;

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

Recap: Array Elements

Modifiable containers for data.

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

NOTE: List indices start from **0**




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

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

Recap: 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]);
```

Recap: Functions

Functions are custom snippets of reusable code:

1. If the **return type** is **void** the function does NOT return.
2. If the **return type** is NOT void, it MUST use **return**.

1

```
// Function to print a number
void print_num(int num) {
    printf("%d\n", num);
}
```

2

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

Recap: Calling Functions

Functions can be called any number of times:

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

```
int x = add(4, 100);
int y = add(60, 30);
int z = add(x, y);
```

Anatomy of a Struct

Structs are custom Data Types.

1. Must be declared with the **struct** keyword.
2. Can contain any number of fields.
3. Every field must have:
   a. Type
   b. Name
4. The **typedef** keyword can be used to create a custom type.

```
typedef struct {
    char name[50];
    int age;
    float height;
} student_t;
```

Using Structs

```
typedef struct {
    int age;
    float height;
} student_t;
```

Structs can be initialised inline or left empty.

```
student_t alice;
alice.age = 24;
alice.height = 1.54;
printf("Alice. Age: %d. Height %f.", alice.age, alice.height);
```

```
student_t bob = {19, 1.76};
printf("Bob has age %d and height %f.", bob.age, bob.height);
```

Using Structs

Every struct field can be read, assigned and overridden.

```
typedef struct {
    int age;
    float height;
} student_t;
```

```
student_t charlie;
charlie.age = 24;
charlie.height = 1.54;
printf("Age %d.",
charlie.age);

charlie.age = 30;
printf("Age %d.",
charlie.age);
```

Exercise

⚠ `make simple.test`

Define these three structs inside (**`simple.c`**).

1. Create a struct named **student_t** with two members: **height** and **age**.
2. Define a struct **point_t** representing a point in 2D space with **x** and **y** integer coordinates.
3. Define a struct **prism_t** representing a 3D prism with **width**, **height** and **depth** float values.

Exercise Solution

```
typedef struct
{
    float height;
    int age;
} student_t;
```

Exercise Solution

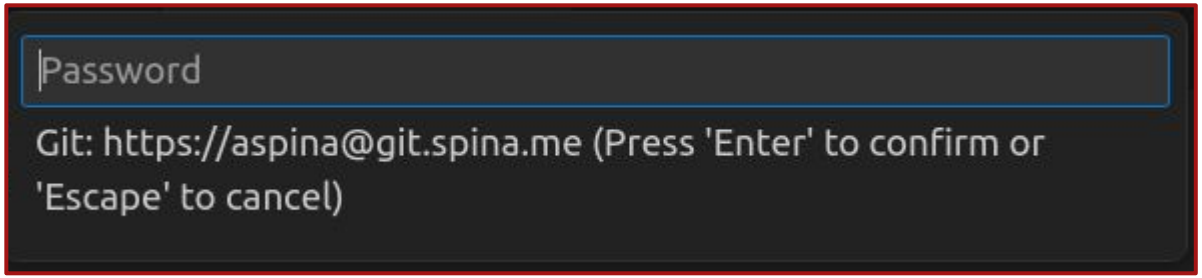
```
typedef struct
{
    int x;
    int y;
} point_t;
```

Exercise Solution

```
typedef struct
{
    float width;
    float height;
    float depth;
} prism_t;
```

Save remotely your Changes

1. `make save`

2. 

3. 

Using Structs

Structs can be passed as input to functions and also returned.

```
typedef struct {
    int age;
    float height;
} student_t;
```

```
int add_ages(student_t first, student_t second)
{
    return first.age + second.age;
}
```

```
student_t create_student(int age, float height)
{
    student_t new_student = {age, height};
    return new_student;
}
```

Exercise

`make medium.test`

Implement the following (**`medium.c`**).

1. Implement function **add_heights** that given two parameters of type **student_t** returns the sum of the heights.
2. Implement function **create_point** that given two parameters **x** and **y** returns a new **point_t** struct with the values set.
3. Implement function **calculate_volume** that given a **prism_t** parameter returns its volume.

Exercise Solution

```c
int add_heights(student_t first_student, student_t second_student)
{
    return first_student.height + second_student.height;
}
```

Exercise Solution

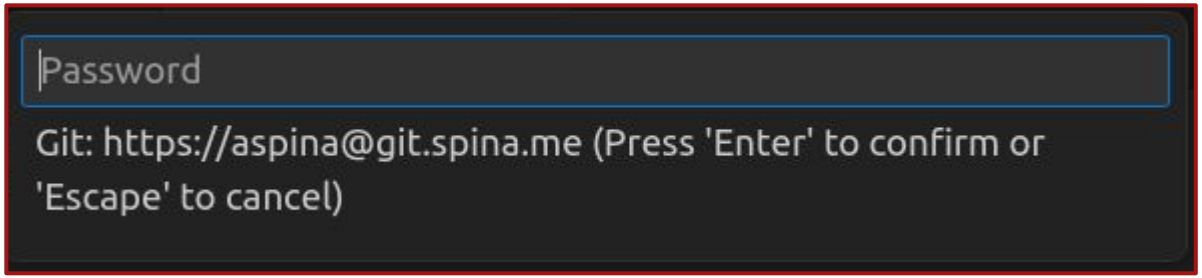
```
point_t create_point(int x, int y)
{
    point_t new_point;
    new_point.x = x;
    new_point.y = y;
    return new_point;
}
```

Exercise Solution

```
float calculate_volume(prism_t prism)
{
    return prism.width * prism.height * prism.depth;
}
```

Save remotely your Changes

1. `make save`

2. 

3. 

Using Structs

Structs can be nested into each-other.

```
typedef struct {
    float width;
    float height;
    float depth;
} prism_t;

typedef struct {
    prism_t dim;
    float weight;
} box_t;
```

```
box_t box = {{10, 20, 30}, 45.8};
printf("Height: %f\n.", box.dim.height);
printf("Weight: %f\n.", box.weight);
```

Exercise

 `make nested.test`

Implement the following (**`nested.c`**).

1. Define a new struct **rectangle_t** that has two fields **left** and **right** both of type **point_t**.
2. Implement function **find_area** that given a **rectangle_t** struct, returns the area of the rectangle.

Exercise Solution

```
typedef struct
{
    point_t left;
    point_t right;
} rectangle_t;
```

Exercise Solution

```c
int find_area(rectangle_t rect)
{
    int width = rect.right.x - rect.left.x;
    int height = rect.right.y - rect.left.y;
    return width * height;
}
```

Using Structs

Structs can also be part of arrays.

```
typedef struct {
    int age;
    float height;
} student_t;
```

Here each value of the array is a struct:

```
student_t students[3] = {{18, 1.75}, {20, 1.85}, {24, 1.68}};


printf("Second Age: %d.\n", students[1].age);
printf("First Height: %f.\n", students[0].height);
```

Exercise

⚠ `make advanced.test`

Implement the following (**`advanced.c`**).

1. Implement function **max_age** that given an array of type **student_t** structs, returns the age of the oldest student.
2. Implement function **are_points_equal** that an array of **point_t** structs, returns true if all points have same **x** and **y**.

Exercise Solution

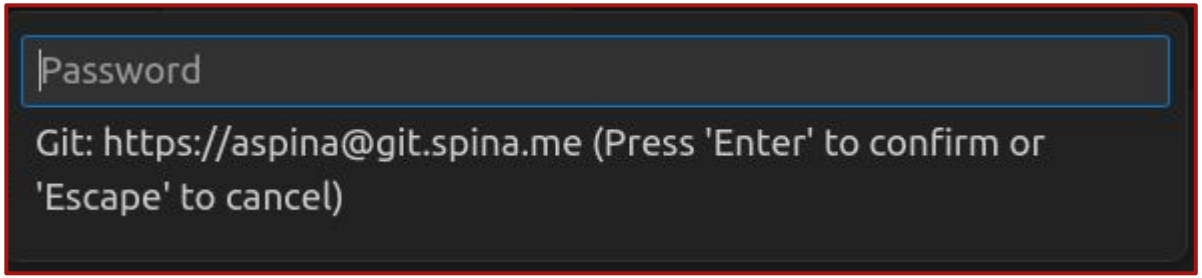
```
int max_age(student_t students[], size_t num_students) {
    int max = students[0].age;
    for (int i = 1; i < num_students; i++)
    {
        if (students[i].age > max)
        {
            max = students[i].age;
        }
    }
    return max;
}
```

Exercise Solution

```
bool are_points_equal(point_t points[], size_t num_points) {
    int first_x = points[0].x;
    int first_y = points[0].y;

    for (int i = 1; i < num_points; i++) {
        if (points[i].x != first_x || points[i].y != first_y) {
            return false;
        }
    }
    return true;
}
```

Save remotely your Changes

1. `make save`

2. 

3. 

Trivia Time!

ahaslides.com/QLQ9O

End of Class

See you all next week!