Introduction to IoT

School Year 2023-2024

Valsalice

Course Structure

1	Introduction and Basics	
2	Basic Data Types and Operators	
3	Control Structures	OCT 31
4	Functions and Scope	NOV 7
5	Arrays and Strings	NOV 14
6	Pointers and Memory Management	NOV 21
A	Preprocessor and Macros	DEC 19
B	Advanced Data Structures	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
10	Introduction to RPL and Network Routing	JAN 16
11	Challenges in Wireless Communication	JAN 23
12	Advanced Protocols: TSCH and 6TiSCH	JAN 30
C	Advanced Topics in Wireless Communication	FEB 6
D	Reliable Data Transfer Challenge	FEB 20, FEB 27, MAR 5

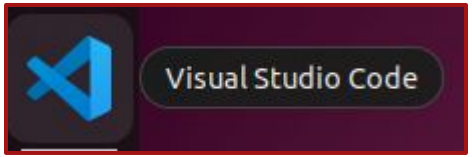
= Core Topics = Optional Topics

Open your Virtual Machines

1. Turn on your Laptops
2. Login using “User”
3. Open the **Virtual Box** program
4. Add the Virtual Machine (**Ctrl + A**)
5. Open the **VirtualBox** folder
6. Select the **nRF52840LAB** file
7. Click **Start**

Prepare the Coding Environment

1. Start the Virtual Machine **nRF52840LAB**
2. Open **Visual Studio Code**



3. From the Terminal:

```
make setup
```

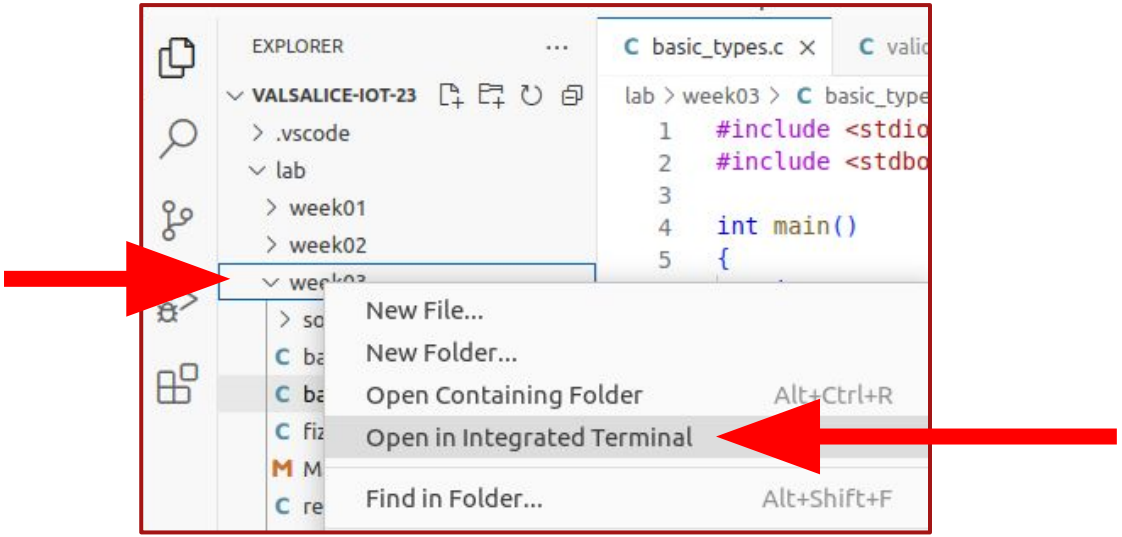
4.
```
valsalice-iot-23 git:(master) make setup
Enter your username:
```

5.
```
Password
```

6.
```
✅ Repository setup complete!
```

Prepare the Coding Environment

7. Open the **`week03`** folder in the terminal



8. You should see the following in the terminal:

```
week03 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.

```
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: Format Specifiers

Format specifiers specify how data should be formatted or interpreted.

```c
int num = 123;
char name[50] = "John";

printf("The integer is: %d\n", num);
printf("The name is: %s\n", name);

scanf("%d", &num);
scanf("%s", name);
```

Type	Format Specifier	Example
char	%c	'A'
string	%s	"House"
int	%d	100
float	%f	6.98

Recap: Changing Value

Variables can be updated and assigned new values.

```c
int x = 123;
printf("Value: %d\n", x);


x = 42;
printf("Updated value: %d\n", x);
```

IMPORTANT: If you change value you must use the same type.

Recap: Arithmetic

Values and variables can be used for arithmetic.

```
int x = 10;
int y = 4;

int w = x + y;
printf("W: %d\n", w);

int z = x / y;
printf("Z: %d\n", z);
```

Exercise

Complete the program (**`basic_types.c`**) that:

1. Asks you to input two numbers
2. Prints out the output of the following operations:
   a. Sum
   b. Difference
   c. Multiplication
   d. Division

To execute your code:

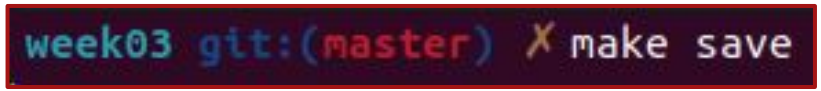
```
make basic_types.run
```

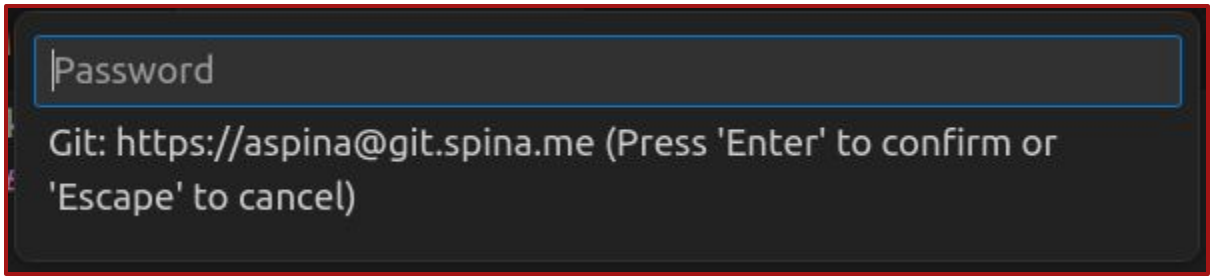
Exercise - Implementation

```c
#include <stdio.h>

int main()
{
    int num1;
    int num2;

    // Ask the user to input two numbers
    printf("Enter the first number: ");
    scanf("%d", &num1);

    printf("Enter the second number: ");
    scanf("%d", &num2);

    // Perform the operations
    int sum = num1 + num2;
    int difference = num1 - num2;
    int multiplication = num1 * num2;
    int division = num1 / num2;

    printf("Sum: %d\n", sum);
    printf("Difference: %d\n", difference);
    printf("Mult: %d\n", multiplication);
    printf("Division: %d\n", division);

    return 0;
}
```

Save remotely your Changes

1. `make save`

2. 

3. 

4. 

Exercise - Questions

1. What happens when **num2** is **zero**?
2. Why does division not have decimals?

Type Casting

You can cast types to other types before operations.

```
int division = num1 / num2;
printf("Division: %d\n", division);
```

```
float division = (float)num1 / num2;
printf("Division: %f\n", division);
```

Comparisons

Values and variables can be compared.

```
int x = 10;
int y = 4;

bool w = x > y;
printf("W: %d\n", w);

bool z = x <= y;
printf("Z: %d\n", z);
```

Equality and Negation

```
int x = 10;
int y = 4;

bool w = x == y;
printf("W: %d\n", w);

bool z = x != y;
printf("Z: %d\n", z);

bool n = !(x > y);
printf("N: %d\n", n);
```

Boolean Operators (so far)

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

Exercise

Complete the program (**`basic_comparisons.c`**) to:

1. Prints out the output of the following operations:
    a. Greater Than
    b. Less Than
    c. Equality
    d. Inequality

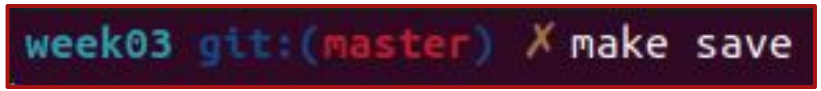
To execute your code: `make basic_comparisons.run`

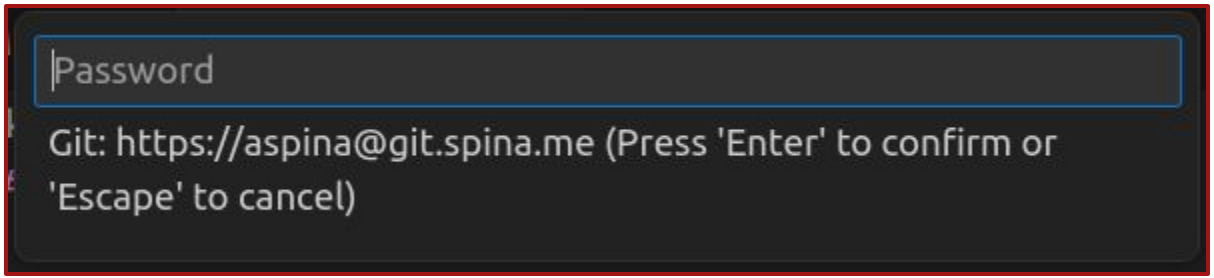
Exercise - Implementation

```
bool greater = num1 > num2;
bool smaller = num1 < num2;
bool equality = num1 == num2;
bool inequality = num1 != num2;

printf("Num1 is greater than num2? %d\n", greater);
printf("Num1 is smaller than num2?: %d\n", smaller);
printf("Num1 is equal to num2?: %d\n", equality);
printf("Num1 is different from num2?: %d\n", inequality);
```

Save remotely your Changes

1. `make save`

2. 

3. 

4. 

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)
```

Exercise

Write a program (**`valid_checker.c`**) that asks the user for a number and checks that the number is between 0 and 100.

- Print "**1**" if the number is valid.
- Print "**0**" if the number is not valid.

To run: `make valid_checker.run`

```
Enter a number: 200
Is the number valid? 0
```

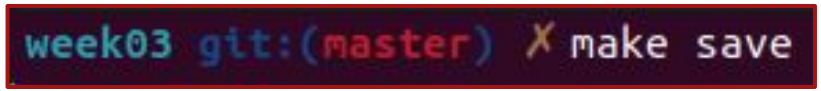
```
Enter a number: 42
Is the number valid? 1
```

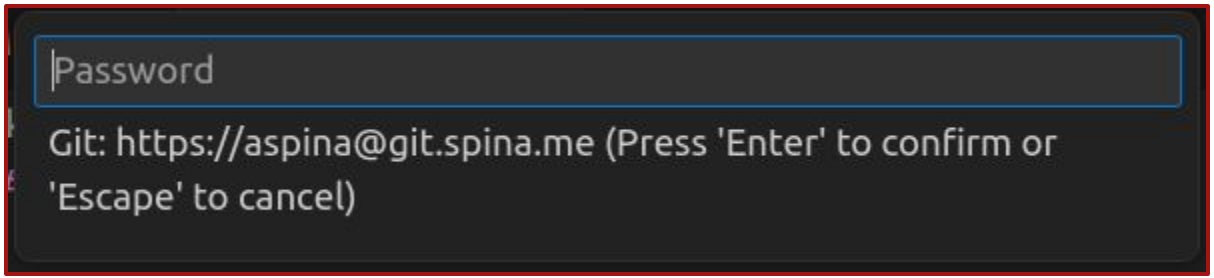
Exercise - Implementation

```c
// Ask the user to input one number:
printf("Enter a number: ");
scanf("%d", &num);

// Validity Check:
bool is_valid = (num >= 0) && (num <= 100);

printf("Is the number valid? %d\n", is_valid);
```

Save remotely your Changes

1. `make save`

2. 

3. 

4. 

If-Statements

Allow for branches in your code!

```c
int x = 5;

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

```c
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.

If-Statements

Anatomy of an if-statement:

1. Uses the **`if`** keyword
2. The condition is between brackets (**`(  )`**)
3. The body is between curly brackets (**`{  }`**)

```
int x = 5;

if (x < 10) {
    printf("Hello");
}
```

Exercise

Write a program (**`repeat_checker.c`**) that asks the user for a number between 0 and 100, if it is not valid it asks the user to input it again.

```
Enter a number between 0 and 100: 80
Well Done!
```

```
Enter a number between 0 and 100: 250
ERROR! Try again: 40
Well Done!
```

Exercise - Implementation 1

```
// Validity Check:
if ((num >= 0) && (num <= 100))
{
    printf("Well Done!\n");
}
else
{
    printf("ERROR! Try again: ");
    scanf("%d", &num);
    printf("Well Done!\n");
}
```

Exercise - Implementation 2

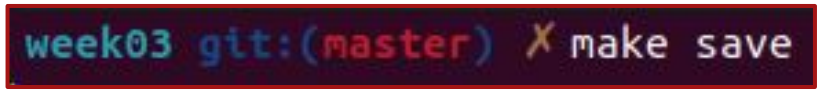
```
// Validity Check:
if ((num < 0) || (num > 100))
{
    printf("ERROR! Try again: ");
    scanf("%d", &num);
    printf("Well Done!\n");
}
else
{
    printf("Well Done!\n");
}
```

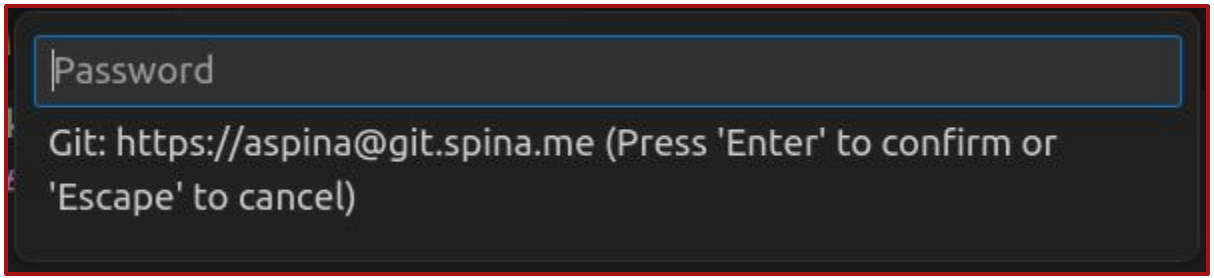
Exercise - Implementation 3

```
// Validity Check:
if ((num < 0) || (num > 100))
{
    printf("ERROR! Try again: ");
    scanf("%d", &num);
}

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

Save remotely your Changes

1. `make save`

2. 

3. 

4. 

End of Class

See you all next week!