Laboratory 3: Hello Serial

Objectives

1. Understand the concept of serial communication.
2. Understand the concept of data format.
3. Understand the ASCII code.
4. Practice more complex sketches (decision)

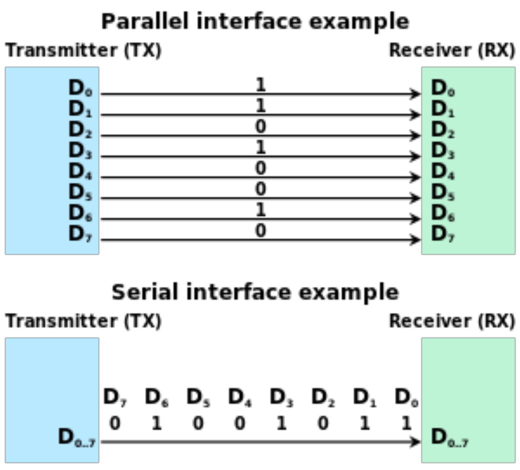
Reference:

https://en.wikipedia.org/wiki/Serial_communication

https://www.arduino.cc/reference/en/language/functions/communication/serial

https://www.arduino.cc/reference/en/#structure

Serial communication



Serial communication is a method for using a sequence of bits to transfer a byte of data. In the digital system, we use 8 bits to represent a byte of data. A byte is generally equal to a character. Thus, there are (theoretically) 256 possible characters in a byte. One standard that provides association between a value and a physical character is ASCII (see next section for more details).

With a serial communication, the data transfer between two devices can be done with just two wires (one for sending and the other one for receiving). This kind of communication is cheaper comparing to parallel communication where 8 wires are needed to transmit a byte of data. Thus, to send a byte of data from one device to another using serial communication, we must send (at least) 8 times (one for each bit). With no clock nor control signal involved in the communication, **the key to the success of serial communication is timing. Both devices must be configured for the same clock speed (specified as bit per second or bps), called baud rate, and the same protocol in order to observe the right data at the right time.** This is the basic of network communication that we used today. **If the sender and receiver use different baud rate, the receiver may interpret incoming data incorrectly**.

ASCII

ASCII stands for the ***American Standard Code for Information Interchange***. It is a character-encoding scheme based on English alphabets, numbers, and punctuation marks. It is nowadays used by several digital devices to represent codes for characters. For example, when type a character **'A'**, the digital computer system simply records an ASCII code (integer value: 65, 01000001B, 0x41) to the memory.

In programming, each character has a numeric value known as its ASCII code. For example, the character **'A'** has an ASCII value of **65**. The next character, **'B'**, has an ASCII value that is one greater than **'A'** (**66**). All English alphabet characters have ASCII codes in sequence: **'A'** is **65**, **'B'** is **66**, and so on. If you have the ASCII code of a character stored in a variable **x**, then **x + k** gives the ASCII code of the character **k** positions after **x**. Digits follow the same pattern: **'0'** + **1** gives **'1'**, **'1'** + **1** gives **'2'**, and so on.

Arduino Control Structures (Decisions) and operators

Arduino is based on the C/C++ programming language. Thus, it shares the same control structures used by C/C++ and those used by Java (and several programming languages). We provide a few examples here to ease your understanding.

- `increment/decrement operator: ++, --`

  Increments or decrements the value of an integer variable by 1.

```
x++; // increment x by 1 and return the old value of x
++x; // increment x by 1 and return the new value of x
```

- `boolean: true(non-zero), false(zero)`

  Use `==, !=, >, >=, <, <=` to compare two values.

```
a == 5
b != a-4
c >= b
```

- `logical operator: &&, ||, !`

  Use **&&** (and), **||** (or) to combine two boolean values. Use `!` to negate logical value.

```
(3 < x) && (x <=5)
(x < -10) || (x > 5)
!(x >= 4) same result as (x < 4)
```

- `'if'`

  Thinking of `'if'` as an action to perform when the test expression is `true`. For example,

```
if (someVariable > 10) {
  // do something here
}
```

- `'if/else'`

  The `'if/else'` allows greater control over `'if'` by grouping multiple tests. For example,

```
if (someVariable > 10) {
  // do something here
} else if (someVariable > 5) {
  // do something here
} else {
  // do something here
}
```

- `'switch-case'`

  When there are more than two options, you can use multiple **`if`** statements, or you can use the **`switch`** statement. **`switch-case`** allows you to choose between several discrete options. **`switch-case`** in Arduino is just like the **`switch-case`** in C language.

```
switch (var) {
    case label1:
        // statements for label1
        break;
    case label2:
        // statements for label2
        break;
    default: // optional
        //statements for default value (when other cases do not match)
        break;
}
```

  `var` is an integer variable (or a character) whose value to compare with various case labels. If its value is equal to **`label1`**, then the statements for **`label1`** will be executed. If its value is equal to **`label2`**, then the statements for **`label2`** will be executed, and so on. If the value is not equal to any values listed, then the statements corresponding to the **`default`** value will be executed. If you don't add the **`break`** statement after each case, then following case it will be executed, till the first **`break`** is encountered or end of the **`switch`** block.

Serial communication in Arduino

Arduino has already supported serial communication with the serial library. The following are some functions used for serial communication.

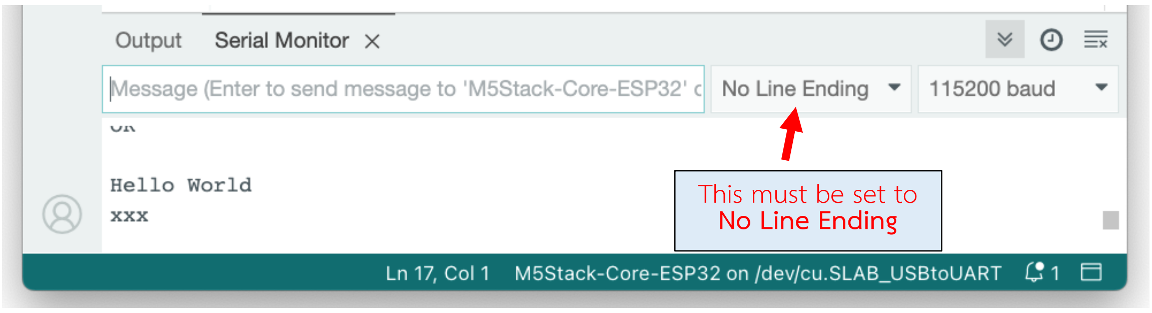
- **`Serial.begin(speed)`**: Sets the data rate in bits per second (baud) for serial data transmission.
- **`Serial.print(data, [format such as BIN, OCT, DEC, HEX])`**: Prints data to the serial port as human-readable ASCII text.
- **`Serial.println(data, [format such as BIN, OCT, DEC, HEX])`**: Prints data to the serial port as human-readable ASCII text with **new line** at the end.

- `Serial.available()`: Get the number of bytes (characters) available for reading from the serial port.
- `Serial.read()`: Reads a byte (of ASCII code) from the serial (equivalent to a character). To read data from Arduino, you use `if (Serial.available() > 0) { }` to see if there is a data sending from the host. Put the `Serial.read()` inside `if` block. Note that, if you enter more than one character, Serial.read() only gets one character at a time, the remaining characters will be returned in next **Serial.read()**.
- `Serial.write(byte)`: Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the **characters** representing the digits of a number use the `print()` function instead.
- For more information see reference.
- The following example is a `"Hello, World"` sketch for serial communication in Arduino, and screen captured result from Arduino.

```
#include <M5Stack.h>

void setup() {
   M5.begin();
   Serial.println("-----------");
   Serial.println("Hello World");
   M5.update();
}

void loop() {
  M5.update();
  if (Serial.available() > 0) {
    int readByte = Serial.read();
    Serial.print('x');
  }
}
```


Output
Serial Monitor
Message (Enter to send message to 'M5Stack-Core-ESP32' c
No Line Ending
115200 baud
This must be set to No Line Ending
Ln 17, Col 1
M5Stack-Core-ESP32 on /dev/cu.SLAB_USBtoUART

```
Hello World
xxx
```

The figure above shows the result of the previous sketch when type in any three characters in the input box of the serial monitor. The number of x's reflects the number of characters you type in.

Lab Exercises (Always save each task into different sketch)

Task 1: BASIC serial communication output: **`print()`**, **`println()`**

1.1 Create a sketch that sends inputs from the M5Stack development kit to serial communication output. In your sketch, read data from Button A and send the result to the serial output to display. If the serial output is not readable, check the baud rate.

```
#include <M5Stack.h>

void setup() {
  M5.begin();
  M5.Lcd.setTextSize(3);
  M5.Lcd.println("Lab 3: Task 1");
  Serial.begin(115200);
  Serial.println(" ... ");
}

void loop() {
  Serial.print("The button is ");
  if (M5.BtnA.isPressed()) {
      Serial.println("*****Pressed*****");
  } else {
      Serial.println("Released");
  }
}
```

1.2 Upload the sketch and see the result. You will see that the output **"The button is *****Pressed*****"** does not show in serial monitor when Button A is pressed. According to Lab 2, something is missing in the code. Modify to make it work as expected.

1.3 Change the **`isPressed()`** function to **`wasPressed()`**. Observe the difference and answer the question below. If the serial monitor output is difficult to observe, change baud rate to a lower number i.e., 9600.

Task 2: Count number of clicks

Modify the sketch in Task 1 to print the word “**Clicked**” when Button A is clicked. Also print the number of clicks so far. The pseudocode of the flow of sketch is given. You need to change into sketch syntax. (See Lab 2 **ledState** declaration for **INITIALIZE** part).

```
INITIALIZE count to 0

SETUP:
  setup M5Stack

LOOP:
  if button A is clicked
    increment count by 1 (count = count + 1 or ++count)
    display "Clicked - " using Serial.print()
    display count’s value using Serial.println()
  end_if
```

Task 3: Serial read() to control LED

Connect the LED to a digital PIN like in the **Laboratory 2.** The LED should be OFF when start. Create a new sketch that will do the following:

- read a value from the serial communication port.
- If a value is between '0' and '9' (`'0'<=value<='9'`), light up the LED and show message "ON" to both LCD screen and serial monitor (Note that the **value** is a character not number).
- Otherwise, turn the LED off and show message "OFF" to LCD screen and serial monitor.

For example, the following example should light up the LED.

```
Task 3: Enter a number to light up the LED
Please input a number.
>5
```

Task 4: Serial read() to control LED color using 'switch-case'

Connect all LEDs to three digital PINs (**use pin 21, 22, and 19**, use other pin might interfere with the internal connection). The LED should be OFF when start. Create a new sketch that do the following (Hint: use sketch from task 4 of Lab 2):

- Read a value from the serial communication port.
- Use **switch-case** to check the value, when the value is:
  - 'r' or 'R', light up the red LED color and turn of other LED colors.
  - 'g' or 'G', light up the green LED color and turn of other LED colors.
  - 'b' or 'B', light up the blue LED color and turn of other LED colors.
  - Show the LED color name to both LCD screen and serial monitor.
  - Otherwise, turn the LED off and show message "OFF" to LCD screen and serial monitor. Text on LCD screen should be easy to read.

Group

Lab 3: Serial Communication

Section: _______ Date: ____________

Members

Name: ______________________________ Student ID: ______________

Name: ______________________________ Student ID: ______________

Name: ______________________________ Student ID: ______________

Task		Graded by
1	BASIC serial communication output; **`print()`** and **`println()`**	
2	Count number of clicks	
3	Serial **`read()`** to control LED (**`'0'<=value<='9'`**) → `ON`, otherwise `OFF` Show LED state to LCD (Initially, LCD must be OFF)	
4	Serial **`read()`** to control LED colors (**`'rgbRGB'`**)	

5 Explain the difference between the two M5Stack functions related to buttons: `isPressed()`, `wasPressed()`.

6. Reflection about this lab. See assignment in **MyCourseVille**.