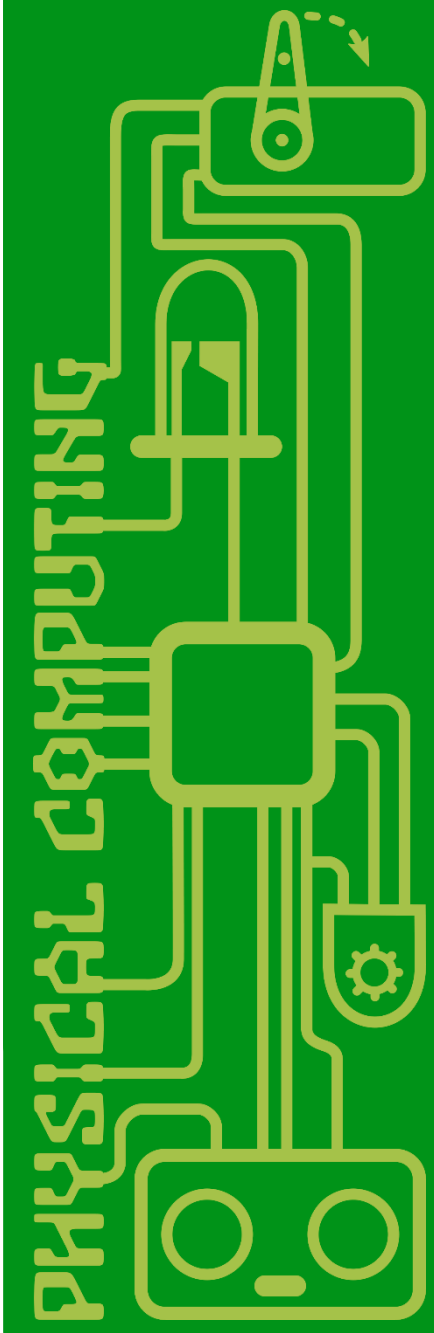
Serial input and output © 2023 by Ivan Novosel is licensed under CC BY-NC-SA 4.0. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/

ROBOTICS

Serial input and output



Author: Ivan Novosel

Institution: V. gymnasium, Zagreb

Co-funded by the Erasmus+ Programme of the European Union

Disclaimer: This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Introduction

We have used serial output to see the state of a sensor before when we introduced potentiometers. In this exercise we will dive deeper and see how we can both send and receive data on Arduino, and on the computer.

We will also learn how to connect a simple LCD screen to an Arduino so we can see the messages we send to Arduino from the computer.

1. Two-way communication using Serial monitor
2. Show the message on the LCD
3. Using Python as Serial monitor

UART communication

Serial communication means the bits of the message are sent one after another.

An Arduino board has more than one way to communicate with other components but one of them you are using every time you plug it into the USB port. **UART** (short for: Universal Asynchronous Receiver / Transmitter) **is a piece of hardware that is used for serial communication and all Arduinos**, and most other boards of that type, have UART to USB chip on them.

Pins 0 and 1 are also normal digital I/O pins, and can be used as such, but we can't use them for serial communication at the same time!

UART uses 2 wires to communicate, and they are marked as TX (transmit) and RX (receive). On UNO and Nano boards RX is connected on pin 0, and TX is on pin 1. These pins also have LEDs connected to them so you can see those blinking when there is communication going over them. When we upload any program to Arduino these lines are used to send our program, so if you connect anything to them it can interfere with serial communication.

Picture 1 – take note of RX and TX markings next to the pins, and the LEDs

Pins 0 (RX) and 1 (TX) are used to program the Arduino. That's why we usually don't use them!

You could also connect these pins to some other device with UART (RX of Arduino to TX of the device and vice versa) and have direct serial communication.

Today we will focus on the communication with the computer via Arduino IDE.

We have used `Serial` before to print sensor values to the serial port and see it through the Serial Monitor in Arduino IDE. `Serial` is the object which we use to access the functionality of UART. When working with UART it is good to know in more detail how it works, but it's outside the scope of this document to cover everything. The most important things are mentioned below, and you have external links that explain more.

How serial communication works on Arduino, **Stack Exchange**, URL: https://bit.ly/49uoSIh

Buffers are memory locations used for temporary storage of messages.

Baud rate is the speed of communication. 1 Baud = 1 symbol / s. In binary symbols are 0 and 1, so for Arduino Baud = bps

We have buffers for receiving and sending messages, they have 64 Bytes by default.

If receiving buffer is full no new bytes will be received!

The baud rate on Arduino and on the computer MUST be the same!

Receiving messages on Arduino

Let's make a simple example of communication where we send a short message to Arduino and send it back to the serial monitor. Just connect Arduino to the computer and upload:

TASK: Type in some messages of different length.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    String message = Serial.readString();
    message.trim();
    Serial.print("Message is: ");
    Serial.println(message);
  }
}
```

Serial.available() checks if there is anything in the buffer to read.

Serial object reference, check for other functions! **Arduino.cc**, URL: https://bit.ly/3QJ5hez

Here we use the `Serial.readString()` function to read the message as a string. When we type in a message in the serial monitor it will have a new line character `'\n'` at its end. It would print as a new line so we use to `trim()` remove it from the end of the string.

This is the base for receiving strings, but if we are getting numbers, we will need to convert them into int or float. Use the `int(string)` or `float(string)` to get a number that we can calculate things. If you don't need concurrent execution, you can also use `Serial.parseInt()` or `Serial.parseFloat()` to directly read a number.

Receiving messages concurrently

How to process incoming serial data without blocking, **gammon.com.au**, URL: https://bit.ly/40UIq4I

The above example is easy to use, but the problem is that while we are reading with `readString()` everything else is on pause – the execution of our program is waiting. That is ok if messages are short, but for anything longer we need to read in small chunks, so we don't block other tasks.

Serial communication is slow! It can take several milliseconds to receive a message.

In C++ strings are arrays of type `char` that end with a null character `'\0'`. The idea here is to make an array and then write into it, byte by byte.

For algorithms that solve this problem check the link in the sidebar. You can use these examples if you are building something that requires concurrent execution.

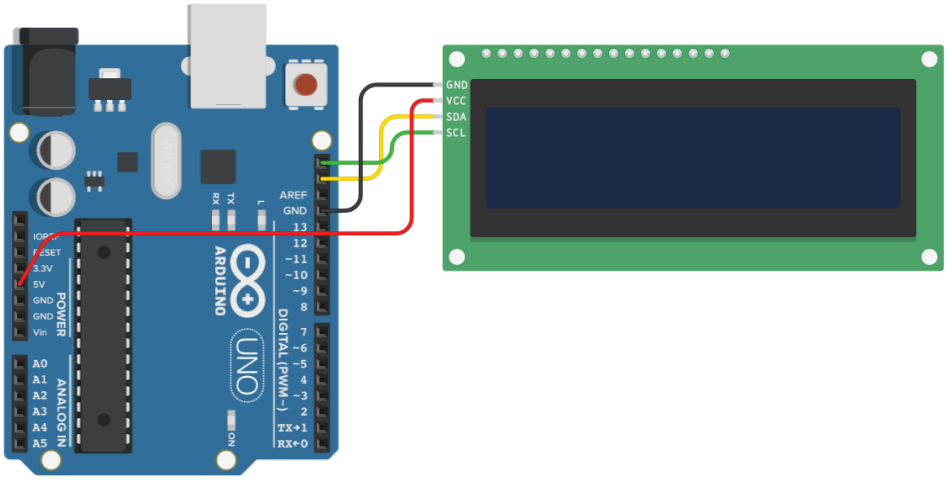
Using an LCD screen with Arduino

We can use the serial monitor inside the Arduino IDE as an interface for our projects but sometimes it is nice to see our message on a separate screen, or you are building a project that is not going to be connected to a USB port, but you need to see what is happening. For that we can use small 16x2 LCD screens.

I2C address scanner, **gammon.com.au**, URL: https://bit.ly/3NvsrV9

Most of the screens come with an I2C interface. I2C is synchronous protocol for serial communication that Arduino can use, and all devices that use it use two special pins SDA (serial data) and SCL (serial clock). Every device has its own address so you can connect up to 118 devices if they have their addresses set up correctly.

For I2C you need to know the right address for your device to use it.



Picture 2 – How to connect an LCD screen that has a PCF8574 I2C interface on the back. This picture is for an Arduino UNO, they have special SDA and SCL pins, on Arduino Nano pins A4 (SDA) and A5 (SCL) are used.

Before using I2C enabled devices it is good to check their address, for that you can use a program that scans all the possible addresses (link is in the sidebar on the page before) and checks if any device is available on a certain address. Once you know it's address you don't need to check for it again, it is set up in hardware (it's possible to change on some devices).

A minimal example for writing a simple message on the screen is:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(36, 16, 2);

void setup()
{
  lcd.init(); // starts the LCD
  lcd.backlight(); // lights up the backlight
}

void loop() {
  lcd.print("Hello!");
}
```

Before running this, you may need to install the **LiquidCrystal I2C** library, as it is not installed by default. Tasks:

1. Test if it works and remove possible bugs.
2. Read the documentation for LiquidCrystal library (find it on your own on Arduino.cc)
3. Make a program that will accept messages from Serial monitor and write it on the LCD screen nicely.

Simulation - LCD screen with serial input, **Tinkercad**, URL: https://bit.ly/3NxFANn

These are necessary libraries for I2C (Wire.h) and LCD to work.

Sets up an object of LiquidCrystal_I2C type – 36 is the address, 16 and 2 are the number of characters on the screen.

print() works the same as for Serial, but it will always start on the same place on the screen.

Serial monitor using pySerial

The built-in serial monitor in Arduino IDE is practical, but it has its limitations. If we need to store the data received from Arduino or present it in a different way, we need to build our own program that can do that.

On the link in the sidebar you can download a simple serial monitor made with the pySerial module. This is a program that you can modify to your needs. In the next exercise we will modify it to visualize data coming from an ultrasound sensor, but for now you just need to explore how it works:

pySerial custom serial monitor code, **Github**, URL: https://bit.ly/3Nws50i

1. Download the program and install the **pySerial module** if it isn't already installed on the computer.
2. Connect a potentiometer to the Arduino, and program it to send the values read from the potentiometer over the serial port.

We have done this before! Check the resistive sensors exercise if you don't remember how to do this :)

3. Run the serial monitor and record the data in a file.

- **Serial communication is buffered, don't let the buffer overfill.**
- **Serial communication is slow, chunk message reading.**
- **You must know the address for IC2 enabled devices.**
- **pySerial module for Python can read from any UART enabled device.**