1. Basics of programming (C++)

C++– general-purpose programming language.

First step towards programming seems difficult and it is - for the teacher and the student - so this step should be done very carefully. To avoid technical difficulties associated with programming languages, we will write our first programs in pseudocode.

A way of writing an algorithm that retains the structure characteristic of code written in a programming language is called **pseudocode**. It gives up strict syntactic rules in favor of simplicity and readability.

In order to simplify learning programming as much as possible, we will limit ourselves to a few pseudocode commands:

- *As long as **condition** is true, repeat **code*** – a loop that will execute ***code*** over and over again as long as a certain ***condition*** is met,
- *If **condition** then do **code1*** – a conditional statement that will execude ***code1*** (once) only when a specific condition ***condition*** is met. This instruction may be followed by an additional instruction *otherwise **code2***, which will be executed if the previous condition is not met or *otherwise, if **another_condition** then do **code3***.
- *Use pin as input/output* – if we want to use one of the pins, we must set it as input or output,
- *Read pin **logical value*** – allows us to read whether a given input is logical 0 or 1 (ie.: is button connected to *pin* pressed or not),
- *Set output pin as value **1 or 0*** – allows you to set a given output to 0 or 1,
- *Remember that **name** means **value*** – variable that remember some value (ie. number),
- *Add, subtract, multiply belowl,*
- *Wait **time***

Additionally, comments are also useful when writing more complex programs. Every line starting with //, # and any text inside /* … */ will be ignored.

The whole difficulty of programming now comes down to translating what we would like to do into pseudocode.

Exercise 1.1

Review examples 1 and 2, and then write pseudocode to indicate using an LED whether the selected combination of three buttons out of five has been pressed. The LED is connected to pin 8, while the buttons are connected to pins 2, 3, 4, 5, and 6. When an incorrect combination is pressed, the LED should not light up.

Conditions can be combined, for example: If a is 1 and b is 1, then ***code***.

Example 1.1

Write a program in pseudocode: Light up the LED for 0.5 seconds, then turn it off for 1 second. The LED is connected to pin 7. Solution:

```
// To make the task easier for ourselves and not have to remember
// where everything is connected (right now we only have one LED
// connected to some pin, but what if there were 5 LEDs, and
// somewhere else there were other things?), let's start by saying
// that everywhere we use the word "diode", we mean "7":

      Remember that diode means 7.

// Due to technical reasons, pins cannot be input and output at the
// same time. Therefore, before using them, it must be established
// whether we want to use them as input or output:

      Use pin diode as OUTPUT.

// As for technical matters, we already have everything described,
// so all that remains is to perform the blinking itself.

// Since we want the LED to blink and never stop, it will be easiest
// to use a loop for this purpose. In the loop, the LED will light up,
// shine for a while, then turn off, and stay off for a while:

      As long as condition always is true, repeat code {
            Set diode pin output value as 1.
            Wait 0.5s.
            Set diode pin output value as 0.
            Wait 1s.
      }
```

Example 1.2

Write a program in pseudocode: after pressing the button, the diode lights up for 3 seconds. The diode is connected to pin 7, the button to pin 3. If the button is pressed, the pin outputs logic 1.

To solve this task, we will use part of example 1:

```
//we will remember in the programe which pin the diode is connected to
//and to which button:

      Remember that diode this 7.
      Remember that button this 3.

//Let's set the pins as outputs and inputs appropriately:

      Use pin LED as OUTPUT.
      Use pin button as INPUT.

//Again, we want the program to execute indefinitely:
// wait for the button to be pressed, turn on the diode, wait, turn off
// and wait for the button to be pressed again:

      Repeat endlessly {
            //Let's read and remember whether the signal from button is 1 or 0:
            Remember that pressed means value read from pin button.

            If pressing equals 1, then execute {
                  Set pin diode on 1.
                  Wait 3s.
                  Set pin diode on 0.
            }
      }
```

Step two comes down to translating pseudocode into a programming language, which will later be translated (by a computer) into machine code.

High-level language (autocode) - a type of programming language whose syntax and keywords are designed to maximize the understanding of the program code for humans, thereby increasing the level of abstraction and distancing itself from hardware nuances.

Pseudocode

```
Repeat without end:
      Turn on diode
      Wait 1s.(1000ms)
      Turn off diode
      Wait 1s.(1000ms)
```

C++ (Arduino)

```
While(1){  //logical one, always true
  digitalWrite(diode,HIGH);
  delay(1000);
  digitalWrite(diode,LOW);
  delay(1000);
}
```

Arduino consists of an 8-bit Atmel AVR microcontroller with additional components to to make uploading code easier and for other purposes. **Microcontroller** is a single-chip microcomputer, an integrated microprocessor system implemented in the form of a single chip containing a processor, RAM (random acces memory), input-output systems and program memory.

The program from example 1 written for Arduino:

```
int diode = 7;                //variable definitionhey(globalhey)

void setup() {                //function executed once,after turning on the
                              // power
  pinMode(diode, OUTPUT);     //sets pin 7 as output
}

void loop() {                 //a part of code performed over and over again
  digitalWrite(diode, HIGH);  //set pin 7 to value HIGH (logic 1)
  delay(500);                 //Wait 500ms = 0.5s
  digitalWrite(diode,LOW);    //set pin 7 to LOW (logical 0)
  delay(1000);                //wait 1s
}
```

As you can see, most of the instructions and the entire program are very similar to the solution from the first example. However, a few comments are in order:

- Everything that is inside the braces {} of the function void *setup()*, will be executed only once after starting Arduino. This is a great place for e.g. setting pins as input/output or starting RS communication (more on this later).
- Everything inside the curly braces {} of the function *void loop()* will be executed over and over again when Arduino starts – it's called *main loop*,
- In order not to complicate learning, we will place variable declarations at the beginning of the program – before *void setup()*, and all variables will be of type *int*, unless otherwise specified in this document. The int variable can take values from -32768 to 32767 (only integer numbers). The declaration takes the form:

int	diode	= 7	;
variable type	variable name	value assignment (optional)	command end

Translation from pseudocode (Arduino programming)

In order for our programs to work, we need to translate them into C++. So let's start with a blank program template where we will place our translations:

```
// place for variable declarations
// if we will be using variables somewhere (e.g., to store the read
// value from an input pin), we should inform the program about it before
// void setup(){}. For example: int press1;
// or int button3 = 4;

void setup() {
                        //function executed once,after turning on the power
                        // Here we will put things that can be done
                        // only once because they won't change later.
                        // E.g. pinMode(…).
}

void loop() {
                        //a function performed over and over again (without end)
                        // Here you will find things that the program has
                        // perform continuously (e.g. checking what signal
                        // is on input and setting something on output)
}
```

Very often during programming, the word **condition** will appear. A **condition** is a certain logical or arithmetic operation that returns a value of 0 (condition <u>not</u> met) or 1 (condition met):

a > b or b < a	*a* bigger than *b*
a >= b	*a* greater or equal *b*
a == b	*a* equal *b (important, double equals!)*
a != b	*a* not-equal *b*

Combining terms: AND: if (***condition1*** && ***condition2***){**code** to execute if true}

OR: if (***condition1*** || ***condition2***){**code**}

Translations for previous pseudocode instructions

Please remember that every statement (except loops and conditional functions) must end with a semicolon (;).

As long as condition** is true, r**epeat code – conditional statement that will be executed only when a specific condition is met, e.g.:

```
int variable = 10;

if(variable == 11){                 //mind double "="!
      digitalWrite(diode1, HIGH);
      digitalWrite(diode2, LOW);
}else if(variable == 10){           //optional. Can be repeated more
                                    // than once
      digitalWrite(diode1, LOW);
      digitalWrite(diode2, HIGH);
}else{                              //optional
      digitalWrite(diode1, LOW);
      digitalWrite(diode2, LOW);
}
```

It can be interpret as: set the variable to 10, after that check whether the variable has the value 11. If it is, light up diode 1, turn off diode 2. If not, check if it is 10 – if so turn off diode 1 and turn on diode 2. If none previus condition was met turn off both diodes.

Use pin as… – if we want to use one of the pins, we must set it as input or output.

```
pinMode(pin,OUTPUT);    //set pin as an output (e.g. diode)
pinMode(pin,INPUT);     //set pin as input (e.g. button)
```

Read pin logical value – allows us to read whether a given input is logical 0 or 1,

```
digitalRead(10);  //checks whether pin 10 is logical 0 or 1
```

Rading input only wouldn't make much sense, but you can save the result of the function to a variable or use it as a CONDITION of the function (*if*, *while* e.t.c.):

```
//declare the variable pressed and the variable with information
// where the button is attached
int pressed = 0;
int button = 7;

//remember whether the button is pressed or not
pressed = digitalRead(button);

//if the button was pressed, turn on the diode
if(pressed == 1){
      digitalWrite(diode, HIGH);
}

//if the button was not pressed, turn off the LED
if(pressed == 0){
      digitalWrite(diode, LOW);
}
```

Set output pin as… – allows you to set a given output to 0 or 1:

```
digitalWrite(13, HIGH);       //sets pin 13 to logical 1
digitalWrite(13, LOW);        //sets pin 13 to logical0
```

Remember that... it's... –variables, storing numbers (characters):

```
variable = 12;    //to remember the new value of a variable, we use a sign
                  // (single!) is equal to. Variable from this point on
                  // will store the value 12.
```

Repeat { … } – the loop that will be performed as long as condition is fillfilled. One such loop is the while loop: *while(condition){function_body}, e.g.*: :

```
variable = 0;
while(variable < 10){
      digitalWrite(diode, HIGH);
      delay(1000);
      digitalWrite(diode, LOW);
      delay(1000);
      variable = variable + 1;
}
```

Which can be read as: set the variable to 0, then check if the variable is less than 10 - if so, turn on the LED and wait a second, turn off the LED and wait a second. Increase the variable by one. End of the loop - return to the beginning and again: check whether the variable is less than 10 etc... If the condition is not met, the program*skips*for the brace }.

In an empty Arduino program, next to the `void setup()` function there is such a *Repeat* function – a `void loop()` function, the content of which is executed endlessly.

Add, ***subtract***, ***etc.***:

```
variable = 3+2;         //the variable will take the value 5
variable = 3 – 2;       //the variable will take the value 1
variable = 3 * 2;       //the variable will take the value 6
variable = 3 / 2;       //the variable will be 1 (as rounded down 1.5)
                        //int type can store only integer values,
                        // so it rounds rounds down

variable++;             //increase the variable by 1
variable--;             //decrease the variable by 1
```

Wait…:

```
delay(1000);            //the program will stop for 1s. 1s = 1000ms
```

Exercise 1.2

Translate to C++ example 1.2 and pseudocode from exercise 1.1.

2. Arduino basics

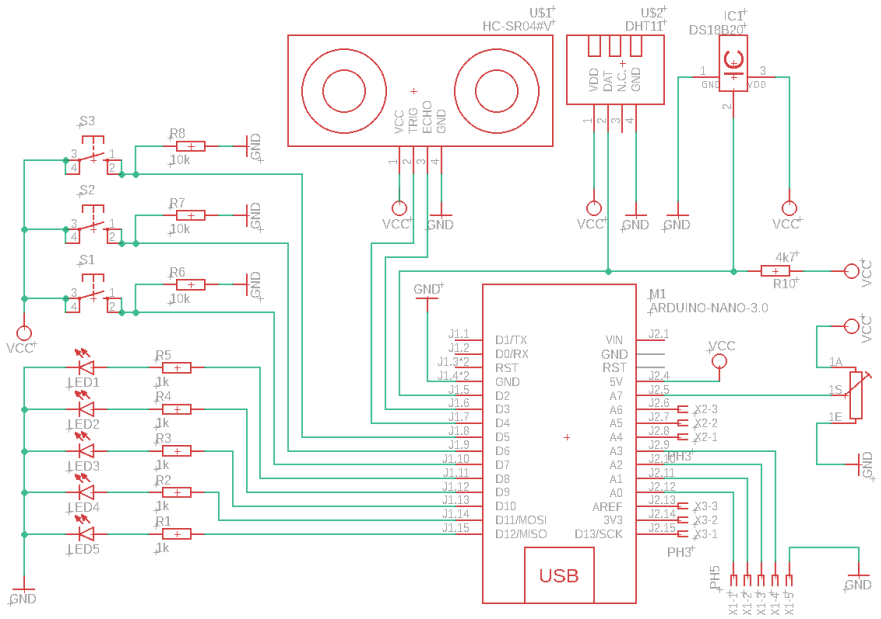
To learn programming using Arduino, you need some basic elements:

- Computer with Arduino IDE software (https://www.arduino.cc/en/Main/Software),
- Arduino Nano or UNO (or clone) with USB cable,
- Arrangement of buttons, diodes and other components.

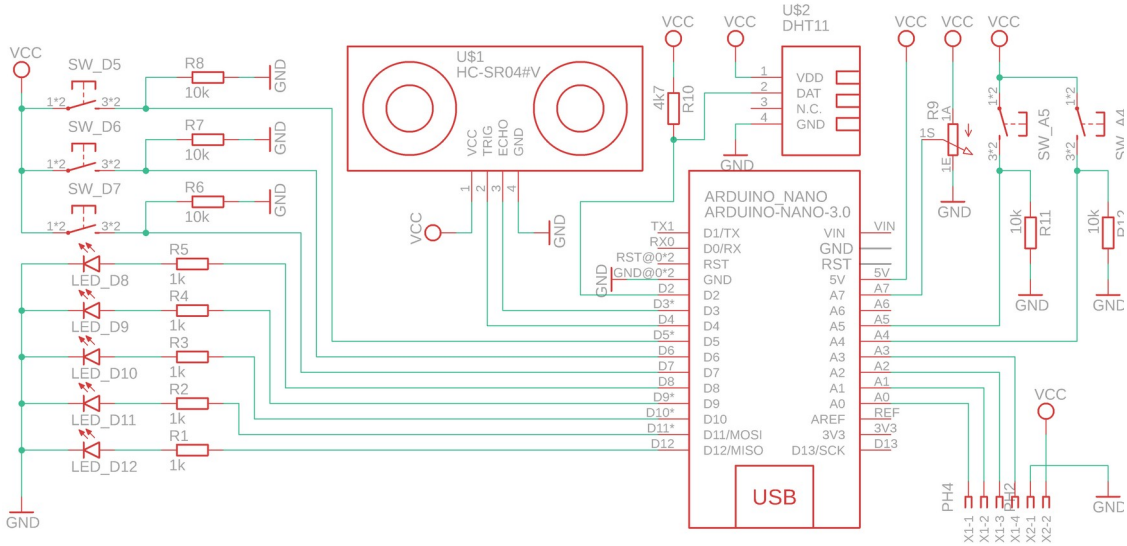
OR simulator (tinkercad.com) when we don't have access to physical parts.

During the classes, ready-made systems will be used:

Version 2:

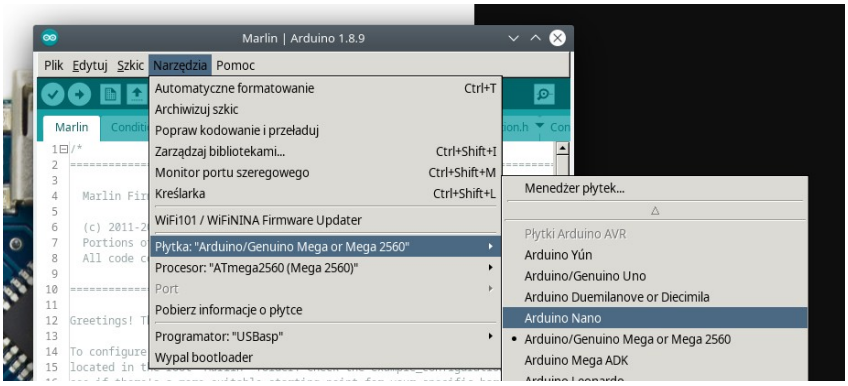


Version 3:

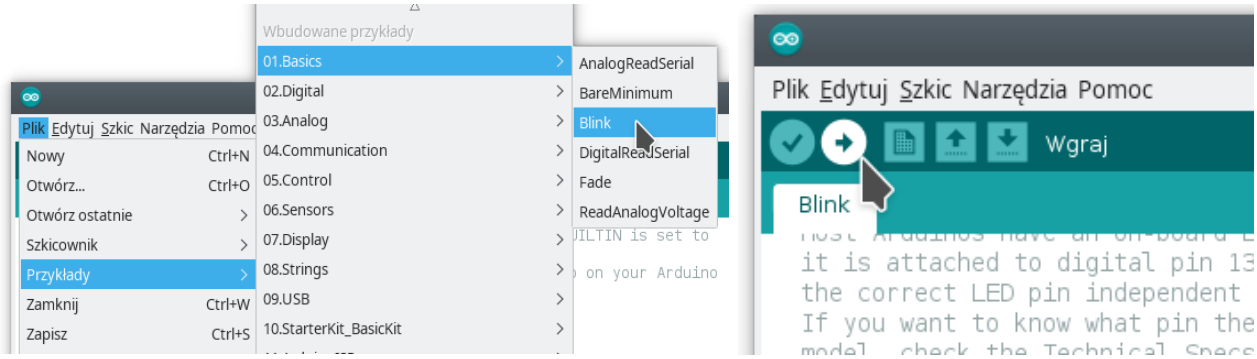


Arduino IDE

To check if everything works launch the Arduino IDE, then in the menu *Tools* choose the appropriate tiles: Board – *Arduino Nano*, Processor – *ATmega328P (Old Bootloader)*, Port – one with highest number.



Then upload the sample program *Blink*:



Arduino online

If you don't have access to physical equipment, you can use online simulator: tinkercad.com/circuits

However, please bear in mind that:

- currently only Arduino UNO is available (which is compatible with NANO),
- Arduino UNO does not have the A7 output, the potentiometer must be connected to another output marked with the letter A. Remember this when writing the programs.
- for starter, just connect the diodes, buttons and potentiometer with resistors.

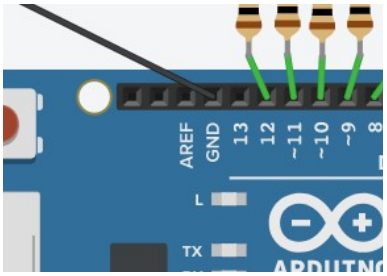
For actual programming, select from the right *Code* and switch from block view to text view.

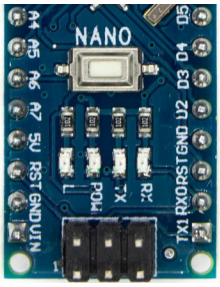
Examples are available on the website
https://github.com/PMKrol/WTDAutomatyka/tree/main/snap/arduino/current/Arduino/libraries/WTD/examples.

Setting/reading ports

Arduions port are places where other devices can be connected: buttons, LED's, LCD's and many other.

```
pinMode(port, STATE);          - sets port as INPUT or OUTPUT
digitalRead(port);             - reads the value (0 or 1) from port
                                      (i.e. button)
digitalWrite(port, STATE);     - sets port output, to logical 0 or 1 (5V)
                                      (i.e led)
```





`LED_BUILTIN`

in the case of Arduino NANO and UNO, it is a variable with the value 13, to which the L diode mounted on both Arduinos is connected

port is the pin number with the letter D omitted. In the case of Ax ports, the letter A should be entered in the program.

Exercise 2.1

After starting the Arduino IDE and connecting it Arduino, from the File menu, select Examples → 01.Basics → Blink. Analyze the program and then upload the it to Arduino.

Does the program behave as expected?

Exercise 2.2

a) What should be changed in the program to replace the diode L so diode connected to D12 blinks? Verify Your idea.

b) Modify the program in such a way that the diodes blink:

- all at once,
- one by one,
- one by one back and forth.

c) Imagine two diodes, one blinking once per second, second blinking twice per second. Draw a graph of light versus time (for both diodes – with different colored pen). Write program.

Combining conditions

If we want to place more than one condition inside the *if* instruction (or other conditional):

```
(digitalRead(3) == 1 || digitalRead(4) == 1)
```

– gives true if input 3 OR 4 is logical 1 (TRUE)

```
(digitalRead(3) == 0 && digitalRead(4) == 0)
```

– gives true if input 3 AND 4 are logical 0 (FALSE)

Extended conditional statement `if/else/else if`

Conditional statement *if* can be expanded by *else*, whose body is executed when the previous conditions are not met. *if* also can be extended by *else if(other condition)*, which is executed when the previous conditions are not met AND *other condition is met*. *else if* can be used more than once, *else* only once.

```
if(condition1){
        //performed when satisfied condition1
}else if(condition2){
        //executed when condition1 is not met and condition2 is met
}else{
        //performed when previous conditions are not met
}
```

After *condition* do not enter a semicolon (;)!

Exercise 2.3

Write a program that only lights up one diode:

a) 1, 2 or 3, when corresponding button is pressed,

b) 4th if first and second button is pressed at once. 5th diode, when all three buttons are pressed.

Variables

Variables are used to conveniently remember program data under a name. Since there are different types of data, there are also different types of variables.The basic type (for us) will be:

- `int` – short for „integer”. Stores numbers from -32768 to 32767

We will treat all variables as global variables – available from anywhere in the program. Global variables are declared at the beginning of the program (before `void setup()`) as follows:

```
variable_type variable_name;
```

or, when we want to declare its value immediately:

```
variable_typevarname = value;
```

For example, the after declaration: `int diode5 = 12;`, we are abe to use *word* diode5 anywhere in the program and it will be understand as 12 or any new value if we change it.

Calculations on variables

Calculations on variables behave like usual: `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (rest from division).

Equal sign (`=`) means assigning *what is on the right* to *what is on the left* – unlike comparison (`==`). E.g.:

```
variable8 = variable5 + 3;
```

Will store sum of value5 and three under name *variable8*.Then the command `digitalWrite(variable8,HIGH);` will set port *that sum* port to high (and for example turn on the diode connected to it).

Additional exercise

Write Arduino turns on, light up the third diode. When button *one* is pressed turn off current diode and turn on *next diode*. When button *three* is pressed, turn off current diode and light up *previous* diode.

Define new *variable* type `int` with any name. Increase value of *variable* by 1, when button 1 is pressed and decrease the value of *variable*, if button 3 is pressed. Depending on what value it takes turn on the appropriate diode (and turn off the others).

Additional exercise 2

Read abount function called `millis()` which returns value of type `unsigned long` containing the time since start of Arduino in milliseconds. Do exercise 5.2c. without `delay()`.

3. Arduino - Communication

Arduino has the ability to communicate with various other devices. The basic communication is communication with the computer via the serial port. This allows to read and send data, control components, etc. Most often, communication via the serial port is used to monitor what is happening in Arduino.

Communication via serial port

Serial port – computer port through which data is transferred in the form of one string of bits. This port is usually equipped with a special circuit (the so-called universal asynchronous transceiver) that translates strings of bits into bytes and vice versa. In other words, it is a device that allows you to send and receive a sequence of bytes (letters, numbers and other characters). The Arduino serial port is connected to a USB converter, which allows direct communication via USB with a personal computer.

```
Series.begin(9600);           //starts serial communication,
                              // this command needs to be in setup().

Series.println("string");     //sends string of characters
                              // with newline at the end
                              // through the serial port.

Series.print("string");       //sends string without newline at the end.
```

For technical reasons, in order for Arduino to send to the computer, it must be "announced" by adding the line to the setup function `Serial.begin(9600);`. Then, anywhere in the program, you can issue the command *send message with content*, using e.g. `Serial.print("contents");`.

Exercise 3.1

a) Analyze the program code below. What message will be sent to the computer if we press the button connected to port 5? What if we release it? What will happen if we press the button for a long time?

```
intbuttonA=5;

void setup() {
  Serial.begin(9600);
  pinMode(buttonA, INPUT);
}

void loop() {
  if(digitalRead(buttonA)){
    Serial.println("Button A is pressed");
  }else{
    Serial.println("Button A is not pressed");
  }
  delay(100);
}
```

Open **File →Examples → WTD → Komunikacja_1** or copy the program as a new sketch, then upload it to Arduino. Start serial communication: **Tools →** Serial port monitor.

Does it behave as expected?

b) modify the above or write a new program that prints the appropriate text when the appropriate two or three buttons are pressed. Add information about releasing the button.

Analog reading

Arduino, in addition to the digital functions already known, has *capacity* reading analog values. These inputs are usually marked as A0-A7. To read *analog* value at the input, use the appropriate function:

```
analogRead(port);    //reads analog value at port. Returned
                     // value (0-1023) corresponds to 0-5V voltage.
```

Exercise 3.2

- write a program that will sent via serial text *"temperature*" every 0.5s.
- Scheck in the diagram to which input the middle leg of the potentiometer is connected. *Tell* Arduino whether this will be an input or output port (see **Setting/reading ports**).
- Replace text "*temperature"* with `analogRead(PORT)`,where PORT is potentiometer's middle pin (e.g. A0, A1, A7).
- Assume that the potentiometer is an analog (linear) temperature device, while the function `analogRead()` returns: 0 for temperature 0°C,1023 in temperature 40°C. How to calculate temperature? Write a program that sends via monitor *current temperature*.
- Modify the program so it turns on diode after exceeding the temperature of 25°C.

Arduino can also read information (single characters) sent from the computer to it. There is a command for this`Serial.read()`. Just like`Serial.write()`yes and read understands a limited set of characters (see ASCII).

```
Series.available();     //checks how many characters are waiting
                        // for interpretation

Series.read();          //reads one character from the string sent to
                        // Arduino from computer
```

Exercise 3.3

```
int sign = 0;                       //Place where the received character will be saved.
int diode1 = 8;

void setup() {
   Serial.begin(9600);              //Start the serial port
                                    // at a speed of 9600 bits per second.

   pinMode(diode1, OUTPUT);
}

void loop() {
   if (Serial.available() > 0) {    //If a new character is received...

      character = Serial.read();    //Remember what character was read
                                    // but only one - the rest are 'waiting'.

      if(char == '1'){              //Attention!
         digitalWrite(diode1, HIGH); // Single character MUST be between
                                    // single apostrophe.
         Serial.println("I turn on LED 1.");
      }
   }
}
```

Open **Examples → WTD → Communication_2** or copy the above program as a new sketch, then upload it to Arduino. Start Serial port monitor. Send *proper* sign via monitor. Send *other* sign.

Does it behave as expected?

a) Modify the program so that the remaining diodes can be turned on.

b) Modify the program so that all the diodes can be turned off – by command and button.

c) **Additional.** Create variables that *stores* current state of each 5 diodes. Modify the program so that entering e.g. *1* cause *change* of state of the first diode (variable and output. For example, sending two 1s will turn on and then turn off diode 1).

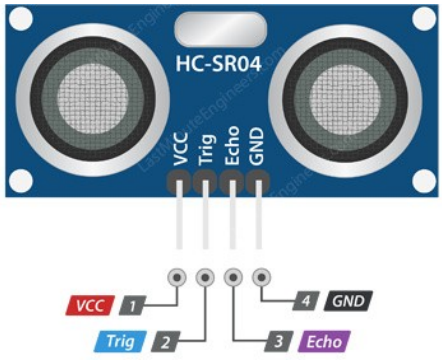
4. Arduino

Communication with other devices

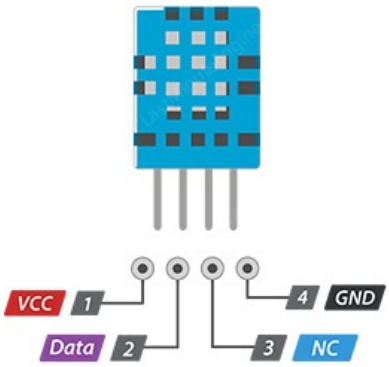
The serial port is not the only standard for communication between devices. Depending on the needs, various solutions can be used: one-way or two-way, serial or parallel, digital or analog, over short or long distances, etc.

The most popular standards and devices have their own implementations in the Arduino environment, so most of the time ready to use solutions are available. One thing to know what device You want to use and find appropriate libraries.

Popular devices, for which the libraries are available are:



Ultrasonic rangefinder
HC-SR04



Thermometer + hygrometer
DHT22

Micro Servo
SG-90

Ultrasonic rangefinder is a device that measures the time between transmitting and receiving an ultrasonic wave. This time (measured by Arduino, between applying a signal to the TRIG input and the appearance of *answer* on the ECHO output of the device), after taking into account the speed of sound (approx. 340 m/s) and conversion, it gives the distance. **HC-SR04** is an example of device that does not use any of the comunication standards.

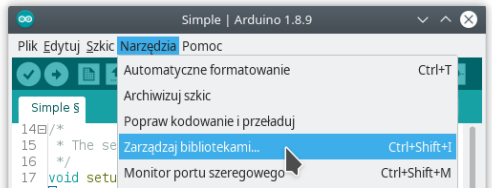
DHT22 is a device communicating via the OneWire interface.

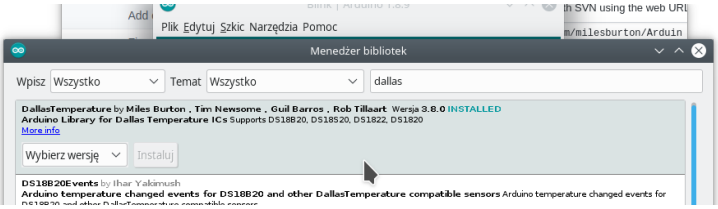
Micro servo is a servo mechanism often used in modeling and various DIY devices. It only requires information in what position it should be set. After receiving such information, the servo automatically sets itself to the given position and controls it continuously.

Libraries

Additional libraries are needed to complete exercises and must be installed (in case there are not already installed) using *Tools → Library manager*, where you should search for the appropriate ones and install it (www.arduino.cc/en/Guide/Libraries):

***DHTNEW** by **Rob Tillart**, **DallasTemperature**,*
***HCSR04** (github.com/Martinsos/arduino-lib-hc-sr04),*
***OneWire**.*





Please ask Your teacher to connect devices..

4.1

a) Ultrasonic rangefinder

- Select **File → Examples →WTD→ HCSR04**. Analyze the program.
- Based on the example, write a program, which will send information about the current distance and turn on the diode when the distance is less than 20 cm.
- Modify the program in such a way that the number of turned on diodes corresponds to the measured distance (e.g.: 10 cm - first diode, 20 cm - second diode, etc.).

b) Temperature and humidity sensor

- Select: **File → Examples →WTD → DHT**. Analyze the program.
- Based on the example, write a program, which sends information about current humidity and temperature. Turn on one diode when humidity is over 50% (or other value). **Attention!** If the humidity is 999%, ask teacher for help.
- Expand the program so that when some temperature is exceeded (e.g. 27 degrees C), another diode lights up.
- Expand the program so that when the temperature AND humidity are exceeded, all diodes light up.
- **Extra** (difficult): modify last point so all diodes in this case blinks.

c) Servo mechanism

- Select **Examples → WTD → Servo**. Analyze the program and verify its operation.
- Modify the program so that the initial value of variable `potVal` is 90 degrees. Use two buttons in such a way that one decreases the value of `potVal` by one, and the second increase it. The angle of the servo should be the same as the value of `potVal`.
- Expand the program with a button that sets the servo drive to the initial position and 0 degree.

Sources:

https://www.arduino.cc/reference/en/
Autodesk Eagle,Version 9.3.2, © 2019 Autodesk
akademia.nettigo.pl/zmienne_podstawy_jezyka_arduino/
tinkercad.com/
pl.wikipedia.org/wiki/Port_szeregowy
Arduino 1.8.9
en.wikipedia.org/wiki/Servomechanism
lastminuteengineers.com

Patryk Król