

# Programowanie C (CP1S01005)

---

Politechnika Białostocka - Wydział Elektryczny  
Cyfryzacja przemysłu, sem. I, studia stacjonarne I stopnia  
Rok akademicki 2024/2025

**Wykład nr 4 (22.11.2024)**

dr inż. Jarosław Forenc

## Plan wykładu nr 4

- Funkcje w języku C
  - zmienne: automatyczne (auto), rejestrowe (register), zewnętrzne (extern), statyczne (static)
  - struktura procesu w pamięci komputera, ramka stosu
  
- Arduino
  - podstawowe informacje
  - historia
  - moduły Arduino, Arduino Uno R3, ATmega328P
  - moduł wykorzystywany na zajęciach
  - środowisko Arduino IDE
  - środowisko Visual Studio Code, rozszerzenie PlatformIO IDE
  - struktura programu, typy danych

# Pamięć a zmienne w programie

- Ze względu na czas życia wyróżnia się w programie:
  - **obiekty statyczne** - istnieją od chwili rozpoczęcia działania programu aż do jego zakończenia
  - **obiekty dynamiczne** - tworzone i usuwane z pamięci w trakcie wykonania programu
    - automatycznie (bez udziału programisty)
    - kontrolowane przez programistę
- O typie obiektu (**statyczny** lub **dynamiczny**) decyduje klasa pamięci obiektu (ang. storage class)
  - **auto** - zmienne automatyczne
  - **register** - zmienne umieszczane w rejestrach procesora
  - **extern** - zmienne zewnętrzne
  - **static** - zmienne statyczne

## Zmienne automatyczne - auto

- Miejsce deklaracji: najczęściej początek bloku funkcyjnego ograniczonego nawiasami klamrowymi { i }
- Pamięć przydzielana automatycznie przy wejściu do bloku i zwalniana po wyjściu z niego
- Zakres widzialności: ograniczony do bloku, w którym zmienne zostały zadeklarowane (**zmienne lokalne**)
- Dostęp do zmiennych z innych bloków możliwy przez wskaźnik
- Jeśli zmienne są inicjalizowane, to odbywa się ona przy każdym wejściu do bloku, w którym zostały zadeklarowane
- Nie ma potrzeby jawnego używania **auto**, gdyż domyślnie zmienne wewnątrz bloków funkcyjnych są lokalne

```
auto int x;
```

## Zmienne rejestrowe - register

- Zazwyczaj o miejscu umieszczenia zmiennej automatycznej decyduje kompilator:
  - pamięć operacyjna - wolniejszy dostęp
  - rejestry procesora - szybszy dostęp
- Programista może zasugerować kompilatorowi umieszczenie określonej zmiennej automatycznej w rejestrach procesora
- Najczęściej dotyczy to zmiennych:
  - często używanych
  - takich, dla których czas dostępu jest bardzo ważny

```
register int x;
```

## Zmienne zewnętrzne - extern

- Miejsce deklaracji: poza blokami funkcyjnymi, najczęściej na początku pliku z kodem źródłowym
- Pamięć na zmienne jest przydzielana, gdy program rozpoczyna pracę i zwalniana, gdy program kończy się
- Zakres widzialności: globalny - od miejsca deklaracji do końca pliku z kodem źródłowym (**zmienne globalne**)
- Jeśli inna zmienna lokalna, ma taką samą nazwę jak globalna, to lokalna przesłania widoczność zmiennej globalnej
- W większości implementacji języka C zmienne **extern** są automatycznie inicjalizowane zerem
- Etykieta **extern** może być pominięta (chyba, że program składa się z kilku plików z kodem źródłowym)
- Zalecane jest ograniczenie stosowania zmiennych globalnych

## Zmienne statyczne - static

- Miejsce deklaracji: w bloku funkcyjnym jako automatyczne lub poza blokami funkcyjnymi, jako globalne
- Istnieją przez cały czas wykonywania programu, nawet po zakończeniu bloku funkcyjnego, w którym zostały zadeklarowane
- Zakres widzialności: zależny od sposobu deklaracji (automatyczne lub globalne)
- Zmienne **static** są automatycznie inicjalizowane zerem
- Mogą być inicjalizowane podczas deklaracji (tylko stałą wartością), inicjalizacja jest wykonywana tylko raz, podczas kompilacji programu

```
static int x = 10;
```

# Klasy pamięci zmiennych

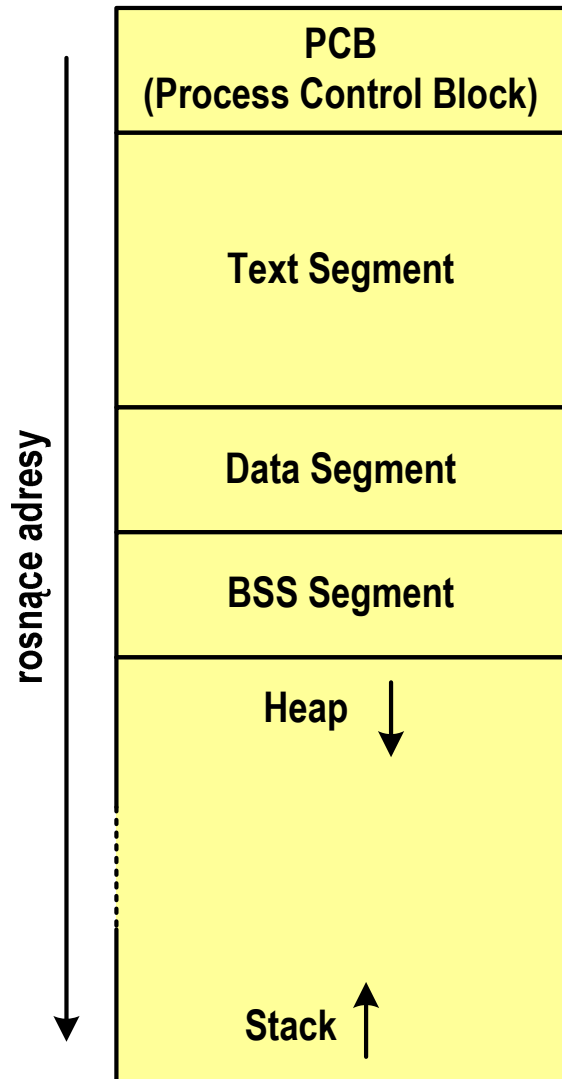
```
int a;                /* extern int a; - zmienna globalna */
void foo();

int main(void)
{
    int b;            /* auto int b; - zmienna lokalna */
    register float a; /* zmienna automatyczna, rejestrowa */
    foo(); foo(); foo();
    return 0;
}

void foo()
{
    static int c = 1; /* zmienna statyczna */
    {
        double a;    /* zmienna lokalna */
    }
    c++; a++;
}
```

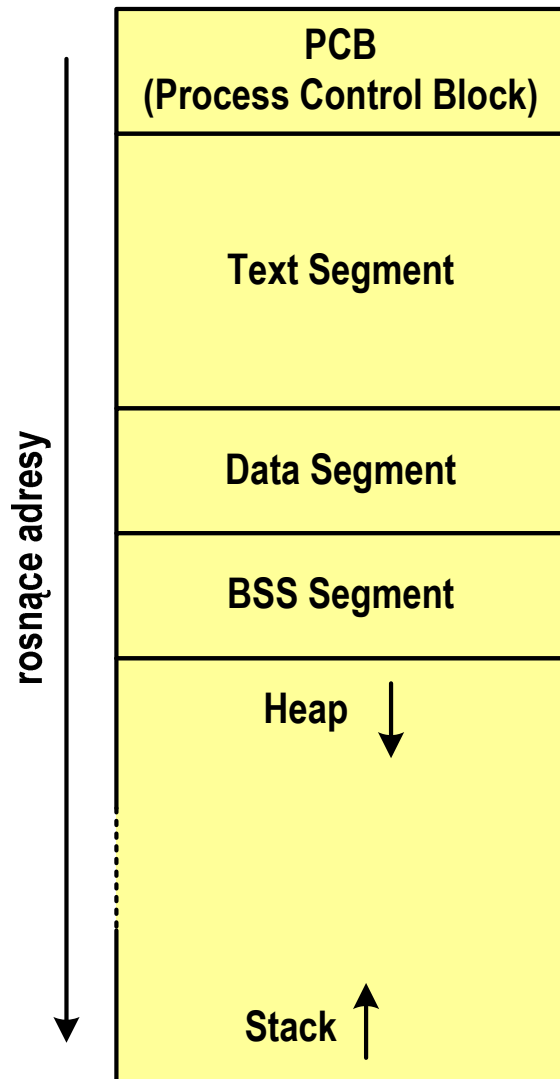


# Struktura procesu w pamięci komputera



- **PCB** - blok kontrolny procesu
  - obszar pamięci operacyjnej zarezerwowany przez system operacyjny do zarządzania procesem
- **Text Segment**
  - kod programu czyli instrukcje w postaci binarnej
- **Data Segment**
  - zmienne globalne i statyczne zainicjalizowane niezerowymi wartościami
- **BSS Segment** (Block Started by Symbol)
  - zmienne globalne i statyczne domyślnie zainicjalizowane zerowymi wartościami

# Struktura procesu w pamięci komputera



- **Heap** - sverta
  - obszar zmiennych dynamicznych
  - pamięć w obszarze sterty przydzielana jest funkcjami `calloc()` i `malloc()`
- **Stack** - stos
  - zmienne lokalne (automatyczne)
  - parametry funkcji i adresy powrotu z funkcji (stack frame)

# Zmienne w pamięci komputera

```
int a;                                /* BSS Segment */
void foo();

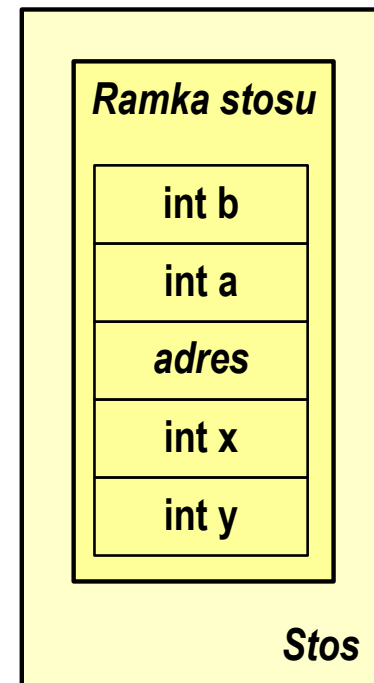
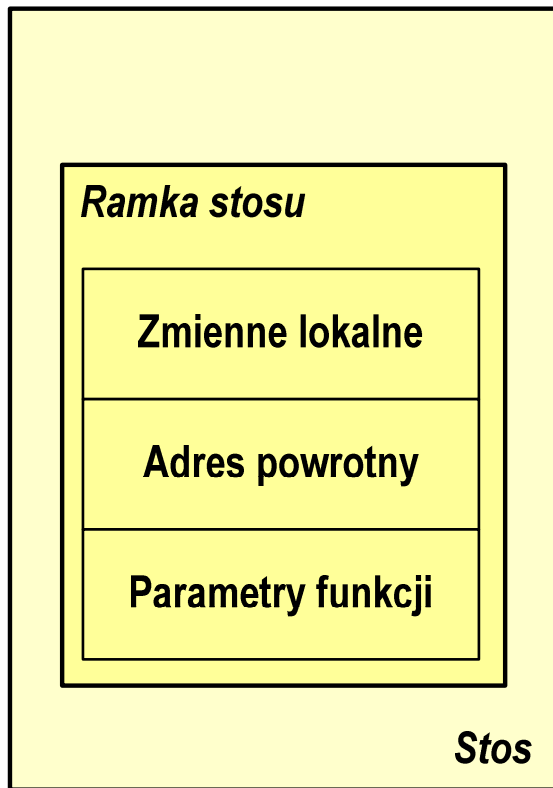
int main(void)
{
    int b;                             /* Stack */
    float *a;                          /* Stack */
    a = (float *) malloc(400);         /* Heap - 400 bajtów */
    return 0;
}

void foo()
{
    static int c = 1;                  /* Data Segment */
    {
        double a;                     /* Stack */
    }
    c++; a++;
}
```

# Ramka stosu (stack frame)

- Każde wywołanie funkcji powoduje odłożenie na stosie tzw. **ramki stosu**

```
void fun(int x, int y)
{
    int a, b;
}
```



# Arduino - podstawowe informacje

- Platforma programistyczno-sprzętowa dla systemów wbudowanych
- Strona internetowa: [www.arduino.cc](http://www.arduino.cc)
- Projekt typu open-source (Open Hardware)
- Pojedynczy moduł: mikrokontroler montowany w pojedynczym obwodzie drukowanym, z wbudowaną obsługą układów wejścia-wyjścia i interfejsem mikrokontroler-komputer PC (USB)
- Standaryzowany język programowania: bazuje na Wiring i jest podobny do C/C++
- Środowisko programistyczne: Arduino IDE



## Arduino - historia (1/2)

- Interactive Design Institute Ivrea (IDII), Ivrea, Włochy
- **2003** - student Hernando Barragán, w ramach realizacji pracy magisterskiej, tworzy platformę programistyczną **Wiring** (opiekunowie pracy: Massimo Banzi, Casey Reas)
- **Cel projektu:** stworzenie prostych, tanich narzędzi do tworzenia projektów cyfrowych przez osoby nie będące inżynierami (artystów, projektantów)
- **Platforma:** płytko drukowana z mikrokontrolerem ATmega128 oraz środowisko programistyczne (IDE) umożliwiające łatwe programowanie mikrokontrolera
- **2005** - Massimo Banzi, David Mellis (student) i David Cuartielles rozszerzają platformę Wiring poprzez dodanie obsługi tańszego mikrokontrolera ATmega8 - projekt ten zostaje nazwany **Arduino**

## Arduino - historia (2/2)

- **2008** - Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino i David Mellis tworzą firmę **Arduino LLC**, która miała być właścicielem znaku towarowego Arduino
- Produkcją i sprzedażą płytek miały zajmować się zewnętrzne firmy płacące tantiemy Arduino LCC
- „The Untold History of Arduino” by Hernando Barragán:
  - <https://arduinohistory.github.io/>

# Arduino - informacje w Internecie

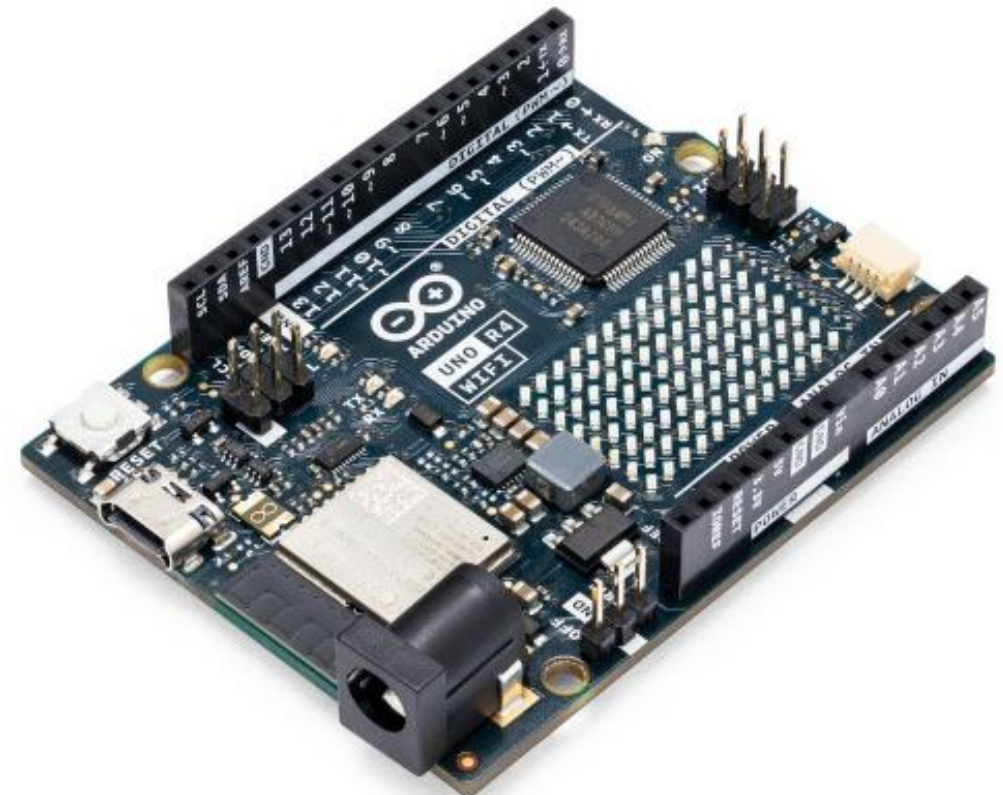
- Strona internetowa: [www.arduino.cc](http://www.arduino.cc)
- Arduino open source community: [github.com/arduino](https://github.com/arduino)
- Kursy internetowe:
  - Forbot (Kurs Arduino, poziom I): <https://forbot.pl/blog/kurs-arduino-podstawy-programowania-spis-tresci-kursu-id5290>
  - Forbot (Kurs Arduino, poziom II): <https://forbot.pl/blog/kurs-arduino-ii-wstep-spis-tresci-id15494>
  - Kurs Arduino (Youtube, kanał KoValsky majstruje): <https://www.youtube.com/watch?v=TzTmWqoN9i8>
  - Kurs Arduino (Youtube, kanał Forbidden Bit): <https://www.youtube.com/playlist?list=PLgjf5CWt5eA9mCBgZWinYSTVw960RYAqg>



# Arduino - moduły

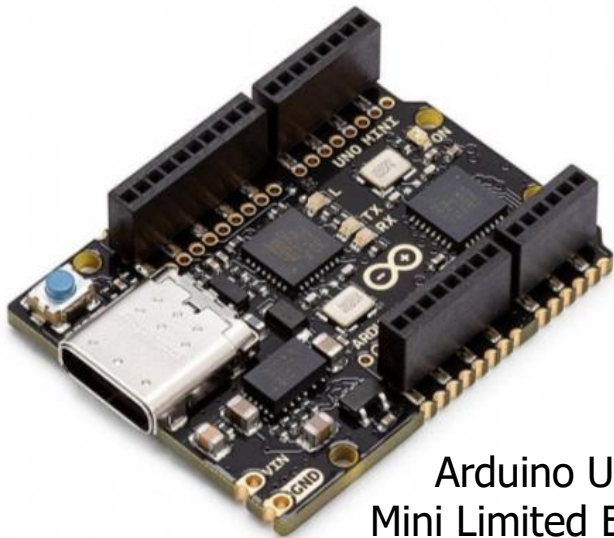


Arduino Uno Rev3

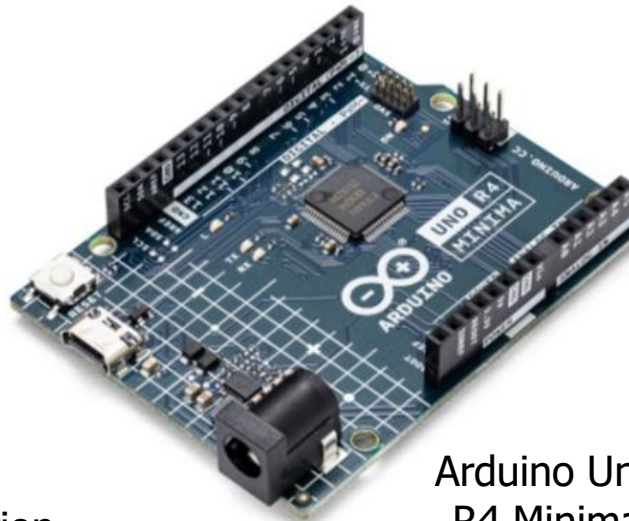


Arduino Uno R4 WiFi

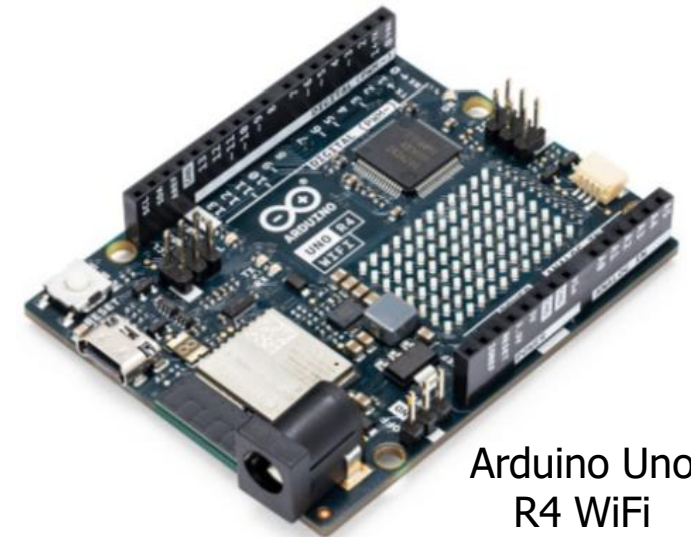
# Arduino - moduły



Arduino Uno  
Mini Limited Edition



Arduino Uno  
R4 Minima



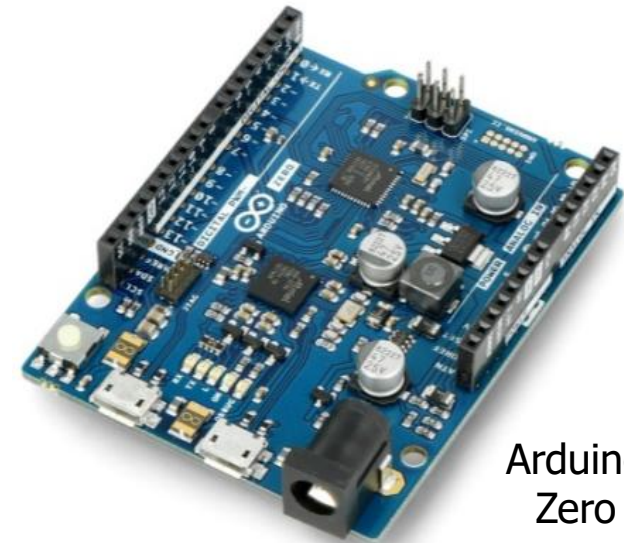
Arduino Uno  
R4 WiFi



Arduino  
Leonardo



Arduino  
Micro



Arduino  
Zero

# Arduino - moduły



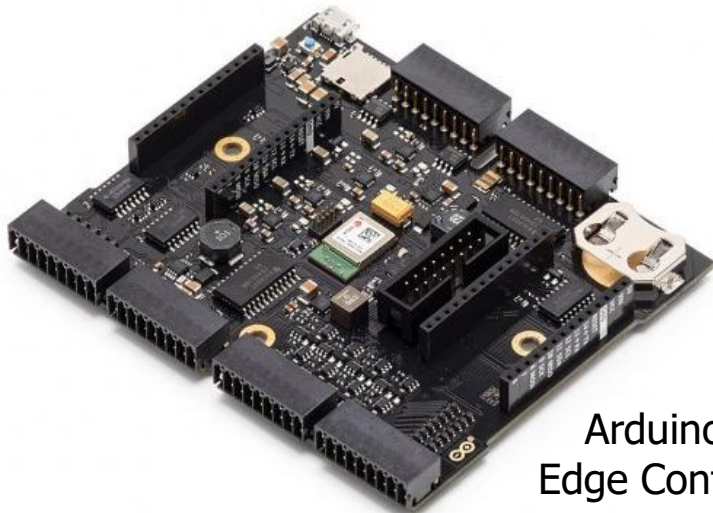
Arduino  
MKR Zero



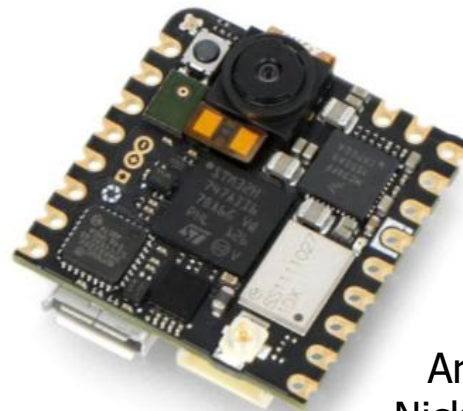
Arduino  
Nano



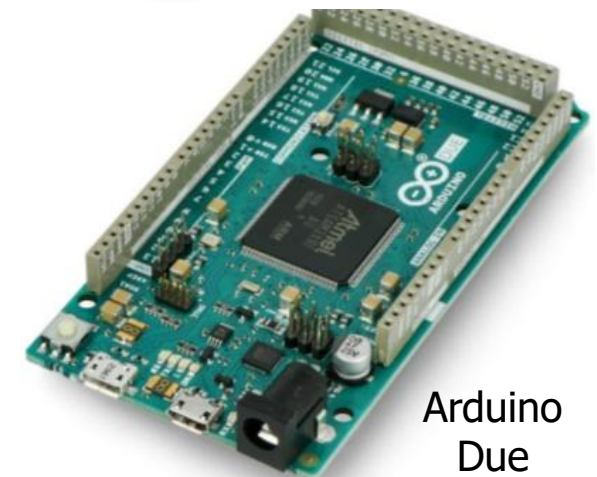
Arduino  
Portenta C33



Arduino  
Edge Control

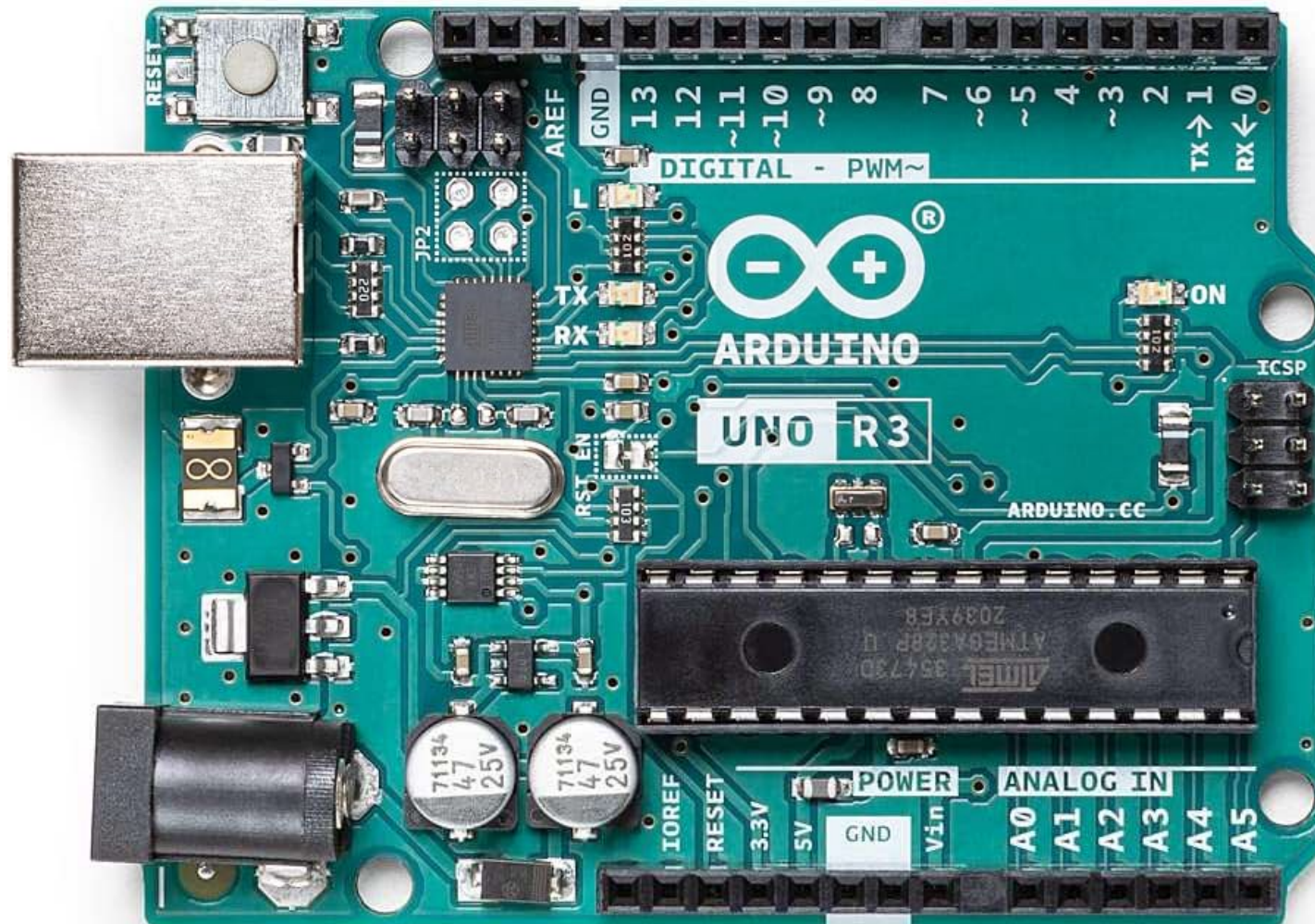


Arduino  
Nicla Vision



Arduino  
Due

# Arduino Uno R3



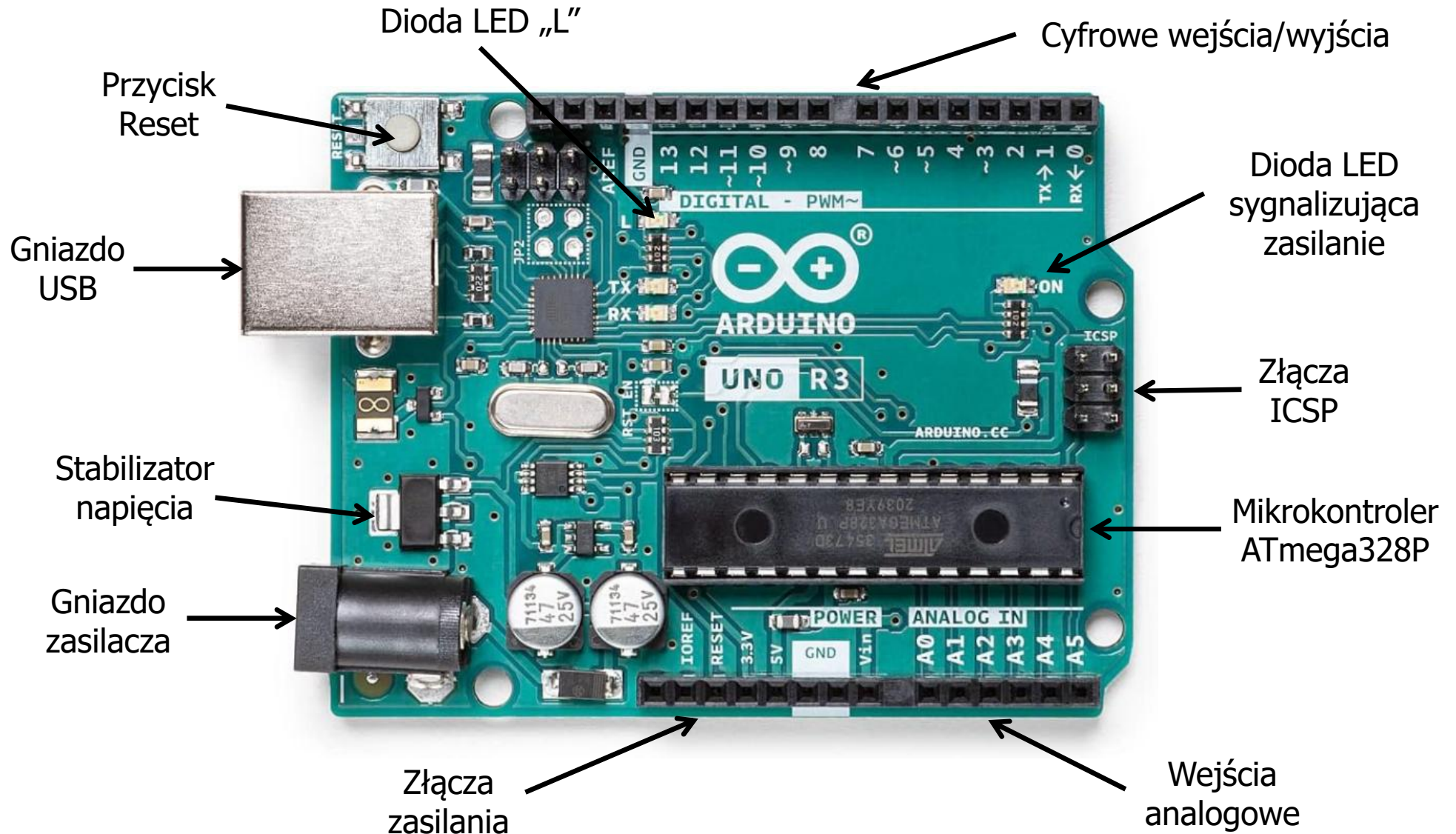
# Arduino Uno R3 - specyfikacja

- Zasilanie:
  - złącze koncentryczne 5,5/2,1 mm (napięcie od 7 do 12 V)
  - złącze USB typu B - napięcie 5 V
- Mikrokontroler: ATmega328P w obudowie DIP-28:
  - pamięć SRAM: 2 kB
  - pamięć Flash: 32 kB (0,5 kB zarezerwowane dla bootloadera)
  - pamięć EEPROM: 1 kB
  - częstotliwość zegara: 16 MHz
- 20 uniwersalnych wyprowadzeń wejść/wyjść:
  - 14 cyfrowych wejść/wyjść (w tym 6 z możliwością generowania 8-bitowego sygnału PWM)
  - 6 wejść analogowych o rozdzielczości 10 bitów (mogących pełnić funkcję cyfrowych wejść/wyjść)

# Arduino Uno R3 - specyfikacja

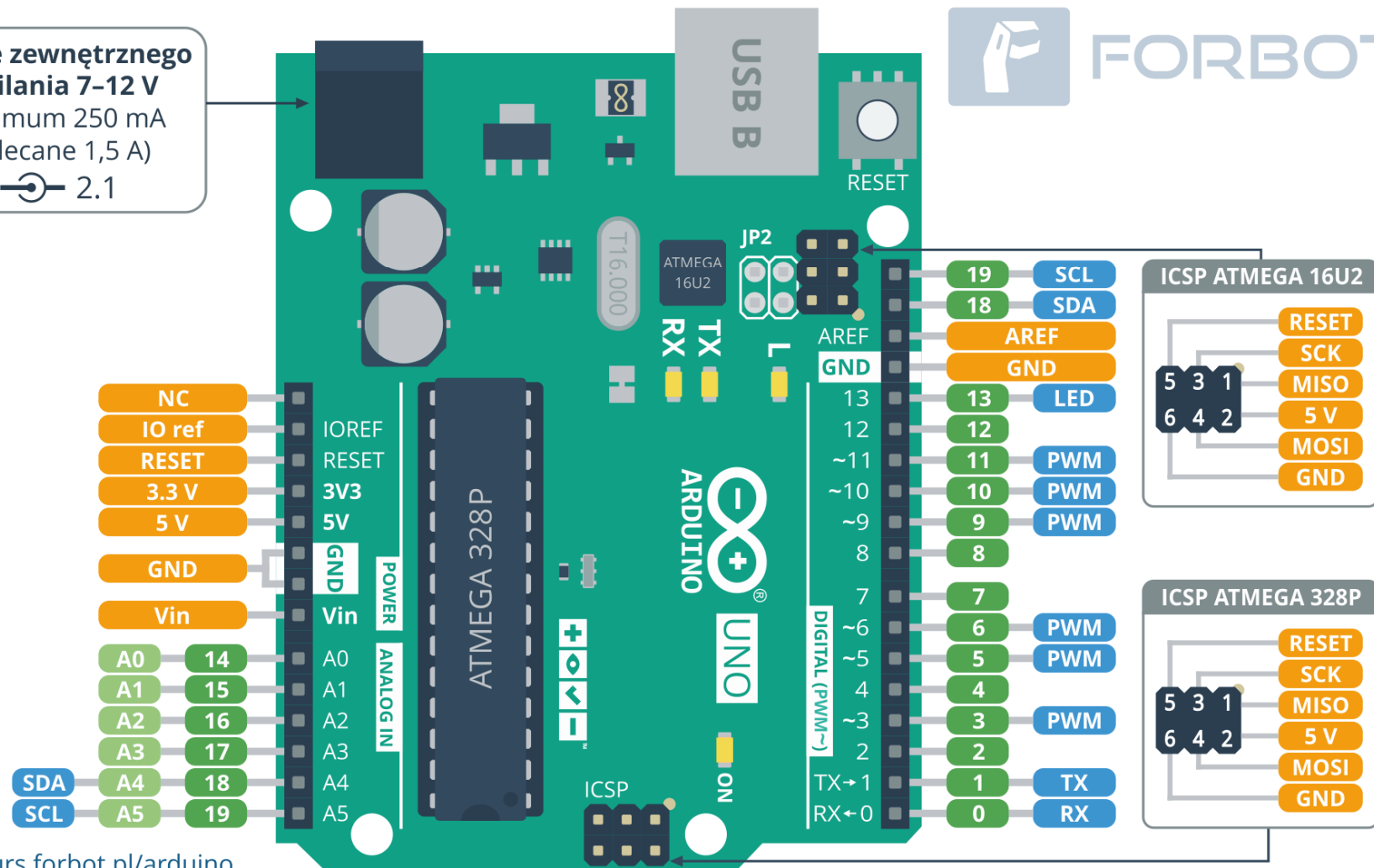
- Interfejsy komunikacyjne: UART, I2C, SPI
- Zewnętrzne przerwania: 2
- Maksymalny prąd:
  - dla wyjścia 5 V: 500 mA
  - dla wyjścia 3,3 V: 50 mA
  - dla poszczególnych GPIO: 20 mA
- Rozmiary płytki: 68,6 × 53,4 mm

# Arduino Uno R3 - elementy na płycie



# Arduino Uno R3 - opis wyprowadzeń

Złącze zewnętrznego zasilania 7-12 V  
minimum 250 mA  
(zalecane 1,5 A)  
2.1

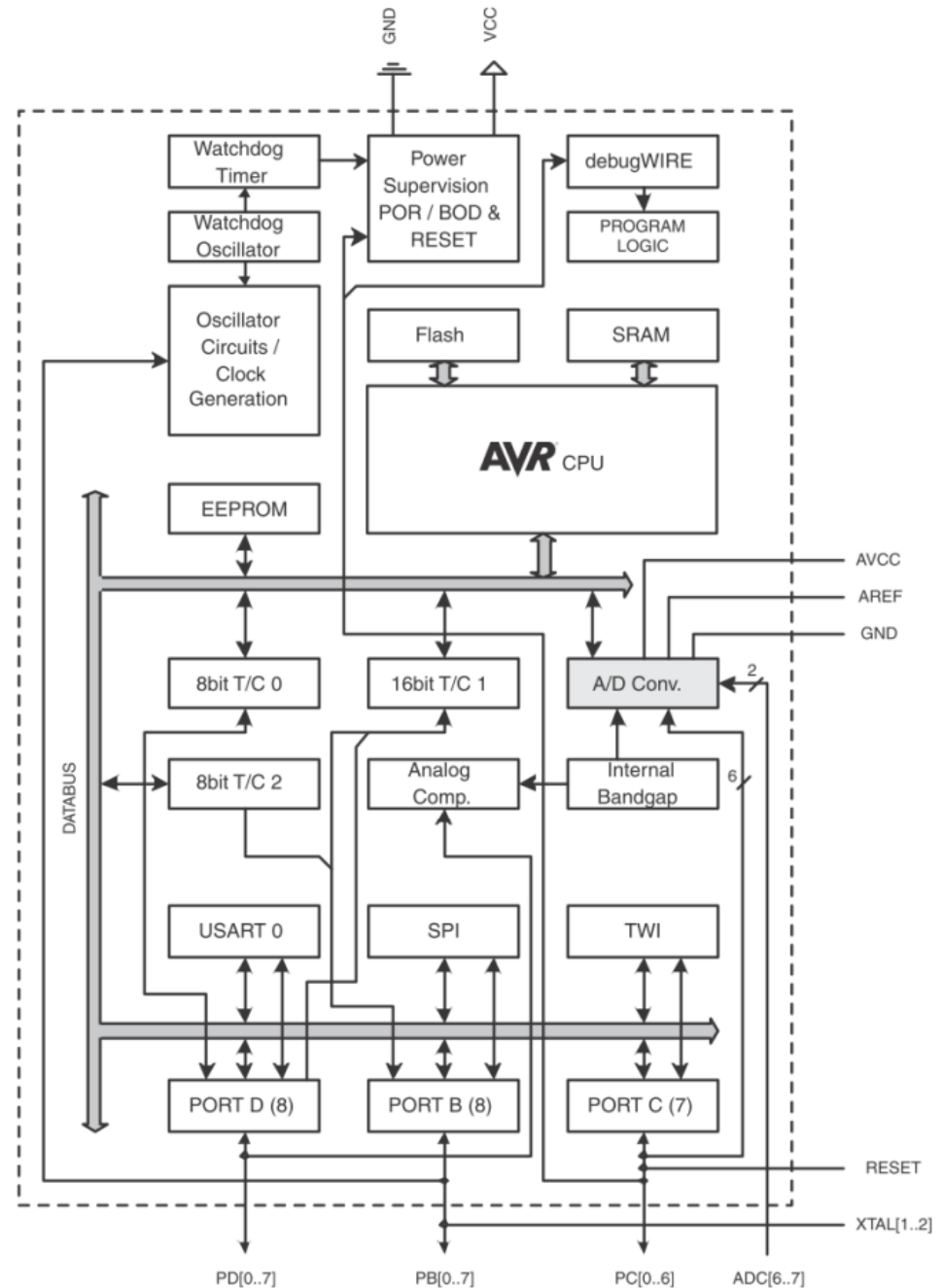




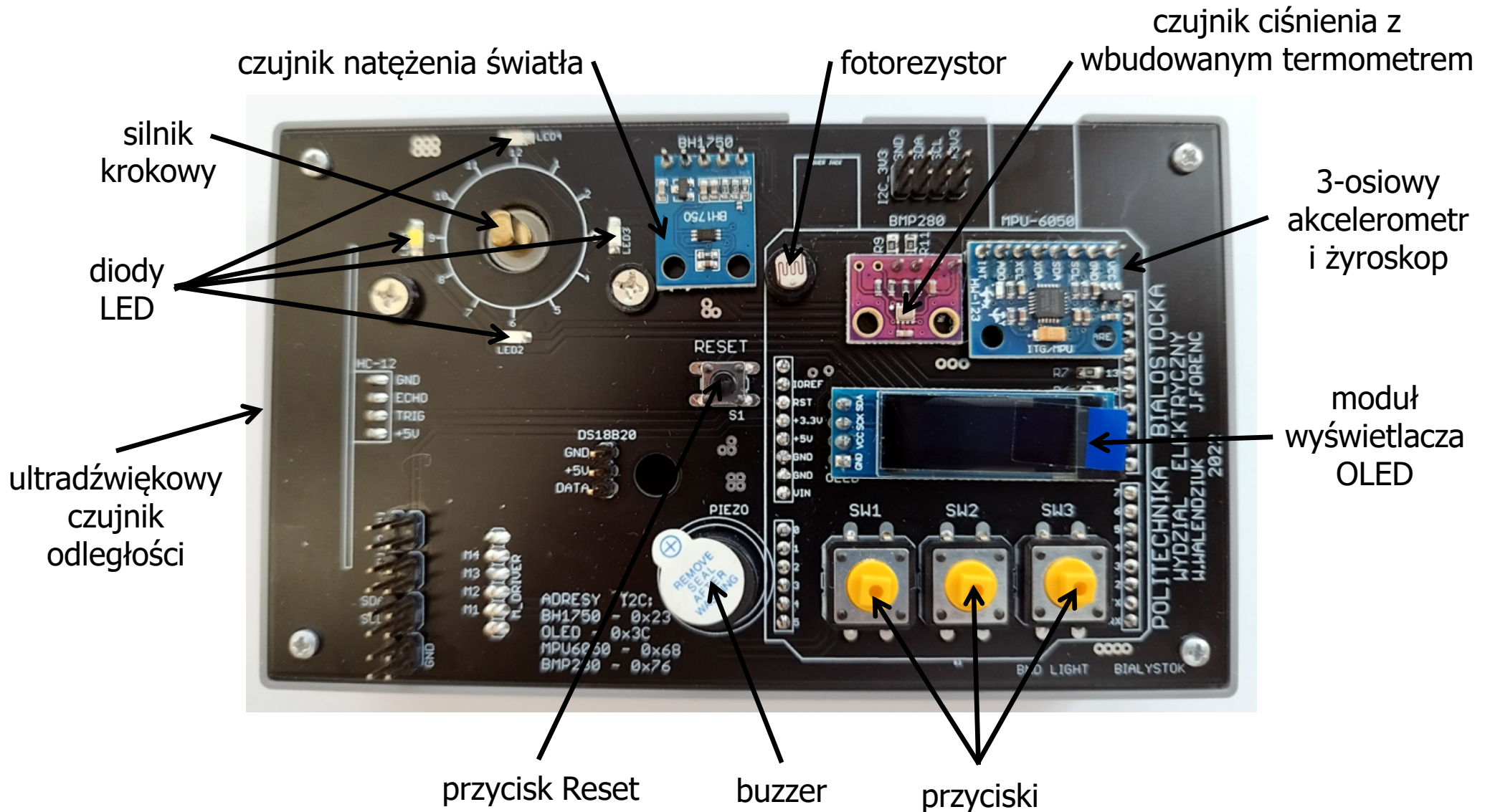
# ATmega328P

## ■ Specyfikacja:

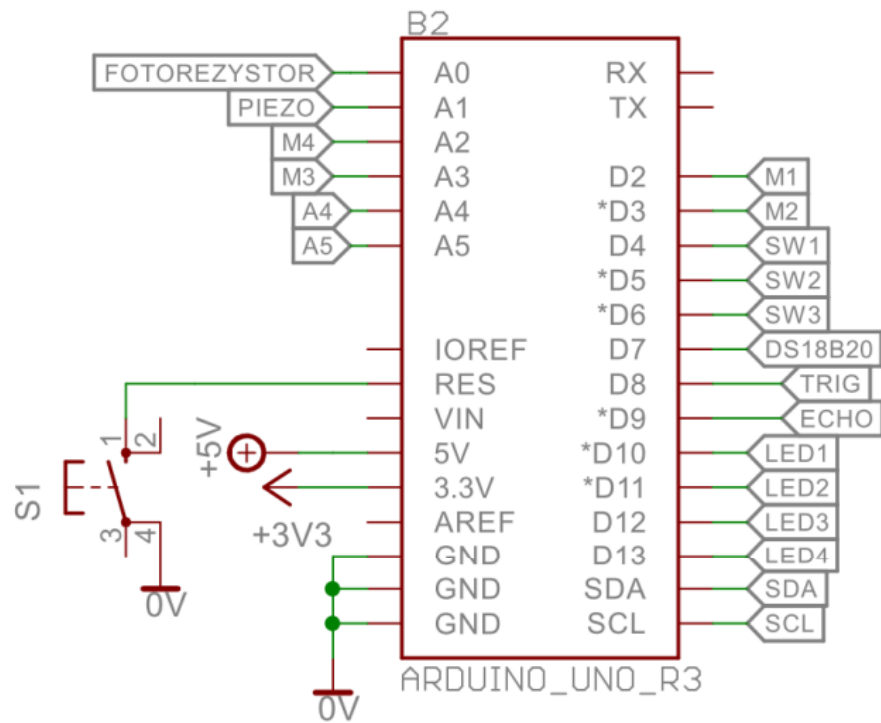
- ❑ zasilanie: 1,8 V - 5,5 V
- ❑ taktowanie: do 20 MHz
- ❑ pamięć Flash: 32 KB
- ❑ 23 linie wyjścia/wejścia
- ❑ dwa 8-bitowe liczniki
- ❑ jeden 16-bitowy licznik
- ❑ 6 kanałów PWM
- ❑ 6 kanałów 10-bitowego przetwornika A/D
- ❑ sprzętowe interfejsy komunikacyjne: USART, SPI, TWI (I2C)
- ❑ obudowa DIP



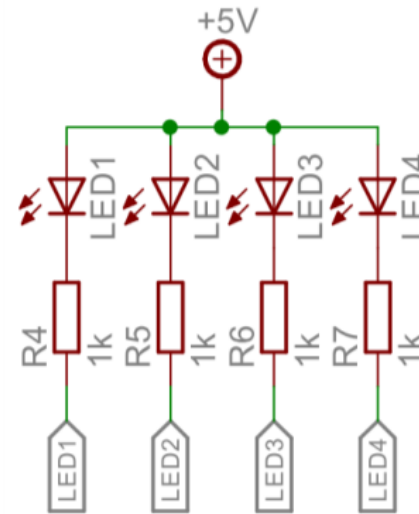
# Moduł stosowany na zajęciach



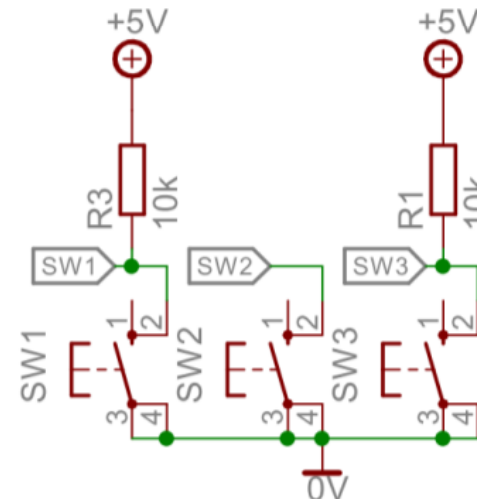
# Moduł stosowany na zajęciach



Schemat podłączenia wyprowadzeń modułu Arduino

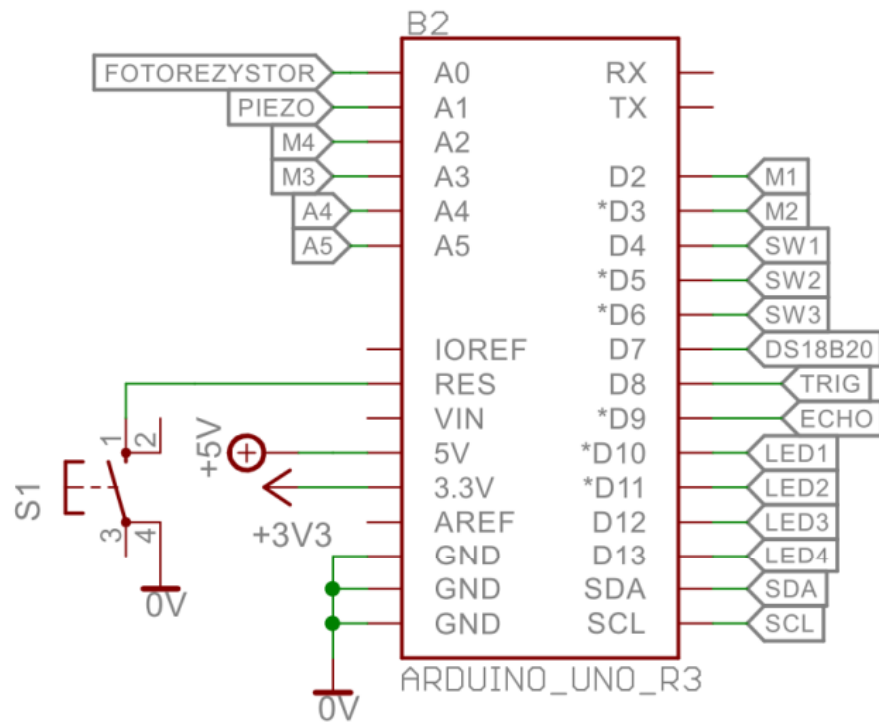


Schemat podłączenia diod LED

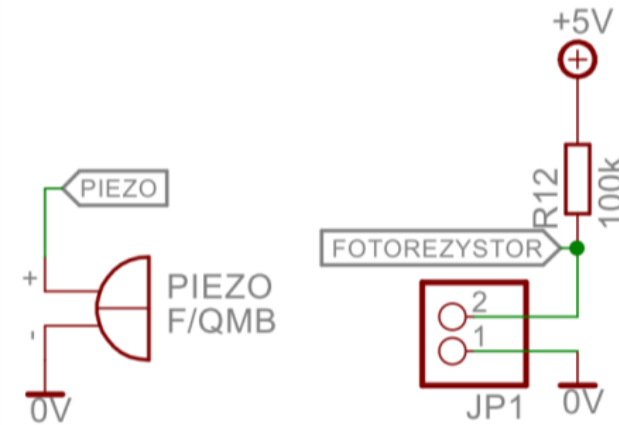


Schemat podłączenia przycisków

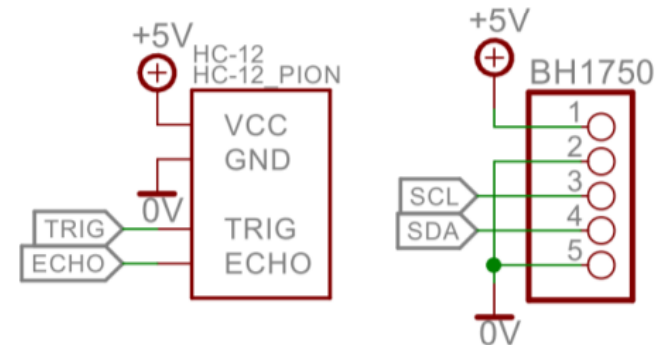
# Moduł stosowany na zajęciach



Schemat podłączenia wyprowadzeń modułu Arduino

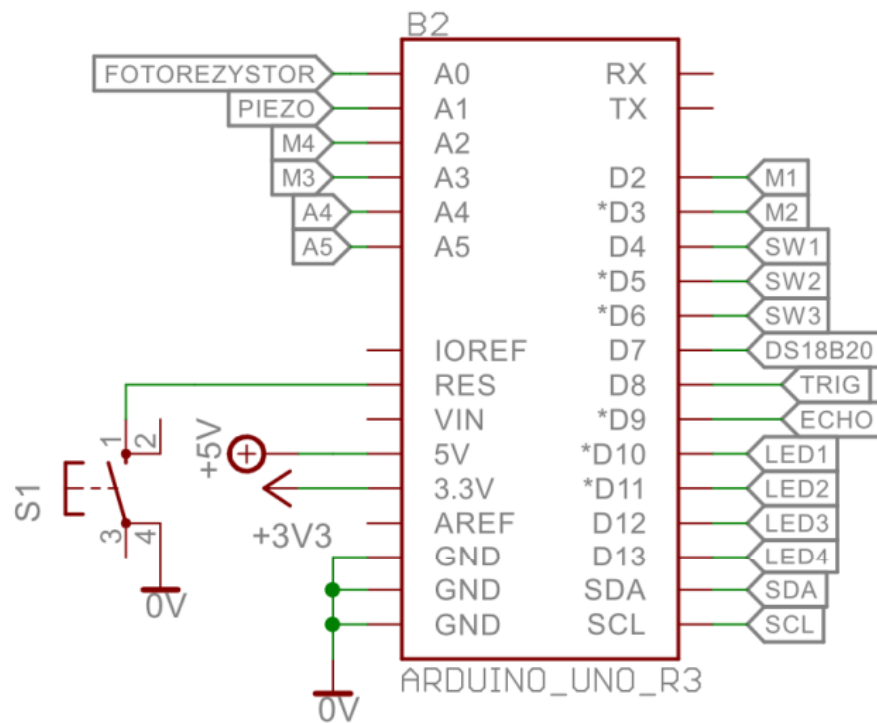


Schemat podłączenia buzzera i fotorezystora

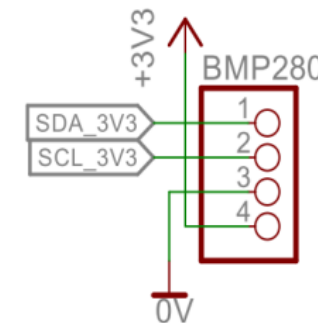


Schemat podłączenia czujnika odległości i czujnika natężenia światła

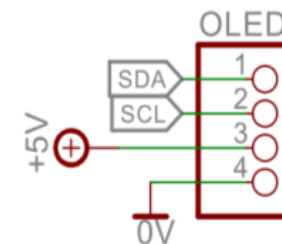
# Moduł stosowany na zajęciach



Schemat podłączenia  
wyprowadzeń modułu Arduino



Schemat podłączenia  
czujnika ciśnienia i temperatury



Schemat podłączenia  
wyświetlacza OLED

# Arduino IDE

- <https://www.arduino.cc/en/software> (najnowsza wersja: 2.3.3)



## Arduino IDE 2.2.1

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

### SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

### DOWNLOAD OPTIONS

**Windows** Win 10 and newer, 64 bits

**Windows** MSI installer

**Windows** ZIP file

**Linux** AppImage 64 bits (X86-64)

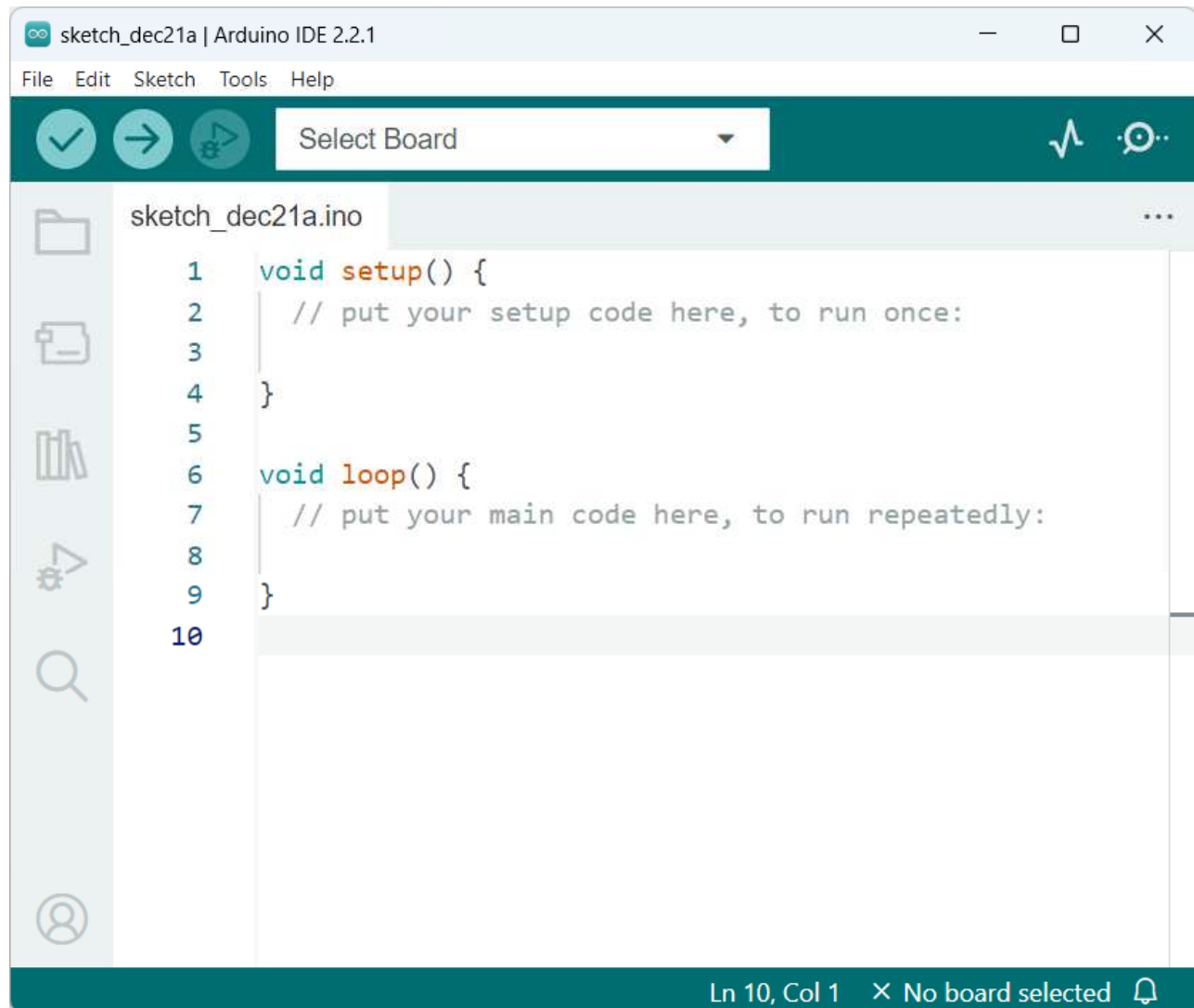
**Linux** ZIP file 64 bits (X86-64)

**macOS** Intel, 10.14: "Mojave" or newer, 64 bits

**macOS** Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

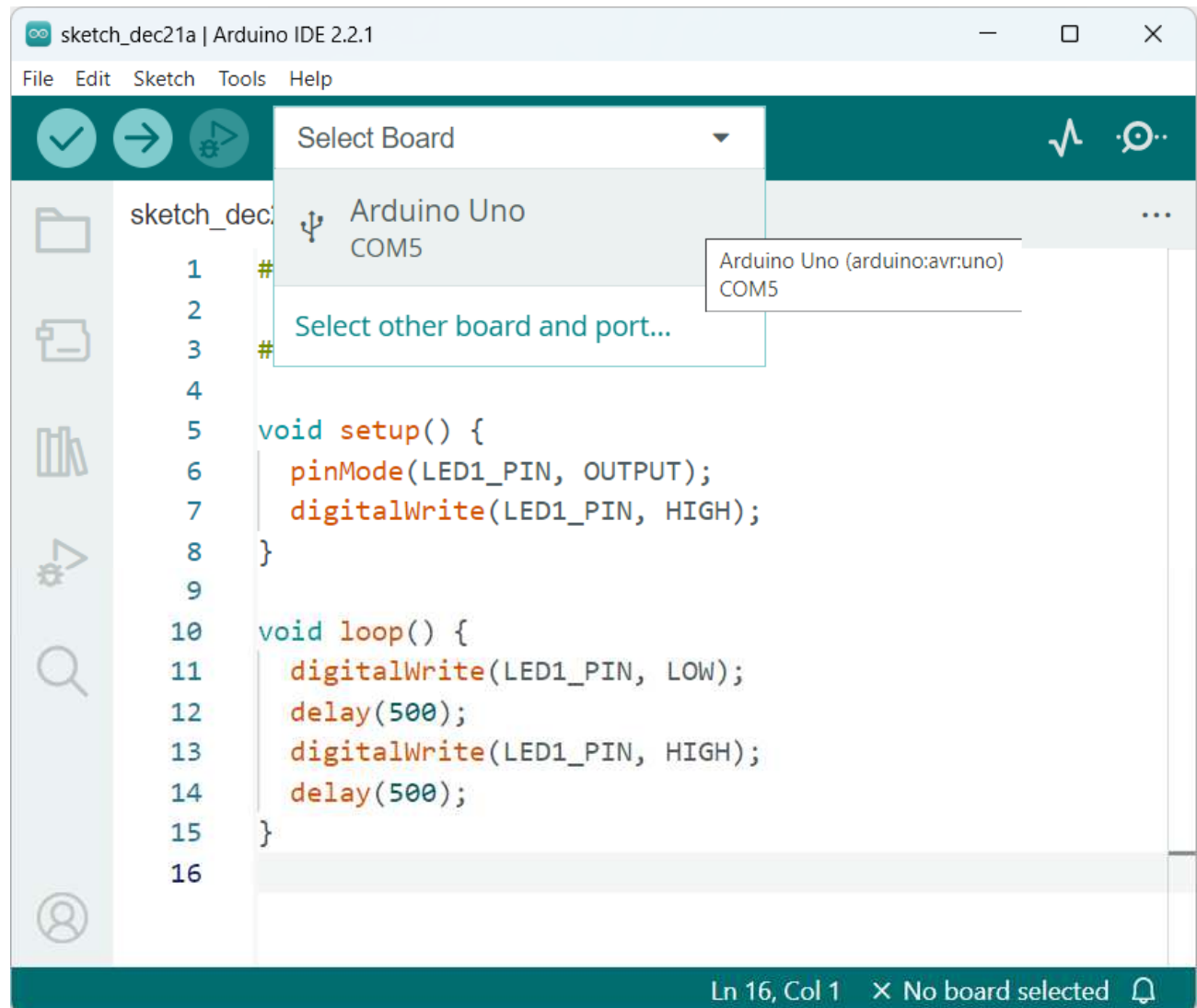
# Arduino IDE 2.2.1



Środowisko Arduino  
IDE z przykładowym  
kodem programu


# Arduino IDE 2.2.1

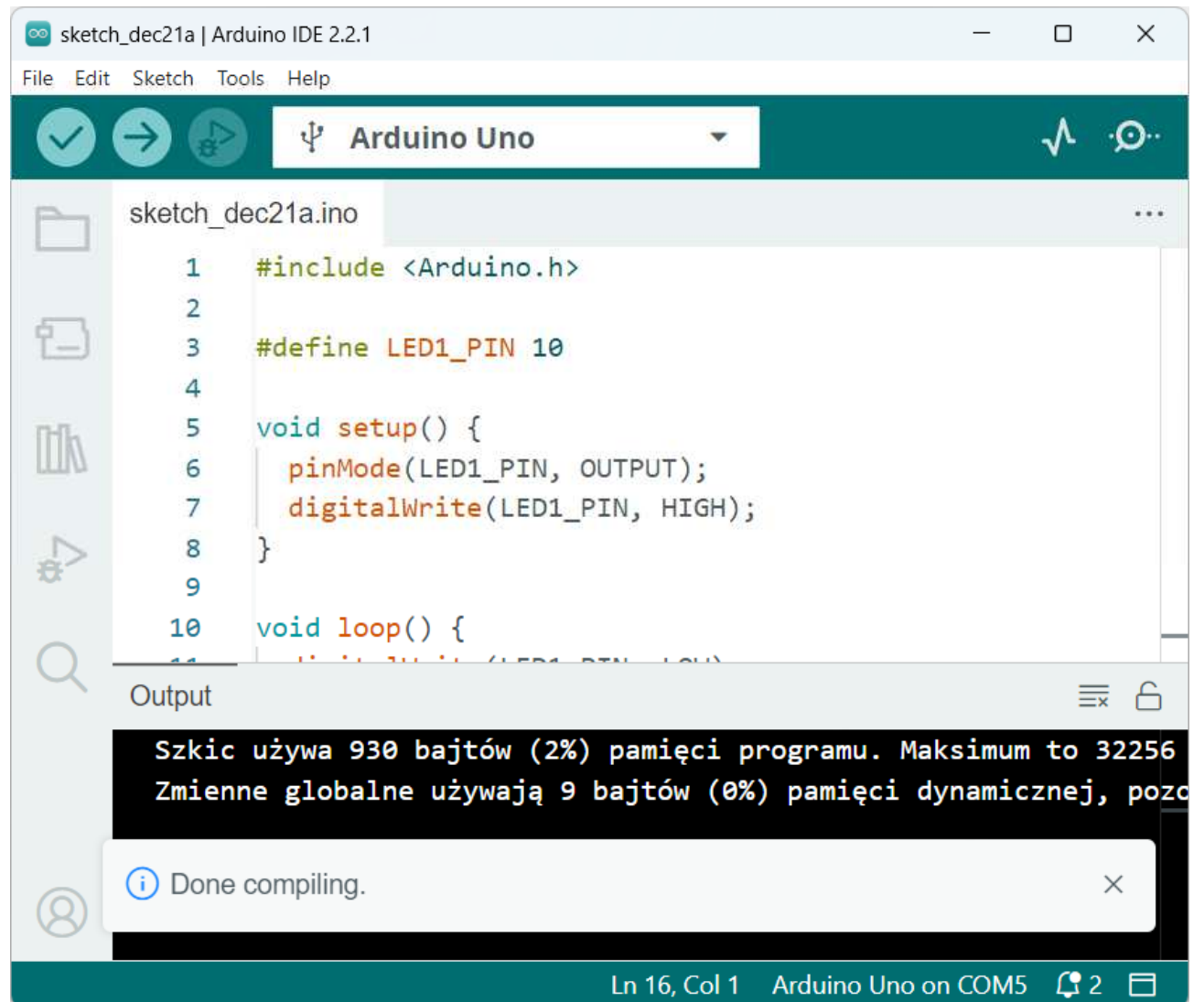
- wprowadzamy kod programu
- wybieramy typ płytki





# Arduino IDE 2.2.1

- wprowadzamy kod programu
- wybieramy typ płytki
- 
- Verify - sprawdzenie i kompilacja programu



```
sketch_dec21a | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Arduino Uno
sketch_dec21a.ino
1 #include <Arduino.h>
2
3 #define LED1_PIN 10
4
5 void setup() {
6   pinMode(LED1_PIN, OUTPUT);
7   digitalWrite(LED1_PIN, HIGH);
8 }
9
10 void loop() {
11   digitalWrite(LED1_PIN, LOW);
12 }
```

Output

Szkic używa 930 bajtów (2%) pamięci programu. Maksimum to 32256  
Zmienne globalne używają 9 bajtów (0%) pamięci dynamicznej, pozostaje 1023 bajtów.

Done compiling.

Ln 16, Col 1 Arduino Uno on COM5 2

# Arduino IDE 2.2.1

- wprowadzamy kod programu
- wybieramy typ płytki



- Verify - sprawdzenie i kompilacja programu



- Upload - sprawdzenie, kompilacja i przesłanie programu do Arduino

```
sketch_dec21a | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Arduino Uno
sketch_dec21a.ino
1  #include <Arduino.h>
2
3  #define LED1_PIN 10
4
5  void setup() {
6    pinMode(LED1_PIN, OUTPUT);
7    digitalWrite(LED1_PIN, HIGH);
8  }
9
10 void loop() {
11   digitalWrite(LED1_PIN, LOW);
12 }
```

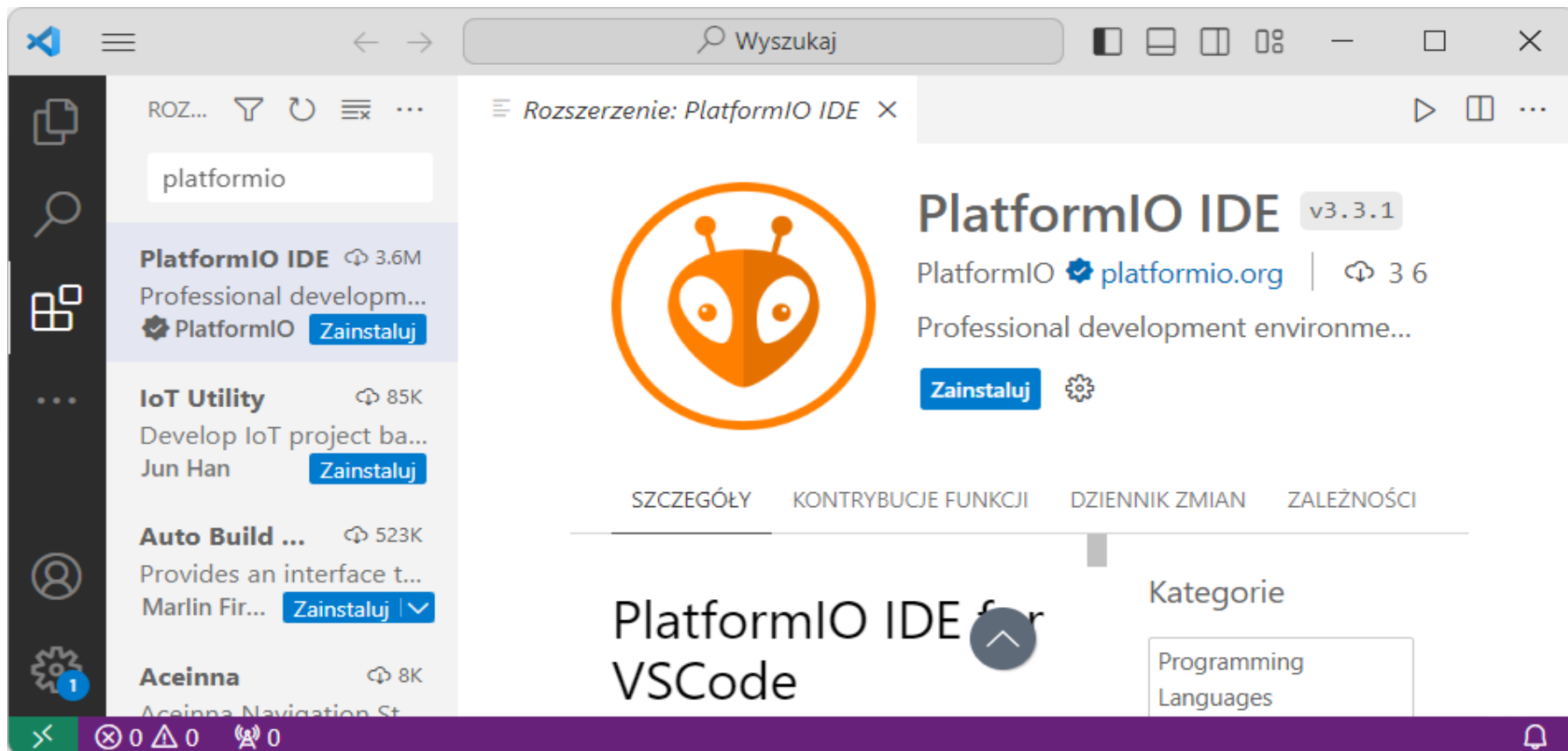
Output

Szkic używa 930 bajtów (2%) pamięci programu. Maksimum to 32256  
Zmienne globalne używają 9 bajtów (0%) pamięci dynamicznej, pozostaje 1023 bajtów.

Done uploading.

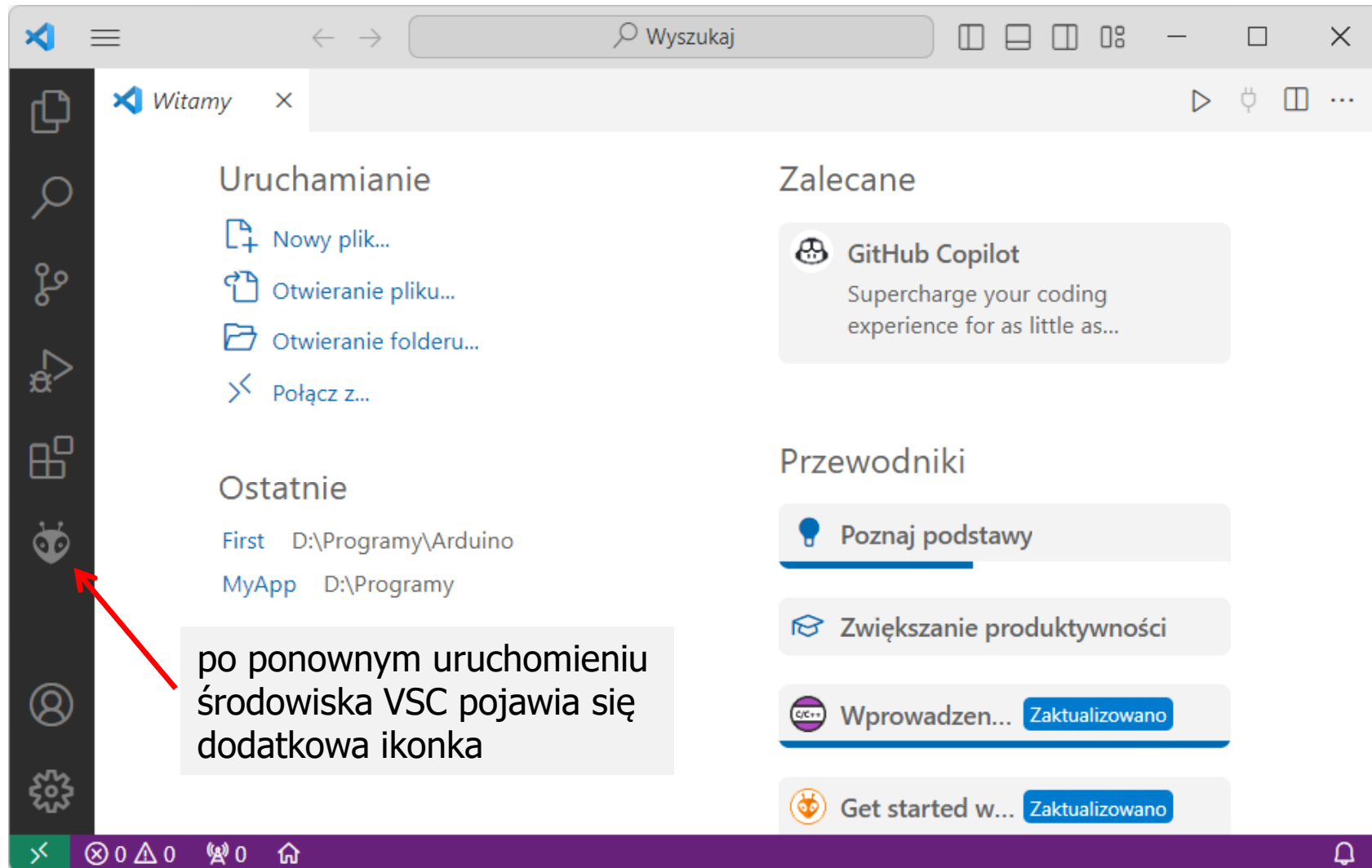
Ln 16, Col 1 Arduino Uno on COM5 2

# Visual Studio Code + PlatformIO IDE 3.3.1

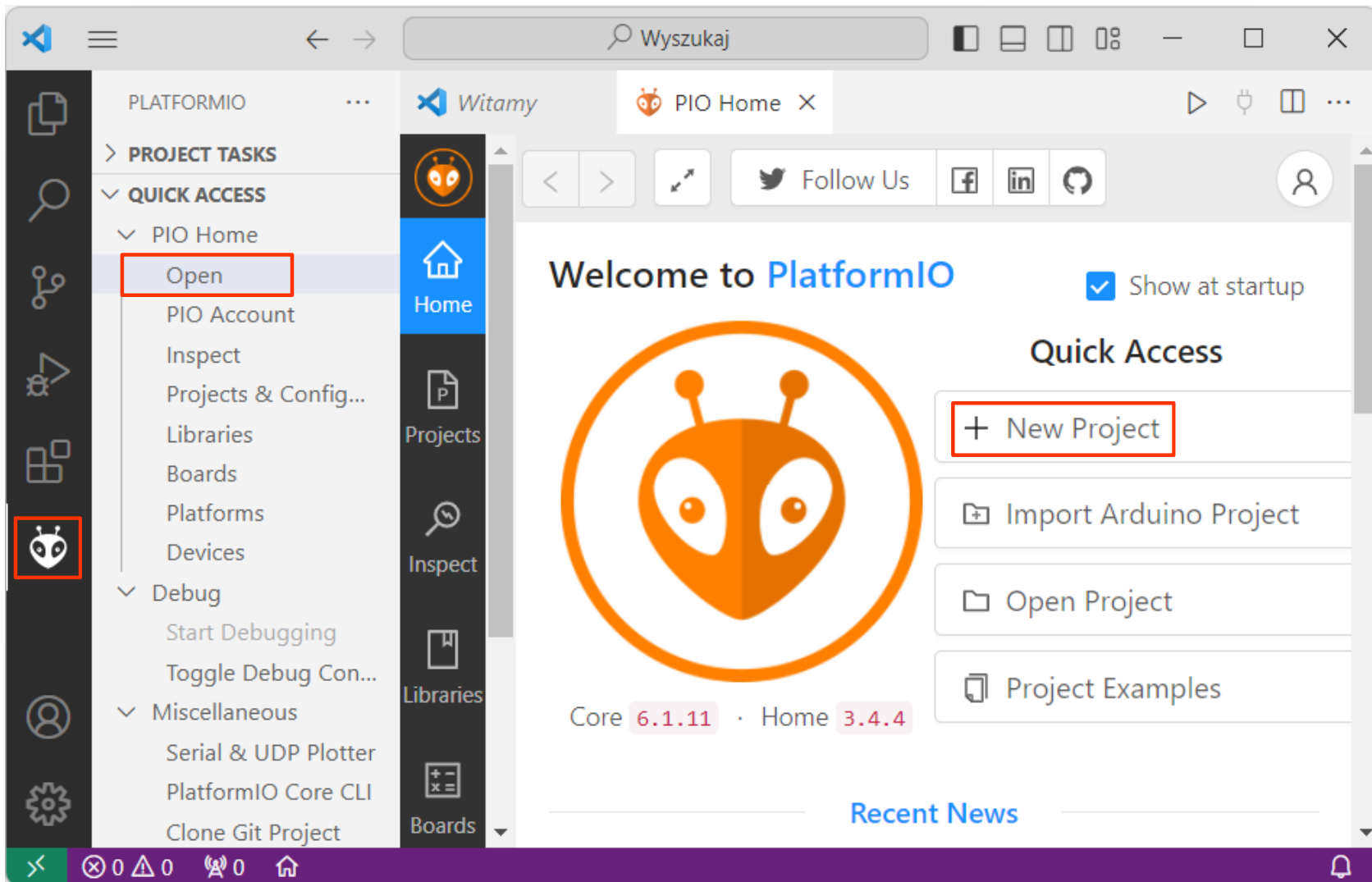


- ❑ rozszerzenia: **Ctrl + Shift + X**
- ❑ wyszukanie i instalacja rozszerzenia **PlatformIO IDE**

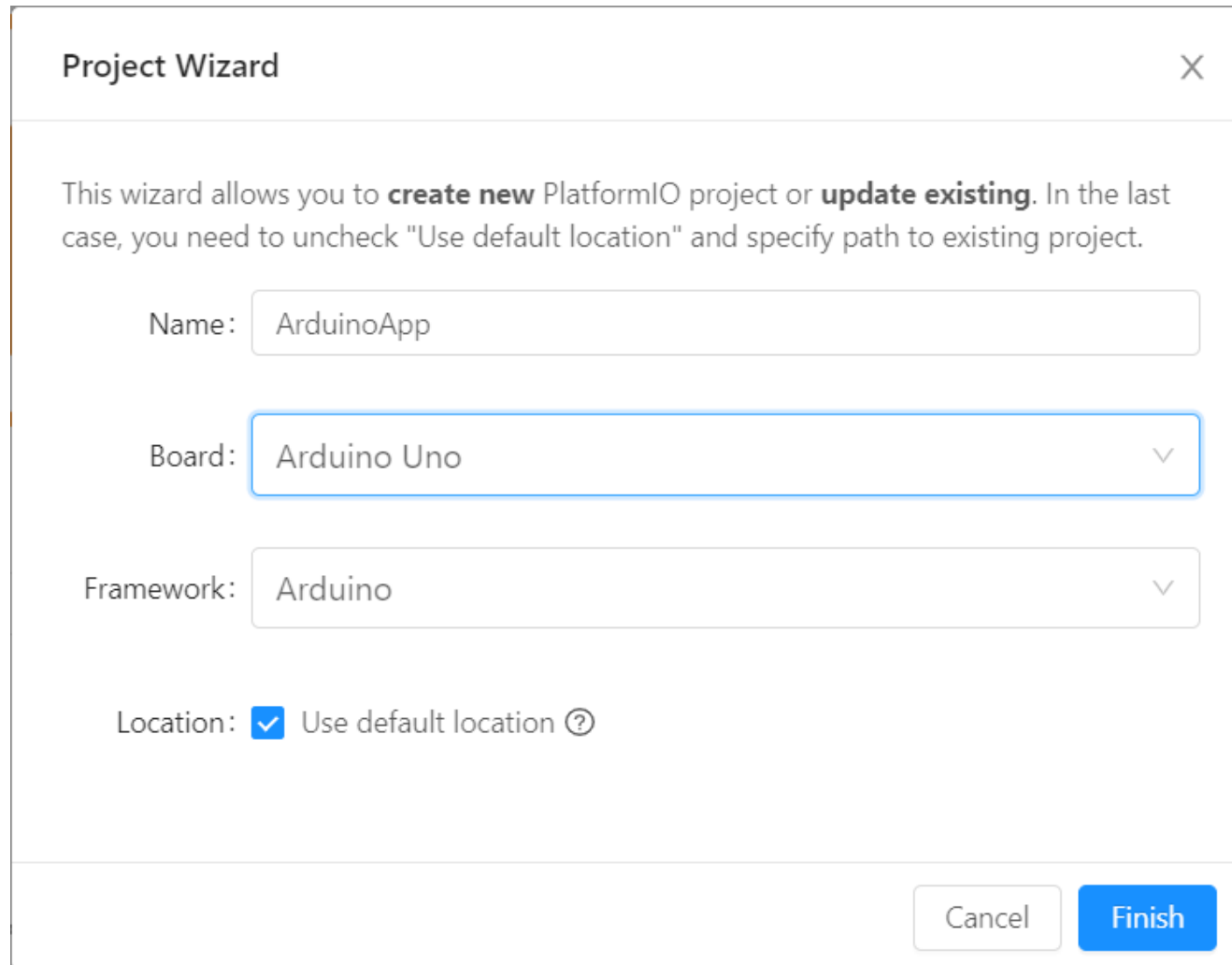
# Visual Studio Code + PlatformIO IDE 3.3.1



# VS Code: stworzenie projektu



# VS Code: stworzenie projektu



The image shows a 'Project Wizard' dialog box in VS Code. It has a title bar with 'Project Wizard' and a close button (X). The main text reads: 'This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.'

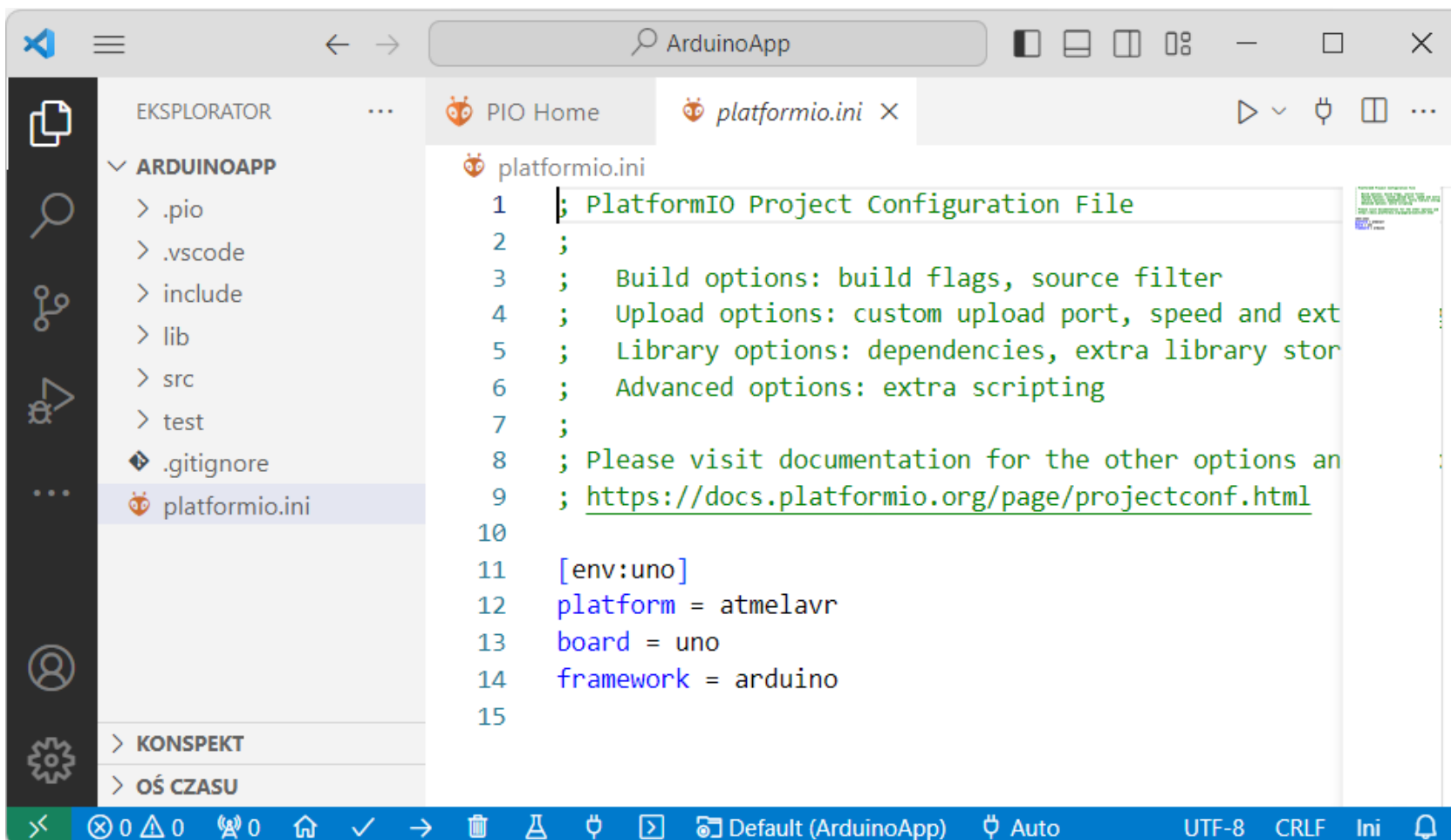
There are three input fields:

- Name:
- Board: - Framework:

At the bottom, there is a 'Location' section with a checked checkbox and the text 'Use default location' followed by a help icon (?).

At the bottom right, there are two buttons: 'Cancel' and 'Finish'.

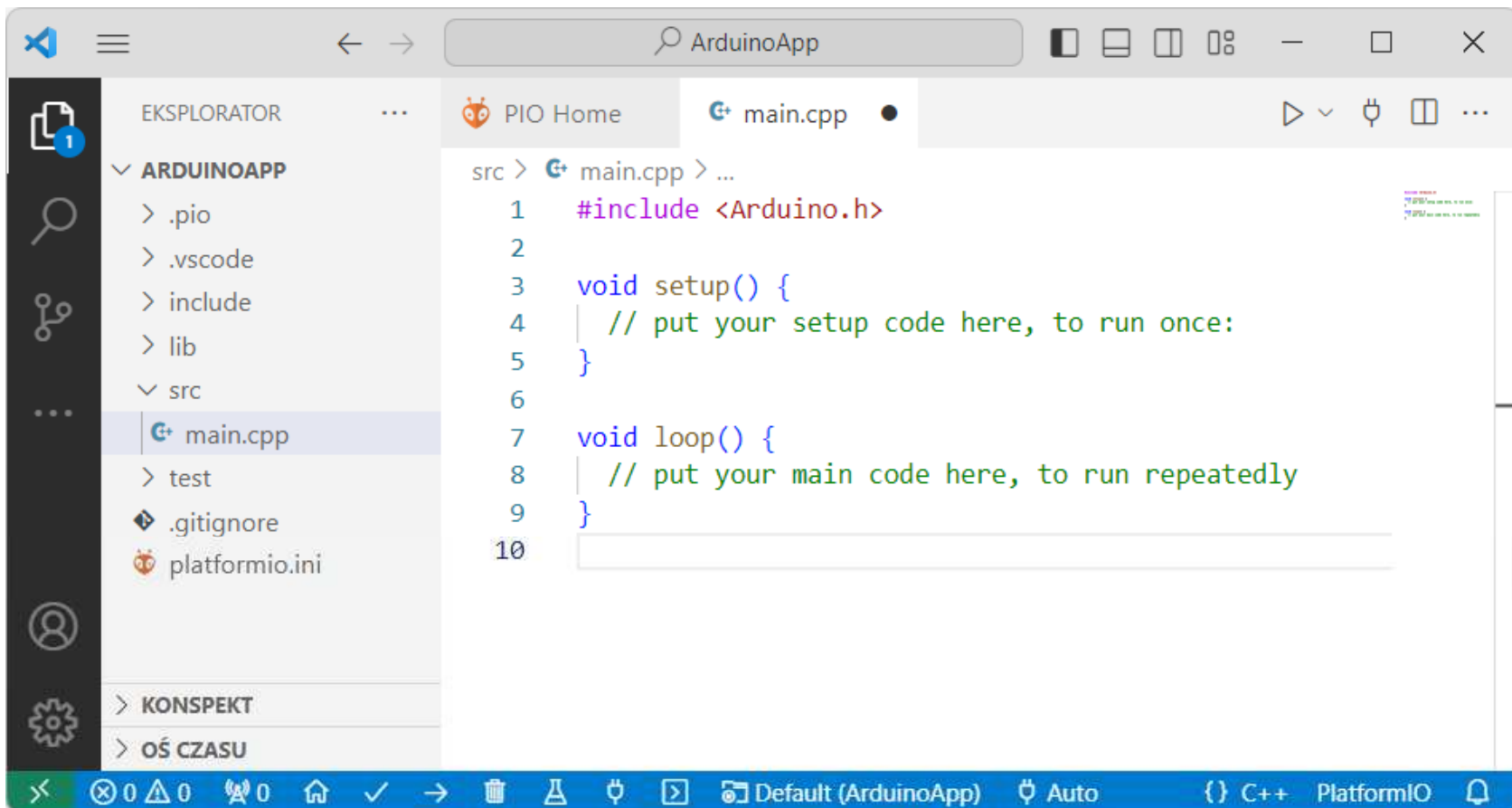
# VS Code: stworzenie projektu



```
platformio.ini
1  ; PlatformIO Project Configuration File
2  ;
3  ; Build options: build flags, source filter
4  ; Upload options: custom upload port, speed and ext
5  ; Library options: dependencies, extra library stor
6  ; Advanced options: extra scripting
7  ;
8  ; Please visit documentation for the other options an
9  ; https://docs.platformio.org/page/projectconf.html
10
11 [env:uno]
12 platform = atmelavr
13 board = uno
14 framework = arduino
15
```

- **platformio.ini** - główny plik konfiguracyjny projektu

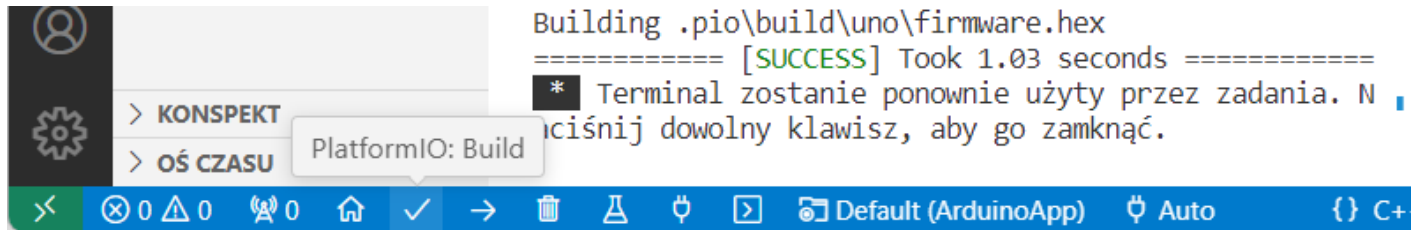
# VS Code: stworzenie projektu



- **src** → **main.cpp** - plik z kodem źródłowym programu



## VS Code: kompilacja i przesłanie programu

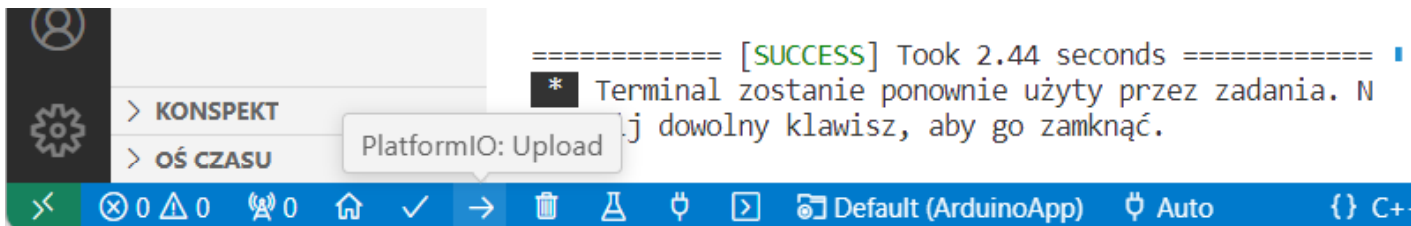


The screenshot shows the VS Code interface with the terminal window open. The terminal output indicates a successful build of the firmware. The text in the terminal is as follows:

```
Building .pio\build\uno\firmware.hex  
===== [SUCCESS] Took 1.03 seconds =====  
* Terminal zostanie ponownie użyty przez zadania. Naciśnij dowolny klawisz, aby go zamknąć.
```

The PlatformIO: Build button is highlighted in the toolbar.

- **Build** - sprawdzenie i kompilacja programu



The screenshot shows the VS Code interface with the terminal window open. The terminal output indicates a successful upload of the firmware. The text in the terminal is as follows:

```
===== [SUCCESS] Took 2.44 seconds =====  
* Terminal zostanie ponownie użyty przez zadania. Naciśnij dowolny klawisz, aby go zamknąć.
```

The PlatformIO: Upload button is highlighted in the toolbar.

- **Upload** - sprawdzenie, kompilacja i przesłanie programu do Arduino

# Arduino - ogólna struktura programu

## ■ Ogólna struktura programu

```
#include <Arduino.h>

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

- `setup()` - funkcja wykonywana tylko jeden raz po uruchomieniu Arduino
- `loop()` - funkcja uruchamiana w pętli (w kółko)

## Arduino Uno - typy danych

| Nazwa                | Rozmiar (bajty) | Uwagi  |
|----------------------|-----------------|--|
| <code>bool</code>    | 1               | przechowuje jedną z dwóch wartości: <code>true</code> lub <code>false</code> |
| <code>boolean</code> | 1               | oznacza to samo co <code>bool</code>   |
| <code>byte</code>    | 1               | 8-bitowa liczba całkowita bez znaku, zakres: 0 .. 255                        |
| <code>char</code>    | 1               | jeden znak w postaci kodu ASCII, np. 'A' → 65                                |
| <code>double</code>  | 4               | liczba zmiennoprzecinkowa, oznacza to samo co <code>float</code>             |
| <code>float</code>   | 4               | liczba zmiennoprzecinkowa, zakres: -3,4E+38 ... 3,4E+38                      |
| <code>int</code>     | 2               | liczba całkowita ze znakiem, zakres: -32.768 ... 32.767                      |
| <code>long</code>    | 4               | liczba całkowita ze znakiem, zakres:<br>-2.147.483.648 ... 2.147.483.647     |

## Arduino Uno - typy danych

| Nazwa                      | Rozmiar (bajty) | Uwagi   |
|----------------------------|-----------------|---|
| <code>short</code>         | 2               | liczba całkowita ze znakiem, zakres: -32.768 ... 32.767             |
| <code>size_t</code>        | -               | typ reprezentujący rozmiar obiektu w bajtach                        |
| <code>string</code>        | -               | łańcuch znaków jako tablica znaków, np. <code>char Str[10]</code> ; |
| <code>String()</code>      | -               | klasa reprezentująca łańcuchy znaków                                |
| <code>unsigned char</code> | 1               | 8-bitowa liczba całkowita bez znaku, zakres: 0 .. 255               |
| <code>unsigned int</code>  | 2               | 16-bitowa liczba całkowita bez znaku, zakres: 0 .. 65.535           |
| <code>unsigned long</code> | 4               | liczba całkowita bez znaku, zakres: 0 ... 4.294.967.295             |
| <code>void</code>          | -               | stosowany tylko w funkcjach, gdy nie zwraca ona wartości            |
| <code>word</code>          | 2               | 16-bitowa liczba całkowita bez znaku, zakres: 0 .. 65.535           |

Koniec wykładu nr 4

Dziękuję za uwagę!