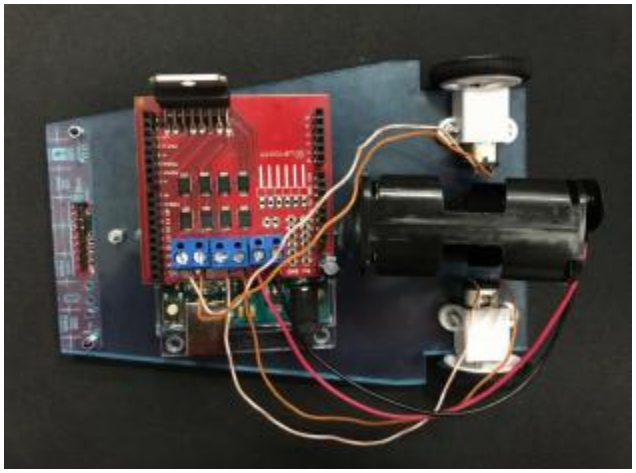
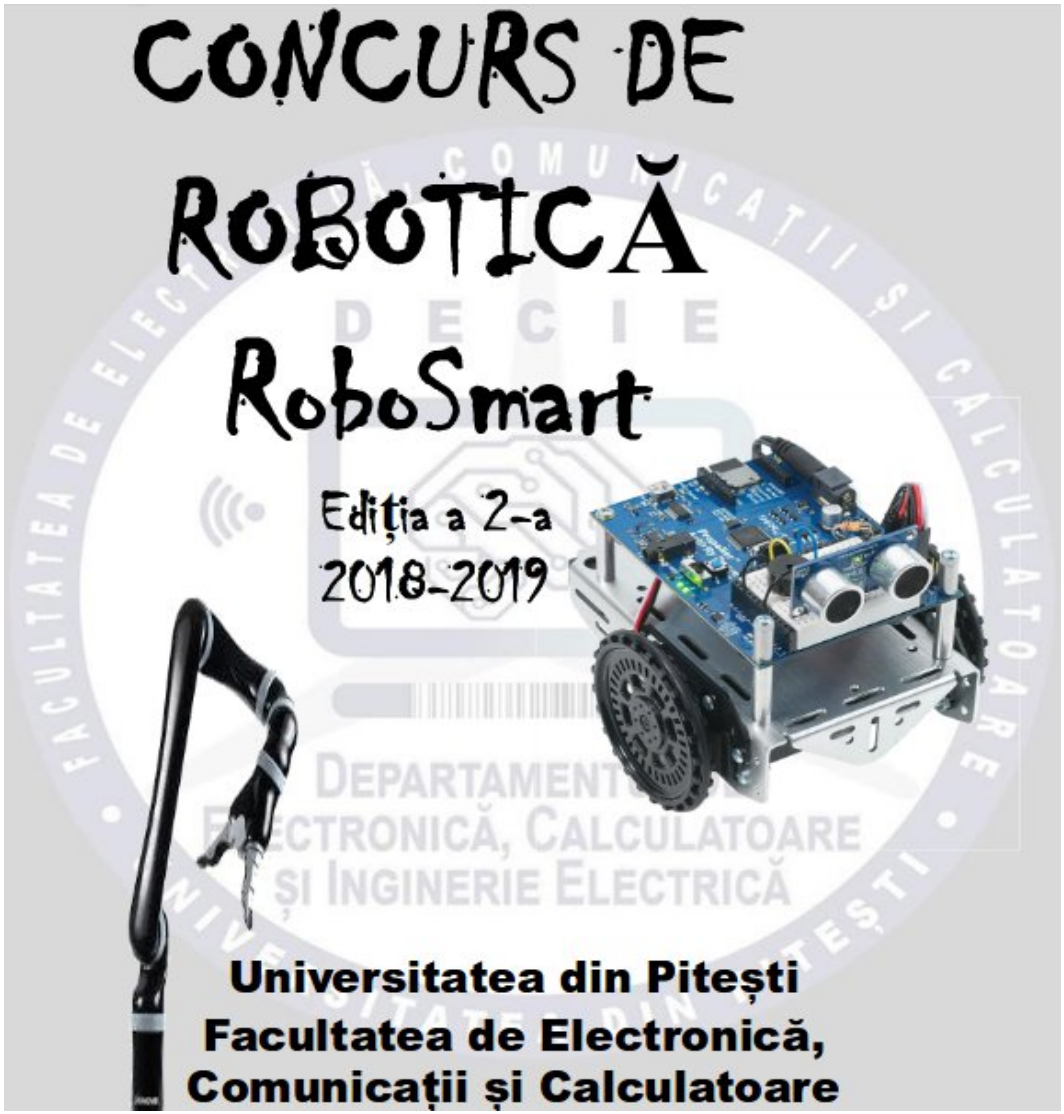
RoboSmart





Inspectoratul Școlar Județean Argeș

Sponsori
TEXTRON FLEET MANAGEMENT

Arduino – Programarea robotului

- Structura unui program Arduino
- Controlul pinilor machetei
- Comanda motoarelor
- Noţiuni avansate
- Portul serial

Prezintă:

Florin-Marian Bîrleanu

http://igotopia.ro

Structura unui program Arduino

Sketch = Structure + (Values + Functions)

- **Program** = *Instrucţiune1 Instrucţiune2* ...
- *Instrucţiune* = Definiţie de **funcţie**

  sau

  Definiţie de **constantă** sau **variabilă** (globală)
- Obligatoriu: Definiţia funcţiei setup()

  Definiţia funcţiei loop()

Structura unui program Arduino

- => Cel mai scurt program Arduino:

```
void setup() {
}

void loop() {
}
```

- Ce înseamnă **setup**() şi **loop**()?

```
void main() {
  setup();
  while (1) {
  loop();
  }
}
```

Structura unui program Arduino

- Funcţia **setup**():
  - E apelată **automat** la pornirea programului
  - (Adică atunci când se alimentează macheta sau când se apasă butonul de reset)
  - Trebuie puse acolo **iniţializări**, lucruri care vor rula o singură dată

Structura unui program Arduino

- Funcţia **loop**():
  - E apelată într-o buclă infinită
  - Aici se fac citiri de intrări, calcule, scrieri de ieşiri
  - Dar când se termină programul? (Niciodată!)
    - La “stingerea” plăcuţei.

Structura unui program Arduino

- Constante:

```
#define NUME valoare
const tip nume = valoare;
```

← :-(

← :-)

- Variabile: = globale

```
tip nume;
```

sau:

```
tip nume = valoare;
```

129, 0127, 0x12F, B01110101 (0b01110101)

Tipuri (noi): boolean, byte, word, String

Variabilele într-un program Arduino

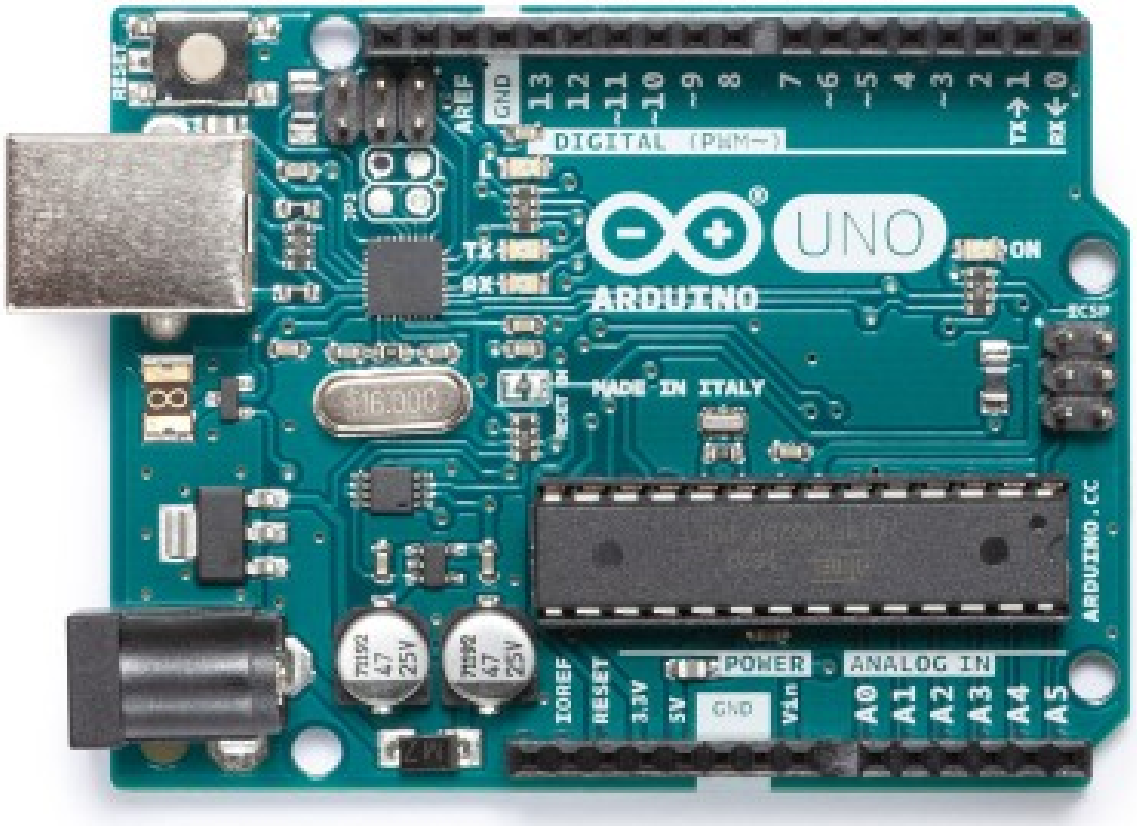
- Globale ← vizibile peste tot
- Locale ← vizibile în blocul lor

- Statice ← ca globale, pentru o funcţie
- Volatile ← cu acces concurent

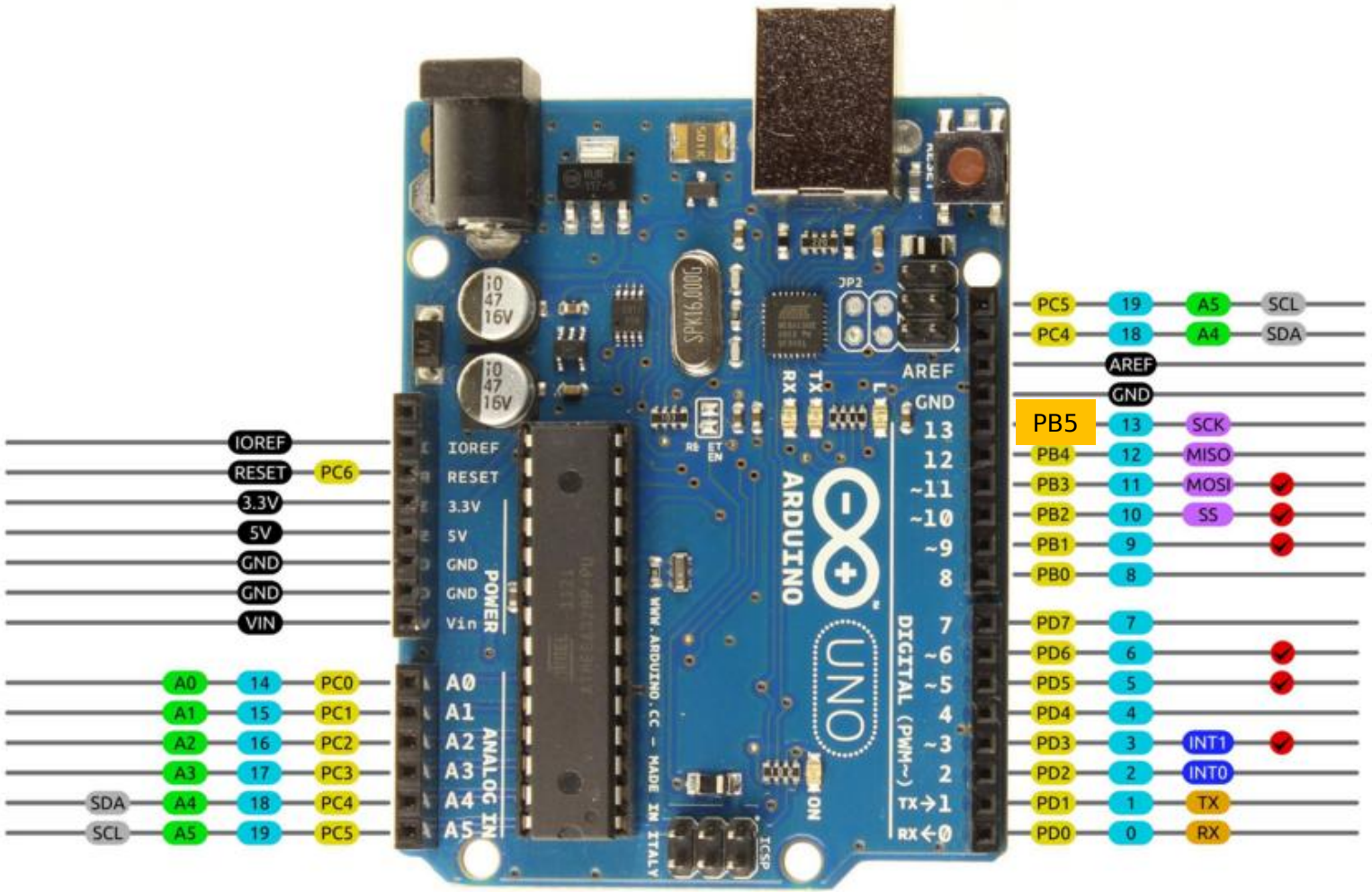
volatile *împiedică eventuale optimizări* pe care le-ar putea face compilatorul; *obligă recitirea valorii din memorie* la fiecare accesare a variabilei

Controlul pinilor în Arduino

- Pini digitali
- Pini analogi
- Pini PWM



Controlul pinilor în Arduino



Pinii digitali

- Intrări sau ieşiri digitale ← (0,1, 2, …, 13, A0, A1, …, A5)
- Valori posibile:
  0 (LOW), 1 (HIGH)

- Configurarea:
  pinMode(nr_pin, {INPUT, OUTPUT, INPUT_PULLUP});

- Scrierea: ← Aprinderea unui **LED**
  pinMode(nr_pin, OUTPUT);
  digitalWrite(nr_pin, HIGH);

- Citirea: ← Citirea unui **buton**
  pinMode(nr_pin, INPUT_PULLUP);
  {HIGH, LOW} = **digitalRead**(nr_pin);

Pinii analogici

- Intrări analogice ← (A0, A1, …, A5)
- Valori posibile:

  0 (0V), 1023 (5V)

  – ADC/10b => rezoluţia = 5V/1024
  – O citire durează 100 µs
  – **analogReference**({DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, EXTERNAL});

- Citirea: ← Citirea unei tensiuni

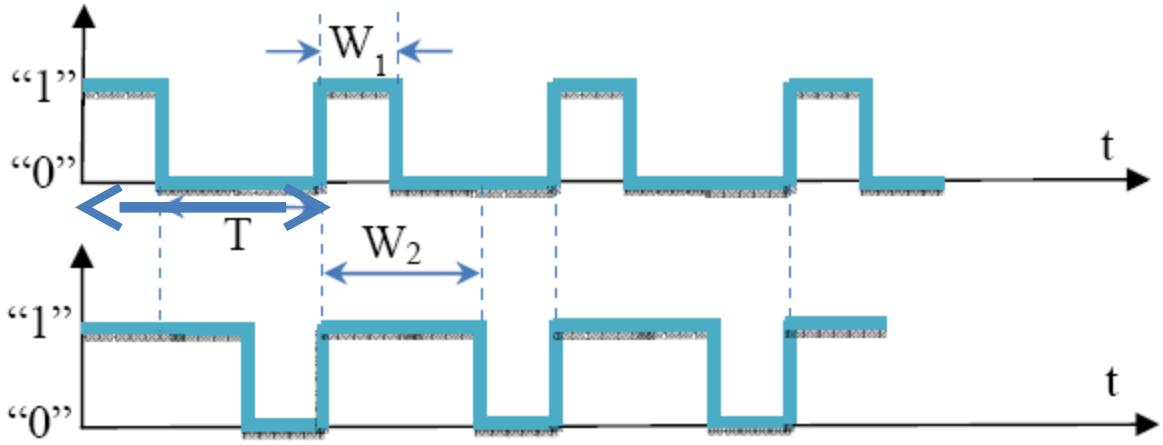
  int valoare = **analogRead**(nr_pin);

Pinii PWM

- Ieşiri “analogice” ← (3, 5, 6, 9, 10, 11)
- “Valori” posibile:

  0 (oprit, τ=0), 255 (pornit la maxim, τ=1)

$$\tau = \frac{W}{T}$$



- Scrierea:

  analogWrite(nr_pin, {0, 1, 2, 3, …, 255});

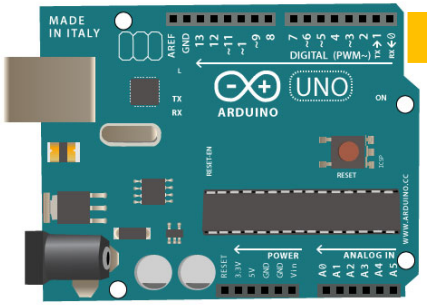
Comanda motoarelor

- Motor =
  - masină electrică ce converteşte energia electrică în energie mecanică
  - rotor + stator
- Motor de c.c.

Parametri: 6 V, 120 mA (max: 1.6 A)

Comanda motoarelor

- Ce e driver-ul de motor?



40 mA

2000 mA

Comanda motoarelor

- Sens rotaţie?
  - Înainte: MxA ← HIGH; MxB ← LOW
  - Înapoi: MxA ← LOW; MxB ← HIGH
  - La stânga: Md – înainte; Ms – oprit
    Sau: M_d – înainte; M_s – înapoi
  - La dreapta: Md – oprit; Ms – înainte
    Sau: M_d – înapoi; M_s – înainte

Comanda motoarelor

- Viteză rotaţie?
  - Oprit:

    MxA ← LOW; MxB ← LOW
  - Pornit (la maxim):

    MxA ← HIGH; MxB ← LOW
  - La v% din maxim:

    MxA ← PWM(v); MxB ← LOW

Notiuni avansate

- Controlul direct al pinilor
  - Operaţii pe biţi
- Lucrul cu întreruperi
  - Temporizări
- C pentru Arduino
  - Utilizarea memoriei de program pentru date

Controlul direct al pinilor

- Pro:
  - Mai rapid
  - Se pot controla simultan mai mulţi pini
  - Programul rezultat ocupă mai puţin
- Con:
  - Mai dificil de înţeles
  - Mai deschis către erori
  - Mai puţin portabil

Controlul direct al pinilor

- 3 **porturi**:
  - **B** (pinii 8-13)
  - **C** (pinii ADC)
  - **D** (pinii 0-7)

- Fiecare e controlat de către 3 **regiştri**:
  - DDRx – Port x Data Direction Register (R-W)
  - PORTx – Port x data register (R-W)
  - PINx – Port x Input pins register (R)

Controlul direct al pinilor

- Exemplu:

```
DDRD   = B11111110;
  ooooooooi          ← o – output; i – input

PORTD = B10101001;
  HLHLHLL?           ← H – high (1); L – low (0)
byte D  = PIND;

  76543210           ← numerotarea biţilor
```

Operaţii la nivel de bit

- Extragere biţi
- Setare biţi
- Resetare biţi
- Inversare biţi

Operaţii la nivel de bit

Extragere biţi (← bitRead)

```
  0 1 1 1 1 0 1 0   &
  0 0 0 0 1 0 0 0     ← masca
=> 0 0 0 0 1 0 0 0    ← rezultatul

  0 1 1 1 1 0 1 0   &
  0 0 0 0 0 1 0 0     == (1 << 2)
=> 0 0 0 0 0 0 0 0
```

Operaţii la nivel de bit

Setare biţi (← bitSet)

```
  0 1 1 1 1 0 1 0  |
  0 0 0 0 1 0 0 0    ← masca
=> 0 1 1 1 1 0 1 0     ← rezultatul

  0 1 1 1 1 0 1 0  |
  0 0 0 0 0 1 0 0    == (1 << 2)
=> 0 1 1 1 1 1 1 0
```

Operaţii la nivel de bit

Resetare biţi (← bitClear)

```
  0 1 1 1 1 0 1 0  &
  1 1 1 1 0 1 1 1     ← masca
=> 0 1 1 1 0 0 1 0    ← rezultatul

  0 1 1 1 1 0 1 0  &
  1 1 1 1 1 0 1 1 == ~00000100 == ~(1<<2)
=> 0 1 1 1 1 0 1 0
```

Operaţii la nivel de bit

Inversare biţi

```
  0 1 1 1 1 0 1 0   ^
  0 0 0 0 1 0 0 0       ← masca
=> 0 1 1 1 0 0 1 0      ← rezultatul

  0 1 1 1 1 0 1 0   ^
  0 0 0 0 0 1 0 0
=> 0 1 1 1 1 1 1 0
```

Lucrul cu întreruperi


```
while (true)
{
    if (! INTRERUPERE)
        ExecutaUrmatoareaInstructiune();
    else
        ExecutaISR();
}
```

Întrerupere la apăsarea unui buton

```
volatile byte v = 0;
void inverseaza() { v = 1-v; }
void setup() {
   pinMode(13, OUTPUT);
   pinMode(2, INPUT_PULLUP);
   attachInterrupt(digitalPinToInterrupt(2),
   inverseaza, FALLING);
}
void loop() {
   digitalWrite(13, v);
}
```

Notă:
delay() şi **millis()** nu merg aici.

Întrerupere la 1 Hz (temporizare)

- Se programează direct **timerele**
- => Nu or să mai meargă: millis(), delay(), analogWrite()

- Se lucrează direct cu **regiştrii**: TCCRxA, TCCRxB, TCNTx, OCRxA, TIMSKx

(Detalii în ATMEL ATmega328P datasheet…)

Întrerupere la 1 Hz (temporizare)

```
void setup() {
   pinMode(13, OUTPUT);
   cli(); // noInterrupts();
   // set timer 1 interrupt @ 1Hz:
   TCCR1A = 0;
   TCCR1B = 0;
   TCNT1 = 0; // init counter to 0.
   OCR1A = 15624; //16MHz/1024-1
   TCCR1B |= (1<<WGM12);
   TCCR1B |= (1<<CS12) |
(1<<CS10); // 1024 prescaler
   TIMSK1 |= (1<<OCIE1A);
   sei(); // interrupts();
}
void loop() {
   ... // whatever
}
```

```
boolean toggle1 = 0;

ISR (TIMER1_COMPA_vect) {
   toggle1 = 1 – toggle1;
   digitalWrite(13, toggle1);
}
```

TCCR0A |= (1<<WGM01); // pt. timer 0
TCCR2A |= (1<<WGM21); // pt. timer 2

Detalii despre aceşti registri: în datasheet-ul microcontrolerului

Temporizări (foarte) scurte

- delay(…);
- millis()
- delayMicroseconds(…); ~ 2 µs (2000 ns)
- __asm(“nop\n”); ~ 62.5 ns

C pentru Arduino

- Tipuri de date: byte, boolean, word, String, int, char, long, float
- Constante numerice:
  - predefinite: false, true, LOW, HIGH, INPUT, OUTPUT, INPUT_PULLUP, LED_BUILTIN
  - 123, B01110101, 013, 0xAB (eventual cu u, l sau ul la final)
- sizeof

  (Ex.: char str[] = “Arduino”; // sizeof(str) ← 8)
- goto etichetă

Salvarea constantelor în memoria de program

- #include <avr/pgmspace.h>
- const byte data[] PROGMEM = {1, 2, 3, 4, 5};
- pgm_read_byte_near(data+0);
- Serial.print(F("Lots of text ..."));

  (F = macro ce face ca textul acesta să fie salvat în memoria de program (FLASH))

Portul serial

- Iniţializare:

- Serial.begin({300, 600, 1200, 2400, 4800, 9600, ...});
- while (! Serial) ; // aşteaptă…

- Scriere:

- Serial.print(49); // ‘49’
- Serial.write(49); // ‘1’

- Citire:

- if (Serial.available()) { intVar = Serial.read(); }

Despre ce am vorbit:

- Care sunt **elementele unui program** Arduino
  (*constante*, *variabile*, *funcţii*)
- Cum se utilizează **pinii**: *digitali*, *analogici*, *PWM*
  (prin funcţii *vs* control direct (prin regiştri))
- Cum se comandă **motoarele**
  (sens, viteză, direcţie)
- Cum se fac **temporizări**
- şi comunicaţii **seriale**