

Introducere în programare. Aplicații (CDL 2023 – 2024)

Autor: prof. ing. Pocneț A. Daniel

Partea întâi: Limbajul Python

CUPRINS¹

1. Bibliografie

- 1.1. <https://docs.python.org/3/library/index.html> (*The Python Standard Library*)
- 1.2. <https://www.w3schools.com/python/default.asp>
- 1.3. <https://www.geeksforgeeks.org/python-programming-language/>
- 1.4. "Learn Python for beginners" (*Paul McWhorter, youtube.com*)
- 1.5. "Python Tutorial – Python Full Course for Beginners" (*ProgrammingwithMosh, youtube.com*)
- 1.6. "Python programming for everyone", "Intro To Tkinter for Python GUI Apps" (www.codemy.com, *tutorial python*)

2. Python. Instalare.

3. Tipuri de date.

4. Structuri de control.

5. Operatori.

6. Funcții definite de utilizator.

7. Lucrul cu fișiere.

8. Recursivitate.

9. Package ul tkinter. Dezvoltarea unei aplicații tip calculator windows.

¹ Partea întâi a modului "Introducere în programare. Aplicații" urmărește inițierea elevilor în abc ul programării. Pentru aceasta a fost ales limbajul Python. Partea întâi acoperă doar programarea structurată în Python. Aspectele cheie ale OOP (Object Oriented Programming) vor fi acoperite într-un CDL viitor în măsura în care va exista o cerere pentru un astfel de CDL.

2. Python. Instalare

Există mai multe variante de medii de dezvoltare (IDE) pentru scrierea programelor în Python, cele mai cunoscute fiind Visual Studio IDE respectiv Visual Studio Code. Cum procedăm pentru a crea mediul de dezvoltare Python? Există mai multe metode însă de regulă:

- Instalăm Python: <https://www.python.org/> , Downloads/Download for Windows, Python 3.12.2.
- Instalăm Visual Studio: <https://visualstudio.microsoft.com/downloads/> , Community/Free download.

După ce am terminat cu instalarea, mai departe trebuie să vedem ce alte package uri / module mai trebuiesc instalate² (*by default, Python are o serie bogată de module însă, în funcție de specificul aplicației, este uneori necesară instalarea unor module suplimentare*).

Exemple de **module / package uri** default:

- **math**. Modulul este necesar pentru utilizarea funcțiilor matematice “standard”.
- **random**. Modulul este necesar pentru utilizarea generatorului de numere pseudo aleatoare.
- **array**. Modulul este util pentru manipularea eficientă a tablourilor numerice.
- **os**. Modulul este util pentru manipularea fișierelor și directoarelor (*de exemplu, pentru a vedea dacă un fișier există pe disc, pentru ștergerea fișierului de pe disc, șamd, trebuie importat modulul os*).
- **tkinter**. Este interfața standard Python către toolkit ul Tk GUI. Acesta se importă astfel:

import tkinter sau **from tkinter import ***

- **turtle**. Permite crearea de imagini și forme pe o panză (canvas) virtuală.
- **tkMessageBox**. Permite afisarea de mesaje în căsuțe de mesaj (message box). Acesta se importă astfel:

from tkinter import *
from tkinter import messagebox

² https://www.youtube.com/watch?v=Ya_hSNxHeVo . Aici arată cum să instalezi package urile numpy, matplotlib, etc.

3. Tipuri de date

Înainte de a trece în revistă tipurile de date să vedem ce înseamnă **variabilă**.

Orice program (cod) implică firește și variabile. Exemple de variabile:

- simpla reiterare în buclă a unei porțiuni de cod implică existența unei variabile *index*,
- parcurgerea elementelor unei liste necesită o variabilă *index*,
- citirea datelor de la tastatură implică variabile,
- manipularea fișierelor (*citire / scriere*) necesită variabile,
- adăugarea / ștergerea de elemente la o listă implică variabile (*lista însăși este o variabilă*),
- într-o aplicație de tip windows, elementele de control (*butoane, câmpuri de introducere a datelor, etichete, șamd*) sunt tot variabile³.

Dar ce implică aceste variabile? Ce caracteristici au variabilele? Ce se întâmplă la rularea programului?

Variabilele au tip, domeniu de vizibilitate, durată de viață.

După tip, variabilele pot fi:

- **automatice**. Este cazul variabilelor locale funcțiilor. Acestea se alocă atunci când se începe execuția funcției. Zona de memorie alocată este eliberată (*variabila "dispare"*) la revenirea din apelul funcției (*la încheierea procesului de execuție al funcției*). Ca durată, variabilele locale (automatice) sunt temporare.
- **globale**. Sunt variabile create în afara tuturor funcțiilor. Ele pot fi modificate atât în afara funcțiilor din cod cât și în interiorul acestora (*pentru a modifica o variabilă globală în corpul unei funcții ea trebuie redeclarată ca global. Dacă doar utilizăm variabila globală dar nu o modificăm atunci nu este necesară redeclararea ca global în corpul funcției*). Variabilele globale există (*în memorie*) pe toată durata rulării programului; domeniul de vizibilitate este întreg codul.

Să vedem acum ce tipuri de date pot deține aceste variabile.

Tipuri de date:

- text (șir de caractere): **str**.
- numeric: **int, float, complex**.
- boolean: **bool**.
- pentru manipularea binară a datelor: **bytes, bytearray**.
- tablou: **list, array, tuple, dictionary**.

Exemplul 1

```
import random

# colectie este o lista. Lista poate contine elemente de tipuri
# diferite. Se pot adauga, sterge, modifica elementele listei.
```

³ Elementele respective sunt de fapt obiecte. Obiectul extinde noțiunea de variabilă el cuprinzând pe lângă date și metode (funcții).

```

colectie = []
# colectie_imuabila este o 'tuple'. Odata asigurate elementele
# unui 'tuple', ele nu pot fi sterse, adaugate, modificate. Dupa
# cum se observa, un 'tuple' poate contine elemente de tipuri diferite.
colectie_imuabila = ("Ana", "Anca", "Doina", "Dalida", "Larisa", "Maria",
                    "Elena", "Oana", 0.1, 0.01, 0.001, 0.0001, 10, 100,
                    1000, 10000)

# adauga un element la lista colectie. Remarcati redeclararea ca
# global a listei colectie. Acest lucru permite modificarea listei.
def add_item(item):

    global colectie
    colectie.append(item)

# sterge un element din colectie. Daca elementul exista, functia
# returneaza 1 altfel (elementul cautat nu exista), functia returneaza 0.
def del_item(item):

    global colectie
    if item in colectie:
        colectie.remove(item)
        return 1
    else:
        return 0

# returneaza elementul de index item_index din lista colectie:
def get_item(item_index):

    global colectie
    if item_index < len(colectie):
        return colectie.pop(item_index)

# printeaza pe monitor elementele listei primite ca argument.
# Spre deosebire de primele trei functii (care opereaza pe variabila
# globala colectie), functia print_collection() opereaza pe o
# copie a variabilei primite ca argument.
def print_collection(collection_name):

    for i in range(len(collection_name)):
        print(collection_name[i])

##### Programul propriu - zis #####

# populeaza lista cu 6 elemente alese intamplator din colectie_imuabila
# dar astfel incat toate cele 6 elemente sa fie diferite intre ele:
for i in range(6):

    exit_while = False
    while exit_while == False:
        num = random.randint(0, len(colectie_imuabila) - 1)
        if colectie_imuabila[num] not in colectie:
            add_item(colectie_imuabila[num])
            exit_while = True

```

```

# afiseaza elementele lui colectie:
print_collection(colectie)

# insereaza o linie noua la afisare:
print("")

# se incearca stergerea unui element particular din lista:
print("Acum vom sterge elementul 'Ana' din lista.")
if del_item("Ana") == 0:
    print("Ana nu se afla in lista !")
else:
    print("Ana a fost eliminata din lista !")
    print("Lista contine acum elementele: ")
    print_collection(colectie)

```

Exemplul 2

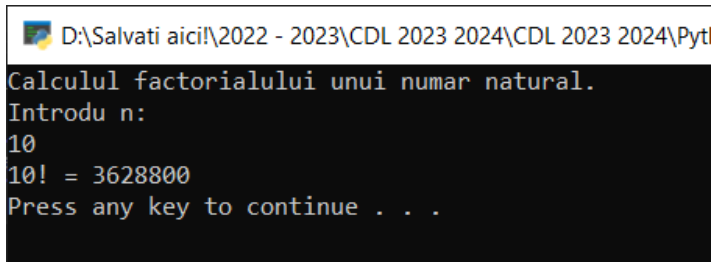
```

print("Calculul factorialului unui numar natural.")
print("Introdu n: ")
n = int(input())

def factorial(natural_number):
    if natural_number == 0:
        return 1
    else:
        return natural_number * factorial(natural_number - 1)

print(str(n) + "! = " + str(factorial(n)))

```



```

D:\Salvati aici!\2022 - 2023\CDL 2023 2024\CDL 2023 2024\Pyt
Calculul factorialului unui numar natural.
Introdu n:
10
10! = 3628800
Press any key to continue . . .

```

Remarcăm în Exemplul 2, forțarea lui `n` la șir de caractere cu ajutorul lui `str()`, înaintea concatenării cu șirul de caractere `"! = "`. Același lucru se întâmplă în cazul valorii întregi returnate de apelul `factorial(n)`.

4. Structuri de control

Acestea sunt structurile "standard" întâlnite în orice limbaj de programare. Ele permit parcurgerea unei structuri de tip tablou (*list, tuple, etc*), implementarea unei ramificații simple sau imbricate, etc.

4.1. Instrucțiunile **for**, **while**

For se utilizează de exemplu pentru parcurgerea tablourilor. Există mai multe feluri de a parcurge un tablou.

```
for i in range(n):
    instrucțiune 1
    .....
    instrucțiune k
```

Instrucțiunile din corpul for anterior (*care dintre acestea aparțin corpului se judecă după indentarea față de cuvântul cheie **for***) se vor executa de *n* ori; *i* va merge de la 0 la *n*-1.

```
for i in range(n, m):
    instrucțiune 1
    .....
    instrucțiune k
```

Acest for este similar cu cel dintâi cu mențiunea că de această dată, *i* merge de la *n* la *m*-1; instrucțiunile din corpul for se vor executa de (*m* - *n*) ori.

```
for i in reversed(range(n)):
    instrucțiune 1
    .....
    instrucțiune k
```

Acest for este similar cu **for i in range(n)** cu mențiunea că, de această dată, *i* merge de la *n*-1 la 0 (*decrementare*).

```
lista_sin = [0.0871, 0.1736, 0.2588, 0.3420, 0.4226, 0.5000, 0.5735, 0.6427, 0.7071,
             0.7660, 0.8191, 0.8660, 0.9063, 0.9396, 0.9659, 0.9848, 0.9961, 1.000]
for i in lista_sin:
    if i == 0.3420:
        print("sinus( " + str(30) + " este: " + str(i))
```

Instrucțiunile din corpul for vor fi reluate de atâtea ori câte elemente are lista *list_sin*. Interesant, variabila *i* de parcurgere a buclei for nu este **int** (*întreagă*) ci **float** (*virgulă mobilă*). În cazul,

```
lista_produce = ["ceas", "automobil", "casa", "barca"]
for i in lista_produce:
    if i == "automobil":
        print("Porsche 911 GT2 RS")
```

, variabila de parcurgere a buclei for va lua pe rând valorile "ceas", "automobil", "casa" respectiv "barca".

Instrucțiunea **while** se utilizează tot pentru parcurgeri repetitive ale unui grup de instrucțiuni. Spre deosebire de **for**, de data aceasta nu mai avem un contor al buclei care, atunci când ajunge la capăt va determina părăsirea buclei. În cazul **while**, instrucțiunile respective se vor relua (*reexecuta*) cât timp este îndeplinită o anumită condiție.

while condiție == **True**:

instrucțiune 1

.....

instrucțiune k

Instrucțiunile din corpul **while** se vor reexecuta cât timp condiție este **True**. Se va ieși din bucla **while** când condiție va deveni **False** (*firește, pentru aceasta trebuie ca, undeva în corpul while, "cineva" să facă acea condiție falsă*).

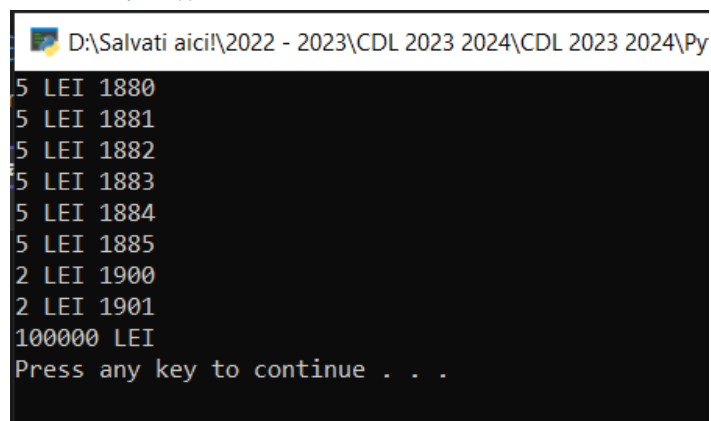
NOTĂ: dintr-o buclă **for** se poate ieși prematur (înainte ca indexul de parcurgere să "pointeze" în afara buclei) cu **break**. De asemenea, se poate "sări" (skip) iterația curentă cu **continue**.

Exemplul 1

```
# colectie este o 'tuple':
colectie = ('1 LEU 1870', '1 LEU 1876', '5 LEI 1880', '1 LEU 1881',
            '5 LEI 1881', '5 LEI 1882', '5 LEI 1883', '5 LEI 1884',
            '5 LEI 1885', '2 LEI 1900', '2 LEI 1901', '100000 LEI')
```

```
# afiseaza doar piesele care nu sunt de 1 LEU din colectie:
```

```
for i in colectie:
    if i[0:5] == '1 LEU':
        continue
    print(i)
```



```
D:\Salvati aici!\2022 - 2023\CDL 2023 2024\CDL 2023 2024\Py
5 LEI 1880
5 LEI 1881
5 LEI 1882
5 LEI 1883
5 LEI 1884
5 LEI 1885
2 LEI 1900
2 LEI 1901
100000 LEI
Press any key to continue . . .
```

Exemplul 2

```
# NOTA: code inspirat de pe
# https://www.geeksforgeeks.org/take-matrix-input-from-user-in-python/
rows = int(input("Introdu numarul de linii: "))
cols = int(input("Introdu numarul de coloane: "))

matrix = []
```



```

def build_matrix(n, m, mat):
    print('Introdu elementele matricii - Enter dupa fiecare element:')
    for i in range(n):
        crt_row = []
        for j in range(m):
            crt_row.append(int(input()))
        mat.append(crt_row)

def print_matrix(n, m, mat):

    print('Listeaza matricea: ')
    for i in range(n):
        for j in range(m):
            print(mat[i][j], end = " ")
        print()

build_matrix(rows, cols, matrix)
print_matrix(rows, cols, matrix)

```

```

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python39_64\python
Introdu numarul de linii: 3
Introdu numarul de coloane: 3
Introdu elementele matricii - Enter dupa fiecare element:
1
2
3
4
5
6
7
8
9
Listeaza matricea:
1 2 3
4 5 6
7 8 9
Press any key to continue . . . 

```

4.2. Instrucțiunile if ... else, if ... elif ... else

```

if expresie1:
    # corpul 1 de
    # instrucțiuni
else:
    # corpul 2 de
    # instrucțiuni

```

Corpul 1 de instrucțiuni se execută dacă expresie 1 este adevărată altfel, se execută corpul 2 de instrucțiuni.

```

if expresie1:

```

```

    # corpul 1 de
    # instrucțiuni
elif expresie 2:
    # corpul 2 de
    # instrucțiuni
.....
elif expresie n:
    # corpul n de
    # instrucțiuni
else:
    # corpul n+1 de
    # instrucțiuni

```

Dacă expresie 1 este adevărată, se execută corpul 1 de instrucțiuni. Altfel (expresie 1 este falsă), dacă expresie 2 este adevărată, se execută corpul 2 de instrucțiuni. Altfel (expresie 2 este falsă), dacă expresie 3 este adevărată, se execută corpul 3 de instrucțiuni. Șamd. Dacă toate cele n expresii sunt false, se execută corpul (n+1) de instrucțiuni. De menționat că, "ramura" else poate lipsi!

NOTĂ: structura if ... elif ... else implementează o prioritizare. Astfel, dacă mai multe expresii sunt adevărate, se va executa corpul de instrucțiuni asociat primei expresii care este adevărată.

Fișă de lucru.

Scrieți un program care rezolvă ecuația de gradul al doilea. Coeficienții ecuației se citesc de la tastatură.

```

import math

print("Acest program rezolva ecuatia de gradul al doilea. ")
print("Rezolvati ecuatia sau parasiti aplicatia? (y/n) ")

rezolva = input()
purcede = False
while purcede == False:
    if rezolva == 'y' or rezolva == 'n':
        purcede = True
    else:
        rezolva = input()

if rezolva == 'y':
    while True:
        print("Introdu coeficientii ecuatiei de gradul al doilea ")
        print("in ordinea a, b, c - dupa fiecare coeficient se da enter ")

        a = float(input())
        b = float(input())
        c = float(input())

        delta = b*b - 4*a*c
        if delta < 0:

```

```

print("ecuatia are radacini complex conjugate, ")
Re = -b/(2*a)
Im1 = math.sqrt(-delta) / (2*a)
Im2 = -Im1
print("x1 = ", end="")
print(Re, end="")
print(" +j*", end="")
print(Im1, end="")
print(" x2 = ", end="")
print(Re, end="")
print(" +j*", end="")
print(Im2)
else:
    print("ecuatia are radacini reale, ")
    x1 = (-b + math.sqrt(delta))/(2*a)
    x2 = (-b - math.sqrt(delta))/(2*a)
    print("x1 = ", end="")
    print(x1, end="")
    print(" x2 = ", end="")
    print(x2)

print("Doriti reluarea programului? (y/n) ")
rezolva = input()
purcede = False
while purcede == False:
    if rezolva == 'y' or rezolva == 'n':
        purcede = True
    else:
        rezolva = input()
if rezolva == 'n':
    break

```

Remarcăm:

- Părăsirea forțată a buclei while (cu **break**) atunci când nu se mai dorește reluarea programului cu alți coeficienți.
- Conversia datei introduse de utilizator la **float**, **int**, etc: `a = float(input())`
- Bibliotecile se includ cu **import**: `import math`
- Utilizarea funcțiilor din biblioteca **math**: `Im1 = math.sqrt(-delta) / (2*a)`
- Mai multe date printate pe același rând:

```

Re = -b/(2*a)
Im1 = math.sqrt(-delta) / (2*a)
Im2 = -Im1
print("x1 = ", end="")
print(Re, end="")
print(" +j*", end="")
print(Im1, end="")
print(" x2 = ", end="")
print(Re, end="")
print(" +j*", end="")
print(Im2)

```

```
PythonApplication2.py ×
D:\Salvati aici!\2022 - 2023\CDL 2023 2024\CDL 2023 2024\Python files\PythonApplication2\PyApp2VirtualEnvironment
Acest program rezolva ecuatia de gradul al doilea.
Rezolvati ecuatia sau parasiti aplicatia? (y/n)
y
Introdu coeficientii ecuatiei de gradul al doilea
in ordinea a, b, c - dupa fiecare coeficient se da enter
1
-2
1
ecuatia are radacini reale,
x1 = 1.0 x2 = 1.0
Doriti reluarea programului? (y/n)
y
Introdu coeficientii ecuatiei de gradul al doilea
in ordinea a, b, c - dupa fiecare coeficient se da enter
2
2
-4
ecuatia are radacini reale,
x1 = 1.0 x2 = -2.0
Doriti reluarea programului? (y/n)
y
Introdu coeficientii ecuatiei de gradul al doilea
in ordinea a, b, c - dupa fiecare coeficient se da enter
1
2
3
ecuatia are radacini complex conjugate,
x1 = -1.0 +j*1.4142135623730951 x2 = -1.0 +j*-1.4142135623730951
Doriti reluarea programului? (y/n)
n
Press any key to continue . . .
```

5. Operatori

NOTĂ: consultați https://www.w3schools.com/python/python_operators.asp . Desprindeți prioritățile operatorilor și modul lor de asociere.

6. Funcții definite de utilizator

Am văzut deja în capitolele anterioare cum se introduc funcțiile definite de utilizator. Sintaxa:

```
def nume_functie (argumentele functiei separate prin virgula):
    global var_1          # in cazul in care functia modifica
    .....                # valorile unor variabile globale, acestea
    global var_k          # se redeclara cu global.
    # corpul functiei
    # NOTA: daca functia trebuie sa returneze ceva (sir de caractere, data
    #        numerica, etc) atunci se foloseste cuvantul return.
```

Fișă de lucru.

Următorul cod (*incomplet*) trebuie să rezolve un sistem de ecuații liniare compatibil determinat (*Cramer*), de ordinul 3. Pentru aceasta, trebuie introdusă de la tastatură matricea sistemului. Completați codul astfel încât acesta să realizeze taskul precizat.

```
system_matrix = []
free_term_vector = []

def build_matrix(A, n, m):

    for i in range(n):
        crt_row = []
        for j in range(m):
            crt_row.append(int(input()))
        A.append(crt_row)

def copy_matrix(A, B, n, m):

    # scrie corpul functiei (copiază matricea A în matricea B;
    # functia nu returnează nimic).
    # Atentie: lista B trebuie stearsă mai întâi!

def print_system(A, b):

    # scrie corpul functiei (afisează sistemul A * x = b)

def compute_det(A):

    return A[0][0] * A[1][1] * A[2][2] + A[0][2] * A[1][0] * A[2][1] + \
        A[2][0] * A[0][1] * A[1][2] \
        - A[2][0] * A[1][1] * A[0][2] - A[0][0] * A[2][1] * A[1][2] \
        - A[2][2] * A[1][0] * A[0][1]

def replace_column(A, column, b):

    # scrie corpul functiei (functia înlocuiește coloana
    # de index column din matricea A cu vectorul coloana b și
    # returnează matricea obținută).
```

```

##### Programul propriu - zis #####

print('Introdu elementele matricii - Enter dupa fiecare element:')
build_matrix(system_matrix, 3, 3)

print('Introdu elementele termenului liber - Enter dupa fiecare element:')
build_matrix(free_term_vector, 3, 1)

print('Sistemul dat: ')
print("")
print_system(system_matrix, free_term_vector)

det_A = compute_det(system_matrix)
if det_A != 0:
    print('Are solutie: ')
    system_matrix_copy = []
    for i in range(3):
        copy_matrix(system_matrix, system_matrix_copy, 3, 3)
        print(compute_det(replace_column(system_matrix_copy, i,
            free_term_vector)) / det_A)
else:
    print('Sistem incompatibil!')

```

Rezolvare:

```

def copy_matrix(A, B, n, m):

    B.clear()
    for i in range(n):
        temp = []
        for j in range(m):
            temp.append(A[i][j])
        B.append(temp)

def print_system(A, b):

    for i in range(3):
        print(' ' + str(A[i][0]) + ' * x + ' + str(A[i][1]) + ' * y + ' \
            + str(A[i][2]) + ' * z = ' + str(b[i][0]))

def replace_column(A, column, b):

    for i in range(3):
        A[i][column] = b[i][0]

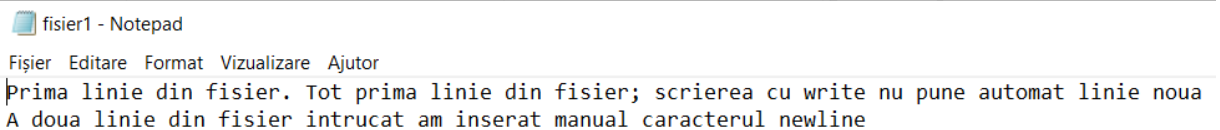
    return A

```

7. Lucrul cu fișiere

Următorul cod crează fișierul “fisier1.txt” și îl deschide pentru scriere. Dacă fișierul respectiv există deja, ceea ce este în el se va suprascrie (*fișierul este deschis cu ‘w’*):

```
file1 = open('D:\\Salvati aici!\\2022 - 2023\\CDL 2023 2024\\  
            CDL 2023 2024\\Python docs\\fisier1.txt', 'w')  
file1.write('Prima linie din fisier.')  
file1.write(' Tot prima linie din fisier; scrierea cu write nu pune  
            automat linie noua')  
file1.write("\n")  
file1.write('A doua linie din fisier intrucat am inserat manual  
            caracterul newline')  
file1.close()
```



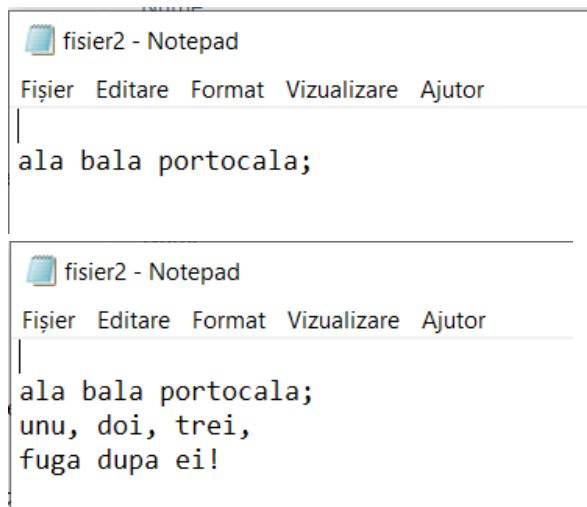
Prima linie din fisier. Tot prima linie din fisier; scrierea cu write nu pune automat linie noua
A doua linie din fisier intrucat am inserat manual caracterul newline

Următorul cod șterge fișierul creat anterior (fisier1.txt); remarcați importul modulului **os**:

```
import os  
if os.path.exists('D:\\Salvati aici!\\2022 - 2023\\CDL 2023 2024\\  
                CDL 2023 2024\\Python docs\\fisier1.txt'):  
    os.remove('D:\\Salvati aici!\\2022 - 2023\\CDL 2023 2024\\  
            CDL 2023 2024\\Python docs\\fisier1.txt')  
else:  
    print('File does not exist !')
```

Următorul cod crează fișierul “fisier2.txt” în modul de adăugare în fișier. La fiecare execuție a programului se va adăuga în fișier textul introdus de utilizator.

```
file1 = open('D:\\Salvati aici!\\2022 - 2023\\CDL 2023 2024\\CDL 2023 2024\\Python docs\\fisier2.txt', 'a')  
  
print('Introduceti textul dorit; incheiati introducand END.')  
loop_again = True  
input_str = ""  
  
while loop_again == True:  
    input_str = str(input())  
    file1.write("\n")  
    if input_str[len(input_str) - 3 : len(input_str)] == "END":  
        file1.write(input_str[0 : len(input_str) - 3])  
        file1.close()  
        loop_again = False  
    else:  
        file1.write(input_str)
```

Fișă de lucru

Scrieți un cod care deschide fișierul `fisier3.txt` și caută în acesta un model oarecare (*șir de caractere*), de exemplu introdus de la tastatură. Programul va crea fișierul `fisier4.txt` în care va scrie toate liniile din `fisier3.txt` care conțin modelul căutat.

8. Recursivitate

Am văzut deja un exemplu de funcție recursivă în exemplul în care calculăm factorialul unui număr natural.

Fișa de lucru

Scrieți un program care citește un șir de caractere de la tastatură și îl inversează. De exemplu, șirul 'notar' va genera șirul 'raton', șirul '230 V 50 Hz' va genera șirul 'zH 05 V 032', șamd.

Rezolvare:

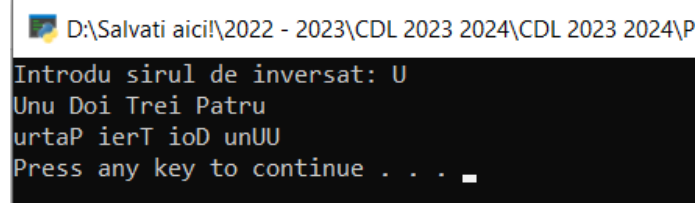
```
def revert_string(string):  
  
    if len(string) == 0:  
        return string  
    else:  
        return revert_string(string[1:]) + string[0]
```

```
print('Introdu sirul de inversat: ')  
input_string = str(input())  
print(revert_string(input_string))
```

O altă metodă constă în inversarea capetelor șirului până când ajungem la șirul de lungime zero:

```
def revert_string(string):  
  
    if len(string) == 0 or len(string) == 1:  
        return string  
    elif len(string) == 2:  
        return string[len(string) - 1] + string[0]  
    else:  
        return string[len(string) - 1] +  
            revert_string(string[1:len(string) - 1]) +  
            string[0]
```

```
print('Introdu sirul de inversat: ')  
input_string = str(input())  
print(revert_string(input_string))
```



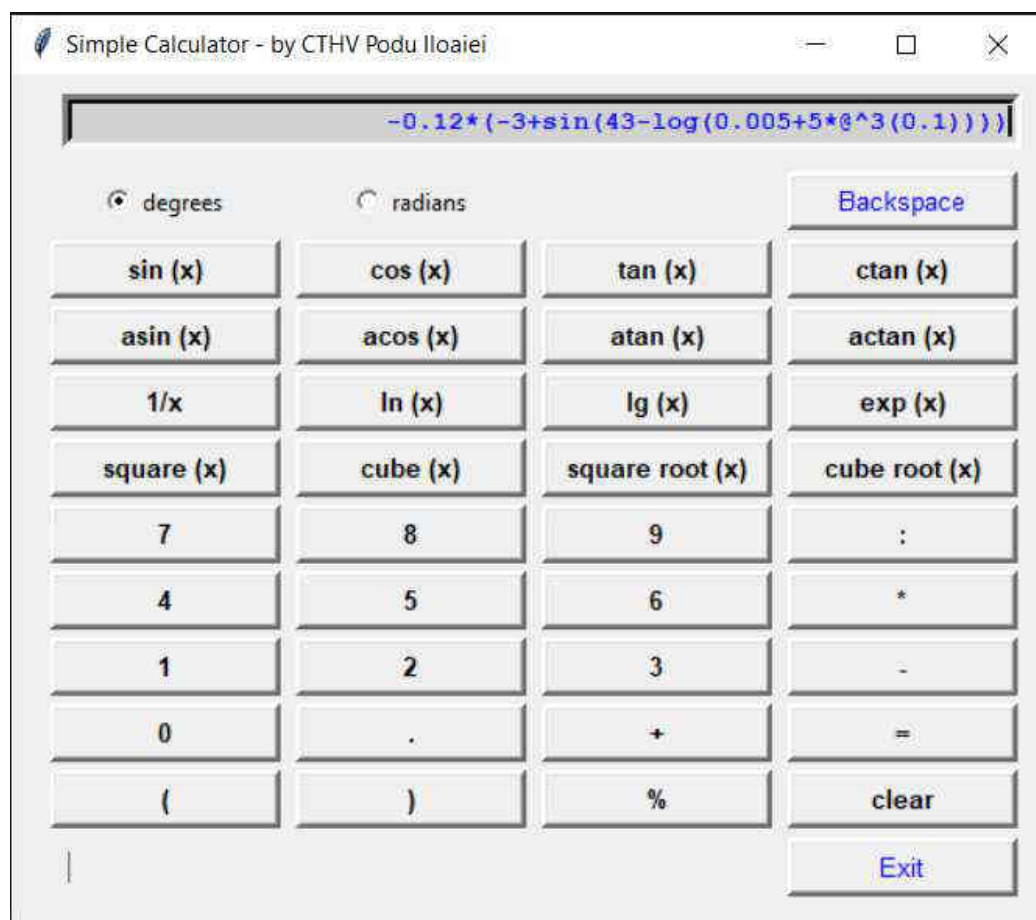
```
D:\Salvati aici!\2022 - 2023\CDL 2023 2024\CDL 2023 2024\P  
Introdu sirul de inversat: U  
Unu Doi Trei Patru  
urtaP ierT ioD unUU  
Press any key to continue . . .
```

9. Package ul tkinter. Dezvoltarea unei aplicații de tip calculator windows

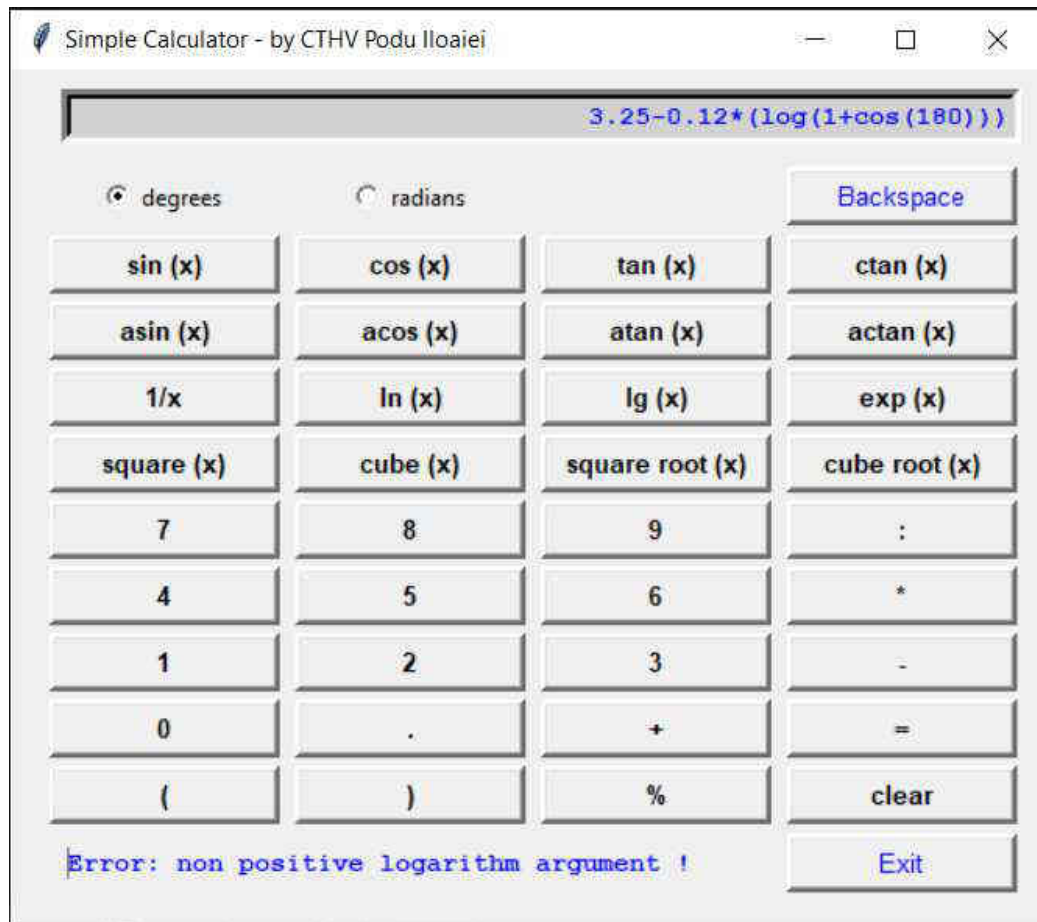
Acest package este librăria Python standard utilizată pentru crearea de interfețe grafice (GUI) pentru aplicații tip desktop.

Proiect:

Realizați o aplicație similară calculatorului din Windows. Aplicația va permite introducerea datelor (*operanzi, operatori, operatori funcții, paranteze*) atât de la butoane cât și prin scrierea directă a expresiei de calculat în câmp de afișare a rezultatului. Aplicația evaluează expresia introdusă și afișează rezultatul atunci când fie se dă click pe butonul "=" fie se apasă tasta Enter (*de la tastatura PC ului*) – în acest caz, în câmpul de introducere a datelor / afișare, ultimul caracter introdus trebuie să fie caracterul "=").



Aplicația va trebui să afișeze și diverse mesaje de eroare atunci când este cazul; afișarea acestor mesaje se va face într-un câmp separat, plasat în partea inferioară a ferestrei aplicației:



Partea a doua: **Programarea platformelor cu microcontroller**

Resurse minime pentru desfășurarea activităților aferente⁴:

- laptop uri,
- aplicația Arduino IDE plus librăriile corespunzătoare (free),
- cateva platforme Arduino, senzori, relee, memorii micro SD, şamd⁵.

Buget de timp alocat:

- partea a doua se va întinde de-a lungul a 12 săptămâni, 2h/săptămână.

⁴ Elevii care urmeaza CDL ul trebuie să cunoască limba engleză cel puțin la un nivel satisfăcător (cerință obligatorie). De asemenea, elevii în cauză trebuie să aibă cunostințe minimale de electronică analogică, măsurări, electronică digitală.

⁵ Pentru detalii vezi fisierul "Deviz materiale CDL partea a II-a.docx".

CUPRINS

1. Bibliografie

1.1. Paul McWhorter, "Arduino lessons" ([youtube.com](https://www.youtube.com)).

1.2. Arduino - HOME (<https://www.arduino.cc>).

2. Prezentarea platformei Arduino și a aplicației de programare Arduino IDE.

3. Introducerea datelor de la tastatură; afisarea datelor pe Serial Monitor.

4. Citirea /scrierea datelor de pe / pe un card de memorie.

5. Întreruperi interne de la timere. Aplicație: realizarea unui ceas electronic cu afisor LCD1602.

6. Cresterea frecvenței PWM ului Arduino prin modificarea registrilor timerului aferent.

7. Transceiverul NRF24L01. Aplicație: acționarea unui releu de la distanță, utilizand module NRF24L01.

8. Transceiverul HC05 (bluetooth).

9. Accelerometrul GY-ADXL337. Aplicație: citirea analogică a unghiului plan.

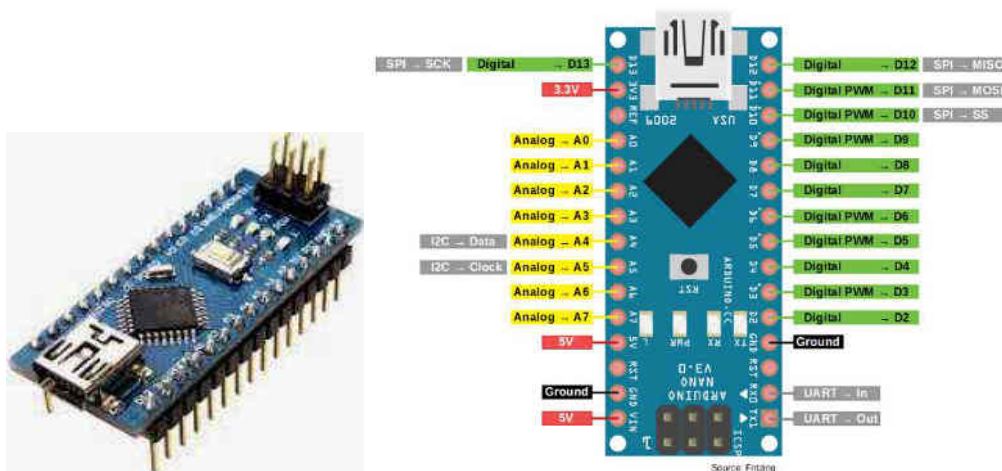
2. Platforma Arduino. Aplicația IDE⁶

2.1. Generalități

Arduino este o platformă cu microcontroler ce se utilizează în robotică, electronică, automatizarea proceselor. Parametri:

- microprocesor pe 8 biți, 16 MHz.
- 2 KB memorie de date, 32 KB memorie program.
- interfețe UART (RX, TX), SPI (SDA, SCL) și I²C (SS, MOSI, MISO, SCK)⁷.
- 14 pini digitali de I/O (D0 – D13)⁸. Dintre acestia, pinii D2 și D3 pot fi utilizați drept linii de întrerupere externă.
- 8 pini analogici de I/O (A0 – A7)⁹. Pinii analogici au atașați câte un convertor analog – numeric pe 10 biți (*necesar la citirea pinilor analogici*).
- cativa pini cu iesire PWM¹⁰.
- un pin de ieșire de 3.3V respectiv un pin de ieșire de 5V.
- mufa micro – USB tip B necesară interfațării cu calculatorul pentru programarea MCU¹¹,
- alimentare externă prin pinul VIN (7 – 14 V).

Curentul maxim ce poate fi furnizat de placile ARduino este de 40 mA / pin de ieșire dar nu mai mult de 200 mA în total (*toți pinii de ieșire utilizați*). Ca regulă generală, este bine să nu depășim 20 - 30 mA / pin de ieșire.



⁶ Surse: <https://www.theengineeringprojects.com/2015/43/arduino.html> , <https://www.arduino.cc> .

⁷ Diverse module, senzori, au astfel de interfețe seriale prin intermediul cărora pot fi configurați respectiv se pot accesa datele. De exemplu, modulul HC05 Bluetooth are interfața UART, senzorul MEMS MPU9250 are interfața SPI, samd.

⁸ Sensul pinilor se configurează la programarea microcontrolerului.

⁹ Vezi nota 4.

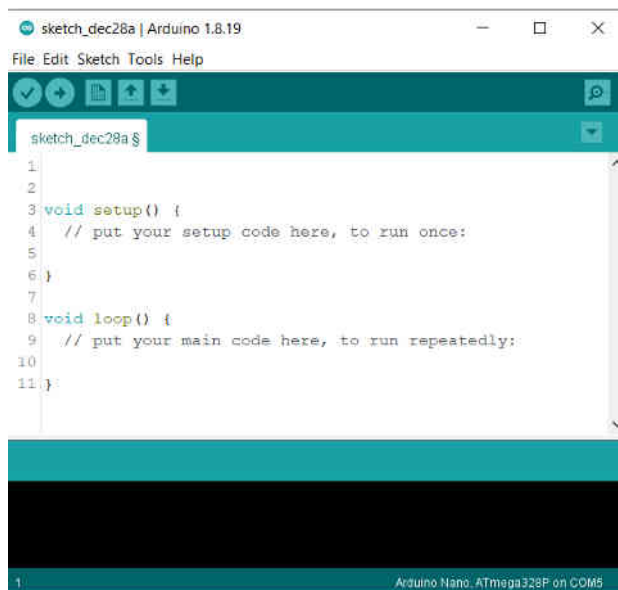
¹⁰ Frecvența PWM (default) pentru pinii PWM Arduino este de 490 / 980 Hz. Pentru majoritatea aplicațiilor, aceasta valoare este suficient de mare. Există și situații când este nevoie de ieșiri PWM de frecvență mai mare (KHz – zeci KHz); în acest caz, se poate crește valoarea frecvenței PWM prin scrierea unor regiștri aferenți Timer ului 1.

¹¹ Toate platformele Arduino se pot alimenta cu energie prin mufa micro – USB (dacă nu se dispune de alimentare externă prin pinul VIN).

De menționat că există o gamă largă de platforme Arduino: nano, uno, mega, due, șamd. Dintre acestea, nano, UNO și MEGA sunt echipate cu MCU pe 8 biți, 18 MHz; diferă între ele doar memoria, numărul de pini respectiv de linii UART (MEGA are 256 KB memorie program, 8 kB memorie de date, mai mulți pini respectiv 4 canale UART).

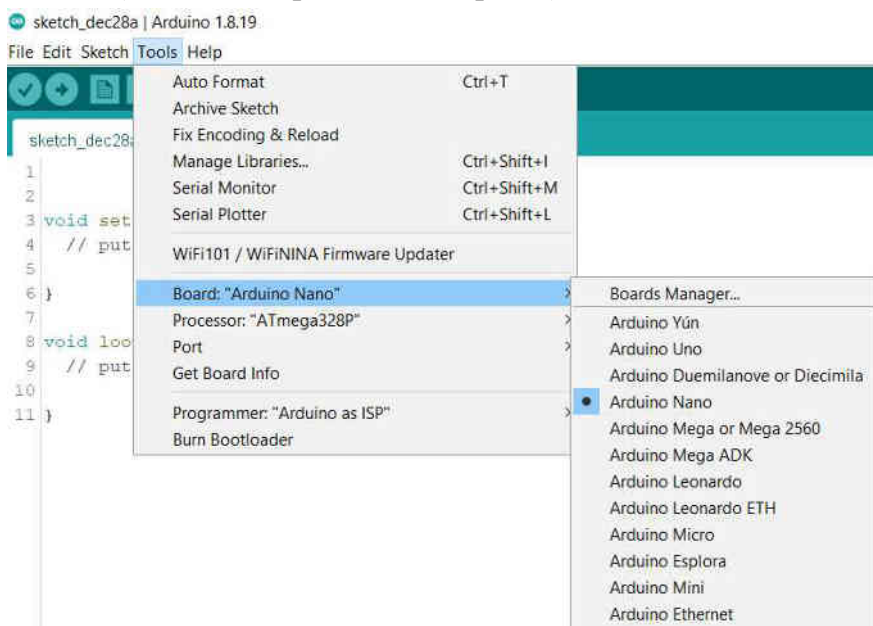
2.2. Mediul Arduino IDE. Structura programelor pentru Arduino. Elemente de limbaj

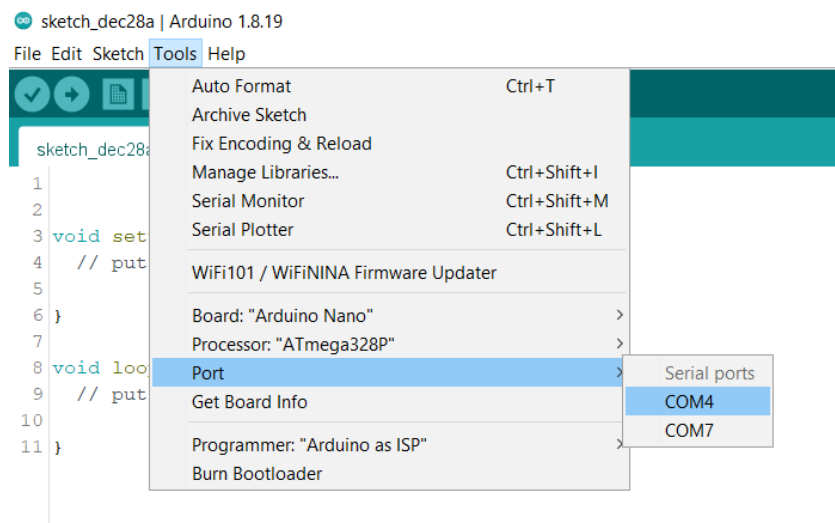
Fereastra aplicației Arduino IDE arată astfel:



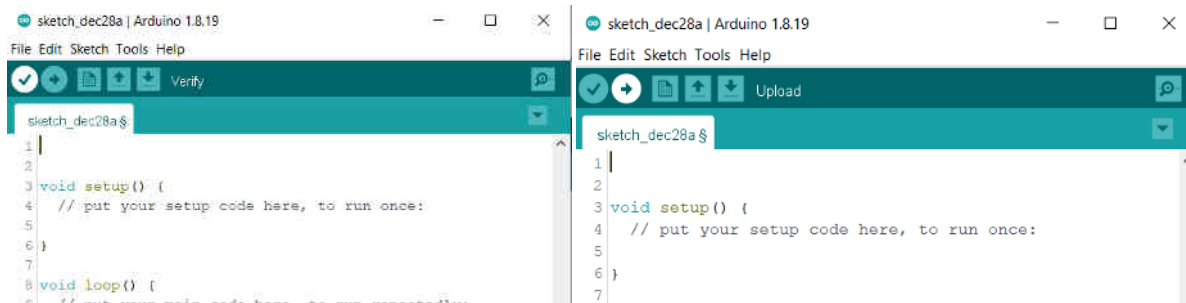
Elementele importante din fereastra principală sunt:

- **meniul Tools.** De aici selectăm tipul de placă, tipul de procesor cu care este echipată respectiv portul USB la care conectăm cablul de downloadare a programului (în memoria procesorului plăcii).





- **butoanele Verify, Upload.** Un click pe Verify va compila programul detectand eventualele erori. Pentru încărcarea programului în memoria procesorului plăcii se va da click pe Upload.

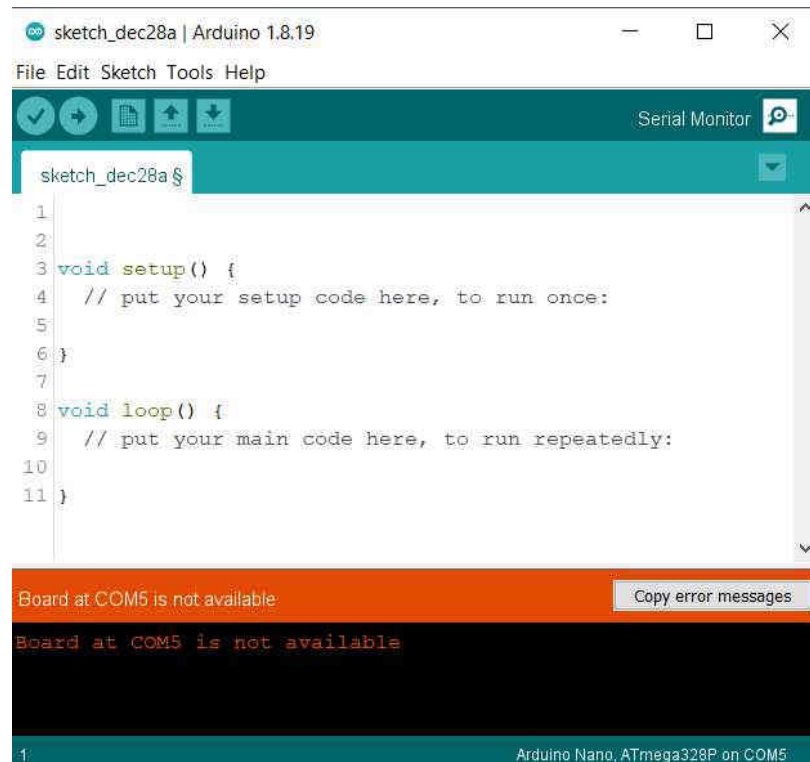


- **Serial Monitor.** Este o interfață serială prin intermediul căreia putem interacționa cu programul Arduino care rulează¹²: se pot afișa date din programul executat (*mesaje text, valori ale diverselor variabile de interes*), se pot introduce date de la tastatură. Practic, cu Serial Monitor se poate controla, între anumite limite, execuția programului Arduino care rulează.

***NOTĂ:** pentru ca Serial Monitor să poată fi utilizat în timpul rulării programului trebuie ca viteza de comunicație serială setată din Serial Monitor să aibă aceeași valoare ca viteza de comunicație serială setată prin program.*

În poza următoare se poate vedea mesajul de eroare afișat la deschiderea Serial Monitor atunci când nu există nici o placă Arduino conectată la portul serial curent:

¹² Există anumite restricții în ce privește schimbul de date cu Serial Monitor. De exemplu, nu pot afișa date în Serial Monitor, din interiorul unei rutine de tratare a întreruperilor.



- **spațiul de lucru.** Este fereastra în care scriem programul Arduino.
- **fereastra de mesaje** (*fundal negru*).

Structura programelor pentru Arduino

Un program Arduino se compune din 3 părți:

- **antetul.** Este porțiunea de dinaintea funcției **setup()**. Aici vom scrie incluziunile bibliotecilor utilizate de program, vom declara pini, variabile program respectiv vom realiza instanțieri ale diverselor module pe care programul Arduino le utilizează. Tot în "antet" utilizatorul poate scrie propriile funcții pe care să le utilizeze (*apeleze*) în program.

```

4
5 #include <TimerOne.h>
6 #include <Wire.h> // for working with I2C.
7 #include <LiquidCrystal_I2C.h>
8
9 #define ALARM_PIN 6 // PWM pin. Enable for this pin is the
10 // alarm_ringing variable.
11
12 LiquidCrystal_I2C lcd1602_i2c(0x27, 16, 2); // The address of I2C is 0x27 by default
13 // (for instance if A0, A1, A2 are not soldered)
14 // LCD 2 rows, 16 columns.
15 // How do I know the address is 0x27?
16 long interrupt_period = 1000000; // Interrupt triggers every second.
17 bool tick;
18
19 byte hours_cnt;
20 byte minutes_cnt;
21 byte seconds_cnt;
22
23 byte alarm_hours;
24 byte alarm_minutes;
25 byte alarm_cnt; // the alarm will ring 15 seconds then it'll
26 bool alarm_ringing;
27
28 bool set_time;
29 bool set_alarm;
30 bool alarm_ON;
31 bool alarm_rang; // prevent alarm firing second time.
32 bool setting_tm_ala;
33
34 void setup() {
35
36 Serial.begin(115200);

```

În exemplul din figura de mai sus “antetul” conține:

- incluziunea a trei librării – TimerOne.h (*pentru lucrul cu întreruperi*), Wire.h (*pentru lucrul cu interfața I²C*) respectiv LiquidCrystal_I2C.h (*pentru lucrul cu afisorul LCD1602 / 2004*).
- declararea pinului ALARM_PIN ca fiind pinul digital D6.
- instanțierea unui obiect LCD1602.
- variabile program.
- **funcția loop()**. Aceasta se execută o singură dată, la începutul rularii programului. Din acest motiv, în corpul acestei funcții se fac inițializări, configurări ale unor regiștri, se montează întreruperi, șamd.

```

33
34 void setup() {
35
36     Serial.begin(115200);
37
38     analogWrite(ALARM_PIN, 0); // PWM with 0% duty cycle - that's buzzer off.
39
40     lcd1602_i2c.init();
41     lcd1602_i2c.backlight();
42
43     // display double dots on LCD (always in the same position):
44     lcd1602_i2c.setCursor(2, 0);
45     lcd1602_i2c.print(":");
46     lcd1602_i2c.setCursor(5, 0);
47     lcd1602_i2c.print(":");
48     lcd1602_i2c.setCursor(5, 1);
49     lcd1602_i2c.print(":");
50     lcd1602_i2c.setCursor(9, 1);
51     lcd1602_i2c.print(":");
52
53     // initialize time:
54     hours_cnt = 0;
55     minutes_cnt = 0;
56     seconds_cnt = 0;
57     alarm_cnt = 0;
58
59     displayHours(0, 0, 0); // start column, row, value.
60     displayMinutes(3, 0, 0);
61     displaySeconds(6, 0, 0);
62
63     displayAlarm(0, 1, 0, 0, false); // starting from 1st column, second row,
64                                     // alarm hour, alarm OFF.
65     tick = false;
66     set_time = false;
67     set_alarm = false;
68     alarm_ON = false;
69     alarm_rang = false;
70     alarm_ringing = false;
71     setting_tm_ala = false;
72
73     Timer1.initialize(interrupt_period);
74     Timer1.attachInterrupt(Timer1_ISR);
75 }

```

În exemplul anterior, în funcția `loop()` se setează viteza de comunicație serială la 115200 baud, se pune pe '0' pinul de ieșire care pilotează alarma (*un buzzer*), se inițializează LCD-ul, șamd.

- **funcția `main()`.** Tot ce punem în corpul acestei funcții se execută la nesfârșit.

```

void loop(){
    // instrucțiune 1;
    // instrucțiune 2;

```

```

.....
// instrucțiune n;
}

```

Execuția funcției `loop()` de mai sus înseamnă executia lui instrucțiune 1, instrucțiune 2, ..., instrucțiune n, instrucțiune 1, instrucțiune 2, ..., instrucțiune n, instrucțiune 1, samd (ciclul se repetă la nesfarsit).

Variabile locale, variabile globale

Daca o variabilă este declarată (cu sau fără initializare) în secțiunea “antet” (la începutul programului, după includerea librărilor necesare) atunci ea este “vizibilă” oriunde în fișierul (programul) respectiv. Aceasta înseamnă că ea poate fi accesată / utilizată / modificată în corpul oricărei funcții din programul respectiv. Din pdv. al duratei de viață, o astfel de variabilă există la nesfârșit (pană cand în memoria program a procesorului Arduino se încarcă un alt program). Spunem că variabila respectivă este **variabilă globală**.

Variabilele declarate în corpul unei funcții sunt locale (*automatice*). Ele sunt “vizibile” doar în corpul funcției respective; la încheierea procesului de executie a funcției respective (*revenirea din apel*), variabilele locale sunt distruse. Durata de viață a unei variabile locale este asadar egală cu durata execuției funcției în corpul careia a fost declarată.

RETINEM:

- **variabila globală este eternă; domeniul de vizibilitate este întreg fișierul (programul).** Ea ocupă fizic loc în memoria de date a procesorului, pană cand un alt program este încărcat în memoria de date.
- **variabila locală / automatică este temporară; domeniul de vizibilitate este local (corpul funcției în care a fost declarată).** Ea ocupă fizic loc în memoria heap doar pe durata existenței acesteia; la revenirea din apelul funcției, spațiul de memorie ocupat se eliberează.

Tipuri de date¹³

Mai intai, ce este o dată? Orice variabilă declarată în program, globală sau locală, este o dată; ea are un anumit tip:

- **bool.** O dată de tip bool poate fi **true** sau **false**.
Exemplu: `bool received_data = false;`
- **byte.** O dată de tip byte este un numar natural ≤ 255 .
- **char.** Tip de data utilizat pentru caractere.
- **int.** Tip de dată utilizat pentru numere întregi pe 2 bytes adica întregi în gama $\{-32768, \dots, +32767\}$.

```

Exemplu: int i = -3000;           // întreg initializat cu valoarea -3000.
        int initial_index = -42000; // se va reprezenta trunchiat deoarece
                                     // -42000 < -32768.
        long initial_index = -42000L; // se va trata -42000 ca un int extins

```

¹³ In acest paragraf – tipuri de date – cuvintele bold sunt cuvinte cheie ale limbajului.

// la 4 bytes (reprezentare corecta).

- **long**. Extinde tipul de data int la 4 bytes.
- **float**. Tip de dată pentru numere în virgulă mobilă (numere reale). Se reprezintă pe 4 bytes. Se pot reprezenta numere reale în gama -3.4×10^{38} , $+3.4 \times 10^{38}$.

Exemplu: `float sensor_data; // stochează data citită de la senzor.`
`float default_bias = 2.016;`

- **array**. Este o colecție de variabile ce pot fi accesate printr-un index (este un vector).

Exemplu: `int myPins[] = {4, 5, 6, 7}; // myPins este un vector de 4 int,`
`// myPins[0] = 4, myPins[1] = 5,`
`// myPins[2] = 6, myPins[3] = 7;`
`char MSG[6] = "Hello"; // sirul "Hello" are 5 caractere (char)`
`// însă vectorul MSG contine 6 date`
`// , a 6-a este caracterul '\0' care este`
`// terminatorul de sir.`

NOTĂ: pentru lista completă a tipurilor de date (și multe altele), consultă <https://www.arduino.cc/reference/en>.

Constante

Există tipuri predefinite de date:

- **HIGH, LOW**. Utilizați în citirea / scrierea pinilor Arduino digitali.

Exemplu: `#define ENA_PIN 8`
`.....`
`void setup(){`
`.....`
`digitalWrite(ENA_PIN, LOW);`
`.....`
`}`

- **INPUT, OUTPUT, INPUT_PULLUP¹⁴**. Utilizați în stabilirea tipului de pin: de intrare, de ieșire respectiv de intrare cu rezistor (*intern placii Arduino*) de pullup.

Exemplu: `#define ENA_PIN 8`
`#define VALID_INPUT 9`
`.....`
`void setup(){`
`.....`
`pinMode(ENA_PIN, OUTPUT);`
`pinMode(VALID_INPUT, INPUT);`
`digitalWrite(ENA_PIN, LOW); // doar pinii de iesire pot fi`
`// scrisi.`
`.....`
`}`

- **true, false**. Utilizate în lucrul cu variabile booleene (*de tip bool*), testarea unei condiții de adevăr, etc. False este totuna cu 0.

¹⁴ În cazul intrărilor (în placa Arduino) care necesită legare la +5v / +3.3v printr-un rezistor de pull up este mai bine să prevedem fizic, pe PCB, un rezistor de 4.7 KΩ decât să utilizăm setarea pinului respectiv ca INPUT_PULLUP. Ceea ce e totuna cu a spune că ne vom rezuma doar la specificatorii INPUT respectiv OUTPUT.

Calificatori de variabilă

- **const.** Variabila declarată cu **const** odata asignată devine constantă: ea poate fi accesată și utilizată în program însă valoarea ei nu poate fi schimbată.
Exemplu: `const float PI = 3.1415; // variabila PI, de tip float, odată`
`const ENA_PIN = 8; // asignată cu valoarea 3.1415, devine constantă.`
`// se preferă în locul declarației cu #define (vezi pag. 10).`
- **static.** Vezi ulterior.
- **volatile.** Variabilele pe care le modifică rutinele de tratare a întreruperilor este bine să fie declarate cu **volatile**.

Funcții predefinite

- pentru pinii de I/O (*digitali, analogici*):
 - pinMode(pin, mode);**
Stabilește tipul pinului respectiv - mode poate fi INPUT, OUTPUT sau INPUT_PULLUP.
 - digitalRead(pin);**
Returnează 0 sau 1 în funcție de valoarea citită de pe pinul digital de intrare pin (0, *dacă pinul pin este LOW*, 1 *dacă pinul pin este HIGH*).
 - digitalWrite(pin, value);**
Setează pinul digital pin cu valoarea value (*value poate fi LOW sau HIGH*).
 - analogRead(pin);**
Returnează valoarea pinului analogic de intrare pin (*un întreg între 0 și 1023*). Acest lucru se întâmplă deoarece pinii analogici au atașați câte un convertor analog - numeric pe 10 biți.
 - analogWrite(pin, value);**
Scrie valoarea value (0 - 255) pe pinul pin. Pe pinul respectiv se va genera un semnal PWM al cărui factor de umplere este $\lambda = \text{value}/255$. Plăcile Arduino UNO, nano și mini au pinii analogici PWM 3, 5, 6, 9, 10, 11 (490 Hz cu excepția pinilor 5 și 6 al căror PWM are frecvența de 980 Hz).
- funcții de timp:
 - delay(value);**
value este o dată de tip unsigned long (este un int pe 4 biți, fără semn, adică în gama 0 - $2^{32}-1$). **Funcția delay() trebuie evitată ori de câte ori este posibil; ea oprește execuția programului pentru o durată de timp precizată de value [ms].** Firește, anumite task-uri critice ca de pildă întreruperile sunt executate totuși la timp.
 - delayMicroseconds(val);**
Se comportă similar cu delay() cu mențiunea că val are tipul unsigned int (*întreg pozitiv pe 2 bytes*) și este exprimat în microsecunde. Cu toate acestea, parametrul val nu trebuie să depășească valoarea 16383 (*vezi Language reference*).

micros();

Returnează durata de timp scursă de la începutul rulării programului, în microsecunde; data returnată este un unsigned long. Pe procesoarele de 16 MHz, rezoluția este de 4 microsecunde; contorul intern se va “da peste cap” după cca. 70 minute.

millis();

Se comportă ca micros() cu mențiunea că valoarea returnată este în milisecunde (*de asemenea, tipul datei returnate este unsigned long*).

NOTĂ: se va evita utilizarea funcțiilor *delay()* respectiv *delayMicroseconds()* utilizandu-se în schimb funcțiile *micros()* și *millis()*. **Atenție! aceste funcții sunt grefate pe Timer0; modificarea registrilor asociați lui Timer0 se va traduce în afectarea execuției funcțiilor micros() respectiv millis()! Ca regulă generală, ori de câte ori vom avea de-a face cu Timere, vom lăsa neatins Timer0!**

- întreruperi externe:

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);

Funcția trebuie apelată pentru a instala o rutină ISR de tratare a întreruperii. Rutina ISR se va executa ori de câte ori, pe linia externă de întrerupere desemnată prin pin (*doar pinii 2 respectiv 3 se pot utiliza*) se întâmplă un eveniment de tipul desemnat prin mode (*mode poate fi LOW, CHANGE, RISING, FALLING*).

detachInterrupt(digitalPinToInterrupt(pin));

Detasează întreruperea montată cu attachInterrupt().

Exemplu:

```
#define MCTR_LEN_PIN 4
#define MCTR_REN_PIN 5
#define MCTR_LPWM_PIN 9
#define MCTR_RPWM_PIN 10
#define MENCODER_PINA 2
#define MENCODER_PINB 3

int encoderPULSE_CNT;
int pulseTOMAKE = 4000;
bool climbUP;
bool motorSTOP = false;
bool motorStopped = false;
bool printCNT;

// motorSense = false / true <=> motor rotates left / right,
// dutyFactor in 0.0 - 0.99 range.
void motorRUN(bool motorStop, bool motorSense, float dutyFactor){
```

```

if(motorStop == true){
    digitalWrite(MCTR_LEN_PIN, LOW);
    digitalWrite(MCTR_REN_PIN, LOW);
    analogWrite(MCTR_LPWM_PIN, (int) (0.0*255));
    analogWrite(MCTR_RPWM_PIN, (int) (0.0*255));
}
else{
    digitalWrite(MCTR_LEN_PIN, HIGH);
    digitalWrite(MCTR_REN_PIN, HIGH);

    if(motorSense == true){          // motor rotates right
        analogWrite(MCTR_LPWM_PIN, (int) (0.0*255));
        analogWrite(MCTR_RPWM_PIN, (int) (dutyFactor*255));
    }
    else{                          // motor rotates left
        analogWrite(MCTR_RPWM_PIN, (int) (0.0*255));
        analogWrite(MCTR_LPWM_PIN, (int) (dutyFactor*255));
    }
}
}
}

```

```

void setup() {

```

```

    Serial.begin(115200);

```

```

    // Reduce the prescaler from x64 (the default value) to x8.
    // This will increase PWM frequency from 0.490 KHz to 3.92 KHz.
    // Note that the PWM frequency could be increased by 64 times but, since
    // the BTS7960 driver supports PWM of maximum 25 KHz, the prescaler
    // cannot be x1 but x8 instead.

```

```

    TCCR1B = (TCCR1B & B11111000) | B00000010;

```

```

    // set those pin as output pins:

```

```

    pinMode(MCTR_LEN_PIN, OUTPUT);
    pinMode(MCTR_REN_PIN, OUTPUT);
    pinMode(MCTR_LPWM_PIN, OUTPUT);
    pinMode(MCTR_RPWM_PIN, OUTPUT);

```

```

    // set encoder's output A and B as input pins.

```

```

    // NOTE: I've tied this input pins to +5V through 4.7 Kohm pullup resistors.
    pinMode(MENCODER_PINA, INPUT);

```

```

pinMode(MENCODER_PINB, INPUT);

// Initially, drive them all LOW:
digitalWrite(MCTR_LEN_PIN, LOW);
digitalWrite(MCTR_REN_PIN, LOW);
digitalWrite(MCTR_LPWM_PIN, LOW);
digitalWrite(MCTR_RPWM_PIN, LOW);
delay(1000);

climbUP = true;
printCNT = false;

// mount external interrupt on pin 2:
encoderPULSE_CNT = 0;
attachInterrupt(digitalPinToInterrupt(MENCODER_PINA), ISR_A, RISING);
}

void ISR_A(){

    int temp = digitalRead(MENCODER_PINB);
    if(temp > 0)
        encoderPULSE_CNT--;
    else
        encoderPULSE_CNT++;

    if(climbUP == true){
        if(encoderPULSE_CNT == pulseTOMAKE)
            climbUP = false;
    }
    else{
        if(encoderPULSE_CNT == 0)
            motorSTOP = true;
    }

    printCNT = true;
}

void loop() {

    if(motorSTOP == false){
        motorRUN(false, climbUP, 0.20); // 20% duty.
    }
}

```

```

        if(printCNT == true){
            Serial.println(encoderPULSE_CNT);
            printCNT = false;
        }
    }
    else{
        if(motorStopped == false){

            // stop the motor:
            motorRUN(true, false, 0.0); // stop the motor.
            detachInterrupt(digitalPinToInterrupt(MENCODER_PINA));
            Serial.println("Motor stopped now.");

            motorStopped = true;
        }
    }
}

```

În exemplul anterior, funcția `motorRUN()` controlează un motor electric de cc. cu encoder de poziție unghiulară încorporat: motorul poate fi pornit sau oprit, se poate roti spre stanga sau spre dreapta respectiv se poate configura factorul de umplere a tensiunii PWM de la bornele motorului (*motorul se alimentează printr-un driver; pinii 4, 5, 9 și 10 constituie interfața de control a driverului*).

La alimentarea motoarelor de cc. cu colector mecanic, dacă frecvența tensiunii PWM de alimentare a motorului este prea mică, apare fenomenul de *whinning* (*miauna motorului*). Pentru eliminarea acestui neajuns trebuie crescută frecvența PWM de la 490 Hz (*valoarea default pt Arduino*) la cativa KHz sau chiar zeci de KHz. În exemplul considerat, acest lucru se face în funcția `setup()` prin suprascrierea registrului `TCCR1B`: `TCCR1B = (TCCR1B & B11111000) | B00000010`.

Tot în funcția `setup()` putem vedea montarea rutinei `ISR_A` pe pinul 2; rutina se va executa ori de câte ori pinul de intrare 2 trece din LOW în HIGH (*pinii 2 și 3 constituie interfața Arduino cu encoderul de poziție unghiulară al motorului*).

Elemente de limbaj

Ca orice limbaj de programare, și limbajul utilizat pentru scrierea aplicațiilor Arduino are structuri de control și operatori.

Structuri de control:

- **if, if else, if else if else.** Acestea sunt structuri de control decizionale.


```

      if (condiție) {
          // instrucțiuni;           // se execută grupul de instrucțiuni dacă,
      }                               // condiție este adevărată.
      if (condiție) {

```

```

    // instrucțiuni A;           // se execută fie grupul de instrucțiuni A, fie
}                                // grupul de instrucțiuni B, în funcție de valoarea
else {                           // de adevăr a lui condiție.
    // instrucțiuni B;
}

if (condiție 1) {                // dacă, condiție 1 este adevărată, se execută
    // instrucțiuni A;          // grupul de instrucțiuni A, altfel,
}                                // dacă, condiție 2 este adevărată, se execută
else if (condiție 2) {           // grupul B de instrucțiuni, altfel,
    // instrucțiuni B;          // se execută grupul C de instrucțiuni.
}

else {
    // instrucțiuni C;
}

```

Atenție! Acest tip de structură de control – if ... else if ... else if ... else ... – va executa primul grup de instrucțiuni a cărui condiție de adevăr este satisfăcută (adevărată), chiar dacă mai multe condiții au valoare true! De exemplu, dacă, condiție 1 și condiție 2 sunt ambele adevărate, se va executa grupul A de instrucțiuni.

- **switch (var) {**
 case valueA:
 // grupul A de instrucțiuni;
 break;
 case valueB:
 // grupul B de instrucțiuni;
 break;
 default:
 // grupul C de instrucțiuni;
 break;
 }

Această structură de control este similară cu structura if ... else if ... else. Diferența constă în aceea că, în switch, se compară valorile variabilei var, pe rând, cu valueA, value B, șamd, până când se găsește egalitatea caz în care se execută grupul respectiv de instrucțiuni (*dacă var difera de toate valorile valueA, valueB, etc, se va executa grupul de instrucțiuni din default*). În if ... else if ... else if ... else ..., se testează valoarea de adevăr a condițiilor respective.

De menționat că, clauza default este opțională!

- **for, while, do while.** Acestea permit parcurgerea unui vector, executarea de un anumit număr de ori (sau cât timp o expresie booleană este adevărată) a unui grup de instrucțiuni.

Exemple:

```
// Dim an LED using a PWM pin
```

```

int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup() {
  // no setup needed
}

void loop() {
  for (int i = 0; i <= 255; i++) {
    analogWrite(PWMpin, i);
    delay(10);
  }
}

```

În exemplul anterior, se generează pe pinul de ieseire 10 un tren de impulsuri PWM cu factor de umplere ajustabil. Inițial, factorul de umplere este 0% și crește cu $100/256 \approx 0.39\%$ la fiecare 10 ms (*vezi apelul `delay(10)`*) până când ajunge la 100% (*bucla for se execută de 256 ori*).

```

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include "printf.h"

```

```

#define CSN_PIN      8
#define CE_PIN       7

```

```

const byte SENDER1_ADDR[6] = "00111";
const byte RECEIVER_ADDR[6] = "00001";
const float START = 5678;
const float SENSORS_INIT_DONE = 9999;

```

```

float received_data[4];           // received_data[0] holds Sensor 1 angle,
                                   // received_data[1] holds Sensor 1 angular speed,
                                   // received_data[2] holds Sensor 2 angle,
                                   // received_data[3] holds Sensor 2 angular speed.

float tx_control_data;           // used to send "START" command to the transmitters.
float rx_control_data;           // rx_control_data holds the transmitter response to
                                   // the "START" command - it should be "ACK".

```

```

float sensor1_angle;
float sensor1_angularspeed;
float sensor2_angle;
float sensor2_angularspeed;

```

```

int key_entered;

```

```

bool sender2_ack_received;

```

```

bool write_report;

```

```

uint8_t pipe;
uint8_t payload_length;          // payload length [bytes].

unsigned long current_entry_time;
unsigned long previous_entry_time = 0;

RF24 rx_NRF(CE_PIN, CSN_PIN);    // the receiver NRF module.

void setup() {

    // communication rate with Serial Monitor
    // (by default is Serial 0 port):
    Serial.begin(500000);

    // initialize tx NRF radio module:
    if(!rx_NRF.begin()){
        Serial.println(F(" NRF radio on Arduino principal does not respond! "));
        while(1){ }           // do not advance further.
    }

    // set the speed to 1Mbit/s:
    rx_NRF.setDataRate(RF24_2MBPS);

    // set the power level:
    rx_NRF.setPALevel(RF24_PA_MIN);

    // generic delay...
    delay(15000);

    // STEP1: display a message - requesting user input:
    key_entered = 0;
    Serial.println(F("Press key 1 to start transmission."));
    while(key_entered != 1){
        while(Serial.available() == 0);           // awaiting user's key press...
        key_entered = Serial.parseInt();           // see which key was pressed.
        Serial.read();
    }

    // STEP2: send "START" command to SENDER1 (as a result, SENDER1's Arduino
    //      nano does some Sensor1 initialisation; when finishes, SENDER1
    //      notice SENDER2 by sending the START command. When SENDER2

```

```

// receives START command, it undergoes a similar routine for
// initialising Senor2; when finishes, it notifies the RECEIVER
// by sending an ACK).
rx_NRF.openWritingPipe(SENDER1_ADDR);
rx_NRF.stopListening();
tx_control_data = START;
write_report = rx_NRF.write(&tx_control_data, sizeof(tx_control_data));
if(write_report == true){
    Serial.println(F("Transmission START command sent to SENDER1.));
    rx_NRF.openReadingPipe(0, RECEIVER_ADDR);
    rx_NRF.startListening();
}
else{
    Serial.println(F("START command could not be sent to SENDER1.));
    while(1){ }
}

// STEP3: wait SENDER2 to reply with SENSORS_INIT_DONE (meaning both Sensors
// initialized properly).
sender2_ack_received = false;
while(sender2_ack_received == false){
    if(rx_NRF.available() == true){
        rx_NRF.read(&rx_control_data, sizeof(rx_control_data));
        if(rx_control_data == SENSORS_INIT_DONE){
            Serial.println(F("Sender2 ACK response received. Sensors have been
                               initialized.));
            sender2_ack_received = true;
        }
    }
}

// STEP4: from now on, the RECEIVER only reads data from SENDER2.
rx_NRF.flush_rx();
rx_NRF.setPayloadSize(sizeof(received_data));
rx_NRF.setRetries(0, 2);           // NEEDED OR NOT ???
sender2_ack_received = false;
}

```

În exemplul anterior se pot vedea mai multe feluri de a utiliza construcția while. De exemplu, construcția **while (1) { }** blochează la nesfârșit continuarea programului deoarece corpul de instrucțiuni grupat între acolade (*vid in acest caz*) se va executa la nesfârșit, condiția ce validează execuția corpului { } fiind în permanenta adevărată (1 este true). În exemplul cu pricina, construcția while (1) { }

s-a utilizat deoarece nu are sens să se continue programul dacă nu s-a reușit inițializarea modulului de RF.

Tot în acest exemplu, în STEP 1 vedem utilizarea lui `while` pentru a bloca executarea mai departe a programului până când, de la tastatură se introduce un anumit număr (*chiar în interiorul aceluși `while` există un alt `while` care blochează avansul programului până când se introduce ceva de la tastatură – vezi `while(Serial.available() == 0);`*).

```
do {  
    // instrucțiuni  
} while (condiție);
```

realizează același lucru ca `while (condiție) { // instrucțiuni }` cu deosebirea că, în cazul **`do ... while`**, corpul de instrucțiuni dintre acolade se va executa cel puțin o singură dată (*mai întâi se execută corpul de instrucțiuni, apoi se (re)evaluează condiție după care, dacă este adevărată, corpul de instrucțiuni se va executa încă o dată, șamd*).

Operatori

- **aritmetici:** `+`, `-`, `*`, `/`, `%` (restul împărțirii modulo), `=` (operatorul de atribuire).
Acești operatori aritmetici sunt binari ("unesc" doi operanzi). Dacă unul dintre cei doi operanzi este o dată de tip `float` sau `double` (număr real) atunci automat calculul se va face în virgula mobilă (se convertește forțat și al doilea operand la `float` sau `double` iar rezultatul operației aritmetice va fi `float` sau `double`).

Exemplu: `float PI = 3.1415;`
`int alpha = 2;`
`int fraction = alpha / (2*PI);`

În exemplul considerat, `2*PI` se convertește la `float` (*2 este de tip `int` – întreg*) iar `alpha`, deși de tip `int`, este convertit tot la `float`. Rezultatul împărțirii va fi tot un `float`. **Atenție! la atribuire, `fraction` (care este de tip `int`) va primi partea întreagă a rezultatului (care este 0 în acest caz)! Dacă nu dorim asta atunci trebuie să scriem `float fraction = alpha / (2*PI)`.**

Atenție! Există și o prioritate a operatorilor! În cazul în care operatorii au aceeași prioritate intervine asocierea. Aceasta poate fi de la dreapta la stanga sau invers. Pentru a evita memorarea mecanică a priorității operatorilor respectiv a tipului de asociere, vom utiliza paranteze. În exemplul anterior, dacă am fi avut:

`int fraction = alpha / 2*PI;`

, rezultatul ar fi fost 3! (*s-ar fi executat întâi `alpha / 2`, rezultat 1, după care `1*PI` care da 3.1415, ar fi fost trunchiat la 3, `fraction` fiind un `int`*).

- **de comparare:** `!=` (diferit de), `==` (identic egal), `<=`, `>=`, `<`, `>`.
De menționat că **operatorul `==` este egalul propriu-zis, testarea egalității, nu operatorul de atribuire (`=`)!**

- **logici (booleeni):** *!* (negare logică), *&&* (SI logic), *||* (SAU logic). Evident, utilizarea operatorilor logici are drept rezultat o expresie logică.

Exemplu:

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH) { // if BOTH the switches read HIGH
  // statements
}
```

În exemplul considerat, dacă ambii pini de intrare 2 și 3 sunt HIGH, se execută instrucțiunile dintre acolade.

- **la nivel de bit (bitwise operators):** *~* (complementare), *|* (SAU bit cu bit), *&* (SI bit cu bit), *^* (XOR bit cu bit), *<<* (deplasare stanga pe biti), *>>* (deplasare dreapta pe biti).

Exemplu:

```
// Note: This code uses registers specific to AVR microcontrollers (Uno, Nano, Leonardo,
Mega, etc.)
// it will not compile for other architectures
void setup() {
  DDRB = DDRB | 0b00100000; // set PB5 (pin 13 on Uno/Nano, pin 9 on Leonardo/Micro, pin
11 on Mega) as OUTPUT
  Serial.begin(9600);
}

void loop() {
  PORTB = PORTB ^ 0b00100000; // invert PB5, leave others untouched
  delay(100);
}
```

În `loop()`, prin `PORTB = PORTB ^ 0b00100000`; datorită operatorului XOR (^), care lasă neschimbat bitul respectiv la XOR cu zero, se va complementa doar bitul 5 (biții sunt 0, 1, 2, ..., 7, de la dreapta la stanga, prefixul 0b semnifică faptul că urmează o data binară) – XOR cu 1 schimbă valoarea de adevăr a bitului.

3. Introducerea datelor de la tastatură; afișarea datelor pe Serial Monitor

Următorul program Arduino rezolvă ecuația de gradul al doilea, $ax^2+bx+c=0$, a, b, c numere reale.

- la început se afișează pe Serial monitor mesajul "Tasteaza 1 pentru a începe ..."
- imediat după apăsarea tastei 1 pe Serial monitor se afișează mesajul "Introdu coeficienții ecuației, a, b, c (cate un spațiu între coeficienti)..."
- imediat după introducerea coeficienților, se va afișa, pe Serial monitor mesajul "Ecuația are soluții reale" sau "Ecuația are soluții complexe" urmat de soluțiile găsite.

După aceasta, pe linie nouă, în Serial monitor va apărea mesajul "Apasă tasta 1 pentru a rezolva o nouă ecuație sau tasta 2 pentru părăsirea programului".

```
float a, b, c;                // 2nd order equation coefficients.
float x1, x2;                 // hold real solutions.
float x1Re, x1Im, x2Re, x2Im; // hold complex solutions.
float delta;
bool reiterate;
int keyPressed;
String inputString;           // holds one string entered from Serial Monitor.

void setup() {

  Serial.begin(500000);

  reiterate = true;
  keyPressed = 0;

  while (reiterate == true) {

    Serial.println(F("Acest program rezolva ecuatia de gradul al doilea."));
    Serial.println(F(""));

    // ask the USER to press key 1 to start:
    Serial.println(F("Apasa tasta 1 pentru a incepe..."));
    Serial.println(F(""));

    // until USER doesn't press key 1, keep waiting for that ...
    while(keyPressed != 1){
      while(Serial.available() == 0); // wait the USER to type something ...
      keyPressed = Serial.parseInt(); // read the USER's input.
      Serial.read();                  // clear the keyboard buffer
    }
  }
}
```

```

}

// if we're here, the USER pressed key 1 so go forth,
// ask the USER to enter the coefficients:
Serial.println(F("Introdu coeficientii ecuatiei (apasa Enter dupa fiecare coef.)"));
Serial.println(F("NOTA: pentru a continua, trebuie sa introduci toti coeficientii."));

while(Serial.available() > 0){           // clear possible garbage
    Serial.read();                      // on the Serial, prior to
}                                       // entering equation's coefficients.

// get the x squared coefficient:
while(Serial.available() == 0);
inputString = Serial.readStringUntil('\r');
Serial.read();                          // get rid of the "Enter" character.
a = inputString.toFloat();

// get the x coefficient:
while(Serial.available() == 0);
inputString = Serial.readStringUntil('\r');
Serial.read();                          // get rid of the "Enter" character.
b = inputString.toFloat();

// get the free term:
while(Serial.available() == 0);
inputString = Serial.readStringUntil('\r');
Serial.read();                          // get rid of the "Enter" character.
c = inputString.toFloat();

Serial.println(F(""));
Serial.println(F("Ai introdus urmatoorii coeficienti:"));
Serial.print(a, 4);                     // display coefficient a with 4 decimals.
Serial.print(F(" "));                   // put a space after a.
Serial.print(b, 4);                     // show coefficient b with 4 decimals.
Serial.print(F(" "));                   // put a space after b.
Serial.print(c, 4);                     // show coefficient c with 4 decimals.
Serial.println(F(""));                  // newline.
Serial.println(F(""));

// Now solve the equation:
delta = b*b - 4*a*c;
if(delta >= 0.0){

```

```

        Serial.println(F("Ecuatia are solutiile reale:"));
        x1 = (-b + sqrt(delta)) / (2*a);
        x2 = (-b - sqrt(delta)) / (2*a);
        Serial.print(x1, 4);
        Serial.print(F(" "));
        Serial.print(x2, 4);
        Serial.println(F(""));
    }
    else{
        Serial.println(F("Ecuatia are solutiile complexe:"));
        x1Re = -b / (2*a);
        x2Re = x1Re;
        x1Im = sqrt(-delta) / (2*a);
        x2Im = -x1Im;
        Serial.print(x1Re, 4); Serial.print(F(" + i*")); Serial.print(x1Im, 4);
        Serial.print(F(" "));
        Serial.print(x2Re, 4); Serial.print(F(" + i*")); Serial.print(x2Im, 4);
        Serial.println(F(""));
    }

    Serial.println(F(""));
    Serial.println(F("Apasa tasta 2 pentru reluare sau tasta 3 pentru iesire..."));
    while(Serial.available() == 0);
    keyPressed = Serial.parseInt();
    if(keyPressed != 2){
        reiterate = false;           // exit while loop.
        Serial.println(F(""));
        Serial.println(F("Ai ales sa parasesti programul."));
    }
    else
        keyPressed = 0;              // prepare for a new execution.

}
}

void loop() {

}

```

Ce învățăm din exemplul anterior?

- dacă vrem să oprim continuarea programului până cand USER ul introduce ceva de la tastatură (*Serial Monitor*), utilizăm:

```
while(Serial.available() == 0);
```

- dacă vrem să ne asigurăm, înaintea unei transmisii, că bufferul seriei este gol (pentru a nu prinde "gunoarie" printre date valide), utilizăm:

```
while(Serial.available() > 0)
    Serial.read();
```

- afișarea unui text pe Serial monitor se poate face cu `print("Textul de afișat")` sau `println("Textul de afișat")` – aceasta din urmă introduce și un salt la rand nou. Totuși, scrierea datelor pe serial monitor este consumatoare de timp. Pentru speed up, se va proceda astfel:

```
print(F("Textul de afișat"));
println(F("Textul de afișat"));
```

- afișarea pe Serial monitor a unui float se poate face specificand numărul de zecimale. De exemplu,

```
Serial.print(x1, 4);
```

va afișa float ul x1 cu 4 zecimale.

- dacă se așteaptă citirea unui întreg introdus de la tastatură (din Serial monitor) vom utiliza:

```
inputData = Serial.parseInt(); // inputData are tipul int
```

- dacă se așteaptă citirea unui float introdus de la tastatură (din Serial monitor, Enter la sfârșit), vom utiliza:

```
while(Serial.available() == 0);
inputString = Serial.readStringUntil('\r'); // inputString are tipul String.
Serial.read();                               // elimina "Enter" din buffer.
inputFloat = inputString.toFloat();          // inputFloat are tipul float.
```

- citirea unui float introdus de la Serial monitor se poate face, mai simplu, astfel:

```
while(Serial.available() == 0);
a = Serial.parseFloat(); // a este un float.
while(Serial.available() > 0) // necesita curatarea bufferului
    Serial.read();           // atasat seriei.
```

4. Citirea / scrierea datelor de pe / pe un card micro SD

5. Întreruperi (interne) de la timere. Aplicație: realizarea unui ceas electronic cu afisor LCD1602

Arduino nano și UNO au 3 timere (*contoare*) interne; Arduino MEGA are 6 timere. Aceste timere pot fi programate astfel încat să putem gestiona task-uri care trebuiesc executate periodic (*la distante echidistante in timp*).

În exemplul următor, realizăm un ceas electronic cu afișare pe Serial monitor. Se poate potrivi, de la tastatură / Serial monitor, ora (*ora, minutul, secunda*), ora alarmei (*ora, minutul*) respectiv starea alarmei (*on / off*); alarma care sună se concretizează într-un mesaj pe Serial monitor.

```
#include <TimerOne.h>           // for working with Timer1.
#include <Wire.h>                // for working with I2C (LCD1602 has an
                                // I2C interface with Arduino).
#include <LiquidCrystal_I2C.h>   // for working with LCD1602 display.

// #define ALARM_PIN 6          // PWM pin. Enable for this pin is the
                                // alarm_ringing variable.

LiquidCrystal_I2C lcd1602_i2c(0x27, 16, 2); // The address of I2C is 0x27 by default
                                              // (for instance if A0, A1, A2 are not
                                              // soldered - see the back).
                                              // LCD 2 rows, 16 columns.

long interrupt_period = 1000000; // This clock is interrupt (from Timer1) driven;
                                  // interrupt fires once every second (that's
                                  // a million microseconds).

bool tick;

byte hours_cnt;                  // these variable hold
byte minutes_cnt;               // time (hours,
byte seconds_cnt;               // minutes and seconds).
byte alarm_hours;               // the same for alarm.
byte alarm_minutes;
byte alarm_cnt;                 // counts the alarm duration.
bool alarm_ringing;

bool set_time;                  // need to know what is being set:
bool set_alarm;                 // time or alarm.
bool alarm_ON;                  // the alarm status: active or deactivated.
bool alarm_rang;                // prevent alarm firing second time.

void initAndConfigLCD1602(){
```



```

    lcd1602_i2c.init();
    lcd1602_i2c.backlight();

    // display double dots on LCD (always in the same position):
    lcd1602_i2c.setCursor(2, 0);
    lcd1602_i2c.print(":");
    lcd1602_i2c.setCursor(5, 0);
    lcd1602_i2c.print(":");
    lcd1602_i2c.setCursor(5, 1);
    lcd1602_i2c.print(":");
    lcd1602_i2c.setCursor(9, 1);
    lcd1602_i2c.print(":");
}

void resetClockLogic(){

    hours_cnt = 0;
    minutes_cnt = 0;
    seconds_cnt = 0;
    alarm_cnt = 0;
    tick = false;                // clear the semaphore.
    set_time = false;
    set_alarm = false;
    alarm_ON = false;
    alarm_rang = false;
    alarm_ringing = false;
}

void displayHours(byte start_column, byte row, byte hours){

    byte first_hours_digit = hours / 10;
    byte second_hours_digit = hours % 10;
    lcd1602_i2c.setCursor(start_column, row);
    lcd1602_i2c.write(first_hours_digit + 48);
    lcd1602_i2c.setCursor(start_column + 1, row);
    lcd1602_i2c.write(second_hours_digit + 48);
}

void displayMinutes(byte start_column, byte row, byte minutes){

    byte first_minutes_digit = minutes / 10;

```

```

    byte second_minutes_digit = minutes % 10;
    lcd1602_i2c.setCursor(start_column, row);
    lcd1602_i2c.write(first_minutes_digit + 48);
    lcd1602_i2c.setCursor(start_column + 1, row);
    lcd1602_i2c.write(second_minutes_digit + 48);
}

void displaySeconds(byte start_column, byte row, byte seconds){

    byte first_seconds_digit = seconds / 10;
    byte second_seconds_digit = seconds % 10;
    lcd1602_i2c.setCursor(start_column, row);
    lcd1602_i2c.write(first_seconds_digit + 48);
    lcd1602_i2c.setCursor(start_column + 1, row);
    lcd1602_i2c.write(second_seconds_digit + 48);
}

void displayAlarm(byte start_column, byte row, byte al_hours, byte al_minutes, bool al_state){

    lcd1602_i2c.setCursor(start_column, row);
    lcd1602_i2c.print("alarm: ");

    displayHours(start_column + 7, row, al_hours);
    displayMinutes(start_column + 10, row, al_minutes);

    lcd1602_i2c.setCursor(start_column + 13, row);
    if(al_state == false)
        lcd1602_i2c.print("OFF");
    else
        lcd1602_i2c.print(" ON");
}

// the clock's time is set from the Serial monitor
// and updated on LCD:
void setTime(){

    // clear possible junk prior to time setting:
    while(Serial.available() > 0)
        Serial.read();

    // set hours:
    Serial.println("Enter time, hours: ");

```

```

while(Serial.available() > 0)                // clear the input buffer (which now has
    Serial.read();                          // the string "Enter time, hours: ").

while(Serial.available() == 0);
hours_cnt = (byte) Serial.parseInt();
displayHours(0, 0, hours_cnt);               // update entered time hours on LCD.

// set minutes:
Serial.println("Enter time, minutes: ");
while(Serial.available() > 0)               // clear the input buffer (which now has
    Serial.read();                          // the string "Enter time, minutes: ").

while(Serial.available() == 0);
minutes_cnt = (byte) Serial.parseInt();
displayMinutes(3, 0, minutes_cnt);          // update entered time minutes on LCD.

// set seconds:
Serial.println("Enter time, seconds: ");
while(Serial.available() > 0)               // clear the input buffer (which now has
    Serial.read();                          // the string "Enter time, seconds: ").

while(Serial.available() == 0);
seconds_cnt = (byte) Serial.parseInt();
displaySeconds(6, 0, seconds_cnt);          // update time seconds on LCD.
}

// the clock's alarm is also set from Serial monitor
// and updated on LCD:
void setAlarm(){

    // clear possible junk:
    while(Serial.available() > 0)
        Serial.read();

    // set alarm hours:
    Serial.println("Enter alarm, hours: ");
    while(Serial.available() > 0)
        Serial.read();

    while(Serial.available() == 0);
    alarm_hours = (byte) Serial.parseInt();
    displayHours(7, 1, alarm_hours);

```

```

// set alarm minutes:
Serial.println("Enter alarm, minutes: ");
while(Serial.available() > 0)
    Serial.read();

while(Serial.available() == 0);
alarm_minutes = (byte) Serial.parseInt();
displayMinutes(10, 1, alarm_minutes);

// alarm ON / OFF:
Serial.println("Set alarm ON? (Y/N)");
while(Serial.available() > 0)
    Serial.read();

while(Serial.available() == 0);
if(((char) Serial.read()) == 'Y')
    alarm_ON = true;
else
    alarm_ON = false;

// display alarm state:
lcd1602_i2c.setCursor(13, 1);
if(alarm_ON == false)
    lcd1602_i2c.print("OFF");
else
    lcd1602_i2c.print(" ON");
}

// alarm and time are set from Serial Monitor:
void serialEvent(){

    char input_char;

    if((set_time == false) && (set_alarm == false)){
        if(Serial.available() > 0){
            input_char = (char) Serial.read();
            if(input_char == 'T')
                set_time = true;
            else if(input_char == 'A')
                set_alarm = true;
        }
    }
}

```

```

        // clear '\n' from the buffer:
        Serial.read();
    }
}

void setup() {

    Serial.begin(500000);                // Serial's baudrate.
    // analogWrite(ALARM_PIN, 0);        // PWM with 0% duty cycle - that's buzzer off.
    initAndConfigLCD1602();
    resetClockLogic();

    displayHours(0, 0, 0);                // start column, row, value.
    displayMinutes(3, 0, 0);
    displaySeconds(6, 0, 0);
    displayAlarm(0, 1, 0, 0, false);      // starting from 1st column, second row,
                                           // alarm hour, alarm OFF.

    Serial.println(F("Pentru a seta ora (alarma) apasa tasta T (A)"));

    Timer1.initialize(interrupt_period);  // mount the interrupt (the
    Timer1.attachInterrupt(Timer1_ISR);    // interrupt service routine, Timer1_ISR).
}

void Timer1_ISR(){

    tick = true;                          // mark the interrupt as "occured".

    seconds_cnt++;                         // update time counters
    if(seconds_cnt == 60){                 // logic.
        seconds_cnt = 0;
        minutes_cnt++;
        if(minutes_cnt == 60){
            minutes_cnt = 0;
            hours_cnt++;
            if(hours_cnt == 24)
                hours_cnt = 0;
        }
    }
}
}

```

```

void loop(){

    if(tick == true){

        // update time:
        displayHours(0, 0, hours_cnt);
        displayMinutes(3, 0, minutes_cnt);
        displaySeconds(6, 0, seconds_cnt);

        if(alarm_ON == true){
            if((hours_cnt == alarm_hours) && (minutes_cnt == alarm_minutes)){
                if(alarm_rang == false){
                    alarm_ringing = true;
                }
            }
            else if(alarm_rang == true)
                alarm_rang = false;

            if(alarm_ringing == true){
                alarm_cnt++;
                // analogWrite(ALARM_PIN, (byte)40*alarm_cnt);
                Serial.println("Ringing...");
                if(alarm_cnt == 15){
                    alarm_cnt = 0;
                    Serial.println("Alarm stopped.");
                    alarm_ringing = false;
                    // analogWrite(ALARM_PIN, 0);
                    alarm_rang = true;
                }
            }
        }

        tick = false;
    }
    else if(set_time == true){

        setTime();                // get the hour time from Serial monitor.
        set_time = false;         // clear set_time flag.
    }
    else if(set_alarm == true){

        setAlarm();               // get alarm time and state from Serial monitor.
    }
}

```

```

        set_alarm = false;           // clear set_alarm flag.
    }
}

```

În exemplul prezentat, vedem:

- includerea bibliotecilor TimerOne.h, Wire.h, LiquidCrystal_I2C.h pentru lucrul cu intreruperi (*bazate pe Timer1*) respectiv lucrul cu interfața I2C (*display ul LCD1602 are și un chip adaptor pentru I2C – reduce interfața display ului de la 20 pini la 4 pini*) si cu display ul LCD1602.
- cateva funcții pentru lucrul cu LCD1602 (*setCursor(), print(), write()*).
- instalarea intreruperii de la Timer1 (*funcțiile initialize() si attachInterrupt()*).
- rutina de tratare a intreruperii – Timer1_ISR – care actualizează contoarele de timp (*ora, minute, secunde*) ale ceasului, în fiecare secundă (*întreruperea se execută o dată pe secundă*).
- în loop(), de fiecare dată cand întreruperea se triggherează, se reactualizează ora pe afișorul LCD respectiv se setează ora și alarma. Tot in loop() se gestionează durata de alarmare (*cat alarma "tipa"*).

NOTA: în lipsa unui buzzer, s-a afișat mesajul "Ringing ..." pe durata de alarmare.

6. Creșterea frecvenței PWM ului Arduino prin modificarea regiștrilor timerului aferent

La platformele Arduino UNO și nano, pinii 3, 5, 6, 9, 10 și 11 au ieșire PWM. Acest PWM are frecvența scăzută, 490 sau 980 Hz, ceea ce este OK pentru majoritatea aplicațiilor. Există totuși cazuri când am prefera un PWM de frecvență superioară, de câteva KHz sau chiar zeci de KHz. De exemplu, dacă avem o acționare cu motor de cc. alimentat de la un driver (pentru motoare de cc.) iar motorul este cu colector mecanic (comutat mecanic), atunci dacă frecvența PWM generată de driver este prea mică (sute de Hz – 1 KHz), motorul va produce un sunet deranjant – whinnying – datorită comutației. Acest sunet dispare dacă reușim să creștem frecvența PWM la doar câțiva KHz¹⁵.

Concret, cum putem crește frecvența PWM default pentru Arduino? Pentru aceasta trebuie să știm ca acest PWM se implementează cu întreruperi. La Arduino UNO și nano există 3 timere – Timer0, Timer1 și Timer2 – fiecare controlând câte 2 ieșiri PWM:

- Timer0: pinii PWM 5 și 6 (pini digitali),
- Timer1: pinii PWM 9 și 10 (pini digitali),
- Timer2: pinii PWM 11 și 3 (pini digitali).

După cum am mai spus (pag. 11, funcții de timp), **ca regulă generală nu ne atingem de Timer0; există o serie de funcții sistem și întreruperi grefate pe Timer0. Așadar, dacă este necesară comanda unui driver pentru motor de cc., vom utiliza pinii digitali cu capacitate PWM 9 și 10.** Dar pentru aceștia frecvența default este 490 Hz (doar pinii digitali 5 și 6 au frecvența PWM 980 Hz valoare de asemenea prea mică). Putem crește această frecvență de 8 sau chiar de 64 ori (în acest caz vom obține valoarea 31.25 KHz valoare prea mare pentru anumite drivere – vezi BTS7960) suprascriind valoarea default a registrului TCCR1B (Timer / Counter 1 Control Register B).

1.2 TCCR1B – Timer/Counter1 Control Register B

Bit (0x81)	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 15-6. Clock Select Bit Description			
CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{IO}/1$ (no prescaling)
0	1	0	$clk_{IO}/8$ (from prescaler)
0	1	1	$clk_{IO}/64$ (from prescaler)
1	0	0	$clk_{IO}/256$ (from prescaler)
1	0	1	$clk_{IO}/1024$ (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Printscreen luat de pe <https://electronoobs.com>

¹⁵ Există o limită superioară a frecvenței PWM a semnalelor de comandă pentru driver. De exemplu, pentru driverul BTS7960 frecvența maximă PWM este de 25 KHz.

Putem crește sau scădea frecvența PWM ului pe pinii digitali 9 și 10 modificând valoarea biților de prescalare pentru Timer1. Aceștia sunt biții 2:0 ai registrului TCCR1B. Dar mai întâi, **ce înseamnă prescalare?**

După cum sugerează poza anterioară, fără prescalare, periodicitatea întreruperii generate de Timer1 este dictată direct de frecvența microprocesorului ce echipază placa Arduino – 16 MHz pt Atmega 328p. Având în vedere că Timer1 este pe 16 biți ($2^{16} = 65536$) rezultă că, fără prescalare (*biții CS12, CS11, CS10 sunt 001; Timer1 va număra până la capăt*), întreruperea bazată pe Timer1 se produce o dată la $65536 * 10^{-6} / 16$ secunde, adică o dată la 4096 microsecunde. Dar, numărarea se petrece de fapt up, până la 2047 după care down până la zero deci perioada pentru PWM este de fapt 2048 microsecunde adică cca 488.28 Hz, valoare rotunjită la 490 Hz.

Dacă setăm biții CS12, CS11, CS10 la 010, Timer1 va număra în gama 0 – 8191 deci vom obține o frecvență de 8 ori mai mare pentru PWM decât fără prescalare (*cu prescalare 1*): $2048 / 8 = 256$ microsecunde deci un PWM de frecvență 3.9 KHz.

Dacă setăm biții respectivi la 011 vom obține un PWM de 31.25 KHz, *same*.

Este prudent ca la suprascrierea conținutului unui registru să modificăm doar biții de interes restul biților rămânând nealterați. Putem face acest lucru astfel (*vezi exemplul din pag. 12 – 15, întreruperi externe*):

TCCR1B = (TCCR1B & B11111000) | B00000010

Prin TCCR1B & B11111000 se obține o dată binară pe 8 biți având primii 5 biți (*most significant*) din registrul TCCR1B, ultimii 3 biți fiind 0. Mai departe, facând **“SAU” bit cu bit** cu data binară B00000010, în TCCR1B se vor modifica doar ultimii trei biți (*de fapt doar ultimii doi, de la 01 la 10*).

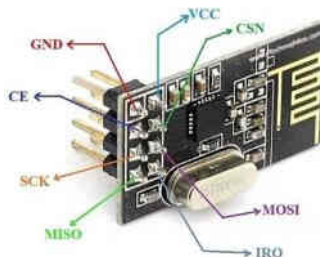
7. Transceiverul NRF24L01. Aplicație: acționarea unui releu de la distanță, utilizand module NRF24L01

Următorul program realizează cuplarea / decuplarea la / de la rețeaua de 230 V 50 Hz , de la distanță, a unui consumator de ca. (*un simplu bec de 20 W*), prin intermediul unui releu comandat de 230 V, 10 A / 5V:

- comanda la distanță se face prin intermediul a doua transceivere radio NRF24L01: unul conectat la un Arduino nano (*comanda se da de la tastatura / Serial monitor*) celalalt, conectat la un alt Arduino nano la care se conectează și releul de acționare (*pentru a înțelege mai bine mecanismul, partea comandată afișează de asemenea mesaje la Serial monitor*).

Modulul NRF24L01 este un transceiver care lucrează în banda 2.4 – 2.5 GHz, alimentat de la 1.9 – 3.6 Vcc, și care poate transmite (*pana la 32 bytes / transmisie*) și recepționa, inclusiv simultan (*transmisie full duplex*); raza de acoperire în aer liber, fără antenă, este de cca. 100 m respectiv cateva sute de metri cu antena. Viteza efectivă maximă de transmisie ("*on air speed*") este de 2Mbit/s, valoare modestă¹⁶.

Interfața NRF24L01 – Arduino (*aceeași atât pentru comandă cat și pentru remote*):



- VCC, legat la pinul 3.3v,
- GND, legat la pinul GND,
- CSN, legat la pinul D8,
- CE, legat la pinul D7,
- MOSI, legat la pinul D11,
- MISO, legat la pinul D12,
- SCK, legat la pinul D13,
- IRQ, neutilizat.

La remote, interfața Arduino – releu JQC-3FF-S-Z (5V / 220 V, 10A):

- VCC, legat la pinul 5v,
- GND, legat la pinul GND,
- IN1, legat la pinul D4 (*pinul de comandă al releului*).

Codul Arduino pentru COMMAND:

¹⁶ Un senzor care furnizează doua date de tip float ($2 \times 4 = 8$ bytes = $8 \times 8 = 64$ bits) va implica transmiterea a 100 – 120 biti (pe langă data utilă există si un overhead, header + CRC). La o analiză de worst case, vom considera și o retransmisie (cel puțin) cu un gap de 250 us. Rezultă cca $60 \times 2 + 250 = 370$ us timp efectiv cheltuit cu transmisia datelor. Considerand și timpul de comutare a radioului între modulele TX și RX de cca. 130 us, obținem aproximativ 0.5 ms durata efectivă implicată de transmisie.

```

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include "printf.h"

#define CSN_PIN      8
#define CE_PIN       7

const byte REMOTE_ADDR[6] = "00111"; // the REMOTE NRF24l01's address.
const byte COMMAND_ADDR[6] = "00001"; // the COMMAND NRF24l01's address.
const float RELAY_ON = 1234; // sent by COMMAND to REMOTE to switch on relay.
const float RELAY_OFF = 5678; // sent by COMMAND to REMOTE to switch off relay.
const float REMOTE_ACK = 1928; // both commands must be ACK-ed by the remote.
// when COMMAND receives a REMOTE_ACK it'll
// display on Serial monitor the appropriate status
// of the relay.

float tx_control_data; // used to send commands to the REMOTE.
float rx_control_data; // holds the REMOTE response.

char input_char;

bool on_off;
bool pending_ACK;

bool write_report;

RF24 tx_NRF(CE_PIN, CSN_PIN); // the COMMAND's radio (NRF24l01).
// For the interface with Arduino nano, see
// Partea a doua - "programarea
// platformelor cu microcontroler.docx".

//*****
void nrf_InitAndConfig(){
    // initialize tx NRF radio module:
    if(!tx_NRF.begin()){
        Serial.println(F("NRF radio on COMMAND does not respond! "));
        while(1){ // do not advance further.
        }
    }
    else{
        Serial.println(F("COMMAND's NRF radio module initialised."));

        tx_NRF.powerDown();
        tx_NRF.powerUp();

        // set the payload size to 4 (bytes):
        tx_NRF.setPayloadSize(4);

        // set the speed to 2Mbit/s:
        tx_NRF.setDataRate(RF24_2MBPS);
    }
}

```

```

        // set the power level:
        tx_NRF.setPALevel(RF24_PA_MIN);

        // set the maximum number of retries
        // in case of failed packet delivery:
        //tx_NRF.setRetries(0, 2);
        //tx_NRF.setRetries(2, 4);
    }
}

void send_RemoteCommand(bool on_off){

    tx_NRF.openWritingPipe(REMOTE_ADDR);
    tx_NRF.stopListening();

    if(on_off == true){
        tx_control_data = RELAY_ON;
    }
    else{
        tx_control_data = RELAY_OFF;
    }

    write_report = tx_NRF.write(&tx_control_data, sizeof(tx_control_data));
    if(write_report == true){
        Serial.println(F("  Command sent to the REMOTE...; awaiting for ACK..."));
        tx_NRF.flush_tx();
        tx_NRF.openReadingPipe(0, COMMAND_ADDR);
        tx_NRF.startListening();
        // sent command need to be ACK-ed;
        // start listening to the REMOTE.
    }
    else{
        Serial.println(F("  Command could not be sent."));
        while(1){}
    }
}

void await_ACK(){

    while(pending_ACK == true){
        if(tx_NRF.available() == true){
            tx_NRF.read(&rx_control_data, sizeof(rx_control_data));
            if(rx_control_data == REMOTE_ACK){
                Serial.println(F("    Command ACK-ed ..."));
                if(on_off == true)
                    Serial.println(F("    Relay ON now."));
                else
                    Serial.println(F("    Relay OFF now."));

                Serial.println(F(""));

                pending_ACK = false;
            }
        }
    }
}

```

```

        on_off = false;
    }
    else{
        Serial.println(F("    Erroneous ACK received."));
    }

    // clear the receiving buffer:
    tx_NRF.flush_rx();
}
}

//***** THE SETUP FUNCTION *****
void setup() {
    // communication rate with Serial Monitor
    // (by default is Serial 0 port):
    Serial.begin(115200);

    // initialise NRF, set the speed and emission power:
    nrf_InitAndConfig();

    Serial.println(F("Relay turns ON / OFF by typing o / f."));
    Serial.println(F(""));

    pending_ACK = false;
    on_off = false;

    // clear possible garbage on Serial:
    while(Serial.available() > 0)
        Serial.read();
}

//***** THE LOOP *****
void loop(){

    // await USER's command:
    while(Serial.available() == 0);

    // get USER's command:
    input_char = (char) Serial.read();

    if((input_char == 'o') || (input_char == 'f')){

        if(input_char == 'o'){
            on_off = true;
            Serial.println(F("USER wants relay to switch ON..."));
        }
        else{
            on_off = false;
            Serial.println(F("USER wants relay to switch OFF..."));
        }
    }
}

```

```

    }

    Serial.println(F("  Preparing the COMMAND to send command to REMOTE..."));

    // send the command to the REMOTE and wait for acknowledgement:
    send_RemoteCommand(on_off);
    pending_ACK = true;
    await_ACK();
}

// if we're here, the USER either gave a valid command
// or garbage; nevertheless, clear the Serial:
while(Serial.available() > 0)
    Serial.read();
}

Codul Arduino pentru REMOTE:

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include "printf.h"

#define CSN_PIN          8
#define CE_PIN           7
#define RELAY_PIN        4

const byte  REMOTE_ADDR[6] = "00111"; // the REMOTE NRF24l01's address.
const byte  COMMAND_ADDR[6] = "00001"; // the COMMAND NRF24l01's address.
const float RELAY_ON = 1234; // sent by COMMAND to REMOTE to switch on relay.
const float RELAY_OFF = 5678; // sent by COMMAND to REMOTE to switch off relay.
const float REMOTE_ACK = 1928; // both commands must be ACK-ed by the remote.
                                // when COMMAND receives a REMOTE_ACK it'll
                                // display on Serial monitor the appropriate status
                                // of the relay.

float tx_control_data; // used to send 'ACK' to the COMMAND.
float rx_control_data; // holds the received command from COMMAND.

bool write_report;

RF24 rx_NRF(CE_PIN, CSN_PIN); // COMMAND's NRF24l01s radio.
                                // For the interface with Arduino nano, see
                                // Partea a doua - programarea
                                // platformelor cu microcontroler.docx.

//*****
void nrf_InitAndConfig(){
    // initialize tx NRF radio module:
    if(!rx_NRF.begin()){

```

```

        Serial.println(F(" NRF radio on REMOTE does not respond! "));
        while(1){    // do not advance further.
    }
    else{
        Serial.println(F("REMOTE's NRF radio module initialised."));

        rx_NRF.powerDown();
        rx_NRF.powerUp();

        // set the payload size to 4 (bytes):
        rx_NRF.setPayloadSize(4);

        // set the speed to 1Mbit/s:
        rx_NRF.setDataRate(RF24_2MBPS);

        // set the power level:
        rx_NRF.setPALevel(RF24_PA_MIN);

        // set the maximum number of retries
        // in case of failed packet delivery:
        //rx_NRF.setRetries(0, 2);
        //rx_NRF.setRetries(2, 4);
    }
}

void send_ACK(){

    Serial.println(F("  Preparing to respond back with ACK..."));
    rx_NRF.openWritingPipe(COMMAND_ADDR);
    rx_NRF.stopListening();
    tx_control_data = REMOTE_ACK;

    write_report = rx_NRF.write(&tx_control_data, sizeof(tx_control_data));

    // if ACK delivered successfully, prepare for taking
    // another command:
    if(write_report == true){
        Serial.println(F("    Answered back with ACK..."));
        Serial.println(F("    Listening again to the COMMAND..."));
        rx_NRF.flush_tx();
        rx_NRF.openReadingPipe(0, REMOTE_ADDR);
        rx_NRF.startListening();
    }
}

void await_COMMAND(){

    if(rx_NRF.available() == true){

        Serial.println(F("Data available at the REMOTE."));
        rx_NRF.read(&rx_control_data, sizeof(rx_control_data));
        rx_NRF.flush_rx();
    }
}

```

```

        if((rx_control_data == RELAY_ON) || (rx_control_data == RELAY_OFF)){
            if(rx_control_data == RELAY_ON){
                Serial.println(F(" Received command RELAY_ON."));
                digitalWrite(RELAY_PIN, LOW); // low active relay command.
            }
            else{
                Serial.println(F(" Received command RELAY_OFF."));
                digitalWrite(RELAY_PIN, HIGH); // low active relay command.
            }

            send_ACK();
        }
    }
}

```

//***** THE SETUP *****

```

void setup() {

    pinMode(RELAY_PIN, OUTPUT);
    digitalWrite(RELAY_PIN, HIGH);

    // communication rate with Serial Monitor
    // (by default is Serial 0 port):
    Serial.begin(115200);

    // initialise NRF, set the speed and emission power:
    nrf_InitAndConfig();

    // initially, the REMOTE must be listening to COMMAND:
    Serial.println(F("Listening to the COMMAND..."));
    delay(10000);
    rx_NRF.flush_rx();
    rx_NRF.openReadingPipe(0, REMOTE_ADDR);
    rx_NRF.startListening();

}

```

//***** THE LOOP *****

```

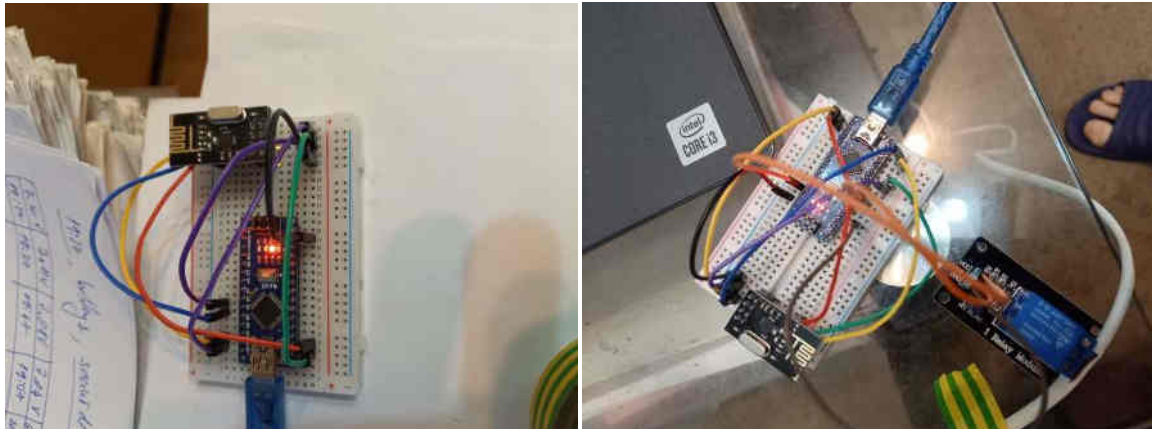
void loop(){

    await_COMMAND();

}

```

Montajul de comandă; montajul comandat (*releu*)



Arduino care comandă; mesaje pe Serial monitor:

```

command_nrf24l01_relay_example | Arduino 1.8.19
File Edit Sketch Tools Help

command_nrf24l01_relay_example

97   tx_NRF.read(&rx_control_data, sizeof(rx_control_data));
98   if(rx_control_data == REMOTE_ACK){
99       Serial.println(F("      Command ACK-ed ..."));
100      if(on_off == true)
101          Serial.println(F("      Relay ON now."));
102      else
103          Serial.println(F("      Relay OFF now."));
104
105      Serial.println(F(""));
106
107      pending_ACK = false;
108      on_off = false;
109  }
110  else{
111      Serial.println(F("      Erroneous ACK rece
112  }
113
114  // clear the receiving buffer:
115  tx_NRF.flush_rx();
116  }
117  }
118  }
119
120
121  //***** THE SETUP FUNCTION *****
122  void setup() {
123
124      // communication rate with Serial Monitor
125      // (by default is Serial 0 port):
126      Serial.begin(115200);
127
128      // initialise NRF, set the speed and emission power
129      nrf_InitAndConfig();
130
131      Serial.println(F("Relay turns ON / OFF by typing o
132      Serial.println(F(""));
133
134      pending_ACK = false;
135      on_off = false;
136
137      // clear possible garbage on Serial:
138  }

```

COM5

```

COMMAND's NRF radio module initialised.
Relay turns ON / OFF by typing o / f.

USER wants relay to switch ON...
Preparing the COMMAND to send command to REMOTE...
Command sent to the REMOTE...; awaiting for ACK...
Command ACK-ed ...
Relay ON now.

USER wants relay to switch OFF...
Preparing the COMMAND to send command to REMOTE...
Command sent to the REMOTE...; awaiting for ACK...
Command ACK-ed ...
Relay OFF now.

USER wants relay to switch ON...
Preparing the COMMAND to send command to REMOTE...
Command sent to the REMOTE...; awaiting for ACK...
Command ACK-ed ...
Relay ON now.

USER wants relay to switch OFF...
Preparing the COMMAND to send command to REMOTE...
Command sent to the REMOTE...; awaiting for ACK...
Command ACK-ed ...
Relay OFF now.

```

Arduino comandat (releu); mesaje pe Serial monitor:

```
COM3

REMOTE's NRF radio module initialised.
Listening to the COMMAND...
Data available at the REMOTE.
  Received command RELAY_ON.
  Preparing to respond back with ACK...
  Answered back with ACK...
  Listening again to the COMMAND...
Data available at the REMOTE.
  Received command RELAY_OFF.
  Preparing to respond back with ACK...
  Answered back with ACK...
  Listening again to the COMMAND...
Data available at the REMOTE.
  Received command RELAY_ON.
  Preparing to respond back with ACK...
  Answered back with ACK...
  Listening again to the COMMAND...
Data available at the REMOTE.
  Received command RELAY_OFF.
  Preparing to respond back with ACK...
  Answered back with ACK...
  Listening again to the COMMAND...
```

Partea de comandă ("COMMAND"), în loop(),

- așteaptă mai întâi ca USER ul să tasteze ceva.
- dacă s-a tastat 'o' sau 'f' înseamnă că,
 - s-a introdus o comandă validă;
 - se apelează funcția `send_RemoteCommand()` care trimite comanda `RELAY_ON` (sau `RELAY_OFF`) către partea comandată ("REMOTE");
 - se așteaptă confirmarea comenzii de către partea comandată (apelul funcției `await_ACK()`).
- se curăță seriala de datele introduse (comandă validă sau garbage).

Partea comandată, în loop(),

- așteaptă comanda de la "COMMAND";
- când sosesc date, dacă ele reprezintă o comandă validă se acționează corespunzător releul după care se trimite feedback părții de comandă (apelul funcției `send_ACK()`).

Evident, s-au prevăzut și o serie de mesaje de debug pe Serial monitor, atât pe partea de comandă cât și pe partea comandată.

De reținut:

- **grija pentru coerența datelor:**
 - apelul funcției `flush_rx()` după fiecare dată recepționată (citită) respectiv apelul funcției `flush_tx()` după fiecare dată trimisă (scrisă), atât pentru partea de comandă cât și pentru partea comandată.
 - după fiecare mesaj introdus de la tastatură, se curăță seriala.
- **stabilirea lungimii mesajelor schimbate de transceivere între ele** (funcția `setPayloadSize()`). Între transceiverele NRF24L01 se pot schimba mesaje de maximum 32 bytes. Dacă nu se apelează funcția `setPayloadSize(number of bytes)` care să limiteze lungimea mesajelor la o valoare mai mică, mesajele transmise vor

avea intotdeauna lungimea maximă – 32 bytes – chiar dacă dorim să transmitem un numar mai mic de bytes. Cum durata de transmisie este influențată și de lungimea mesajelor (*data utilă, payload*), vom stabili lungimea acestora cu ajutorul funcției `setPayloadSize()`, apelată înainte de transmisie / recepție, ori de cate ori este posibil.

- durata transmisiei¹⁷ este influențată și de **numărul maxim de retransmisii respectiv de gap ul dintre acestea** (*în cazul unei transmisii esuate*). Se poate fixa numărul maxim de retransmisii si distanța temporală dintre acestea cu ajutorul funcției `setRetries(n, m)`, unde gap ul inter retransmisii este $250 * n$ [microsecunde] respectiv m este numărul maxim de retransmisii.
- **aceste transceivere sunt deosebit de sensibile la zgomot electric** (zeci KHz – sute KHz) **ce se suprapune peste tensiunea de alimentare**. Dacă după ce s-au verificat conexiunile cu Arduino și codul Arduino, comunicația inter – RF uri nu se poate realiza (*transmisii eşuate*) sau se realizeaza respectiv esuează intermitent, cel mai probabil există surse de zgomot electric în imediata apropiere¹⁸.

¹⁷ Doar pentru ca o transmisie s-a efectuat cu succes (*funcția write()*) nu înseamna ca mesajul transmis a ajuns nealterat la destinație. Exista o serie de factori ce pot influența calitatea comunicației între transceivere NRF24I01.

¹⁸ Alimentarea plăcii Arduino de la un încărcător de smartphone plasat la mai puțin de 50 – 70 cm de modulul de RF, aproape sigur va cauza functionarea defectuoasă a transmisiei RF.

8. Transceiverul HC05 (Bluetooth)

HC05 este un modul Bluetooth cu rază mică de acoperire, cca. 10 m în aer liber.



Pentru conectarea la Arduino, avem următoarea interfață:

- EN, conectat la pinul D2,
- VCC, conectat la pinul +5v,
- GND, conectat la pinul GND,
- TX, conectat la pinul D8,
- RX, conectat la pinul D7, prin intermediul unui divizor rezistiv cu raport 2/3 (nivelul de tensiune acceptat de HC05 pentru palierul de '1' este de cca 3.3 V ori nivelul de '1' al Arduino este de cca 4.8 – 5 V).
- STATE, neconectat.

NOTĂ: modulul HC05 are interfață UART drept pentru care ar putea fi conectat la interfața UART fizică a plăcii Arduino (pinii 0 și 1). Totuși, deoarece seriala este grefată pe UART-ul fizic al plăcii Arduino, conectarea transceiverului Bluetooth HC05 pe interfața fizică UART Arduino ar face imposibilă utilizarea Serial monitor. Soluția este utilizarea bibliotecii SoftwareSerial.h care extinde interfața fizică UART Arduino.

Trebuie menționat că, **pentru utilizarea modulelor HC05 acestea trebuie mai întâi configurate**. Trebuie configurat rolul (MASTER / SLAVE), adresa MAC, baudrate-ul, șamd. Configurarea acestor module se va face în cadrul unei sesiuni dedicate. Pentru a aduce un modul HC05 în modul AT (configurare) se va ține apăsat butonul (vezi imaginea de mai sus) după care se va conecta placa Arduino la computer; pe Arduino se va instala în prealabil un program simplu care citește inputul de la tastatură (Serial monitor) și afișează date de la HC05 pe Serial monitor. **Dialogul cu HC05 în modul AT trebuie desfășurat la 38400 baud!**

Un ultim detaliu: în cazul în care Serial monitor nu se utilizează, conectarea HC05 la Arduino se poate face, după cum am menționat, direct la portul UART fizic al plăcii Arduino. În acest caz, la încărcarea programului în memoria Arduino se va deconecta în prealabil pinul UART RX urmand ca după încărcarea programului în memorie să se reconecteze pinul respectiv; acest lucru se face pentru a evita conflict pe Seriala.

Următorul program trimite datele introduse de la tastatură (Serial monitor) la celălalt capăt al conexiunii wireless (HC05 receptor are Serial monitor deschis). Rolurile celor doi "actori" care comunică se pot interschimba (unul trimite, celălalt recepționează și invers).

Schimbul de roluri se face introducand de la Serial monitor (*cel care transmite*) cuvantul TAKEOVER după care se da Enter.

```
// this is a bit clever example of bluetooth communication as master and slave can switch
// their roles on the fly. Only the current master can initiate this roles swapping; the USER
// must type "TAKEOVER" then press Enter in the Serial monitor.

#include <SoftwareSerial.h>

#define HC05_EN_PIN    2
#define HC05_TX_PIN    7
#define HC05_RX_PIN    8

SoftwareSerial HC05_BLT(HC05_RX_PIN, HC05_TX_PIN);    // the bluetooth object.

const String CODEWORD = "TAKEOVER";
char last_8chars[9] = "abcdefgh";    // the last_8chars array includes the '\0' character.
char current_char;

bool HC05_role;

void reset_CharArray(){
    last_8chars[0] = 'a';
    last_8chars[1] = 'b';
    last_8chars[2] = 'c';
    last_8chars[3] = 'd';
    last_8chars[4] = 'e';
    last_8chars[5] = 'f';
    last_8chars[6] = 'g';
    last_8chars[7] = 'h';
}

void setup() {
    pinMode(HC05_EN_PIN, OUTPUT);

    // bring the bluetooth module in communication mode:
    digitalWrite(HC05_EN_PIN, LOW);

    // the master and slave HC05 speed have been "a priori" set to 38400bauds:
    Serial.begin(38400);
    HC05_BLT.begin(38400);
}
```

```

// initially, this is a master (for the slave, make HC05_role false):
HC05_role = false; // true;
if(HC05_role == true){
    Serial.println(F("Everything you type here will be transmitted at the other end "));
    Serial.println(F("of the wireless channel."));
}
else{
    Serial.println(F("This side of the wireless channel receives data from the other end."));
}

// clear the serial:
while(Serial.available()>0)
    Serial.read();
}

void loop() {

    if(HC05_role == true){          // it's the master.

        if(Serial.available()){

            // read the character from Serial:
            current_char = (char) Serial.read();

            // shift the current char left into the 8 characters array:
            for(int i = 0; i < 7; i++)
                last_8chars[i] = last_8chars[i+1];
            last_8chars[7] = current_char;

            // test a possible role switch:
            String temp8 = String(last_8chars);
            if(temp8.compareTo(CODEWORD) == 0){

                // swapp roles:
                HC05_role = false;

                // clear the shift register content:
                reset_CharArray();
            }

            HC05_BLT.write(current_char);
        }
    }
}

```

```

    }
    else{                                     // it's the slave.
        if(HC05_BLT.available()){
            // read the character from bluetooth:
            current_char = (char) HC05_BLT.read();

            // shift the current char left into the 8 characters array:
            for(int i = 0; i < 7; i++)
                last_8chars[i] = last_8chars[i+1];
            last_8chars[7] = current_char;

            // test a possible role switch:
            String temp8 = String(last_8chars);
            if(temp8.compareTo(CODEWORD) == 0){

                // swapp roles:
                HC05_role = true;

                // clear the shift register content:
                reset_CharArray();
            }

            Serial.write(current_char);
        }
    }
}

```

Trebuie menționat că cele două module HC05 au fost “a priori” interogate și configurate într-o sesiune dedicată. Pentru fiecare in parte:

- s-a citit adresa fizică,
- s-a configurat rolul (*MASTER / SLAVE*),
- s-a configurat baudrate ul(*38400, cu 1 bit de stop, fără CRC*),
- s-a configurat adresa de binding (*fiecare are adresa celuilalt*),
- s-a configurat modul de operare.

// HC05 interrogation commands:

```

//
// AT?                                     // see if HC05 can be interrogated.
// AT+ADDR?                               // see the HC05's MAC address.
// AT+ROLE?                               // 0 / 1 means SLAVE / MASTER.
// AT+UART?                               // see HC05's speed, stop and parity bits.
                                           // default value is 9600, 0, 0.
// AT+BIND?                               // see the binding address.
// AT+CMODE?                              // 0 means connect the module to the specified address (

```

// through the BIND command).

```
// HC05 configuring commands:
//
// AT+ADDR=0021:04:082C86
// AT+ROLE=1
// AT+UART=38400,0,0
// AT+BIND=0021:04:082CE2
// AT+CMODE=0
```

```
#include <SoftwareSerial.h>
```

```
#define MASTER_EN_PIN 2
#define MASTER_TX_PIN 7
#define MASTER_RX_PIN 8
```

```
SoftwareSerial master_BLT(MASTER_RX_PIN, MASTER_TX_PIN);
```

```
void setup() {
```

```
    pinMode(MASTER_EN_PIN, OUTPUT);
```

```
    // bring the bluetooth module in AT mode for
```

```
    // interrogation:
```

```
    digitalWrite(MASTER_EN_PIN, HIGH);
```

```
    // you could only interrogate the HC05 in low baudrate.
```

```
    // however, it does not work in 9600 bauds; use 38400 instead:
```

```
    Serial.begin(38400);           // communication rate with Serial Monitor.
```

```
    master_BLT.begin(38400);       // communication rate with HC05.
```

```
    Serial.println("...observe the LED blinking at every 2 seconds.");
```

```
    Serial.println(F("That means the HC05 is in AT mode; you can question it.));
```

```
    while(Serial.available()>0)
```

```
        Serial.read();
```

```
}
```

```
void loop() {
```

```
    if(Serial.available())
```

```
        master_BLT.write(Serial.read());
```



```

    if(master_BLT.available())
        Serial.write(master_BLT.read());
}

```

De pe laptop ul HC05 Master, s-a introdus de la tastatură (*Serial monitor*) textul:

"I'm the MASTER.

As long as I'm the MASTER I can send messages to the other end of the channel.

Do you see?

I can become the SLAVE and give the ownership to the other end by typing TAKEOVER."

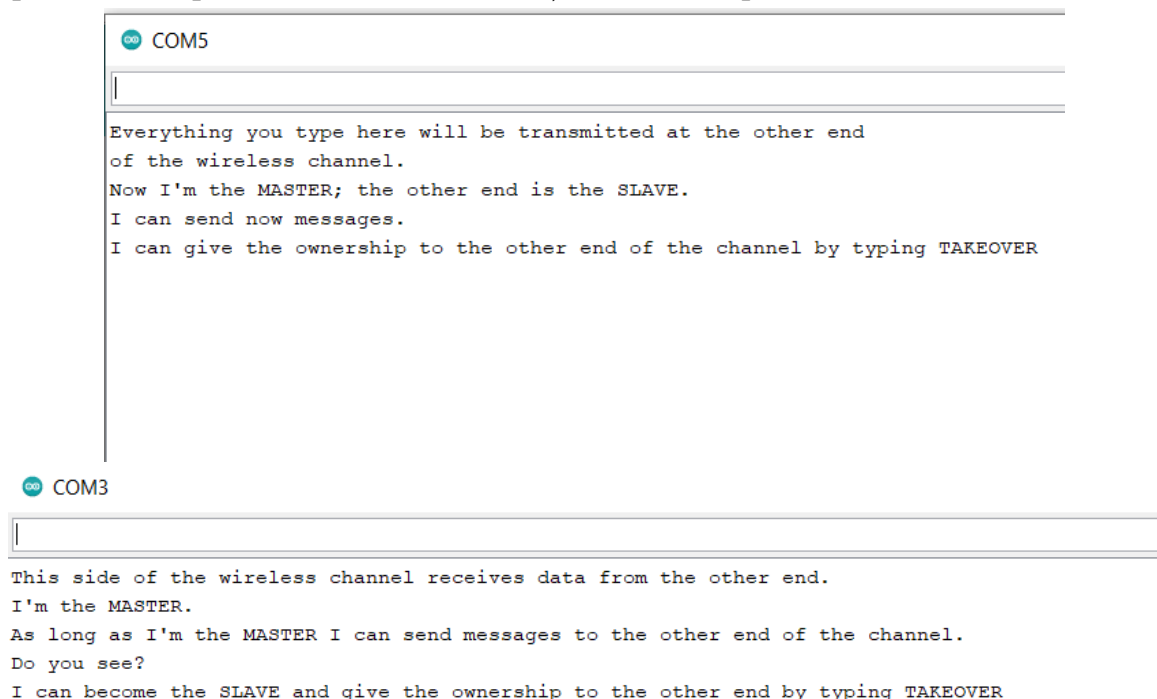
Se poate vedea în poza a doua (COM3) că laptop ul HC05 Slave a recepționat textul menționat. De asemenea, Slave ul receptionand TAKEOVER a devenit Master iar Masterul de la care s-a introdus TAKEOVER a devenit Slave (*rolurile s-au schimbat*). De la noul Master s-a introdus textul:

"Now I'm the MASTER; the other end is the SLAVE.

I can send now messages.

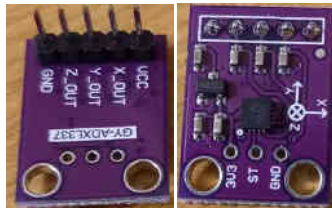
I can give the ownership to the other end of the channel by typing TAKEOVER."

În poza întâi se poate vedea că textul menționat s-a recepționat.



9. Accelerometrul GY-ADXL-337. Aplicație: măsurarea unghiului.

GY-ADXL-337 este un sensor MEMS cu ieșire analogică, de la Analog Devices.



Senzorul se alimentează de la +3.3V; nu se poate configura domeniul de măsură al accelerațiilor pentru cele 3 axe. Utilizarea acestui senzor este simplă, ieșirea fiind analogică.

NOTĂ: pentru fiecare dintre cele 3 axe trebuie aflate limitele între care variază datele de la senzor. Acest lucru se poate face în cadrul unei sesiuni dedicate (off line) pentru măsurarea parametrilor senzorului.

Următorul program utilizează accelerometrul ADXL337 pentru măsurarea statică a doua unghiuri (rotație în jurul axei Y, rotație în jurul axei X):

```
// This is a tilt angle measurement instrument. It uses the GY-ADXL-337 sensor.
// To measure the inclination angle from the Z axis, the Zout and Xout accelerometer data are
// used; the static angle is computed as yRotAngle = atan (Xout / Zout).
//
// After experimenting with the sensor, I found the following values:
//
//                                     Axis pointing down   Axis pointing up
// X (Analog to Int value)           278                   418
// Y (Analog to Int value)           275                   413
// Z (Analog to Int value)           286                   425

#include <math.h>
#include <TimerOne.h>

int ADXL337_XOUT_PIN = A2;           // Sensor outputs, XOUT, YOUT, ZOUT
int ADXL337_YOUT_PIN = A3;           // are connected to Arduino's
int ADXL337_ZOUT_PIN = A4;           // analog pins A2, A3, A4.

const float XADI_MIN = 278.0;         // means -1.0 g
const float XADI_MAX = 418.0;         // means +1.0 g
const float YADI_MIN = 275.0;         // the
const float YADI_MAX = 413.0;         // same
const float ZADI_MIN = 286.0;         // for the
const float ZADI_MAX = 425.0;         // rest of them.
const float X_SLOPE = 2.0 / (XADI_MAX - XADI_MIN);
```

```

const float Y_SLOPE = 2.0 / (YADI_MAX - YADI_MIN);
const float Z_SLOPE = 2.0 / (ZADI_MAX - ZADI_MIN);
const double RAD_2_DEG = 180.0 / 3.1415;    // radians to degrees.

float xG;                                // hold sensor values
float yG;                                // mapped into
float zG;                                // the [-1.0, +1.0] g range.

float xADI;                              // these are sensor
float yADI;                              // values obtained with
float zADI;                              // analogRead().

double xRotAngle;
double yRotAngle;

long interr_period_us = 100000;           // interrupts triggers every 0.1 seconds.
bool tick;

void getSensorADIData(){
    xADI = analogRead(ADXL337_XOUT_PIN);
    yADI = analogRead(ADXL337_YOUT_PIN);
    zADI = analogRead(ADXL337_ZOUT_PIN);
}

// map analog to integer sensor values into the [-1.0, +1.0] g range:
void mapSensorADIData(){
    xG = -1.0 + X_SLOPE * (xADI - XADI_MIN);
    yG = -1.0 + Y_SLOPE * (yADI - YADI_MIN);
    zG = -1.0 + Z_SLOPE * (zADI - ZADI_MIN);
}

void computeRotAngles(){
    yRotAngle = (atan2((double) xG, (double) zG)) * RAD_2_DEG;
    xRotAngle = (atan2((double) yG, (double) zG)) * RAD_2_DEG;
}

void displayRotAngles(){
    Serial.print(yRotAngle, 2); Serial.print(F(" "));
    Serial.print(xRotAngle, 2); Serial.println(F(""));
}

void Timer1_ISR(){

```

```

    tick = true;
}

void setup() {

    Serial.begin(115200);
    tick = false;

    // mount Timer 1 interrupt (triggered once every sample period):
    Timer1.initialize(interr_period_us);
    Timer1.attachInterrupt(Timer1_ISR);
}

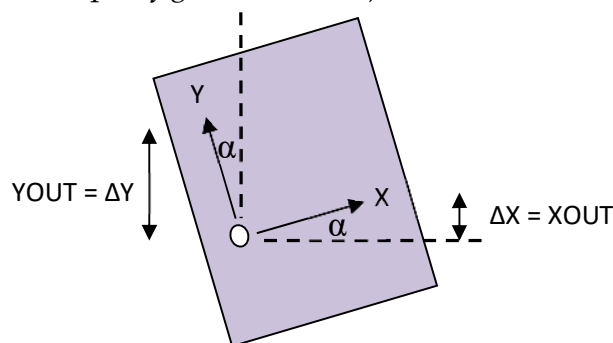
void loop() {

    if(tick == true){

        getSensorADIData();           // read sensor data.
        mapSensorADIData();           // map sensor data.
        computeRotAngles();           // compute static angles.
        displayRotAngles();           // display angles.
        // reset tick:
        tick = false;
    }
}

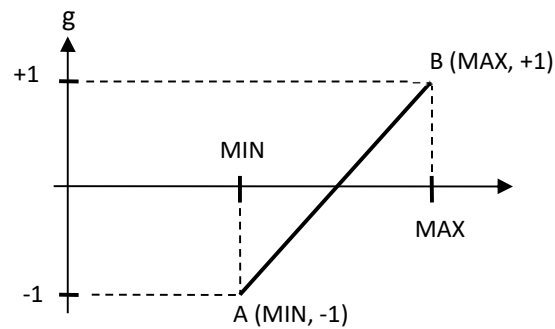
```

Programul este foarte simplu: în `loop()`, o dată la fiecare 0.1s variabila `tick` (setată de intreruperea de la `Timer1` montată în funcția `setup()`) este gasită `true`. Când se întâmplă asta, se citesc datele de la senzor, se mapează aceste accelerații în intervalul $[-1.0, +1.0]$ g după care se calculează unghiurile respective și se afișează (după afișare se resetează variabila `tick`). De menționat că, calculul unghiurilor se face cu relația respectivă – vezi `computeRotAngles()` – doar în ipoteza măsurării unghiurilor statice (sau cu variație lentă în timp). Principiul este simplu (figura următoare):



Conform figurii, $X_{OUT} = a \sin(\alpha)$, $Y_{OUT} = b \cos(\alpha)$, ceea ce implică $X_{OUT}/Y_{OUT} = (a/b) \tan(\alpha)$, de unde obținem unghiul (în codul arduino am considerat $a = b$ ceea ce s-ar putea să nu fie chiar riguros corect).

Cum anume facem maparea intervalelor $[MIN, MAX]$ între care variază datele de la senzor (fiecare axă are un MIN respectiv un MAX) în intervalul $[-1.0g, +1.0g]$? Pentru aceasta vezi figura următoare:



Evident, ecuația segmentului AB este:

$$y = -1.0 + \text{panta} * (x - \text{MIN}),$$

$$\text{panta} = 2 / (\text{MAX} - \text{MIN}),$$

$$x \in [\text{MIN}, \text{MAX}].$$