

# 1. Introducere

---

Introducere în limbajul C  
Arduino Software (IDE)  
Arduino Hardware

---

Dezvoltarea aplicațiilor care au la bază un microcontroler a crescut în ultimii ani, în special datorită introducerii de noi platforme integrate. Una dintre acestea este și platforma cu microcontroler *Arduino*. Aceasta permite dezvoltarea de aplicații software pentru microcontrolere într-un mod mult mai simplu decât în variantele clasice. Acest lucru se datorează în primul rând integrării de funcționalități specifice, care permit achiziția de informații din mediul exterior și reacția corespunzătoare a platformei.

*Arduino* este platformă electronică gratuită (eng. open-source) bazată pe un mecanism software și hardware ușor de utilizat. Plăcile de dezvoltare de acest fel permit citirea de date de la diferite tipuri de intrări (de ex.: senzori de lumină, butoane, informații de pe internet) și le transformă în mărimi de ieșire, care pot controla motoare, aprinde LED-uri sau publică date pe internet.

Locul unde a fost dezvoltat pentru prima oară *Arduino*, în 2003, este *Ivrea Interaction Design Institute* (în Italia). Scopul a fost acela de a se realiza prototipuri, cât mai repede și mai ușor, de către studenți fără cunoștințe de programare sau electronică. Odată câștigată popularitatea, placile de dezvoltare Arduino au început să fie adaptate și utilizate în diferite produse sau servicii: Internet of Things (IoT), imprimante 3D, sisteme embedded complexe.

## 1.1 Introducere în limbajul C

Programarea microcontroler poate fi realizată în diferite limbiage, cel mai răspândit fiind *Limbajul C*. O extensie a acestui limbaj este *Embedded C*, care este orientat spre platformele cu microcontroler. Aceasta conține facilități complexe de accesare a blocurilor de memorie sau de manipulare a mărimilor de intrare/ieșire. În continuare se vor presta informații utile pentru înțelegerea și implementarea lucrărilor de laborator prezentate în acest îndrumar.

### 1.1.1 Directive preprocesor

Aceste directive încep cu caracterul # și sunt interpretate înainte de a se compila programul propriu-zis (compus din instrucțiuni și declarații). În mod frecvent sunt întâlnite două directive [9]:

- directiva *# define*: permite folosirea constantelor simbolice în programe. La compilare, fiecare apariție a simbolului este înlocuită, caracter cu caracter, cu valoarea sa. O astfel de directivă nu se termină cu ';;'
- directiva *# include*: permite includerea unui fișier sursă în alt fișier sursă. Aceste fișiere sunt de obicei fișiere numite *antet* (eng. *header*), ce reunesc declarații de funcții standard.

### 1.1.2 Tipuri de date

Principalele tipuri de date în limbajul C sunt împărțite în trei categorii:

- tipuri de bază: sunt tipuri de date aritmetice și pot fi clasificate în tipuri de date întregi și tipuri de date reale (în virgulă mobilă);
- tipuri derive: acestea includ: tip pointer, tip tablou, tip structură, tip uniune și tip funcție;
- tipul *void*: reprezintă lipsa unui tip de date.

În tabelul 1.1 sunt prezentate tipurile de date întregi, dimensiunea ocupată în memorie și intervalul de valori pe care îl poate lua [8].

Tab. 1.1 Date de tip întreg.

Tip	Valori	Dimensiune în memorie
char	-128 ÷ 127	1 byte
unsigned char	0 ÷ 255	1 byte
signed char	-128 ÷ 127	1 byte
int	-32768 ÷ 32767	2 bytes
unsigned int	0 ÷ 65535	2 bytes
long	-2,147,483,648 ÷ 2,147,483,647	4 bytes

Detalii legate de tipurile de date reale, împreună cu dimensiunea ocupată în memorie și cu valorile posibile sunt prezentate în tabelul 1.2. Ambele reprezentări au aceeași gamă de valori pentru plăcile de dezvoltare utilizate în cadrul laboratorului.

Tab. 1.2 Date de tip real.

Tip	Valori	Dimensiune în memorie
float	1.1E-38 ÷ 3.4E+38	4 byte
double	2.2E-308 ÷ 1.7E+308	8 byte

### 1.1.3 Variabile și constante

*Variabila* este un identificator asociat cu o locație de memorie în care se stochează valori care pot fi modificate pe parcursul existenței ei. Fiecare variabilă are un tip asociat, care

determină dimensiunea ocupată în memorie, valorile pe care le poate lua și setul de operații care se poate aplica asupra variabilei.

Numele variabilei poate fi compus din litere, cifre și caracterul ”\_”. Numele poate începe cu literă sau ”\_”. Limbajul C face distincție între majuscule și minuscule. Tipul de date pe care o variabilă poate să îl preia a fost prezentat în secțiunea anterioară.

În următorul program sunt prezentate o serie implementări:

---

```

1 int i, j, k;
2 char c, ch;
3 float f, salariu;
4 double d;
```

---

În exemplul anterior au fost declarate și definite variabilele *i*, *j*, *k* ceea ce a condus la crearea de către compilator a variabilelor *i*, *j*, *k* de tip *int*.

De asemenea, variabilele pot fi și inițializate (se atribuie o valoare inițială) la declararea lor. De exemplu:

---

```

1 int d = 3, f = 5; //definire si initializare d si f
2 char x = 'x';     // variabila x primește valoarea 'x'.
```

---

*Constantele* [9] reprezintă valori fixe (numerice, caractere sau siruri de caractere), care nu pot fi modificate în timpul execuției programului. Tipul constantei este determinat de compilator pe baza valorii și sintaxei utilizate, putând avea oricare dintre tipurile predefinite de date. Constantele rămân în memorie pe toată durata execuției programului.

În limbajul C sunt două moduri de a scrie o constantă:

- utilizând ”#define”;
- utilizând cuvântul cheie *const*.

În următorul exemplu sunt exemplificate ambele variante:

---

```

1 //varianta 1 utilizind #define
2 #define LUNGIME 10
3 #define INALTIME 78
4
5 //sau folosind cuvantul cheie const
6 const int LUNGIME = 10;
7 const int INALTIME = 78;
```

---

La variabilele prezentate anterior se poate adăuga și un identificator care să prezinte scopul (vizibilitatea) cât și durata de existență a lor. Dintre cei mai utilizați identificatori, din limbajul C, se amintește [8]:

- *register*: permite definirea de variabile locale care pot fi stocate în registrii, în loc de memoria RAM (depinde și de configurația HW). Astfel, variabila are dimensiunea maximă egală cu dimensiunea registrului. Se utilizează pentru variabile care necesită acces rapid (de exemplu numărătoare);

- *static*: variabilele statice sunt create o singură dată atunci când codul specific funcției în care au fost declarate este încărcat în memorie și nu sunt distruse decât atunci când acest cod este eliminat din memorie (o astfel de variabilă rămâne în memorie atâtă timp cât rulează programul);
- *extern*: permite crearea unei referințe globale, între toate fișierele program. Astfel, variabilele de acest tip sunt vizibile în toate fișierele programului.

#### 1.1.4 Operatori și expresii

*Operatorii* sunt simboluri care, aplicate unor operanzi (date - constante, variabile, expresii), au ca efect realizarea unor operații matematice sau logice, care returnează un rezultat. O succesiune validă de operanzi și operatori formează o *expresie*. Următoarele tipuri de operatori sunt disponibili în limbajul C: aritmetici, relaționali, logici, la nivel de bit, de atribuire.

În tabelul 2.1 sunt prezentate unii dintre operatorii utilizați în exemplele din cadrul acestui îndrumar.

Tab. 1.3 Operatori ai limbajului C.

Operator	Descriere	Exemplu
$+, -, *, \backslash$	Operatori matematici de bază	$A=1, B=3: A * B = 3$
$++, --$	Incrementare sau decrementare	$A=4: A++ = 5$
$==$	Verifică dacă valorile operanzilor sunt egale sau nu. Dacă da, condiția e adevărată	$A=1, B=3: (A == B)$ este fals
$!=$	Verifică dacă valorile operanzilor sunt egale sau nu. Dacă nu, condiția e adevărată	$A=1, B=3: (A != B)$ este adevarat
$>, <, >=, <=$	Verifică dacă valoarea este: mai mare, mai mică, mai mare și egală și mai mică și egală	$A=5, B=3: (A >= B)$ este adevarat
$\&\&,   , !$	Operator: SI logic, SAU logic și negație	$A=1, B=0: (A    B)$ este adevarat
$\sim$	Operatorul complement pe bit: inversează reprezentarea sirului de biți	$A=1: \sim A=0$
$\&,  , \wedge, <<, >>$	Operatori pe biți (SI, SAU, SAU exclusiv, siftare la stânga, siftare la dreapta)	$A=60, A<< 2 = 240$ în binar 11110000
$+ =, - =, * =, / =$	Operatori de atribuire	$C += A$ este echivalent cu $C = C + A$

Pe lângă operatorii prezențați anterior se mai pot enumera și cei din tabelul 1.4:

#### 1.1.5 Instrucțiuni

O instrucțiune reprezintă o porțiune de cod care odată executată permite efectuarea unei acțiuni: atribuire, selecție, iterare, etc. Instrucțiunile utilizează în general conceptul

Tab. 1.4 Operatori ai limbajului C.

Operator	Descriere	Exemplu
<code>sizeof()</code>	Returnează dimensiunea unei variabile	int A: <code>sizeof(A)</code> este 2
<code>&amp;</code>	Returnează adresa unei variabile	<code>&amp;A</code> returnează adresa variabilei
<code>*</code>	Pointer la o variabilă	<code>*A</code>
<code>?:</code>	Dacă, condiția este adevarată ? atunci valoarea X : daca nu valoarea Y	if (A>B)? C=5 : C=10;

de adevărat și fals. Cele mai utilizate instrucțiuni sunt cele de decizie (condiționale), de selecție și cele repetitive.

### Instrucțiuni de decizie

Acest tip de instrucțiuni sunt introduse utilizându-se cuvântul cheie `if`, astfel avem următoarele sintaxe generale:

Algoritmul 1.1 Varianta 1

---

```

1 if (expresie)
2     Instructiune; //Daca expresie este adevarata se executa
    ↪ Instructiune

```

---

Algoritmul 1.2 Varianta 2

---

```

1 if (expresie)
2     Instructiune1; //Daca expresie este adevarata se executa
    ↪ Instructiune1
3 else
4     Instructiune2; //Daca expresie nu este adevarata se executa
    ↪ Instructiune2

```

---

Următorul exemplu prezintă utilizarea celor două variante:

Algoritmul 1.3 Utilizare instrucțiuni de decizie

---

```

1 int nVarsta = 5;
2 int nContor = 1;
3 if (nVarsta > 3)
4     nContor++; //Expresia este adevarata , deci se executa linia
    ↪ nContor++
5 else
6     nContor--; //Conditia a fost adevarata , deci nu se executa
    ↪ aceasta linie

```

---

### Instrucțiuni de selecție

Pentru a se realiza o selecție multiplă (în situația în care există mai multe cazuri posibile), se poate folosi o succesiune de instrucțiuni `if` (incluse unele în altele) sau se poate folosi o

instructiune de tipul *switch*. Aceasta din urmă permite introducerea unei enumerări a cazurilor posibile și o expresie de selecție. Forma generală este:

---

Algoritmul 1.4 Instrucțiunea *switch* - sintaxă generală

---

```

1 switch ( expresie ) {
2   case c1:
3     instructiune1 ;
4     break ;
5   case c2:
6     instructiune2 ;
7     break ;
8   ...
9   default :
10    instructiune_default ;
11    break ;

```

---

La început se evaluează *expresia*, iar dacă se găsește în ceea ce precede cuvântul *case* expresia respectivă, se execută codul corespunzător până la linia *break*. Dacă nu este regăsită expresie în nici un *case*, atunci se execută *default*. Această variantă implicită, *default*, nu este obligatorie, dar este indicată întrucât poate trata situații neașteptate.

Utilizarea instrucțiunii *switch* este arată în exemplul următor:

---

Algoritmul 1.5 Exemplu de utilizare a instrucțiunii *switch*

---

```

1 char operatie;
2 int nNr1 , nNr2 , rezultat ;
3 .....
4 switch ( operatie ) {
5   case '+':
6     rezultat = nNr1 + nNr2 ;
7     break ;
8   case '-':
9     rezultat = nNr1 - nNr2 ;
10    break ;
11   case '*':
12     rezultat = nNr1 * nNr2 ;
13     break ;

```

---

## Instrucțiuni repetitive

În limbajul C există trei tipuri de instrucțiuni repetitive: *for*, *while* și *do-while*.

Instrucțiunea *while* este una repetitivă, cu o condiție inițială și cu un număr necunoscut de etape. Sintaxa generală este:

---

Algoritmul 1.6 Instrucțiunea *while* - sintaxă generală

---

```

1 while ( expresie )
2     Instructiune; //Atata timp cat expresie este adevarata se
      ↳ executa Instructiune

```

---

Expresia trebuie să conțină o variabilă care este modificată (actualizată) interiorul buclei *while*. În acest fel, expresia poate să devină falsă și bucla să se oprească. De exemplu:

---

Algoritmul 1.7 Instrucțiunea *while* - exemplu

---

```

1 int nConditie = 1;
2 while ( nConditie < 10 )
3     nConditie++; //Atata timp cat expresie este mai mica de 10 se
      ↳ executa incrementarea

```

---

O buclă cu condiția de forma *while*(1) este o buclă infinită.

Instrucțiunea *for* este de asemenea una repetitivă, cu condiție inițială și cu un număr cunoscut de pași. Sintaxa sa generală este:

---

Algoritmul 1.8 Instrucțiunea *for* - sintaxă generală

---

```

1 for ( expresie1 ; expresie2 ; expresie3 )
2     Instructiune ;

```

---

Instrucțiunea se execută în următoarele etape:

- expresie1: este evaluată la intrarea în buclă;
- expresie2: este evaluată înaintea fiecărei iterări și reprezintă condiția de execuție a instrucțiunii; valoarea logică falsă a ei provoacă ieșirea din instrucțiune;
- expresie3: se evaluatează la sfârșitul fiecărei iterări, pentru a se actualiza parametrii instrucțiunii.

Un exemplu de utilizare este următorul:

---

Algoritmul 1.9 Instrucțiunea *for* - exemplu

---

```

1 int i , nRezultat ;
2 for ( i = 0 ; i < 10 ; i++ )
3     nRezultat++; //se repeta operatia de incrementare pana i = 9 ;

```

---

Instrucțiunea *do-while* este utilizată atunci când se dorește utilizarea unei condiții finale, iar ciclul să se execute cel puțin o dată. Acest lucru este diferit față de instrucțiunile repetitive *while* sau *for*. Sintaxa generală este:

---

Algoritmul 1.10 Instrucțiunea *do-while* - sintaxă generală

---

```

1 do {
2     Instructiune ;
3 } while ( expresie );

```

---

Se execută *instructiune* apoi este evaluată *expresie*. În funcție de aceasta se mai execută sau nu *instructiune*. Un exemplu de utilizare este următorul:

---

Algoritmul 1.11 Instrucțiunea *do-while* - exemplu

---

```

1 int nVar = 0, i = 10;
2
3 do{
4     nVar++; //Se executa aceaste instructiuni pana i ajunge la
        ↪ valoarea 0
5     i--;
6 }while (i != 0);

```

---

### 1.1.6 Tablouri de date

Tablourile de date reprezintă liste care pot stoca colecții fixe de elemente de același fel. Un tablou de date poate fi văzut și ca o colecție de variabile de același fel. Accesul la un element al unui tablou se face folosind unul sau mai mulți indici. În memorie, tablourile ocupă locații contigute. De asemenea, tablourile pot avea una sau mai multe dimensiuni.

Sintaxa de declarare a unui tablou de date în C, cu un anumit tip de date conținut este:

---

Algoritmul 1.12 Tablou unidimensional de date - sintaxă generală

---

```
1 tip nume_tablou_date [dimensiune];
```

---

Declararea acestui tablou uni-dimensional cuprinde dimensiunea sa, care trebuie să fie de tip întreg (mai mare decât zero), iar tipul este oricare dintre tipurile de date acceptate de limbajul C și prezentate în secțiunea § 1.1.2. De exemplu, pentru a se declara un tablou numit DateT cu 10 elemente de tip float, se poate folosi:

---

Algoritmul 1.13 Tablou unidimensional de date - exemplu

---

```

1 float DateT[10]; //declarare tablou unidimensional cu 10 elemente
    ↪ de tipul float

```

---

Declararea unui tablou de date se poate face și cu inițializare, după cum se arată în următorul exemplu.

```

1 float DateT[10]; //declarare tablou fara initializare
2 int DataI[3]={5,7,12}; //declarare tablou cu initializare

```

---

În cadrul tablourilor de date se mai pot folosi tablouri bidimensionale dar și declararea fără a se specifica numărul de elemente (atunci când se realizează inițializarea elementelor tabloului). Următorul exemplu ilustrează acest lucru.

---

Algoritmul 1.14 Tablou bidimensional de date - exemplu

---

```

1 float DateTI[10][3]; //declarare tablou bidimensional cu 10*3
    ↪ elemente , cu date de tip float

```

---

```
2 float vect[ ] = {1.5, 3.2, 7.1, -2.5, 6}; // declarare fara a se
    ↪ preciza dimensiunea
```

---

Tablourile unidimensionale care conțin elemente de tip char sunt folosite pentru memorarea șirurilor de caractere. Pentru a marca sfârșitul unui șir de caractere, după ultimul caracter se adaugă un octet ('\0') numit *terminator de șir*. Declararea unui șir de caractere se face la fel ca la declararea tablourilor unidimensionale, cu date de tip char.

Algoritm 1.15 Tablou bidimensional de date - exemplu

---

```
1 char numeSir[6]; // declarare sir de caractere fara initializare
2 char numeSirNou[3] = { 'I', 'R', '\0'}; // declarare sir cu
    ↪ initializare
3 char anotimp[5] = "Vara";
```

---

### 1.1.7 Funcții

O funcție reprezintă un grup de declarații care împreună pot realiza o anumită sarcină. Limbajul C trebuie să conțină cel puțin o funcție, iar aceasta se numește *main()*.

Utilitatea introducerii funcțiilor provine de la posibilitatea împărțirii codului după anumite funcționalități, alese de fiecare programator după necesitate. În mod logic, aceste funcții trebuie să realizeze o sarcină specifică.

O funcție are două componente:

- *declarația* funcției: prin aceasta se informează compilatorul despre numele funcției, tipul de dată returnat și parametrii;
- *definiția* funcției: reprezintă corpul efectiv al funcției, în care se implementează funcționalitatea acesteia.

Sintaxa generală a unei funcții definite în C este:

Algoritm 1.16 Funcție - sintaxă generală

---

```
1 tip_returnat nume_functie(lista_parametri)
2 {
3     corpul_functiei
4     .....
5 }
```

---

Funcțiile în limbajul C pot să conțină unele dintre următoarele:

- tip\_returnat: este un tip oarecare de dată și reprezintă tipul rezultatului returnat de funcție. Dacă nu este specificat, implicit tipul rezultatului returnat este int. Pentru funcțiile care nu returnează rezultat trebuie să se specifice tipul void;
- nume\_funcție: este un identificator unic al funcției;
- parametrii funcției: reprezintă enumerarea declarațiilor parametrilor sub forma: tip\_parametru nume\_parametru. Tipul parametrului poate fi orice tip valid de date;

- corpul funcției: conține o colecție de instrucțiuni care definesc scopul efectiv al unei funcții.

Următorul exemplu ilustrează definiția unei funcții care calculează maximul dintre două numere, primite ca și parametru. Funcție returnează valoarea maximă dintre cele două.

Algoritm 1.17 Implementare funcție care returnează valoarea maximă

---

```

1 int calculeazaMaxim( int nNr1 , int nNr2 )
2 {
3     //se declara o variabila locala
4     int nResult ;
5     //se verifica care numar este mai mare
6     if ( nNr1 > nNr2 )
7         nResult = nNr1 ;
8     else
9         nResult = nNr2 ;
10    //se returneaza numarul mai mare
11    return nResult ;
12 }
```

---

## 1.2 Arduino Software (IDE)

Mediul de dezvoltare integrat Arduino (sau Arduino IDE) conține un editor de text în care se poate dezvolta codul, o zonă în care sunt afișate mesaje, o consolă și o zonă cu diferite meniuuri. Acesta are capacitatea de a se conecta la partea hardware a modulului de dezvoltare Arduino, atât pentru a se încărca programele dezvoltate de utilizator, cât și pentru a se comunica cu acestea.

### 1.2.1 Instalare Arduino IDE

Editorul în care se vor scrie toate aplicațiile prezentate în acest îndrumar se poate descărca de la adresa: <https://www.arduino.cc/en/Main/Software>. Se alege varianta "Windows ZIP file for non admin install". Următorul pas presupune extragerea arhivei într-un director dorit și instalarea *arduino.exe*. După ce acest pas a fost efectuat, este necesară conectarea unui dispozitiv Arduino la unul dintre porturile USB ale calculatorului. După detectarea de către PC a placii Arduino, se poate selecta instalarea de drivere de pe PC, prin indicarea adresei unde ați făcut dezarchivarea. Din acest director se alege directorul *drivers*.

### 1.2.2 Utilizare Arduino IDE

Rularea executabilului instalat la pasul anterior va conduce la deschiderea unui editor precum cel din figura 1.1.

Programele scrise în Arduino IDE sunt numite **sketches**. Acestea sunt scrise în editorul Arduino și salvate cu extensia *.ino*.

În cadrul editorului de text, pot fi identificate 4 zone importante. Zona 1 corespunde

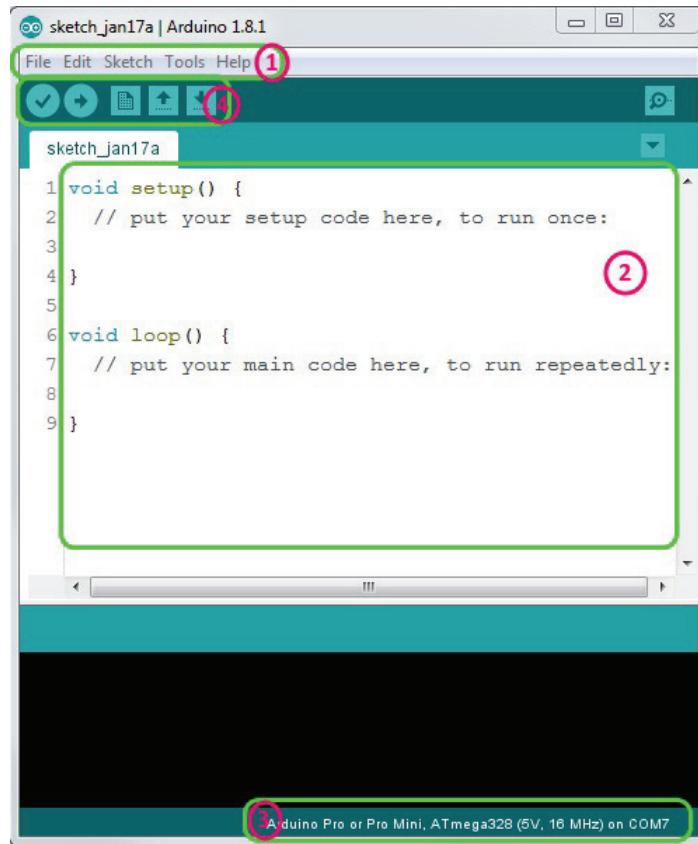


Fig. 1.1 Editorul de text pentru Arduino.

principalelor meniuri pe care le are IDE-ul: File, Edit, Sketch, Tools, Help. Funcționalitățile acestor meniuri sunt următoarele:

- *File*:
  - *New*: permite crearea de noi instanțe ale editorului de texte;
  - *Open*: Permite deschiderea unui program (sketch) existent;
  - *Exemple*: Conține toate exemplele disponibile la instalare în Arduino IDE, cât și exemplele provenite din librăriile instalate ulterior;
  - *Preferences*: Deschide o fereastră în care se pot particulariza anumite funcționalități ale IDE-ului.
- *Edit*: Conține comenzi de transfer de text, comentare/decomentare, cât și funcționalități de căutare.
- *Sketch*:
  - *Verify/Compile*: Verifică codul scris de erori de compilare. De asemenea, realizează un raport cu memoria utilizată de codul și variabilele din programul principal, raportate la microcontrolerul ales;
  - *Upload*: Realizează compilarea și încărcarea pe placa selectată a fisierului binar obținut, utilizându-se un port ales de utilizator (selectabil dintr-un alt meniu);
  - *Include Library*: Permite includerea în proiectul curent a unei librării externe. Libraria poate să fie preinstalată sau poate fi instalată, având arhiva acesteia;
  - *Add File*: Permite adăugarea unui fișier sursă la proiectul curent.

- *Tools:*
  - *Auto Format:* Formatează codul scris (spațiere, indentare);
  - *Serial Monitor:* Deschide o fereastră ce permite monitorizarea datelor transferate între placa de dezvoltare și portul de comunicație ales;
  - *Serial Plotter:* Realizează graficul datelor transmise pe canalul de comunicații serial într-o manieră real-time. Se pot realiza vizualizări simultane pentru mai multe variabile disponibile pe modulul de comunicație serial;
  - *Board:* Permite selectarea placii de dezvoltare (o listă completă a acestora se poate găsi aici: <https://www.arduino.cc/en/Guide/Environment-boards>);
  - *Port:* Acest meniu conține toate dispozitivele seriale (reale sau virtuale) de pe PC-ul considerat.
- *Help:* În acest meniu se pot găsi numeroase documente care se instalează odată cu Arduino IDE.

Zona 2: Reprezintă spațiul de scriere al codului pentru placa de dezvoltare considerată (aplicația propriu zisă). La crearea unui nou program sunt adăugate în mod implicit două secțiuni:

- *setup:* este o secțiune care este rulată doar o singură dată, când este alimentată sau resetată placa de dezvoltare;
- *loop:* este o buclă infinită, executată ciclic atât timp cât placa este alimentată.

Codul C corespunzător celor două secțiuni, la crearea unei noi instanțe Arduino este prezentat în algoritmul 1.18:

---

Algoritmul 1.18 Configurația inițială a unui program Arduino

---

```

1 void setup() {
2     // put your setup code here, to run once:
3 }
4
5 void loop() {
6     // put your main code here, to run repeatedly:
7 }
```

---

Zona 3: Conține informații legate de placa de dezvoltare utilizată și despre portul pe care se realizează comunicația cu aceasta (se poate modifica din meniul Tools/Port). Ultima zonă, zona 4, conține două comenzi, regăsite și în meniul *Sketch*: Verify și upload.

### 1.2.3 Simularea modulelor Arduino

Un prim pas în dezvoltarea aplicațiilor folosind Arduino presupune o etapă de testare primară sau validare. Această etapă poate fi realizată prin implementarea fizică a unei scheme electrice simplificată a proiectului, sau folosindu-se un simulator pentru Arduino. Astfel de simulatoare pentru Arduino sunt disponibile pe piață, atât pentru persoane cu experiență, cât și pentru începători. Testarea aplicațiilor în aceste simulatoare realizându-se fără grija eventualelor probleme de neconformitate cu anumite specificații tehnice.

Platformele de simulare pentru Arduino sunt ideale pentru programatorii și proiectanții care doresc să învețe elemente de bază ale circuitelor electrice. Astfel, cu ajutorul unui simulator pentru Arduino se poate învăța programare orientată Hardware (HW), fără a avea distrugere componente sau echipamente atunci când se fac greșeli.

Un alt avantaj al realizării de simulări pentru Arduino este faptul că permite realizarea de operații de debug, linie cu linie. Astfel, programatorul poate să descopere eventualele probleme sau funcționări incorecte în codul elaborat.

În cadrul îndrumarului se va utiliza simulatorul (emularea) Autodesk TINKERCAD [3]. Principalele componente conținute sunt prezentate în figura 1.2:

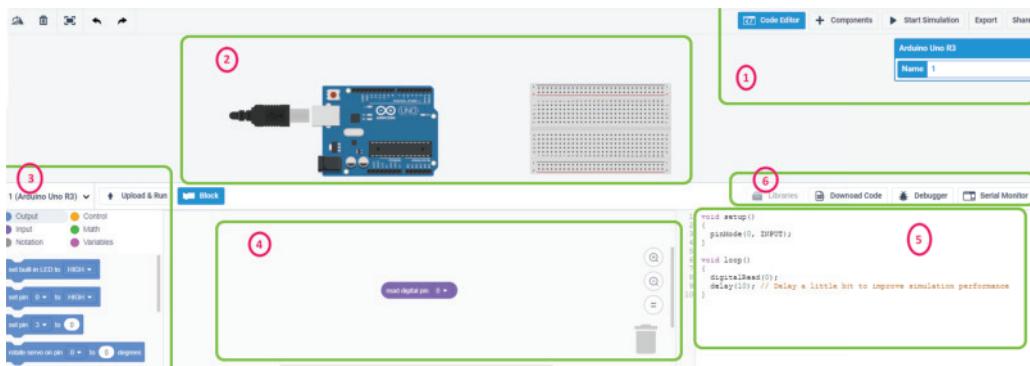


Fig. 1.2 Tinkercad: Simulator pentru Arduino.

Zonile importante pentru realizarea unei simulări sunt următoarele:

- zona 1: corespunde comenzilor legate de activarea/dezactivarea zonelor de scriere cod și listă componente. Pe lângă acestea se găsesc trei butoane: pornirea simulării, exportul modelului dezvoltat (a configurației curente) și partajare (share) cu alte persoane, utilizându-se o adresă de email;
- zona 2: Corespunde zonei de elaborare a schemei HW, ce urmează a fi simulată și al cărei cod poate fi editat în zona 5;
- zona 3: Corespunde unor module predefinite, care pot fi manevrate cu "drag & drop" în zona 4. Odată poziionate în zona 4, se va putea observa și codul corespunzător modulelor, în zona 5;
- zona 4: utilizată pentru a crea o schemă logică, cu module predefinite. Codul corespunzător se găsește în zona 5;
- zona 5: zona de elaborare cod. Aici se poate observa codul corespunzător modulelor adăugate în zona 4. În cazul în care se dorește editarea directă a codului, fără blocuri predefinite, se dezactivează butonul *Block* din zona 3;
- zona 6: Corespunde modulelor de debug pentru codul elaborat sau de monitorizare a modulului de comunicație serială.

Pentru realizarea unei simulări este necesară înregistrarea pe pagina simulatorului Tinkercad [3] ca utilizator nou. Apoi, prin alegerea meniului "Circuits" și selectarea butonului "Create new Circuit" se poate ajunge la meniul prezentat mai sus. Pașii următori presupun alegerea de componente din zona corespunzătoare, editarea codului, pentru modulul Arduino, și încărcarea acestuia în HW. Pasul final este acela de rulare a simulării, prin

apăsarea butonului ”Start Simulation”.

### 1.3 Arduino Hardware

Plăcile de dezvoltare Arduino sunt extrem de variate, având posibilitatea de a se alege diferite funcționalități și procesoare. Toate plăcile de dezvoltare conțin un set de intrări digitale sau analogice care pot fi interfațate cu diferite plăci de extensie pentru Arduino (numite în lb. engleză *shields*) sau circuite conexe. Modulele de comunicație serială (pentru unele modele sunt prezente interfețe USB) sunt utilizate, în cadrul plăcilor de dezvoltare, pentru programarea modulelor Arduino.

Din punct de vedere hardware, multe dintre plăcile de dezvoltare Arduino conțin microcontrolere Atmel [2] pe 8 biți (ATmega8, ATmega168, ATmega328, ATmega1280, ATmega2560) cu diferite variante de echipare, din punct de vedere al memoriei flash, pinilor sau funcționalităților. Mare parte dintre plăci funcționează la 5 voltă și au un oscilator cu quart (rezonator ceramic) de 16Mhz. Informații legate de arhitectura microcontrolerelor ATmega sunt disponibile în secțiunea 2.9 din cartea [5].

Microcontrolerle de pe modulele Arduino sunt pre-programate cu *boot loader* [4], astfel încât să se simplifice încărcarea programelor dezvoltate de programator pe memoria flash de pe cip. Plăcile de dezvoltare Arduino sunt programate utilizându-se interfața USB, având integrat un convertor USB-serial (cum este FTDI FT232 [6]).

#### 1.3.1 Arduino UNO

Modelul UNO de la Arduino reprezintă una dintre cele mai populare plăci de dezvoltare pentru cei care doresc să învețe electronică și programare pe microcontrolere. Această placă de dezvoltare este considerată cea mai robustă, utilizată și bine documentată placă dintre toate cele disponibile, din familia Arduino. Acest model este cel de referință pentru modelele Arduino, este primul din seria placilor de dezvoltare Arduino cu conexiune USB.

Arduino UNO conține un microcontroler din familia ATmega328P [7] și are următoarele caracteristici principale:

- 14 pini digitali de intrare/iesire (6 pot fi utilizati ca ieșiri PWM);
- 6 intrări analogice;
- oscilator cu quart de 16MHz;
- conexiune USB, alimentare jack;
- lucrează la 5 V, poate fi alimentat între 5-12V;
- 32KB de memorie flash (0.5KB pentru bootloader), 2KB SRAM, 1KB EEPROM;

În figura 1.3 se prezintă configurația tuturor pinilor și funcțiile speciale pe care le poate avea fiecare dintre aceștia.

Unii dintre pinii prezenți pe placă au funcții speciale. Dintre aceștia amintim:

- Comunicație serială: pinii 0 (RX) și 1 (TX). Utilizați pentru a transmite și a receptiona date într-o manieră serializată. Aceștia sunt conectați la pinii corespunzători adaptorului USB-TTL;
- Întreruperi externe: pinii 2 și 3. Aceștia pot fi setați să genereze o cerere de întrerupere

atunci când apar tranziții de la o stare la alta a pinilor;

- PWM: pinii 3, 5, 6, 9, 10. Utilizați pentru a furniza un semnal de 8 biți, modulat în frecvență;
- comunicație SPI: pinul 10 (SS), 11(MOSI), 12 (MISO), 13 (SCK). Acești pini suportă comunicația SPI;
- LED 13: pin conectat la un led;
- Interfață I2C: pinii 18 (SDA) și 19 (SCL) pot fi folosiți pentru a transmite sau recepționa date;
- Intrări analogice: notate cu A0-A5, sunt conectate la un modul ce permite convertirea unui semnal analogic în unul digital.

Plăcile de dezvoltare Arduino UNO conțin un fuzibil resetabil, care protejează portul USB al calculatorului de scurtcircuite sau sarcini mai mari de curent. Chiar dacă multe dintre porturile calculatoarelor au astfel de protecții, acesta aduce extra protecție portului USB. Dacă un curent mai mare de 500 mA este aplicat pe port, fuzibilul o să oprească portul/conexiunea până când scurtul sau supraîncărcarea este eliminată.

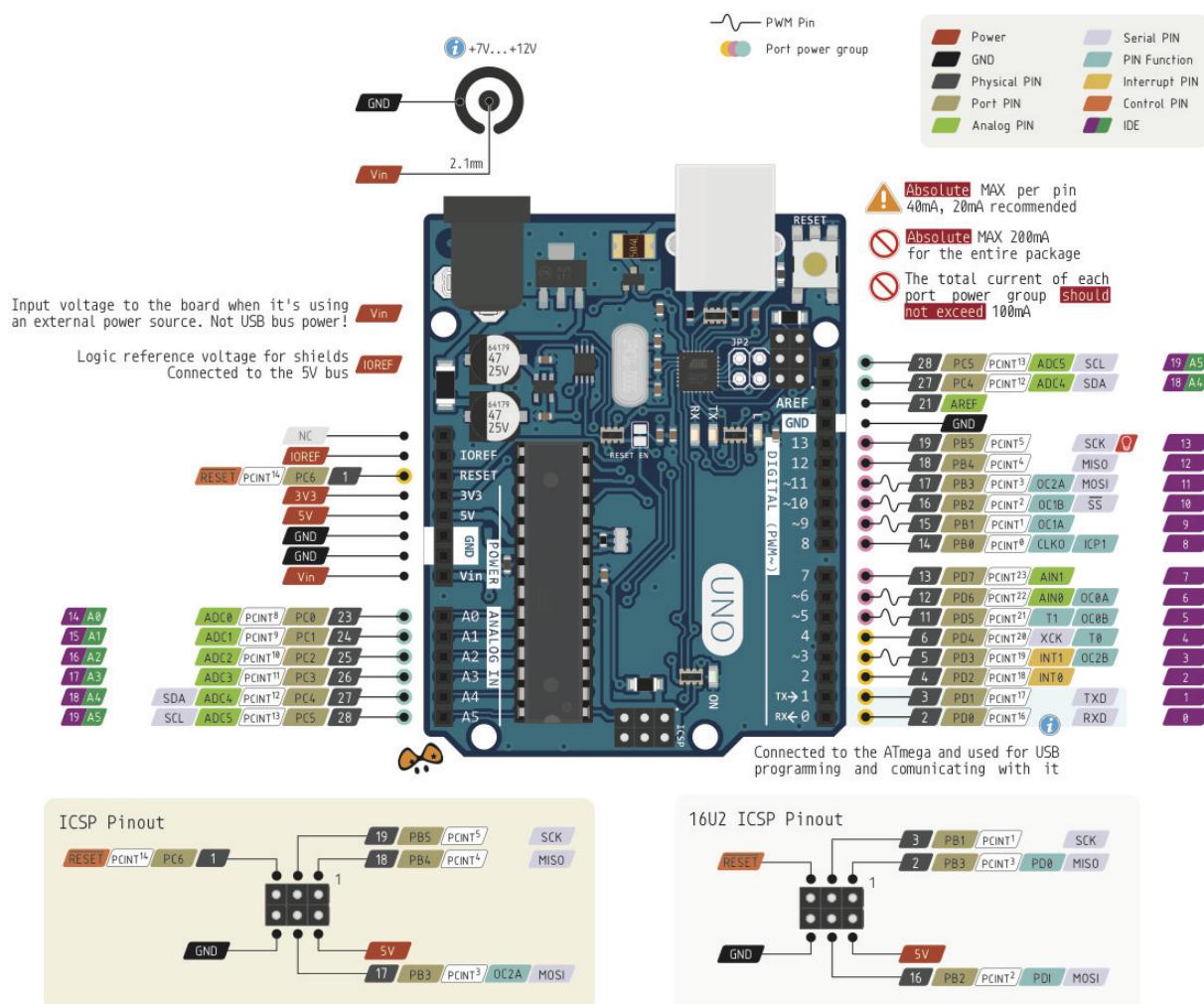


Fig. 1.3 Configurația pinilor pentru Arduino UNO.

### 1.3.2 Senzori compatibili cu modulele Arduino

În cadrul diferitelor aplicații embeded, plăcile de dezvoltare Arduino pot fi utilizate cu ușurință împreună cu diferiți senzori digitali sau analogici precum:

- senzor de temperatură (figura 1.4(a)): permite măsurarea temperaturii și a umidității. Pentru măsurarea temperaturii se pot folosi senzori din familia: DHT11, DHT22, LM35 sau TMP36;
- senzor barometric (figura 1.4(b)): permite determinarea valorilor presiunii atmosferice, a temperaturii cât și a altitudinii. Din această familie de senzori fac parte BMP180 sau BMP280;
- senzor de puls (figura 1.4(c)): permite determinarea ritmului cardiac. Poate fi utilizat pentru monitorizarea ritmului cardiac;
- senzor de ploaie (figura 1.4(d)): acest senzor poate fi utilizat pentru determinarea prezenței apei pe suprafața senzorului;
- senzor wireless (figura 1.4(e)): Utilizat pentru a permite transferul serial de date pe WIFI. Modulul permite realizarea de conexiuni simple pe TCP/IP. Un modul de acest fel este senzorul ESP8266;
- senzor cu ultrasunete (figura 1.4(f)): permite măsurarea distanțelor utilizând viteza de propagare a sunetelor. Cei mai populari senzori de distanță cu ultrasunete sunt HC-SR04 și Ping;
- senzor de prezență (figura 1.4(g)): permite detectarea mișcării, cum este cazul detectării unei persoane în interiorul sau exteriorul unei anumite zone. Un astfel de senzor este PIR Hc-SR501;
- senzor de culoare (figura 1.4(h)): Senzorii din această categorie detectează culoarea și furnizează la exterior un semnal a cărui frecvență este proporțională cu intensitatea luminoasă. Un astfel de senzor este TCS230;
- senzor de gaz (figura 1.4(i)): permite detecția concentrației de GPL, butan, metan, alcool, propan, hidrogen sau fum. Un astfel de senzor este CH4;
- senzor detectie sunete (figura 1.4(j)): poate fi utilizat pentru detectia de sunete din mediul înconjurător, având o sensibilitate ajustabilă. Un astfel de senzor este KY-038;
- senzor detectie viteza (figura 1.4(k)): Este utilizat în combinație cu un disc cu fante. Acest sistem, format din senzor și disc, permite monitorizarea vitezei unui motor (contorizarea fantelor). Sistemul este compus dintr-un element care furnizează lumină (un LED) și un element sensibil la lumină (de exemplu un fototranzistor). Un senzor de acest fel este LM393.
- senzor cu giroscop și accelerometru (figura 1.4(l)): Un astfel de senzor este compus din două module capabile să măsoare orientarea senzorului față de polul N, cât și viteza unghiulară. Tot pe același senzor este un modul capabil să determine accelerările. Un astfel de modul este MPU-6050.

Un alt modul care poate fi adăugat senzorilor prezentati anterior, este un modul de control (nu este un senzor) pentru motoare de curent continuu. Un astfel de modul, cunoscut și sub

denumirea de *driver de motoare* este L298H. Acest modul permite controlul simultan a două motoare. Are la bază circuitul integrat L298 (compus dintr-o punte H). Controlul se poate realiza în ambele direcții, iar curenții maximi sunt de 2A, pentru fiecare motor. Acest driver este ideal pentru aplicații robotice, întrucât necesită puține linii de comandă pentru fiecare motor.

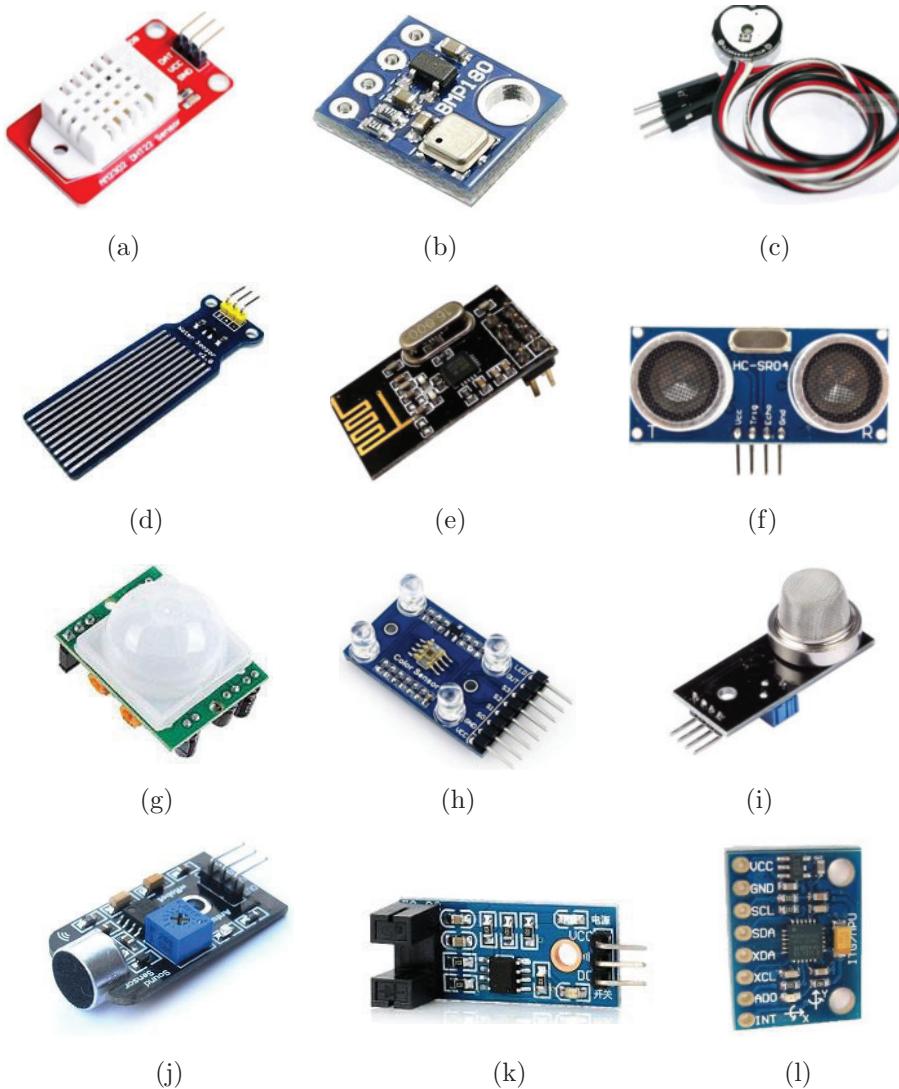


Fig. 1.4 Senzori compatibili cu module Arduino: (a) senzor de temperatură; (b) senzor barometric; (c) senzor de puls; (d) senzor de ploaie; (e) senzor wireless; (f) senzor cu ultrasunete; (g) senzor de prezență; (h) senzor de culoare; (i) senzor de gaz; (j) senzor pentru sunete; (k) senzor pentru măsurarea vitezei, (l) senzor cu giroscop și accelerometru.

#### 1.4 Aplicații propuse

Utilizându-se compilatorul de C disponibil online la adresa: [www.onlinedb.com](http://www.onlinedb.com), să se implementeze următoarele exemple:

1. Să se implementeze un program care realizează inversiunea a două numere citite de la tastatură. Rezultatul se va afișa în consolă.
2. Să se implementeze o funcție ce returnează valoare 1 dacă numărul primit ca parametru

este par și valoarea 0 dacă numărul este impar.

3. Să se implementeze o funcție ce returnează cel mai mare număr dintre cele 3 primite ca și parametrii de intrare ai funcției.

4. Să se implementeze un minicalculator pentru operațiile de bază: adunare, scădere, înmulțire și împărțire.

5. Să se implementeze o funcție care să returneze starea bitului de pe poziția n, primit ca parametru al funcției.