INFORMATICĂ · CLASA A XI-A · SUBPROGRAME · LECȚIA 1

Subprograme (1): funcții care returnează o valoare

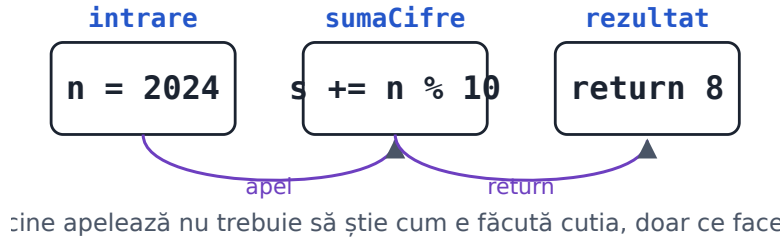
Durată: 60 min Nivel: mate-info Înainte: lecțiile ix-09, ix-10

CE VEI ȘTI DUPĂ LECȚIE

- Înțelegi de ce împărțim un program în subprograme (modularizare, reutilizare).
- Definești o funcție: tip returnat, nume, parametri, corp, return.
- Apelezi funcții din main și din alte funcții.
- Deosebești parametrii formali de cei efectivi și variabilele locale de cele globale.
- Scrii definiția completă a unui subprogram exact cum se cere la Bac (III.1).

1. Ideea: o bucată de program cu nume

Imaginează-ți că trebuie să verifici dacă un număr este prim în trei locuri din același program. Fără subprograme ai copia de trei ori aceeași buclă; dacă găsești o greșeală, o repari în trei locuri. Cu un **subprogram** (o funcție) scrii bucla **o singură dată**, îi dai un nume (`prim`) și o **apelezi** de câte ori ai nevoie.



Funcția ca o cutie

DE CE AȘA?

De ce împărțim programul în funcții? (1) *Reutilizare*: scrii o dată, folosești de multe ori. (2) *Claritate*: `if (prim(x))` se citește ca o propoziție, pe când zece rânduri de buclă trebuie descifrate. (3) *Testare*: poți verifica fiecare funcție separat. (4) *Lucru în echipă*: fiecare scrie câte o funcție, iar ceilalți o folosesc fără să-i cunoască interiorul.

2. Anatomia unei funcții

```cpp
#include <iostream>
using namespace std;

int sumaCifre(int n)          // antet: tip returnat, nume, lista de parametri
{                             // corp
    int s = 0;                // variabilă locală
    while (n > 0)
    {
        s += n % 10;
        n /= 10;
    }
    return s;                 // valoarea „trimisă înapoi” celui care a apelat
}

int main()
{
    int x;
    cin >> x;
    cout << sumaCifre(x);     // apel; x este parametru EFECTIV
    return 0;
}
```

Termen	În exemplu
tip returnat	`int`
nume	`sumaCifre`
parametru **formal** (în definiție)	`int n`
parametru **efectiv** (la apel)	`x` (sau `123`, sau `x + 1`)
variabilă **locală**	`s` (există doar în timpul execuției funcției)
`return`	oprește funcția și întoarce valoarea

Urmărire pentru apelul `sumaCifre(2024)`:

pas	n	s
intrare în funcție	2024	0
după prima iterație	202	4
după a doua	20	6
după a treia	2	6
după a patra	0	8
`return s`	(dispare)	8

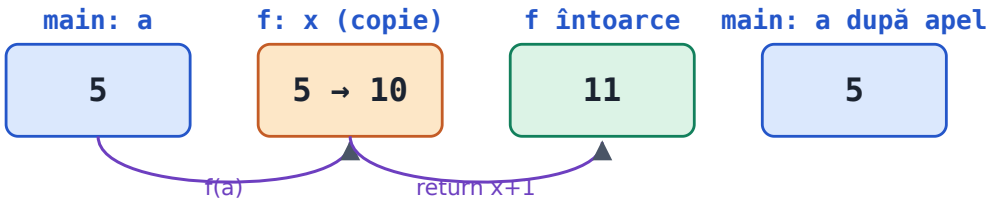
Intrare / ieșire

Intrare	Ieșire
`2024`	`8`
`0`	`0`

DE CE AȘA?

De ce tipul funcției (`int`) apare înaintea numelui? Compilatorul trebuie să știe ce fel de valoare se obține la apel, ca să o poată folosi într-o expresie: `sumaCifre(123) + sumaCifre(45)` este o sumă de două numere întregi, deci valoarea funcției are tip `int` .

Parametrii prin valoare sunt copii



int f(int x) { x = x * 2; return x + 1; } — a din main rămâne 5

Copie

La apel, valoarea parametrului efectiv se **copiază** în parametrul formal. Funcția poate modifica copia, dar variabila originală rămâne neschimbată. (Cum modificăm originalul: lecția următoare.)

3. Funcții `bool` : „este ... ?”

```cpp
#include <iostream>
using namespace std;

bool prim(int n)
{
    if (n < 2) return false;
    for (int d = 2; d * d <= n; d++)
        if (n % d == 0) return false;      // am găsit un divizor: ieșim imediat
    return true;                           // nu am găsit niciunul
}

int main()
{
    int n, x, cnt = 0;
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> x;
        if (prim(x)) cnt++;
    }
    cout << cnt;
    return 0;
}
```

DE CE AȘA?

De ce `return false` în interiorul buclei? `return` termină **toată funcția**. Când găsim un singur divizor, răspunsul este deja „nu e prim"; nu mai are sens să continuăm. **De ce `return true` abia după buclă?** Putem afirma „e prim" doar după ce am verificat *toți* candidații.

Urmărire pentru `prim(15)` și `prim(13)`:

apel	d	d·d ≤ n?	n % d	decizie
`prim(15)`	2	4 ≤ 15 da	1	continuă
	3	9 ≤ 15 da	0	`return false`
`prim(13)`	2	4 ≤ 13 da	1	continuă
	3	9 ≤ 13 da	1	continuă
	4	16 ≤ 13 nu	—	`return true`

Intrare / ieșire

Intrare	Ieșire
`5` `2 9 7 1 13`	`3` (2, 7 și 13 sunt prime)
`3` `1 4 6`	`0`

SFAT

O funcție `bool` are de obicei un nume care sună ca o întrebare (`prim`, `palindrom`, `existaCifra`), iar apelul se citește natural în `if`.

4. Funcții `void`: fac ceva, nu întorc nimic

```cpp
#include <iostream>
using namespace std;

void afiseazaDivizori(int n)
{
    for (int d = 1; d <= n; d++)
        if (n % d == 0) cout << d << ' ';
}

int main()
{
    int x;
    cin >> x;
    afiseazaDivizori(x);          // apel de sine stătător, nu într-o expresie
    return 0;
}
```

Funcția `void` nu are `return valoare;` (poate avea doar `return;` ca să iasă mai devreme) și se apelează ca instrucțiune.

5. Funcții care se ajută între ele

Funcțiile se pot apela una pe alta. Să verificăm dacă un număr este palindrom, întâi mutând „oglindirea" într-o funcție:

```cpp
#include <iostream>
using namespace std;

int oglindit(int n)
{
    int o = 0;
    while (n > 0)
    {
        o = o * 10 + n % 10;
        n /= 10;
    }
    return o;
}

bool palindrom(int n)
{
    return n == oglindit(n);      // o funcție o apelează pe alta
}

int main()
{
    int x;
    cin >> x;
    cout << (palindrom(x) ? "DA" : "NU");
    return 0;
}
```

DE CE AȘA?

De ce `return n == oglindit(n);`? Expresia `n == oglindit(n)` are valoare adevărat/fals, exact ce întoarce o funcție `bool`. Nu mai e nevoie de `if (...) return true; else return false;`. **De ce `10` nu e palindrom?** `oglindit(10) = 1` (cifra 0 de la sfârșit dispare la oglindire), iar `10 ≠ 1`.

6. Unde se scrie funcția: prototipul

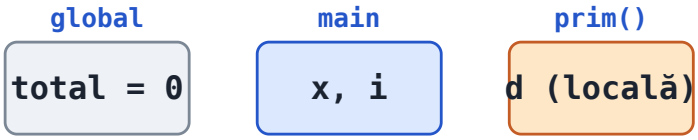
Funcția trebuie să fie cunoscută **înainte** de locul în care e apelată. Ai două variante: o scrii deasupra lui `main`, sau scrii sus doar **prototipul** (antetul urmat de `;`) și definiția mai jos.

```cpp
int sumaCifre(int n);      // prototip: promite că funcția există

int main() { cout << sumaCifre(2024); }

int sumaCifre(int n) { /* ... */ return 0; }
```

7. Variabile locale și globale



tot, pornește de la 0 · locală: doar în funcția ei, valoare necunos

Vizibilitate

	Globală	Locală
unde se declară	în afara oricărei funcții	într-o funcție sau bloc
unde se vede	peste tot, după declarare	doar în funcția/blocul ei
valoare inițială	0	necunoscută („gunoi”)
durata	cât rulează programul	doar cât se execută funcția

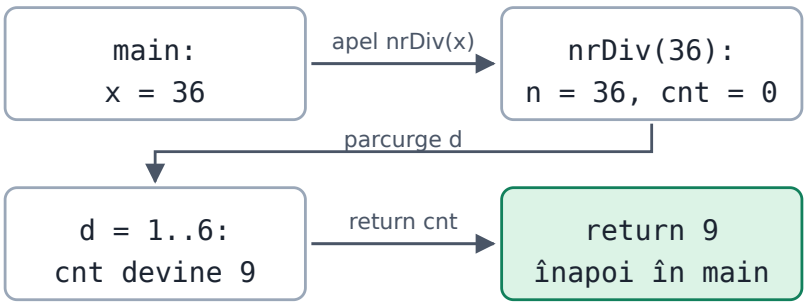
Dacă o variabilă locală are același nume cu una globală, în funcție se folosește cea locală (o „ascunde” pe cea globală).

ATENȚIE — GREȘELI FRECVENTE

Nu folosi variabile globale ca scurtătură pentru a transmite date funcțiilor. La Bac se cere ca subprogramul să lucreze cu **parametrii** din antetul dat, iar o funcție care depinde de variabile globale e greu de testat și de reutilizat.

8. Cum arată cerința la Bac (III.1)

„Subprogramul `nrDiv` are un singur parametru, `n`, prin care primește un număr natural (n ∈ [1, 10^9]). Subprogramul returnează numărul divizorilor lui `n`. Scrieți definiția completă a subprogramului.”



după return, variabilele locale n, cnt, d dispar

Apel

```cpp
#include <iostream>
using namespace std;

int nrDiv(int n)
{
    int cnt = 0;
    for (int d = 1; d * d <= n; d++)
        if (n % d == 0)
        {
            cnt++;                          // divizorul d
            if (d != n / d) cnt++;          // și perechea lui, n / d, dacă e diferită
        }
    return cnt;
}

int main()              // la Bac NU scrii main; aici îl avem doar ca să testăm
{
    int x;
    cin >> x;
    cout << nrDiv(x);
    return 0;
}
```

DE CE AȘA?

De ce se oprește bucla la `d * d <= n`? Divizorii vin în perechi `(d, n/d)`; unul dintre ei este ≤ √n. Parcurgând doar până la √n (≈ 31 623 pași pentru n = 10⁹ față de 10⁹) găsim toate perechile. **De ce `if (d != n / d)`?** Pentru pătrate perfecte (de ex. 36 = 6·6) perechea lui 6 este tot 6; fără verificare l-am număra de două ori.

LA BAC

La Bac se cere **doar definiția** subprogramului: nu scrie `main`, nu citi `n` de la tastatură în subprogram (îl primești ca parametru) și nu afișa rezultatul (îl returnezi). Respectă exact **numele** și **ordinea parametrilor** din enunț.

9. Greșeli frecvente

Greșeala	Efect	Corect
uiți `return` într-o funcție non-`void`	valoare întoarsă necunoscută	toate căile ajung la `return`
`return` în buclă când vrei doar să ieși din buclă	funcția se termină prea devreme	`break` pentru buclă
citești `n` cu `cin` în subprogram	strici formatul cerut la Bac	folosește parametrul
funcția `prim` cu `d <= n`	`n % n == 0`, deci niciun număr nu e prim	`d * d <= n`
apelezi funcția înainte de definiție, fără prototip	eroare de compilare	prototip sau definiție deasupra

Greșeala	Efect	Corect
modifici `n` și speri că se modifică în `main`	nu se modifică (copie)	referință (lecția următoare)

10. La BAC

LA BAC

Subiectul III.1 cere definirea unui subprogram simplu (cifre, divizori, cmmdc, o proprietate a numerelor). Pașii: (1) citește antetul din enunț; (2) alege tipul returnat (`int` , `bool` , `void`); (3) calculează în variabile locale; (4) `return` . La Subiectul I apar grile despre antet corect, valoarea unui apel și despre variabile locale/globale.

11. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Ce afișează?

```
int f(int x) { x = x * 2; return x + 1; }
int main() { int a = 5; cout << f(a) << ' ' << a; }
```

2. Ce afișează `cout << mx(mx(3, 8), 5);` , dacă `int mx(int a, int b) { return a > b ? a : b; }` ?

3. Care este antetul corect al unei funcții `cifre` care primește un număr natural `n` și returnează numărul lui de cifre?

4. Ce afișează?

```
int f(int x) { if (x % 2 == 0) return x / 2; return 3 * x + 1; }
cout << f(f(f(6)));
```

5. Ce afișează?

```
int x = 5;
void f() { int x = 2; x++; }
int main() { f(); cout << x; }
```

6. Funcția `prim` are bucla `for (int d = 2; d <= n; d++)` în loc de `d * d <= n` . Pentru ce valori ale lui `n` returnează `true` ?

7. Scrie definiția funcției `nrCifrePare(n)` , care returnează numărul cifrelor pare ale lui `n` (considerăm `nrCifrePare(0) = 1`).

8. Câți pași face `prim(n)` cel mult pentru `n = 10⁹` ? Dar o variantă cu `d <= n` ?

9. Cât este `sumaCifre(123) + sumaCifre(45)` ?

12. Mai departe (pentru cine vrea mai mult)

Supraîncărcare (overloading). Poți avea mai multe funcții cu același nume, dacă diferă prin parametri: `int aria(int l)` (pătrat) și `int aria(int l, int L)` (dreptunghi). Compilatorul o alege pe cea potrivită.

Parametri impliciti. `int putere(int a, int b = 2)` poate fi apelat ca `putere(5)` (= 25) sau `putere(5, 3)`.

Variabile `static`. O variabilă locală declarată `static` își păstrează valoarea între apeluri (utilă pentru a număra de câte ori a fost apelată funcția).

Fișiere antet. În proiecte mari, prototipurile se pun într-un fișier `.h`, iar definițiile într-un `.cpp`; așa se împarte un program între mai mulți oameni.

Funcții lambda. În C++ modern: `auto patrat = [](int x) { return x * x; };`. Le vei întâlni la `sort` cu comparator (vezi lecția X.12).

Rezumat

- `tip nume(parametri) { … return valoare; }`; `void` dacă nu întoarce nimic.
- Parametri prin valoare = copii; modificările nu se văd în afară.
- Locale: există doar în funcție, necunoscute la început. Globale: peste tot, pornesc de la 0.
- `bool` pentru „este ... ?”; `return` oprește funcția imediat.
- La Bac: doar definiția, cu numele și parametrii din enunț.

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] #14 **primalitate** — ușoară
  Pe pbinfo scrii DOAR funcția — main-ul există deja.
  https://www.pbinfo.ro/probleme/14/primalitate
- [ ] #897 **sumciff** — ușoară
  https://www.pbinfo.ro/probleme/897/sumciff
- [ ] #24 **oglindit2** — ușoară
  https://www.pbinfo.ro/probleme/24/oglindit2

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #16 **suma-divizorilor** — ușoară
  https://www.pbinfo.ro/probleme/16/suma-divizorilor
- [ ] #25 **celmaimarenr** — ușoară
  https://www.pbinfo.ro/probleme/25/celmaimarenr

- [ ] #27 **cifre-zero** — ușoară
  https://www.pbinfo.ro/probleme/27/cifre-zero
- [ ] #76 **fultimacifrapara** — ușoară
  https://www.pbinfo.ro/probleme/76/fultimacifrapara
- [ ] #896 **factorialf** — ușoară
  https://www.pbinfo.ro/probleme/896/factorialf
- [ ] #904 **concat** — ușoară
  https://www.pbinfo.ro/probleme/904/concat
- [ ] #2823 **cifreimpare** — ușoară
  Subiect de Bac.
  https://www.pbinfo.ro/probleme/2823/cifreimpare
- [ ] #3613 **fmultiplu-bac** — ușoară
  Subiect de Bac.
  https://www.pbinfo.ro/probleme/3613/fmultiplu-bac
- [ ] #3614 **fsuma-bac** — ușoară
  Subiect de Bac.
  https://www.pbinfo.ro/probleme/3614/fsuma-bac

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] #469 **Exercițiul #469**
  care este antetul corect al unei funcții
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=469
- [ ] #470 **Exercițiul #470**
  funcția smax: valoarea unui apel
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=470
- [ ] #472 **Exercițiul #472**
  funcția min cu două valori întregi: valoarea unei expresii
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=472
- [ ] #473 **Exercițiul #473**
  subprogramul scif returnează suma cifrelor: valoarea unui apel
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=473
- [ ] #617 **Exercițiul #617**
  antetul corect al subprogramului cifre
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=617

- [ ] #1725 **Exercițiul #1725**
  funcția mini(int x, int y): ce returnează un apel
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1725

Citește și exersează în plus

- Subprograme https://www.pbinfo.ro/articole/3656/subprograme
- Anatomia unei funcții C++ https://www.pbinfo.ro/articole/3662/anatomia-unei-functii-c-c
- Definirea și declararea unei funcții C++ https://www.pbinfo.ro/articole/3664/definirea-si-declararea-unei-functii-c-c
- Declararea variabilelor (locale, globale) https://www.pbinfo.ro/articole/3665/declararea-variabilelor
- cppreference — Functions — referința oficială despre funcții (engleză)
  https://en.cppreference.com/w/cpp/language/functions
- Python Tutor pentru C++ — vezi cum apar și dispar variabilele locale la fiecare apel
  https://pythontutor.com/cpp.html
- LearnCpp — Introduction to functions — explicații detaliate cu multe exemple (engleză)
  https://www.learncpp.com/cpp-tutorial/introduction-to-functions/

Răspunsuri

Exersează pe hârtie

1. `11 5` : `f(a)` lucrează pe o copie (x devine 10, se returnează 11), `a` rămâne 5.

2. `8` : `mx(3, 8) = 8`, apoi `mx(8, 5) = 8`.

3. `int cifre(int n)`. (Returnează un număr, deci tipul este `int`; parametrul este un număr întreg.)

4. `5` : `f(6) = 3` (par: 6/2), `f(3) = 10` (impar: 3·3+1), `f(10) = 5` (par: 10/2).

5. `5` : funcția `f` modifică variabila **locală** `x` (care o ascunde pe cea globală); globala rămâne 5.

6. Pentru `n < 2` funcția returnează `false` din prima verificare. Pentru `n ≥ 2` bucla ajunge la `d = n`, iar `n % n == 0` produce `return false`. Deci funcția **nu returnează `true` pentru niciun număr**, nici măcar pentru cele prime. Corect: `d * d <= n`, ca să nu se ajungă la `d = n`.

7.

```cpp
#include <iostream>
using namespace std;

int nrCifrePare(int n)
{
    if (n == 0) return 1;            // cifra 0 este pară
    int cnt = 0;
    while (n > 0)
    {
        int c = n % 10;              // ultima cifră
        if (c % 2 == 0) cnt++;
        n /= 10;
    }
    return cnt;
}

int main()
{
    int x;
    cin >> x;
    cout << nrCifrePare(x);
    return 0;
}
```

Atenție: paritatea se testează pe **ultima cifră** (`n % 10`), nu pe `n`. Pentru `4081` cifrele pare sunt 4, 0 și 8, deci rezultatul este 3. Cazul `n = 0` se tratează separat, pentru că bucla `while (n > 0)` nu ar număra nimic.

8. Cu `d * d <= n`: cel mult $\sqrt{(10^9)} \approx$ **31 623** pași. Cu `d <= n`: până la 10^9 pași (dar oricum ar returna `false` la `d = n`).

9. `6 + 9 = 15` (1 + 2 + 3 = 6; 4 + 5 = 9).