INFORMATICĂ · CLASA A XI-A · SUBPROGRAME · LECȚIA 2

Subprograme (2): parametri prin referință și subprograme cu tablouri

Durată: 60 min Nivel: mate-info Înainte: lecțiile xi-01

CE VEI ȘTI DUPĂ LECȚIE

- Transmiți parametri prin referință (&) ca să întorci mai multe rezultate.
- Înțelegi diferența valoare / referință urmărind un program (exercițiu de grilă).
- Scrii subprograme care primesc vectori și matrice ca parametri.
- Modifici un număr transmis prin referință (tip Bac: „subprogramul modifică n...").

1. Problema: cum întorc două rezultate?

`return` întoarce **o singură** valoare. Dacă vrem, de exemplu, și cifra minimă, și cifra maximă a unui număr, ne trebuie două rezultate. Soluția: parametri transmiși **prin referință**. Funcția lucrează direct pe variabilele celui care o apelează, nu pe copii.

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

void minMaxCifre(int n, int &mn, int &mx)     // & = prin referință
{
    mn = 9; mx = 0;
    do
    {
        int c = n % 10;
        if (c < mn) mn = c;
        if (c > mx) mx = c;
        n /= 10;
    } while (n > 0);
}

int main()
{
    int x, a, b;
    cin >> x;
    minMaxCifre(x, a, b);        // după apel: a = cifra minimă, b = cifra maximă
    cout << a << ' ' << b;
    return 0;
}
```

Urmărire (`x = 39057` ; `n` e o copie, `mn` și `mx` sunt chiar `a` și `b`):

n	c	mn (adică a)	mx (adică b)
(început)		9	0
39057	7	7	7

n	c	mn (adică a)	mx (adică b)
3905	5	5	7
390	0	0	7
39	9	0	9
3	3	0	9

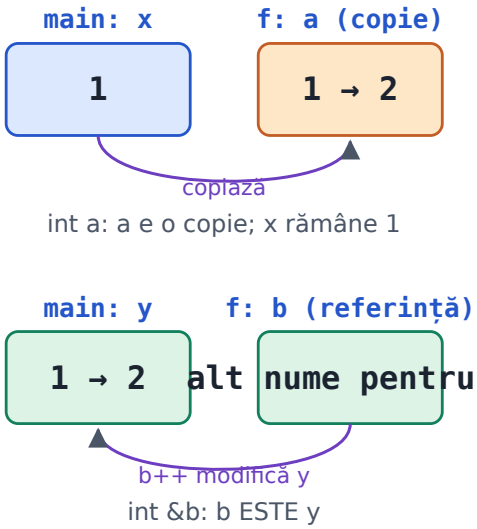
Intrare / ieșire

Intrare	Ieșire
`39057`	`0 9`
`5`	`5 5`

DE CE AȘA?

De ce `mn = 9; mx = 0;` la început? Orice cifră este ≤ 9, deci prima cifră citită va fi sigur ≤ valoarea inițială a minimului; analog, orice cifră este ≥ 0. **De ce `do … while` și nu `while`?** Dacă `n = 0` are o cifră (0), vrem ca ea să fie considerată; cu `while (n > 0)` bucla n-ar rula deloc.

2. Valoare sau referință?



Referință vs valoare

	Prin valoare `int x`	Prin referință `int &x`
ce primește funcția	o **copie**	**chiar variabila** (un alt nume pentru ea)
modificările	nu se văd în afară	se văd în afară
la apel poți pune	orice expresie: `f(a + 1)`, `f(5)`	doar o variabilă: `f(a)`
folosești pentru	date de intrare	rezultate / date de modificat

DE CE AȘA?

De ce la apel nu putem pune `5` pentru un parametru `&`? O referință este un alt nume pentru o variabilă *existentă*. Constanta `5` sau expresia `a + 1` nu sunt variabile, deci nu au unde să primească rezultatul.

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

void schimba(int &a, int &b)
{
    int aux = a;
    a = b;
    b = aux;
}

int main()
{
    int x, y;
    cin >> x >> y;
    schimba(x, y);
    cout << x << ' ' << y;
    return 0;
}
```

Grilă clasică: ce afișează programul?

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

void f(int a, int &b) { a++; b++; }

int main()
{
    int x = 1, y = 1;
    f(x, y);
    cout << x << y;
}
```

LA BAC

Afișează **12**: `x` a fost copiat (rămâne 1), iar `y` a fost transmis prin referință (devine 2). Exact acest tip de verificare apare la Subiectul I.

3. Modificarea unui număr transmis prin referință (format Bac)

„Subprogramul `elimin` are un parametru, `n`, prin care primește un număr natural. Subprogramul elimină din `n` toate cifrele pare, iar numărul obținut este furnizat tot prin `n`. Dacă nu rămâne nicio cifră, `n` devine −1.”

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

void elimin(int &n)
{
    int rez = 0, p = 1;
    bool gasit = false;
    while (n > 0)
    {
        int c = n % 10;
        if (c % 2 == 1)                    // păstrăm cifrele impare
        {
            rez += c * p;                  // o punem pe poziția curentă din rezultat
            p *= 10;
            gasit = true;
        }
        n /= 10;
    }
    if (gasit) n = rez;
    else n = -1;
}

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

Urmărire pentru `n = 3724` (cifrele se iau de la sfârșit):

n	c	păstrăm?	rez	p
3724	4	nu	0	1
372	2	nu	0	1
37	7	da	7	10
3	3	da	37	100
0	—		37	

La final `n = rez = 37`. Pentru `2468` nu s-a păstrat nicio cifră (`gasit` rămâne fals), deci `n = -1`.

DE CE AȘA?

De ce `rez += c * p; p *= 10;`? Cifrele se citesc de la dreapta spre stânga, deci prima păstrată trebuie să ajungă pe ultima poziție (unități), a doua pe zeci etc. `p` (1, 10, 100, ...) este poziția la care scriem. **De ce `gasit` și nu doar `rez == 0`?** Rezultatul poate fi legitim 0? Nu aici (cifrele păstrate sunt impare, deci nenule), dar `gasit` exprimă clar ideea „am păstrat măcar o cifră” și se folosește în același fel în multe probleme.

Intrare / ieșire

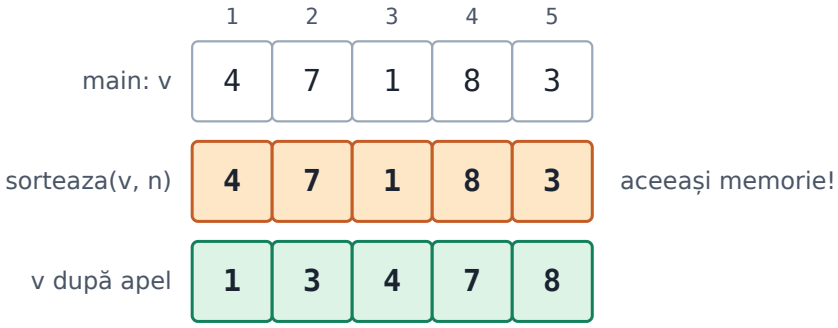
Intrare	Ieșire
`2841`	`1`
`3724`	`37`
`2468`	`-1`

ATENȚIE — GREȘELI FRECVENTE

Când primești `n` prin referință și îl „distrugi” cu `n /= 10`, la final **trebuie** să pui rezultatul înapoi în `n`. Altfel `n` rămâne 0 și `main` primește 0.

4. Vectori ca parametri

Un vector nu se copiază: funcția primește **adresa** lui și lucrează pe vectorul original, fără `&`. Dimensiunea o transmitem separat.



Vectorul nu se copiază

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

int a[101], n;

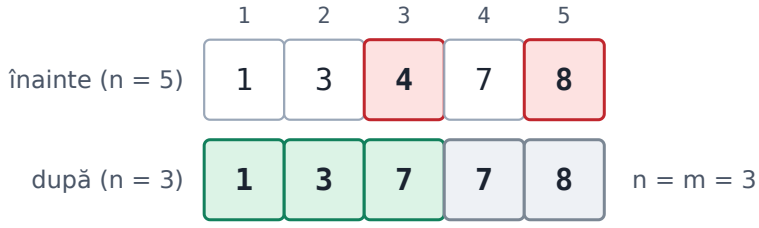
int suma(int a[], int n)              // a[] — fără dimensiune în paranteză
{
    int s = 0;
    for (int i = 1; i <= n; i++) s += a[i];
    return s;
}

void sorteaza(int a[], int n)         // modifică direct vectorul primit
{
    for (int i = 1; i < n; i++)
        for (int j = i + 1; j <= n; j++)
            if (a[i] > a[j]) swap(a[i], a[j]);
}

void stergePare(int a[], int &n)      // n prin referință: se schimbă numărul de elemente
{
    int m = 0;
    for (int i = 1; i <= n; i++)
        if (a[i] % 2 == 1) a[++m] = a[i];
    n = m;
}

void afis(int a[], int n)
{
    for (int i = 1; i <= n; i++) cout << a[i] << ' ';
    cout << '\n';
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cout << suma(a, n) << '\n';
    sorteaza(a, n);
    afis(a, n);
    stergePare(a, n);
    cout << n << '\n';
    afis(a, n);
    return 0;
}
```



Ștergerea

DE CE AȘA?

De ce vectorul se modifică fără `&`, dar numărul `n` are nevoie de `&`? Numele unui vector *este* o adresă: funcția primește adresa și scrie la ea. Pe când `n` este un număr obișnuit: fără `&` funcția primește o copie și, după `stergePare`, `main` ar crede că vectorul are tot 5 elemente. **De ce `a[++m]`?** Se mărește întâi `m`, apoi se scrie: elementele impare se mută pe poziții consecutive de la 1.

Urmărire pentru `stergePare` pe `(1, 3, 4, 7, 8)`:

i	a[i]	impar?	m	vectorul (primele m)
1	1	da	1	1
2	3	da	2	1 3
3	4	nu	2	1 3
4	7	da	3	1 3 7
5	8	nu	3	1 3 7

La final `n = m = 3`.

Intrare / ieșire

Intrare	Ieșire

`5`
`4 7 1 8 3`
	`23`
	`1 3 4 7 8`
	`3`
	`1 3 7`

CAPCANĂ

Dacă subprogramul trebuie să **schimbe numărul de elemente** (ștergeri, inserări), `n` se transmite prin referință (`int &n`).

Matrice ca parametri

Compilatorul trebuie să cunoască numărul de **coloane** ca să calculeze unde se află `a[i][j]`:

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

int a[101][101], n, m;

void citMat(int a[][101], int n, int m)
{
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) cin >> a[i][j];
}

void afisMat(int a[][101], int n, int m)
{
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++) cout << a[i][j] << ' ';
        cout << '\n';
    }
}

int main()
{
    cin >> n >> m;
    citMat(a, n, m);
    afisMat(a, n, m);
    return 0;
}
```

DE CE AȘA?

De ce `a[][101]` și nu `a[][]` ? În memorie matricea e așezată linie după linie. Elementul `a[i][j]` se află la `i·101 + j` căsuțe după început; fără numărul de coloane (101) compilatorul nu poate calcula această poziție.

5. Atenție la indexare (Bac)

La Bac, enunțul precizează des „elementele vectorului sunt indexate de la 0" sau „de la 1". Respectă exact convenția din enunț: `for (i = 0; i < n; i++)` versus `for (i = 1; i <= n; i++)`.

6. Greșeli frecvente

Greșeala	Efect	Corect
`void f(int n)` când vrei să modifici `n`	`main` nu vede schimbarea	`void f(int &n)`
`f(5)` pentru parametru `int &x`	eroare de compilare	variabilă
`f(x, x)` cu doi parametri `&`	cele două nume desemnează aceeași variabilă	atenție la efect
ștergi elemente dintr-un vector, dar `n` e prin valoare	`n` rămâne vechi	`int &n`

Greșeala	Efect	Corect
`int a[][]` pentru matrice	eroare	`int a[][101]`
modifici `n` în funcție și mai ai nevoie de valoarea inițială	valoarea originală se pierde	copie într-o variabilă locală

7. La BAC

> **LA BAC**
>
> „Subprogramul ... furnizează prin parametrul `n` ...” înseamnă referință. La Subiectul I apar grile de tip „ce se afișează” cu parametri mixți (valoare și referință): scrie un tabel cu variabilele din `main` și urmărește care se modifică. La III.1: citește atent ce primești și ce furnizezi și alege tipul (`int`, `int &`) corespunzător.

8. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Ce afișează?

```
void g(int &a, int b) { a = a + b; b = a - b; }
int main() { int x = 3, y = 4; g(x, y); g(y, x); cout << x << ' ' << y; }
```

2. Ce afișează?

```
void h(int &a, int &b) { int t = a; a = b; b = t; }
int main() { int x = 2, y = 5; h(x, y); h(x, y); cout << x << y; }
```

3. Ce afișează?

```
void f(int &a, int &b) { a++; b++; }
int main() { int x = 1; f(x, x); cout << x; }
```

4. De ce apelul `minMaxCifre(39057, 5, 7)` nu compilează?

5. Scrie subprogramul `void ordonareSecv(int a[], int st, int dr)`, care ordonează crescător doar `a[st..dr]`.

6. Un coleg scrie `void sterge(int a[], int n, int poz)` (fără `&`), iar după apel afișează `n` elemente. Ce se întâmplă și cum repari?

7. Scrie `void inserare(int a[], int &n, int poz, int x)`, care inserează `x` pe poziția `poz` în vectorul cu elementele de la 1 la `n`.

8. Ce afișează?

```
void p(int v[], int n) { v[1] = n; n = 0; }
int main() { int v[] = {0, 5, 6}; int n = 2; p(v, n); cout << v[1] << ' ' << n; }
```

9. Mai departe (pentru cine vrea mai mult)

Referință constantă. Pentru date mari, copierea costă. `void afis(const vector<int> &v)` evită copia, iar `const` garantează că funcția nu modifică vectorul.

Pointeri. Referința este, sub capotă, un pointer ascuns. Cu pointeri ai `void schimba(int *a, int *b)` și apel `schimba(&x, &y)`; stil C, folosit în multe cărți (vezi lecția XI.18).

Mai multe rezultate într-un `pair` sau `struct`. În C++ modern o funcție poate returna `pair<int,int>`: `auto [mn, mx] = minMax(n);`. Totuși la Bac e obligatoriu să știi varianta cu `&`.

Efecte secundare. O funcție care modifică parametri prin referință sau variabile globale are „efecte secundare". Cu cât sunt mai puține, cu atât programul e mai ușor de înțeles și de depanat.

Rezumat

- `&` = prin referință: funcția lucrează pe variabila originală; se folosește pentru rezultate.
- Vectorii se transmit automat ca adresă; `n` prin referință dacă se schimbă lungimea.
- Matrice: `int a[][101]` (numărul de coloane este obligatoriu).
- La apel, parametrul efectiv pentru `&` trebuie să fie o variabilă.

Probleme de lucru (pbinfo)

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

- [ ] #21 **suma-cifrelor** — ușoară
  Rezultatul se întoarce prin parametru.
  https://www.pbinfo.ro/probleme/21/suma-cifrelor
- [ ] #905 **detcifre** — ușoară
  Două rezultate: prima și ultima cifră.
  https://www.pbinfo.ro/probleme/905/detcifre
- [ ] #72 **fsumvec** — ușoară
  https://www.pbinfo.ro/probleme/72/fsumvec
- [ ] #899 **ordonaref** — ușoară
  https://www.pbinfo.ro/probleme/899/ordonaref

Temă (pbinfo)

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

- [ ] #23 **oglindit1** — ușoară
  https://www.pbinfo.ro/probleme/23/oglindit1
- [ ] #906 **sumacifre** — ușoară
  https://www.pbinfo.ro/probleme/906/sumacifre

- [ ] #2805 **duplicare** — ușoară
  Subiect de Bac.
  https://www.pbinfo.ro/probleme/2805/duplicare
- [ ] #2808 **produs4** — ușoară
  https://www.pbinfo.ro/probleme/2808/produs4
- [ ] #3119 **impare-bac** — ușoară
  https://www.pbinfo.ro/probleme/3119/impare-bac
- [ ] #33 **multiplu** — ușoară
  https://www.pbinfo.ro/probleme/33/multiplu
- [ ] #35 **count** — ușoară
  https://www.pbinfo.ro/probleme/35/count
- [ ] #42 **stergeelemente** — ușoară
  https://www.pbinfo.ro/probleme/42/stergeelemente
- [ ] #78 **fcitirematrice** — ușoară
  https://www.pbinfo.ro/probleme/78/fcitirematrice

Pentru cei care vor mai mult

Probleme mai grele sau de concurs, pentru cine vrea să meargă dincolo de BAC.

- [ ] #894 **cifminmax** — medie
  https://www.pbinfo.ro/probleme/894/cifminmax
- [ ] #1432 **mutare1** — ușoară
  https://www.pbinfo.ro/probleme/1432/mutare1

Grile din subiectele de BAC (pbinfo)

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

- [ ] #425 **Exercițiul #425**
  subprogram care interschimbă valorile a două variabile: antetul/corpul corect
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=425
- [ ] #581 **Exercițiul #581**
  void P(int &x) { x = x + 5; }: valoarea variabilei după apel
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=581
- [ ] #604 **Exercițiul #604**
  void f(int &a, int b): valorile după apel (valoare vs. referință)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=604

- [ ] #1724 **Exercițiul #1724**
  void t(int &x, int y): ce se afișează după apeluri
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1724

- [ ] #913 **Exercițiul #913**
  antetul corect al subprogramului elimin_impar
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=913

- [ ] #1727 **Exercițiul #1727**
  valorile afișate de un program cu subprograme cu parametri prin referință
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1727

- [ ] #1728 **Exercițiul #1728**
  ce afișează un program cu subprograme date
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1728

Citește și exersează în plus

- Transmiterea parametrilor https://www.pbinfo.ro/articole/3666/transmiterea-parametrilor
- Cum funcționează stiva? https://www.pbinfo.ro/articole/3874/cum-functioneaza-stiva
- Mai mult despre funcții https://www.pbinfo.ro/articole/3667/mai-mult
- LearnCpp — Pass by reference — explicația completă a referințelor, cu exemple (engleză)
  https://www.learncpp.com/cpp-tutorial/pass-by-lvalue-reference/
- Python Tutor pentru C++ — vezi cum o referință arată către aceeași variabilă
  https://pythontutor.com/cpp.html
- cppreference — Reference declaration — referință oficială (engleză)
  https://en.cppreference.com/w/cpp/language/reference

Răspunsuri

Exersează pe hârtie

1. `7 11`. Primul apel: `a` (adică `x`) devine 3 + 4 = 7; `b` e copie, deci `y` rămâne 4. Al doilea apel `g(y, x)`: `a` (adică `y`) devine 4 + 7 = 11; `x` rămâne 7.

2. `25`: prima interschimbare dă `x = 5, y = 2`, a doua îi readuce la `x = 2, y = 5`.

3. `3`: `a` și `b` sunt două nume pentru aceeași variabilă `x`, deci `x` crește de două ori.

4. Parametrii `mn` și `mx` sunt referințe, iar `5` și `7` sunt constante, nu variabile care să poată primi rezultatul.

5.

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

int a[101], n;

void ordonareSecv(int a[], int st, int dr)
{
    for (int i = st; i < dr; i++)
        for (int j = i + 1; j <= dr; j++)
            if (a[i] > a[j]) swap(a[i], a[j]);
}

int main()
{
    int st, dr;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cin >> st >> dr;
    ordonareSecv(a, st, dr);
    for (int i = 1; i <= n; i++) cout << a[i] << ' ';
    return 0;
}
```

6. `n` este o copie în `sterge`, deci `main` își păstrează valoarea veche și afișează un element în plus (resturi). Se repară cu `int &n`.

7.

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

int a[101], n;

void inserare(int a[], int &n, int poz, int x)
{
    for (int i = n; i >= poz; i--)      // de la sfârșit: mutăm spre dreapta
        a[i + 1] = a[i];
    a[poz] = x;
    n++;
}

int main()
{
    int poz, x;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cin >> poz >> x;
    inserare(a, n, poz, x);
    for (int i = 1; i <= n; i++) cout << a[i] << ' ';
    return 0;
}
```

8. `2 2`: vectorul se modifică (`v[1] = n = 2`), iar `n` e o copie (rămâne 2 în `main`).
