INFORMATICĂ · CLASA A X-A · TIPURI STRUCTURATE DE DATE · LECȚIA 12

Structuri de date neomogene: struct (înregistrări)

Durată: 60 min Nivel: mate-info Înainte: lecțiile x-04

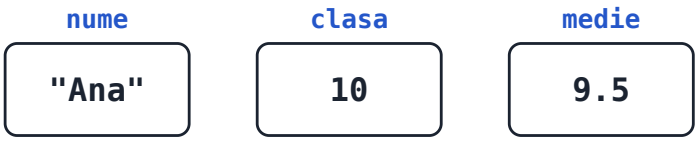
CE VEI ȘTI DUPĂ LECȚIE

- Definești un tip struct cu câmpuri de tipuri diferite.
- Accesezi câmpurile cu operatorul punct (.).
- Folosești vectori de structuri și îi sortezi după un câmp.
- Rezolvi itemii de la Bac cu structuri (declarări, expresii care accesează câmpuri).

1. De ce `struct`?

Un elev are un nume (text), o clasă (număr întreg) și o medie (număr real). Cu ce știm până acum ar trebui trei vectori separați (`nume[]` , `clasa[]` , `medie[]`) și, la sortare, ar trebui să-i interschimbăm „în paralel": ușor de greșit. O **structură** (`struct`) grupează datele înrudite într-un singur tip, ca o fișă.

```
struct elev
{
    char nume[31];
    int clasa;
    double medie;
};
```



e.nume = "Ana"; e.clasa = 10; e.medie = 9.5;

Structura

Declarația `struct elev {…};` **nu creează nicio variabilă**: doar spune cum arată o fișă. Variabilele se declară apoi:

```
elev e;          // o variabilă de tip elev
elev v[100];     // un vector de 100 de elevi
```

DE CE AȘA?

De ce se numește „neomogenă"? Un vector este omogen: toate elementele au același tip. O structură poate avea câmpuri de tipuri diferite (text, întreg, real) și le punem împreună pentru că descriu același obiect.

2. Accesarea câmpurilor: operatorul punct

```
cin >> e.nume >> e.clasa >> e.medie;
if (e.medie >= 9.5) cout << e.nume << " are premiul I";
v[3].clasa = 10;          // al patrulea elev din vector devine elev în clasa a X-a
```

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

struct elev
{
    char nume[31];
    int clasa;
    double medie;
};

int main()
{
    elev e;
    cin >> e.nume >> e.clasa >> e.medie;
    if (e.medie >= 9.5) cout << e.nume << " are premiul I";
    return 0;
}
```

DE CE AȘA?

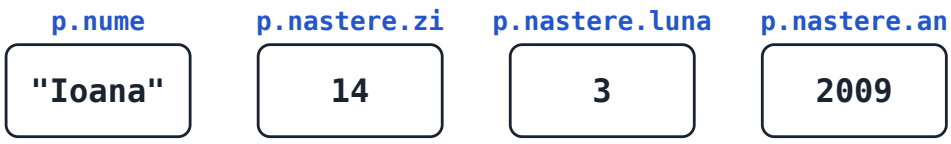
De ce `e.medie` și nu doar `medie`? Într-un program pot exista mai multe variabile de tip `elev` (de exemplu `e1` și `e2`), iar fiecare are propriile câmpuri. Punctul precizează *al cui* câmp folosim.

Intrare / ieșire

Intrare	Ieșire
`Ana 10 9.7`	`Ana are premiul I`
`Dan 9 8.5`	(nimic)

Structuri imbricate

Un câmp poate fi el însuși o structură:



punct după punct: p.nastere.an

Structură în structură

```cpp
struct dataCal { int zi, luna, an; };
struct persoana
{
    char nume[31];
    dataCal nastere;          // câmp care e el însuși o structură
};

persoana p;
p.nastere.an = 2009;          // punct după punct: p, apoi nastere, apoi an
```

LA BAC

Itemi tipici la Bac: „Variabila `x` memorează coordonatele unui punct în plan. Scrieți o expresie care are valoarea 1 dacă punctul se află pe prima bisectoare.” Dacă `struct punct { int a, b; } x;`, răspunsul este `x.a == x.b`. Citește atent **numele exacte** ale câmpurilor și ale variabilelor din enunț.

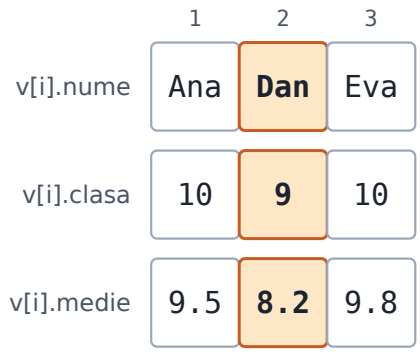
3. Atribuirea și compararea

- O structură se poate **atribui** direct: `e2 = e1;` copiază toate câmpurile, inclusiv șirurile.
- **Nu** se poate compara direct: `e1 == e2` nu compilează. Compari câmp cu câmp.

DE CE AȘA?

De ce atribuirea merge, dar `==` nu? Copierea unei zone de memorie este simplă (se copiază octeții). Egalitatea însă are sens diferit după câmp (la șiruri ne interesează conținutul, nu adresa; la reale, eventual o toleranță). De aceea C++ cere să scrii tu ce înseamnă „egale”.

4. Vector de structuri: citire, calcule, sortare



', 9, 8.2) — un singur element al vectorului

Vector de structuri

Să rezolvăm: se citesc `n` elevi (nume, medie) și se afișează cei cu media **peste media clasei**.

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

struct elev
{
    char nume[31];
    double medie;
};

elev v[101];

int main()
{
    int n;
    double suma = 0;
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> v[i].nume >> v[i].medie;
        suma += v[i].medie;
    }
    double mc = suma / n;                  // media clasei
    for (int i = 1; i <= n; i++)
        if (v[i].medie > mc)
            cout << v[i].nume << '\n';
    return 0;
}
```

Urmărire (intrarea din test):

i	v[i].nume	v[i].medie	suma
1	Ion	9	9
2	Ana	7	16
3	Dan	8	24
4	Eva	10	34

`mc = 34 / 4 = 8.5`. Sunt peste medie: Ion (9) și Eva (10).

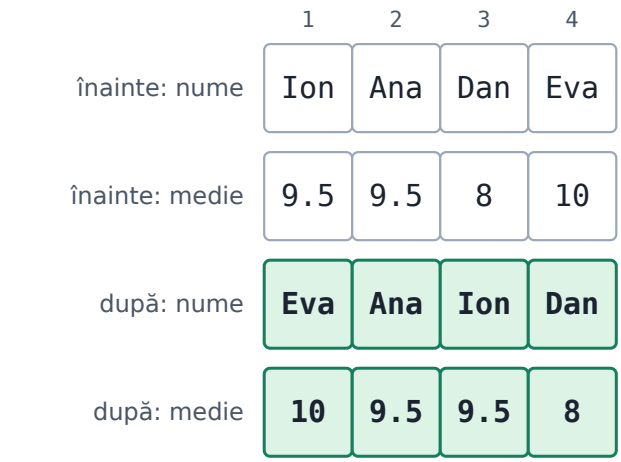
Intrare / ieșire

Intrare	Ieșire

`4`
`Ion 9`
`Ana 7`
`Dan 8`
`Eva 10`
	`Ion`
	`Eva`

`2`
`A 5`
`B 5`
	(nimic: nimeni nu e *strict* peste medie)

Sortarea după un criteriu



gale (Ana, Ion) decide numele: swap(v[i], v[j]) mută ele

Sortare

Descrescător după medie; la medii egale, alfabetic după nume:

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

struct elev
{
    char nume[31];
    double medie;
};

elev v[101];

int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> v[i].nume >> v[i].medie;

    for (int i = 1; i < n; i++)
        for (int j = i + 1; j <= n; j++)
            if (v[i].medie < v[j].medie ||
                (v[i].medie == v[j].medie && strcmp(v[i].nume, v[j].nume) > 0))
                swap(v[i], v[j]);          // elevul ÎNTREG, cu toate câmpurile

    for (int i = 1; i <= n; i++)
        cout << v[i].nume << ' ' << v[i].medie << '\n';
    return 0;
}
```

DE CE AȘA?

De ce condiția are două părți legate cu `||`? `v[i]` trebuie să treacă după `v[j]` în două situații: (1) media lui este mai mică; (2) mediile sunt egale **și** numele lui vine după în ordine alfabetică. Prima parte decide criteriul principal, a doua îl rupe pe cel de egalitate. **De ce `swap(v[i], v[j])` și nu trei interschimbări separate?** Mutăm elevul întreg: nu există riscul să schimbi media unui elev și să uiți numele.

5. Exemple clasice

Fracții

Fracțiile se păstrează ca două întregi, ca să nu pierdem precizie:

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

struct fractie { int a, b; };          // a / b

int main()
{
    fractie f, g, s;
    cin >> f.a >> f.b >> g.a >> g.b;
    s.a = f.a * g.b + g.a * f.b;         // a/b + c/d = (a*d + c*b) / (b*d)
    s.b = f.b * g.b;
    int x = s.a, y = s.b;                // simplificare cu algoritmul lui Euclid
    while (y) { int r = x % y; x = y; y = r; }
    cout << s.a / x << '/' << s.b / x;
    bool maiMica = (1LL * f.a * g.b < 1LL * g.a * f.b);    // f < g, pentru numitori
pozitivi
    cout << ' ' << (maiMica ? "da" : "nu");
    return 0;
}
```

DE CE AȘA?

De ce comparăm `a·d < c·b` și nu `a/b < c/d`? Împărțirea de întregi taie zecimalele (`1/2` ar fi 0), iar cu numere reale apar erori de rotunjire. Prin înmulțire în cruce rămânem la numere întregi și rezultatul este exact. **De ce `1LL *`?** Produsele pot depăși `int`.

Puncte în plan

Care este punctul cel mai îndepărtat de origine? Nu avem nevoie de rădăcină pătrată: comparăm pătratele distanțelor.

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

struct punct { int x, y; };

punct p[101];

int main()
{
    int n, poz = 1;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> p[i].x >> p[i].y;
    for (int i = 2; i <= n; i++)
        if (p[i].x * p[i].x + p[i].y * p[i].y > p[poz].x * p[poz].x + p[poz].y * p[poz].y)
            poz = i;
    cout << p[poz].x << ' ' << p[poz].y;
    return 0;
}
```

DE CE AȘA?

De ce fără `sqrt`? Funcția radical este crescătoare: dacă $d1^2 > d2^2$, atunci și $d1 > d2$. Comparăm direct pătratele, rămânem în întregi și evităm rotunjirile.

6. Greșeli frecvente

Greșeala	Efect	Corect
`elev.nume` (numele tipului în loc de variabilă)	eroare de compilare	`e.nume`
`e1 == e2`	nu compilează	compară câmpurile
`struct elev {…}` fără `;` la sfârșit	eroare de compilare	`};`
`e.nume = "Ana";`	nu se poate atribui un șir	`strcpy(e.nume, "Ana");`
`v[i].medie < v[j].medie` la egalitate fără criteriu secundar	ordine neprecizată pentru egali	adaugă a doua condiție
declari `elev v[100]` și folosești `v[100]`	depășire	`v[101]` sau indici 0..99

7. La BAC

LA BAC

Structurile apar la Subiectul I (grile ca la `pbinfo`: „Care dintre următoarele atribuiri este corectă?”, „Scrieți o expresie care...”) și uneori în Subiectul II. Verifică: numele tipului vs. numele variabilei, numele exacte ale câmpurilor, unde se pune `;`.

8. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Declară o structură `produs` cu câmpurile `cod` (întreg), `pret` (real) și `stoc` (întreg), apoi o variabilă `p`. Scrie o expresie adevărată dacă produsul este pe stoc și costă sub 50 de lei.

2. Se dă `struct punct { int x, y; };` și `struct dreptunghi { punct st_sus, dr_jos; } d;`. Scrie o expresie pentru aria dreptunghiului (cu laturile paralele cu axele).

3. Pentru `struct punct { int x, y; } p;`, scrie expresii care sunt adevărate dacă: (a) `p` este pe prima bisectoare; (b) pe a doua bisectoare; (c) în cadranul II.

4. Ce afișează?

```
struct A { int x; char y[5]; } a = {3, "abc"}, b;
b = a; b.x++;
cout << a.x << b.x << b.y;
```

5. Cum verifici dacă două variabile `e1` și `e2` de tip `elev` (cu `nume`, `clasa`, `medie`) sunt egale?

6. Elevii sunt `Ana 9`, `Dan 10`, `Eva 8`. Care este ordinea după sortarea descrescătoare după medie?

7. Câte valori întregi memorează `x`, declarat `struct { int a, b; double c; int v[3]; } x;`?

8. Calculează `2/3 + 3/4` cu formula din secțiunea 5. Simplifică rezultatul.

9. Se citesc `n` puncte. Scrie secvența care numără câte sunt în cadranul I (`x > 0` și `y > 0`).

10. De ce `swap(v[i], v[j])` este mai sigur decât trei vectori separați?

9. Mai departe (pentru cine vrea mai mult)

`sort` cu comparator. În locul celor două bucle ($O(n^2)$) poți folosi `sort` din `<algorithm>` ($O(n \log n)$), cu o funcție care spune când `a` trebuie să fie înaintea lui `b`:

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

struct elev { char nume[31]; double medie; };
elev v[101];

bool inainte(const elev &a, const elev &b)
{
    if (a.medie != b.medie) return a.medie > b.medie;      // medie mai mare primul
    return strcmp(a.nume, b.nume) < 0;                     // la egalitate, alfabetic
}

int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> v[i].nume >> v[i].medie;
    sort(v + 1, v + n + 1, inainte);
    for (int i = 1; i <= n; i++) cout << v[i].nume << ' ' << v[i].medie << '\n';
    return 0;
}
```

De la struct la clasă. În C++, `struct` poate avea și **funcții** în interior (metode) și constructori: `struct punct { int x, y; double dist() { return sqrt(x*x + y*y); } };`. Asta e începutul programării orientate pe obiecte, pe care o vei întâlni în facultate.

`pair` și `tuple`. Pentru perechi simple nu trebuie să definești un `struct`: `pair<int, int> p = {3, 4}; cout << p.first;`. Este foarte folosit la concursuri.

Baze de date. O listă de structuri (elevi, produse) este exact ce este un tabel dintr-o bază de date (vezi clasa a XII-a): fiecare coloană = un câmp, fiecare rând = o structură.

Rezumat

- `struct nume { tip camp1; tip camp2; … };` (cu `;` la final). Declară variabile separat: `nume x;`.
- Acces cu `.`: `v[i].medie`, `p.nastere.an`.
- Atribuire directă da (`e2 = e1`), comparare directă nu (`==` câmp cu câmp).
- Sortare: compari câmpurile, interschimbi structurile întregi.
- Fracții și puncte: păstrează întregi, compară prin înmulțire în cruce / pătratul distanței.

Probleme de lucru (pbinfo)

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

- [ ] #923 **fractiimax** — ușoară
  https://www.pbinfo.ro/probleme/923/fractiimax
- [ ] #922 **puncte** — ușoară
  https://www.pbinfo.ro/probleme/922/puncte

- [ ] #4193 **bac-fractii** — ușoară
  Format de Bac: o singură instrucțiune / expresie.
  https://www.pbinfo.ro/probleme/4193/bac-fractii

Temă (pbinfo)

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

- [ ] #1013 **aniversari** — ușoară
  https://www.pbinfo.ro/probleme/1013/aniversari
- [ ] #2736 **baza-de-date** — ușoară
  https://www.pbinfo.ro/probleme/2736/baza-de-date
- [ ] #4191 **bac-spectacole** — ușoară
  https://www.pbinfo.ro/probleme/4191/bac-spectacole
- [ ] #4192 **bac-angajati** — ușoară
  https://www.pbinfo.ro/probleme/4192/bac-angajati
- [ ] #2004 **ore** — ușoară
  https://www.pbinfo.ro/probleme/2004/ore
- [ ] #3164 **qclasa** — ușoară
  https://www.pbinfo.ro/probleme/3164/qclasa

Pentru cei care vor mai mult

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

- [ ] #2802 **clasa** — medie
  https://www.pbinfo.ro/probleme/2802/clasa

Grile din subiectele de BAC (pbinfo)

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

- [ ] #107 **Exercițiul #107**
  declararea corectă a unei variabile de tip structură
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=107
- [ ] #126 **Exercițiul #126**
  care atribuiri sunt corecte pentru o declarație dată
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=126
- [ ] #176 **Exercițiul #176**
  struct elev { char nume[30]; float nota; }: ce expresie/atribuire este corectă
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=176

- [ ] #258 **Exercițiul #258**
  struct punct { int x, y; } p: care dintre expresii este corectă
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=258

- [ ] #281 **Exercițiul #281**
  referiri corecte la câmpurile unei structuri
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=281

- [ ] #301 **Exercițiul #301**
  var1, var2 memorează numele și nota a doi elevi: ce expresie dă rezultatul cerut
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=301

- [ ] #63 **Exercițiul #63**
  variabila t memorează lungimile a trei laturi (câmpurile a, b, c)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=63

- [ ] #1087 **Exercițiul #1087**
  indicați secvența care realizează prelucrarea cerută asupra unei structuri
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1087

Citește și exersează în plus

- Structuri de date neomogene https://www.pbinfo.ro/articole/7652/structuri-de-date-neomogene-in-c
- cppreference — struct (class types) — referință pentru structuri și clase (engleză)
  https://en.cppreference.com/w/cpp/language/class
- cppreference — std::sort — sortarea vectorilor de structuri cu un comparator propriu
  https://en.cppreference.com/w/cpp/algorithm/sort
- Python Tutor pentru C++ — vezi cum arată în memorie un vector de structuri
  https://pythontutor.com/cpp.html

Răspunsuri

Exersează pe hârtie

1.

```
struct produs { int cod; double pret; int stoc; } p;
// expresia:
p.stoc > 0 && p.pret < 50
```

2. Latura pe orizontală: `d.dr_jos.x - d.st_sus.x`; pe verticală: `d.st_sus.y - d.dr_jos.y`. Aria: `abs((d.dr_jos.x - d.st_sus.x) * (d.st_sus.y - d.dr_jos.y))` (cu `abs` ca să nu depindem de orientare).

3. (a) `p.x == p.y`; (b) `p.x == -p.y`; (c) `p.x < 0 && p.y > 0`.

4. `34abc`: `b = a` copiază câmpurile, apoi `b.x++` schimbă doar `b` (a rămâne 3).

5. `strcmp(e1.nume, e2.nume) == 0 && e1.clasa == e2.clasa && e1.medie == e2.medie`. Numele se compară cu `strcmp`, nu cu `==`.

6. `Dan` (10), `Ana` (9), `Eva` (8).

7. 5 valori întregi: `a`, `b` și `v[0..2]` (c este real).

8. `(2·4 + 3·3) / (3·4) = 17/12`. Cmmdc(17, 12) = 1, deci e deja simplificat.

9.

```
int cnt = 0;
for (int i = 1; i <= n; i++)
    if (p[i].x > 0 && p[i].y > 0) cnt++;
```

10. Cu un singur `swap`, elevul se mută întreg. La trei vectori trebuie interschimbate trei lucruri în același timp; dacă uiți unul, numele nu mai corespunde mediei.