INFORMATICĂ · PYTHON · CLASA A IX-A · SUBPROGRAME (INTENSIV) · LECȚIA I3

Subprograme în C++ și funcții predefinite din cmath și algorithm

Durată: 100 min Nivel: intensiv · Python + C++ Înainte: lecțiile py9-08, py9-09

CE VEI ȘTI DUPĂ LECȚIE

- Recunoști și scrii părțile unei funcții C++: tip returnat, nume, parametri cu tip, corp, return; folosești void, prototipul și apelul din main.
- Alegi între transmiterea prin valoare și prin referință (&) și compari efectul cu ce se întâmplă în Python cu un int și cu o listă.
- Folosești sqrt, abs, floor, ceil, round, pow din <cmath> și min, max, sort din <algorithm>, cu atenție la double, la conversii și la afișare.

1. Anatomia unei funcții C++

Ideile sunt cele din py9-08 (antet, corp, parametri, apel, `return`, stiva de apeluri). În C++ se schimbă **scrierea**: tipurile se declară peste tot.

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

double media(int a, int b) {        // tip returnat, nume, parametri cu tip
    return (a + b) / 2.0;           // 2.0, nu 2: altfel împărțirea e întreagă
}

void linie(int n) {                 // void: nu returnează nimic
    for (int i = 0; i < n; i++) cout << '-';
}

int main() {
    cout << media(9, 10) << ' ' << media(7, 8) + 1 << '\n';   // 9.5 8.5
    linie(5);                                                 // -----
    return 0;
}
```

	Python	C++
antet	`def media(a, b):`	`double media(int a, int b)`
tipuri	nu se scriu	rezultatul: înaintea numelui; fiecare parametru: `int a, int b` (nu `int a, b`)
fără rezultat	fără `return` (dă `None`)	tipul `void`
mai multe rezultate	`return a, b`	nu direct: prin referințe (secțiunea 3)

> **ATENȚIE — GREȘELI FRECVENTE**
>
> **Valoarea returnată se convertește la tipul din antet.** Dacă scrii `int media(int a, int b) { return (a + b) / 2.0; }`, rezultatul 9,5 devine `9` (se taie zecimalele), fără eroare. Iar o funcție care nu este `void` și se termină fără `return` are comportament **nedefinit**: compilatorul te avertizează cu `-Wall`, nu ignora mesajul.

2. Unde scrii funcția: înainte de main sau cu prototip

Compilatorul citește fișierul de sus în jos. Un apel către o funcție pe care încă n-a văzut-o dă eroarea `'palindrom' was not declared in this scope`. Ai două soluții: scrii definiția **deasupra** lui `main` sau pui sus un **prototip** (antetul urmat de `;`) și definiția mai jos.

Aplicație (din programă): câte numere palindrom sunt într-un șir.

```
int oglindit(int n);                // prototipuri: doar antetul, cu ;
bool palindrom(int n);

int main() {
    int n, x, c = 0;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> x;
        if (palindrom(x)) c++;      // comunică prin argument și valoarea returnată
    }
    cout << c << '\n';
}

int oglindit(int n) {               // definițiile, după main
    int r = 0;
    while (n > 0) { r = r * 10 + n % 10; n /= 10; }
    return r;
}

bool palindrom(int n) { return oglindit(n) == n; }
```

Pentru intrarea `5` și `121 45 7 1221 10` afișează `3`. Prototipul îi spune compilatorului tipul rezultatului și tipurile parametrilor, deci poate verifica apelul înainte să vadă corpul. `main` rămâne scurt și se citește ca un plan.

3. Prin valoare sau prin referință

În py9-09 (secțiunea 3) ai văzut că în Python un `int` nu poate fi schimbat de o funcție, dar o listă da. În C++ **alegi tu**, prin antet:

```
void dubleazaV(int x)  { x = x * 2; }   // prin valoare: lucrează pe o copie
void dubleazaR(int &x) { x = x * 2; }   // prin referință: x ESTE variabila din apel

// în main:  int n = 5;
dubleazaV(n); cout << n << ' ';         // 5
dubleazaR(n); cout << n << '\n';        // 10
```

ce transmiți	Python	C++
un număr	funcția nu-l poate schimba pentru apelant	`int x`: o copie; `int &x`: chiar variabila din apel
o colecție	lista: modificările se văd în apelant	tabloul `int a[]`: modificările `a[i] = …` se văd
o expresie, `f(n + 1)`	merge	la `int x` merge; la `int &x` **nu compilează**

Aplicație (din programă): salariul net. Calcul simplificat: CAS 25%, CASS 10%, impozit 10% din ce rămâne. Când vrei **un** rezultat, îl returnezi. Când vrei **mai multe**, le dai înapoi prin referință:

```
void salariu(double brut, double &net, double &impozit) {
    double cas = 0.25 * brut, cass = 0.10 * brut;
    impozit = 0.10 * (brut - cas - cass);
    net = brut - cas - cass - impozit;
}
```

După `salariu(5000, n, imp)`, `n` este `2925` și `imp` este `325`. Varianta cu un singur rezultat, `double net(double brut)` cu `return`, se poate folosi direct în expresii (`net(a) + net(b)`) și nu poate strica nimic în apelant. Varianta cu `&` dă mai multe rezultate deodată, dar din apel nu vezi ce se modifică: citești antetul.

> **DE ȘTIUT**
>
> Un tablou se transmite fără `&` și totuși funcția îl modifică: `void zero(int a[], int n)` care pune `a[i] = 0` golește tabloul din `main`. Funcția nu știe lungimea tabloului, de aceea primește mereu și `n`.

4. Locale și globale în C++

Regulile din py9-09 rămân: variabilele declarate într-o funcție (și parametrii) sunt locale. Diferențele:

```
int contor = 0;                 // globală: pornește automat de la 0
int x = 1;

void creste() { contor++; }     // modifică globala direct, fără „global”
void f() { int x = 5; x++; }    // x local îl ascunde pe cel global

int main() { creste(); creste(); f(); cout << contor << ' ' << x; }   // 2 1
```

- **Modificarea unei globale**: în Python doar cu `global`; în C++ direct, oricând (de aceea e și mai periculos).
- **Valoarea inițială**: globalele pornesc de la 0, dar **localele neinițializate au valori la întâmplare**. Inițializează mereu `s = 0`, `c = 0`.
- **Vizibilitatea**: o variabilă trăiește doar în blocul `{ }` în care e declarată: `i` din `for (int i = …)` nu mai există după `for`.

Sfatul rămâne cel din py9-09: comunici prin parametri și `return` (sau `&`, dacă sunt mai multe rezultate).

5. Funcții matematice din `<cmath>`

apel C++	rezultat (`cout`)	Python	de reținut
`sqrt(2)`	`1.41421`	`1.4142135623730951`	`cout` arată 6 cifre semnificative
`sqrt(-1)`	`nan` (g++: `-nan`)	`ValueError`	C++ nu oprește programul: validezi tu
`abs(-7)` , `abs(-2.5)`	`7` , `2.5`	la fel	cu `<cmath>` , merge și pe `double`
`floor(-2.7)` , `ceil(2.1)` , `(int) (-2.7)`	`-3` , `3` , `-2`	`math.floor` , `math.ceil` , `int`	conversia la `int` **taie** spre 0
`round(2.5)` , `round(-2.5)`	`3` , `-3`	`2` , `-2`	C++ rotunjește „5” **departe de 0**; Python, spre par
`pow(2, 10)` , `pow(2, 0.5)`	`1024` , `1.41421`	`2 ** 10` → `1024`	`pow` întoarce **mereu** `double`
`pow(10, 9)`	`1e+09`	`1000000000`	`double` mare: `cout` trece la notație științifică

Pentru afișări cu un număr fix de zecimale: `#include <iomanip>` și `cout << fixed << setprecision(2)` (rămâne activ pentru toate afișările următoare). Este echivalentul lui `f"{x:.2f}"` .

Aplicație (din programă): dobânda compusă `A = P · (1 + r/n)^(n·t)` .

```cpp
double suma_finala(double P, double r, int n, int t) {   // r: rata, ca fracție (5% = 0.05)
    return P * pow(1 + r / n, n * t);
}
// în main:
cout << suma_finala(100000, 0.05, 1, 3) << '\n';                      // 115763
cout << fixed << setprecision(2) << suma_finala(1000, 0.05, 1, 3)     // 1157.63
     << ' ' << suma_finala(1000, 0.05, 12, 3) << '\n';                // 1161.47
```

Fără `fixed` , 115 762,50 lei apar ca `115763` : șase cifre semnificative, iar banii s-au „rotunjit”. Iar dacă rata e un `int` în procente, `pow(1 + r / 100, 3)` cu `r = 5` dă `1` : `5 / 100` este împărțire întreagă, deci 0. Scrii `r / 100.0` .

Aplicație (din programă): pachete și distanțe. Câte pachete **complete** de 2 kg ies din 17,5 kg: `floor(17.5 / 2)` = `8` ; câte pungi trebuie pentru toată cantitatea: `ceil(17.5 / 2)` = `9` . Distanța dintre (1, 1) și (4, −3): `sqrt(pow(4 - 1, 2) + pow(-3 - 1, 2))` = `5` , iar distanța „pe străzi”: `abs(4 - 1) + abs(-3 - 1)` = `7` .

6. min, max, sort din `<algorithm>`

	Python	C++
minimul a două valori	`min(a, b)`	`min(a, b)`: **același tip**; `min(3, 2.5)` nu compilează
mai multe valori date direct	`max(4, 1, 9)`	`max({4, 1, 9})` (între acolade)
minimul, suma unei colecții	`min(v)`, `sum(v)`	buclă: `mn = min(mn, s[i])`, `suma += s[i]`
sortare crescătoare	`v.sort()`	`sort(s, s + n)`: de la `s[0]` până **înainte de** `s[n]`
sortare descrescătoare	`v.sort(reverse=True)`	`sort(s, s + n, greater<int>())`

Aplicație (din programă): performanța sportivilor. Se citesc `n` scoruri; afișăm minimul, maximul, media și podiumul.

```cpp
int n, s[100];
cin >> n;
for (int i = 0; i < n; i++) cin >> s[i];   // citirea în tablou
int mn = s[0], mx = s[0], suma = 0;
for (int i = 0; i < n; i++) {
    mn = min(mn, s[i]);
    mx = max(mx, s[i]);
    suma += s[i];
}
cout << "min " << mn << ", max " << mx << ", media "
     << fixed << setprecision(2) << (double)suma / n << '\n';

sort(s, s + n, greater<int>());          // descrescător
cout << "podium: " << s[0] << ' ' << s[1] << ' ' << s[2] << '\n';
```

Pentru `6` și `82 95 70 99 88 91` afișează `min 70, max 99, media 87.50` și `podium: 99 95 91`. În Python: `min(v)`, `max(v)`, `sum(v) / len(v)`, `sorted(v, reverse=True)[:3]`. Atenție la `(double)suma / n`: fără conversie, `525 / 6` ar da `87`.

> **LA BAC**
>
> La Bac, când se cere definiția unui subprogram C++, antetul este dat: respectă tipul returnat, ordinea și **felul** parametrilor (cu sau fără `&`). Un `&` lipsă sau în plus schimbă complet ce face subprogramul.

7. Verifică-te

1. Numește părțile antetului `long long cub(int x)`.
2. Ce afișează `cout << f(x) << ' ' << x;` pentru `int x = 4;` și `int f(int a) { a += 3; return a * 2; }`? Dar dacă antetul devine `int f(int &a)`?
3. De ce nu compilează `max(n, 0.5)` pentru `int n`? Cum o repari?

4. Ce afișează `cout << round(6.5) << ' ' << floor(6.5) << ' ' << ceil(6.5);` ? Ce dă `round(6.5)` în Python?
5. Scrie prototipul unei funcții care primește un tablou de `n` întregi și returnează maximul.
6. Ce afișează `cout << pow(2, 30);` ? Cum obții toate cifrele?

Răspunsuri

1. Tipul returnat `long long`, numele `cub`, un parametru `x` de tip `int`.
2. `14 4`: `a` este o copie. Cu `&`: `14 7`, pentru că `a` este chiar `x`.
3. `max` cere două valori de **același** tip, iar aici sunt `int` și `double`. Scrii `max((double)n, 0.5)` sau `max<double>(n, 0.5)`.
4. `7 6 7`. În Python, `round(6.5)` este `6` (rotunjire spre par).
5. `int maxim(int a[], int n);`
6. `1.07374e+09`. Conversia `(long long)pow(2, 30)` sau `fixed << setprecision(0)` afișează `1073741824`. Pentru puteri întregi mari, mai sigur e un produs în buclă (vezi fișa).

Probleme de lucru în clasă (pbinfo)

Le rezolvăm împreună în ședință, în ordinea de mai jos.

- [ ] `#1282` **radical** — ușoară
  https://www.pbinfo.ro/probleme/1282/radical
- [ ] `#1376` **laturitriunghi** — ușoară
  https://www.pbinfo.ro/probleme/1376/laturitriunghi
- [ ] `#1625` **ec2** — ușoară
  https://www.pbinfo.ro/probleme/1625/ec2

Temă pentru acasă (pbinfo)

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

- [ ] `#922` **puncte** — ușoară
  https://www.pbinfo.ro/probleme/922/puncte
- [ ] `#2603` **primacifraapartiizecimale** — ușoară
  https://www.pbinfo.ro/probleme/2603/primacifraapartiizecimale
- [ ] `#129` **sortare** — ușoară
  https://www.pbinfo.ro/probleme/129/sortare