INFORMATICĂ · PYTHON · CLASA A IX-A · MEMORAREA DATELOR ȘI ORGANIZAREA CODULUI (INTENSIV) · LECȚIA I4

Fișiere text în C++: ifstream și ofstream

Durată: 100 min Nivel: intensiv · Python + C++ Înainte: lecțiile py9-03, py9-10, py9-13

CE VEI ȘTI DUPĂ LECȚIE

- Scrii în C++ un program care citește dintr-un fișier cu >> și scrie rezultatele în alt fișier cu <<, în paralel cu varianta Python.
- Citești corect „până la sfârșitul fișierului” cu while (f >> x) și explici de ce while (!f.eof()) dă o valoare în plus.
- Tratezi cazurile limită: fișier inexistent (if (!f)), fișier gol, date greșite, și compari reacția celor două limbaje.
- Alegi între suprascriere și adăugare (ios::app) și explici de ce contează close().

1. Ce știi deja și ce e nou

Din lecția py9-13 știi ce este un fișier text, cele trei etape (deschidere, transfer, închidere) și modurile `"r"`, `"w"`, `"a"`. **Ideile rămân aceleași în C++**; se schimbă uneltele. În loc de un singur `open()` care întoarce un obiect `TextIOWrapper`, C++ are **două clase** din biblioteca `<fstream>`: `ifstream` (*input file stream*, pentru citire) și `ofstream` (*output file stream*, pentru scriere).

Etapa	Python	C++
biblioteca	(nimic, `open` e predefinit)	`#include <fstream>`
deschidere pentru citire	`f = open("date.in")`	`ifstream f("date.in");`
deschidere pentru scriere	`g = open("date.out", "w")`	`ofstream g("date.out");`
deschidere pentru adăugare	`g = open("log.txt", "a")`	`ofstream g("log.txt", ios::app);`
citire	`f.read()`, `f.readline()` + `split()` + `int()`	`f >> x;` (citește și convertește)
scriere	`g.write(str(x) + "\n")`	`g << x << '\n';`
închidere	`f.close()` sau automat cu `with`	`f.close();` sau automat la ieșirea din `main`

`f` și `g` sunt **obiecte**, exact ca în Python (lecția py9-10): `ifstream f("date.in");` declară obiectul și îl leagă de fișier dintr-un singur pas. Le folosești cu aceiași operatori pe care îi știi de la `cin` și `cout`: `>>` pentru citire, `<<` pentru scriere.

2. Primul program: sum.in → sum.out

Fișierul `sum.in` conține `12 30`. Programul scrie suma în `sum.out`.

```cpp
#include <fstream>
using namespace std;
int main() {
    ifstream f("sum.in");      // 1. deschidere pentru citire
    ofstream g("sum.out");     //    și pentru scriere
    int a, b;
    f >> a >> b;               // 2. transfer: citire
    g << a + b << '\n';        //    transfer: scriere
    f.close();                 // 3. închidere
    g.close();
    return 0;
}
```

```python
with open("sum.in") as f:
    a, b = map(int, f.read().split())
with open("sum.out", "w") as g:
    g.write(str(a + b) + "\n")
```

Ambele programe scriu în `sum.out` rândul `42`.

Ce face `f >> a`: sare peste orice spații și treceri la rând nou, citește caracterele care formează un număr și le **convertește** în tipul variabilei (`int`). Deci `f >> x` face din C++ ce face `int(...)` aplicat pe `read().split()` din Python: nu contează cum sunt așezate numerele pe rânduri.

> **SFAT**
>
> Dacă fișierul are numele corect, dar programul „nu vede" datele, verifică întâi **folderul**: ca și în Python, un nume fără cale se caută în folderul din care rulează programul (la Code::Blocks, folderul proiectului).

3. Un număr n, apoi n valori

Cel mai des întâlnit format: pe primul rând `n`, apoi `n` valori. Exemplu: catalogul unei clase. `note.in` conține `5` pe primul rând și `9 7 10 8 6` pe al doilea. Vrem media cu două zecimale și numărul notelor de cel puțin 9.

```cpp
#include <fstream>
#include <iomanip>
using namespace std;
int main() {
    ifstream f("note.in");
    ofstream g("note.out");
    int n, x, s = 0, mari = 0;
    f >> n;                          // primul număr: câte note urmează
    for (int i = 0; i < n; i++) {
        f >> x;
        s += x;
        if (x >= 9) mari++;
    }
    g << fixed << setprecision(2) << (double)s / n << '\n';
    g << mari << '\n';
    return 0;
}
```

```python
with open("note.in") as f:
    n = int(f.readline())
    note = list(map(int, f.readline().split()))
with open("note.out", "w") as g:
    print(f"{sum(note) / n:.2f}", file=g)
    print(sum(1 for x in note if x >= 9), file=g)
```

`note.out` conține `8.00` și `2`, în ambele variante.

Python	C++	De reținut
`f"{m:.2f}"`	`g << fixed << setprecision(2) << m`	`setprecision` cere `#include <iomanip>`
`sum(note) / n`	`(double)s / n`	fără `(double)`, `/` între `int`-uri dă câtul
`print(…, file=g)`	`g << … << '\n'`	`<<` **nu** pune singur spații sau `\n`

Varianta C++ nu ține notele minte: le prelucrează pe rând, din mers. Varianta Python de mai sus le pune într-o listă; și în Python se poate din mers, ca în lecția py9-13.

4. Până la sfârșitul fișierului: while (f >> x)

De multe ori nu știi câte valori sunt. Un jurnal de temperaturi, `temperaturi.in`, conține valori puse oricum: `3 -2 5`, `7`, un rând gol, `-4 0 6`.

```cpp
#include <fstream>
#include <iostream>
using namespace std;
int main() {
    ifstream f("temperaturi.in");
    int t, zile = 0, s = 0, mn = 1000, mx = -1000;
    while (f >> t) {               // cât timp citirea a reușit
        zile++;
        s += t;
        if (t < mn) mn = t;
        if (t > mx) mx = t;
    }
    cout << zile << " zile, min " << mn << ", max " << mx
         << ", media " << (double)s / zile << '\n';
    return 0;
}
```

```
7 zile, min -4, max 7, media 2.14286
```

De ce merge: expresia `f >> t` face citirea și apoi „valorează” chiar obiectul `f`. Pus într-o condiție, `f` devine `true` dacă ultima operație a reușit și `false` dacă a eșuat (nu mai era niciun număr). Deci bucla se oprește **exact** când citirea eșuează, iar ultima încercare, cea ratată, nu ajunge în corpul buclei.

Echivalentul Python: `for t in f.read().split():` (sau rând cu rând, cu `for linie in f:`).

LA BAC

La Bac, Subiectul III.3 dă un fișier (de tipul `bac.txt`) cu un șir de numere, adesea fără să spună câte sunt. Tiparul `while (f >> x) { … }` este cel așteptat; dacă se cere și eficiență, prelucrezi din mers, fără să memorezi tot șirul.

5. Capcana: while (!f.eof())

Ar părea firesc să scrii „cât timp nu s-a terminat fișierul, citește". Fișierul `eof.in` conține `3 8 5` și un Enter la final.

```
ifstream f("eof.in");
int x, c = 0;
while (!f.eof()) {              // GREȘIT
    f >> x;
    c++;
    cout << x << ' ';
}
cout << "-> " << c << " valori\n";
```

```
3 8 5 5 -> 4 valori
```

De ce: `eof()` („end of file") devine adevărat **abia după ce o citire a încercat să treacă de capătul fișierului**, nu când cursorul a ajuns aproape de el. După ce s-a citit `5`, în fișier mai rămâne `\n`, deci `eof()` e încă fals. Bucla mai face un pas: `f >> x` sare peste `\n`, dă de capăt și **eșuează**, iar `x` își păstrează vechea valoare. Așa apare un `5` în plus și contorul ajunge 4.

Dacă fișierul **nu** are Enter după ultimul număr, aceeași buclă afișează corect `3 8 5 -> 3 valori`. Adică programul greșit pare bun la unele teste și pică la altele: cel mai rău tip de bug (lecția py9-03).

	`while (!f.eof()) { f >> x; … }`	`while (f >> x) { … }`
verifică	„am trecut deja de capăt?"	„citirea de acum a reușit?"
cu Enter la final	o iterație în plus, cu o valoare veche	corect
fișier gol	o iterație cu `x` neinițializat	0 iterații, corect

În Python problema nu apare: `readline()` dă `''` exact când nu mai este nimic de citit (lecția py9-13), iar `for` pe un fișier se oprește singur.

6. Fișier inexistent, fișier gol, date greșite

Aici cele două limbaje se comportă **foarte diferit**. Python se oprește cu o eroare clară. C++ **nu dă nicio eroare**: obiectul intră într-o stare de eșec, iar toate citirile următoare eșuează în tăcere.

Situația	Python	C++
fișierul nu există	`FileNotFoundError`	`f` nu se deschide; `while (f >> x)` face 0 pași
fișierul e gol	`read()` dă `''`, lista e goală	`while (f >> x)` face 0 pași

Situația	Python	C++
date greșite: `4 7 x 9`	`ValueError: invalid literal … 'x'`	se citesc `4` și `7`, apoi bucla se oprește la `x`: suma afișată e `11`
împarți la numărul de valori, care e 0	`ZeroDivisionError`	la `int`: eroare la rulare; la `double`: rezultat `nan` sau `inf`

Un program care citește din `lipsa.in` (inexistent) cu `while (f >> x) c++;` afișează liniștit `am citit 0 valori`. Ca să știi ce s-a întâmplat, **verifică deschiderea**:

```
ifstream f("lipsa.in");
if (!f) {                                   // deschiderea a eșuat
    cout << "Nu pot deschide lipsa.in\n";
    return 1;                               // ieșire din main cu cod de eroare
}
```

```
Nu pot deschide lipsa.in
```

`if (!f)` se citește „dacă `f` este în stare de eșec”, la fel cum `while (f >> x)` întreabă „dacă `f` este în stare bună”. Corespondentul Python este `try` / `except FileNotFoundError`.

> **ATENȚIE — GREȘELI FRECVENTE**
>
> Pentru un program testat serios, ai nevoie de cel puțin trei teste „neobișnuite”: fișier lipsă, fișier gol, fișier cu o singură valoare. Pe pbinfo fișierul există mereu, dar în aplicațiile reale (și la evaluarea pe care o cere programa) nu e așa.

7. Scrierea: suprascriere, adăugare, close()

`ofstream g("x.out");` se comportă ca `"w"`: creează fișierul sau **îl golește** dacă exista. Pentru a adăuga la sfârșit, ca `"a"`, folosești `ios::app`. Exemplu: un server își scrie evenimentele într-un jurnal (`ora cod`, unde codurile de la 500 în sus sunt erori), iar apoi programul numără erorile.

```cpp
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
int main() {
    ofstream g("server.log", ios::app);    // ca "a" în Python
    g << "08:15 200\n" << "08:16 500\n" << "08:20 404\n";
    g.close();                             // închide ÎNAINTE să citești

    ifstream f("server.log");
    string ora;
    int cod, erori = 0;
    while (f >> ora >> cod)                // un cuvânt în string, un număr în int
        if (cod >= 500) {
            erori++;
            cout << "eroare la " << ora << '\n';
        }
    cout << erori << " erori\n";
    return 0;
}
```

La prima rulare afișează `eroare la 08:16` și `1 erori`. La a doua rulare jurnalul are deja 6 rânduri, deci afișează de două ori `eroare la 08:16` și `2 erori`. Cu `ofstream g("server.log");` (fără `ios::app`) ar fi afișat la fiecare rulare `1 erori`.

Observă `f >> ora` cu `ora` de tip `string`: `>>` citește un **cuvânt** (până la primul spațiu sau Enter), exact ca un element din `split()`.

De ce contează `close()`. `ofstream` nu scrie pe disc la fiecare `<<`: adună textul într-o zonă de memorie (*buffer*) și îl trimite pe disc la închidere. Dacă ștergi rândul `g.close();` din programul de mai sus, la prima rulare `f` găsește fișierul încă gol și afișează `0 erori`. La sfârșitul lui `main` obiectele se închid singure, dar dacă **tot în program** vrei să citești ce ai scris, închide explicit.

> **DE ȘTIUT**
>
> Pentru a citi **un rând întreg** (cu spații cu tot), C++ are `getline(f, linie);` cu `linie` de tip `string`, care corespunde lui `linie = f.readline().rstrip("\n")`. Pentru numere, rămâi la `>>`.

8. Verifică-te

1. Ce bibliotecă incluzi pentru fișiere în C++ și ce clase folosești pentru citire, respectiv pentru scriere?
2. `q2.in` conține `10 20 30` și un Enter. Ce sumă calculează `while (!f.eof()) { f >> x; s += x; }`? Dar `while (f >> x) s += x;`?
3. Care este diferența dintre `ofstream g("a.txt");` și `ofstream g("a.txt", ios::app);`? Care sunt corespondentele în Python?
4. Fișierul conține `5 abc 7`. Ce sumă dă `while (f >> x) s += x;` în C++? Ce se întâmplă cu `sum(int(x) for x in f.read().split())` în Python?
5. Traduce în C++: `with open("x.in") as f: n = int(f.readline())`.
6. Un program C++ citește dintr-un fișier care nu există și afișează `0`. De ce nu a dat nicio eroare și cum îl faci să anunțe problema?

Răspunsuri

1. `<fstream>`; `ifstream` pentru citire, `ofstream` pentru scriere.
2. Cu `eof()`: `90` (ultimul `30` se adună de două ori). Cu `while (f >> x)`: `60`.
3. Primul golește fișierul (ca `"w"`), al doilea adaugă la sfârșit (ca `"a"`).
4. C++: `5` (bucla se oprește la `abc`, iar `7` nu mai e citit). Python: `ValueError: invalid literal for int() with base 10: 'abc'`.
5. `ifstream f("x.in"); int n; f >> n;` (plus `#include <fstream>`).
6. În C++ deschiderea eșuată nu oprește programul: `f` intră în stare de eșec și citirile nu reușesc. Verifici imediat după deschidere cu `if (!f) { … }`.

Probleme de lucru în clasă (pbinfo)

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

- [ ] #18 **numarul-de-divizori** — ușoară
  https://www.pbinfo.ro/probleme/18/numarul-de-divizori
- [ ] #2275 **minimdoua** — ușoară
  https://www.pbinfo.ro/probleme/2275/minimdoua
- [ ] #1881 **platou4** — ușoară
  https://www.pbinfo.ro/probleme/1881/platou4

Temă pentru acasă (pbinfo)

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

- [ ] #3391 **sirmaxim** — ușoară
  https://www.pbinfo.ro/probleme/3391/sirmaxim
- [ ] #2269 **maximpar** — ușoară
  https://www.pbinfo.ro/probleme/2269/maximpar
- [ ] #8 **maxim3** — ușoară
  https://www.pbinfo.ro/probleme/8/maxim3