

# Cerințe proiect laborator POO

[Cerințe etapa 1](#) | [Cerințe etapa 2](#) | [Cerințe etapa 3](#)

## Prezentare generală

Nota voastră la laboratorul de POO se acordă pe baza unui **proiect individual**, dezvoltat pe parcursul primelor 13 săptămâni din semestru<sup>1</sup>. Proiectul va fi evaluat în **trei etape**, cu cerințe și deadline-uri distincte pentru fiecare<sup>2</sup>.

Proiectul constă în dezvoltarea unei **aplicații C++** care **să gestioneze un model de date**, folosind paradigma de **programare orientată pe obiecte**. Va trebui să definiți atât clasele care să reprezinte entitățile din modelul vostru, cât și metodele/funcțiile corespunzătoare care să implementeze „logica de business”.

## Notare

Pentru fiecare etapă, veți primi o **notă de la 1 la 12**. Nota 10 se acordă pentru implementarea corectă a tuturor **cerințelor de bază**, putând primi până la două puncte în plus dacă rezolvați și **cerințele suplimentare** (vor fi indicate corespunzător).

Nota finală este media notelor de pe fiecare etapă. Trebuie să aveți minim **nota 5** ca să promovați laboratorul și să puteți participa la colocviu/examen.

## Alegerea temei

Recomandarea mea este să alegeți o temă care vă pasionează, ca să fiți motivați să lucrați la proiect și să îl duceți până la capăt.

## Exemple de teme

- Magazin online (produse, cumpărători, comenzi, reduceri etc.)
- Gestiunea resurselor umane (firmă, angajați, echipe, salarii etc.)

---

<sup>1</sup>În ultima săptămână veți da un colocviu (test pe calculator), până la acel moment trebuie să aveți situația la laborator încheiată.

<sup>2</sup>Cerințele sunt gândite în așa fel încât să continuați programul din etapa anterioară, adăugând funcționalități în plus la codul existent.

- Gestiunea unei clinici/unui spital (medici, pacienți, consultații, medicamente etc.)
- Project management (proiect, task, echipă, membru al echipei etc.)
- Gestiunea școlarității (studenți, note, discipline, profesori etc.)
- Bibliotecă (carte, autor, categorie, împrumut etc.)
- Music player (album, artist, melodie, playlist etc.)
- Grădină zoologică (animale, bilete, hrană pentru animale etc.)
- Formula 1 (echipe, mașini, piloți, raliuri etc.)
- Cinematograf (filme, actori, vizionări, bilete etc.)

Puteți alege orice alt domeniu doriți. Indiferent de tema aleasă, va **trebui să o confirmați** cu mine (în persoană când veniți la laborator, sau pe e-mail/Teams dacă nu puteți ajunge) **până la sfârșitul săptămânii 3** de facultate.

De preferat ar fi să vă alegeți teme distincte. Cel mult doi studenți pot avea o temă identică/similară. Chiar și în acest caz, vor trebui să aibă o structură diferită a claselor.

**Recomandare:** după ce v-ați ales tema și ați confirmat-o, puteți începe prin a vă face o „schită” cu clasele de care veți avea nevoie, ce date și metode vor reține fiecare și care vor fi legăturile dintre ele. Puteți face asta pe hârtie sau folosind o aplicație cum ar fi [Excalidraw](#).

**Recomandare:** familiarizați-vă cât mai curând cu și începeți să folosiți debugger-ul din mediul vostru de lucru. O să vă petreceți 20% din timp scriind efectiv cod și peste 80% din timp încercând să înțelegeți de ce nu funcționează cum v-ați fi așteptat.

## Cerințe etapa 1

**Deadline:** cerințele trebuie implementate până la sfârșitul **săptămânii 5** de facultate (**26 martie 2023**). Pentru a fi punctați, va trebui să le și prezentați la laboratorul din **săptămâna 6**.

### Criterii generale

(1p)

- Trebuie să aveți în main un cod corespunzător care să testeze/utilizeze funcționalitățile implementate.
- Clasele/metodele care nu sunt utilizate sau la care nu se face referire nicăieri **nu vor fi punctate**.
- Încercați să păstrați codul curat:
  - Folosiți nume de variabile/funcții/tipuri de date cu sens pentru oameni (e.g. `nr_angajati` în loc de `n`).
  - Folosiți formatarea automată oferită de editorul vostru de text și încercați să păstrați un stil uniform.
  - Folosiți mai degrabă mai multe clase/metode mai scurte, decât o singură clasă/metodă foarte lungă.

### Versionarea codului

(1p)

- Codul sursă **trebuie** să fie încărcat pe GitHub.
- Trebuie să îmi dați și mie **acces de citire** la repository-ul în care aveți proiectul, dacă nu este public.

Instrucțiunile pentru cum puteți adăuga colaboratori la un repo se găsesc [aici](#). Mă puteți adăuga prin username (GabrielMajeri) sau prin e-mail (`constantin.majeri@s.unibuc.ro`).
- **Recomandare:** adăugați de la început un fișier `.gitignore` la repo-ul vostru ca să nu includeți accidental fișierele compilate / binare în Git. Puteți găsi [aici](#) un exemplu de fișier `.gitignore` pentru C++.
- **Recomandare:** păstrați commit-urile concise și independente. Găsiți [aici](#) un set de bune practici pentru commit-urile de Git.

## Documentație

(1p)

- În repo-ul de pe GitHub trebuie să aveți și [un fișier README](#), de preferat formatat cu [Markdown](#), în care să includeți cel puțin:
  - **Numele** proiectului
  - **Tema** aleasă
  - O listă cu **clasele** pe care le-ați implementat și o scurtă descriere a ce reprezintă fiecare
  - O listă cu **funcționalitățile** pe care le are aplicația voastră la momentul actual (ex.: „capabilă să citească și să rețină o listă de angajați”, „poate calcula prețul mediu al produselor aflate în stoc” etc.)
- **Recomandare:** actualizați acest fișier pe parcurs ce dezvoltați proiectul. Vă va fi mult mai ușor să-l prezentați altor persoane.

## Clase

(2p)

- Trebuie să respectați **principiul encapsulării** (nu aveți voie să aveți date membre publice). Metodele pot fi publice sau private.
- Trebuie să definiți **minim 5 clase**, relevante pentru tema aleasă.

Sunt luate în considerare și clasele utilitare, cum ar fi o clasă pentru șiruri de caractere sau una pentru vectori alocați dinamic.
- **Minim 3 dintre clase** trebuie să fie corelate prin **compunere** (ex. să aveți o dată membru de tip Adresă în clasa Contact, să aveți un vector de Angajat în clasa Companie etc.)
- Pentru etapa 1, **nu** aveți voie să folosiți clasele container/utilitare din biblioteca standard (`std::vector`, `std::string` șamd). Puteți refolosi în schimb clasele implementate de voi la laborator.

## Metode

(2p)

- **Minim 3 getteri și 3 setteri** pentru datele membru (pot fi pentru câmpuri diferite, din clase diferite).
- **Minim 2 metode de „logică de business”** care să aibă sens pentru tema voastră (e.g.: calculează prețul unui produs aplicând o reducere, returnează salariul mediu al angajaților din firmă etc.)

- Pentru **minim 2 clase** trebuie să implementați o metodă de **afișare** pe ecran a datelor din acea clasă (în ce format vreți voi).

Alternativ, se acceptă și doar să supraîncărcați operator<< pentru aceste clase și să implementați acolo afișarea.

- Pentru **minim o clasă** trebuie să supraîncărcați operator<< pentru afișare (dacă ați făcut acest lucru la subpunctul anterior, nu este nevoie să o mai faceți încă o dată).

## Constructori

(0,5p)

- **Minim 3 clase** trebuie să aibă constructori fără parametru, respectiv parametrizați, care să inițializeze datele membre.

## Constructori de copiere și operator=

(0,5p)

- Pentru **minim două clase** trebuie să implementați explicit constructorul de copiere și să supraîncărcați operator=.
- **Cel puțin una** dintre aceste clase trebuie să fie dintre cele care folosesc **alocare dinamică**.

## Alocare dinamică

(1p)

- **Minim o clasă** trebuie să facă alocare dinamică de memorie (fie pentru un singur obiect, fie pentru un vector de obiecte).
- Toată memoria alocată dinamic trebuie **eliberată** corespunzător. Veți fi depunctați pentru *memory leaks*.
- **Minim o clasă** trebuie să aibă un **destructor** (netrivial).

## Bonus

(2p)

Fiecare cerință bonus valorează **1 punct**. Puteți rezolva la alegere oricare două dintre cele trei pentru a obține punctajul maxim.

- Implementați un **meniu interactiv** în main, care să permită:
  - **Citirea** de la tastatură și **crearea a cel puțin 3 tipuri de obiecte** dintre cele definite de voi;
  - **Afișarea** acestor obiecte, după ce au fost citite de aplicație;

- **Apelarea unei metode** pe ele (alta în afară de cea de afișare).
- Implementați **constructorul de mutare** și supraîncărcați **operator= de mutare** pentru cel puțin o clasă care gestionează memorie alocată dinamic.
- Definiți și utilizați o **clasă imbricată**, asemănător cu clasa Nod, definită la exercițiul cu IntList de la laborator (dar trebuie să aibă sens pentru tema voastră).

**Oficiu**

(1p)

## Cerințe etapa 2

**Deadline:** cerințele trebuie implementate până la sfârșitul **săptămânii 9** de facultate (**2 mai 2023**). Pentru a fi punctați, va trebui să le și prezentați la laboratorul din **săptămâna 10**.

Începând cu etapa a 2-a, **aveți voie** să folosiți clasele container din biblioteca standard (`std::string`, `std::vector` etc.)

### Moștenire

(2p)

- Definiți **minim două ierarhii diferite de moștenire** (două ierarhii de moștenire sunt considerate diferite dacă nu au aceeași clasă de bază și aceeași clasă care moștenește clasa/clasele de bază).
- Utilizați **minim doi modificatori de acces diferiți** pe clasa care se moștenește (`public/protected/private`); alegerea vă aparține, dar trebuie să fie justificată în funcție de nevoile proiectului.
- **Minim o clasă** care să folosească moștenire multiplă (să extindă mai multe clase/interfețe).
- Apelați **cel puțin o dată** un constructor (cu parametri) dintr-o clasă de bază, folosind o listă de inițializare în constructorul clasei copil.
- **Minim două date membru** și **minim o metodă** care să aibă modificatorul de acces `protected` (în mod util, să fie accesate/apelate dintr-o clasă care le moștenește).

### Interfețe și metode virtuale

(2p)

- Definiți și extindeți (moșteniți) **minim o interfață** (clasă fără date membru, doar metode pur virtuale și un destructor virtual) care să aibă **minim două metode** (alternativ: minim două interfete, fiecare cu cel puțin o metodă).
- Definiți și extindeți (moșteniți) **minim o clasă de bază abstractă** (clasă care poate avea date membru, dar are cel puțin o metodă pur virtuală).
- Trebuie să aveți / să identificați în proiect **cel puțin o situație** în care să fie nevoie de și să se apeleze destructorul virtual (i.e. să ștergeți un obiect alocat dinamic de tipul unei clase moștenitoare, la care să vă referiți prin intermediul unui pointer la clasa de bază).
- Definiți **cel puțin 4 metode virtuale** care să fie suprascrise în clasele moștenitoare. Pot fi pur virtuale sau cu o implementare implicită. Se iau în con-

siderare și metodele definite la celelalte subpuncte, exceptând destructorii virtuali.

### Polimorfism la execuție

(2p)

- Identificați **minim 4 locuri** în care să aibă loc polimorfism la execuție (*dynamic dispatch*) în proiectul vostru (e.g. apelul unor metode virtuale prin intermediul unor pointeri/referințe către clasa de bază).
- Identificați **minim 4 instanțe de upcasting** în codul vostru (e.g. atribuirea unor obiecte de tipul unor clase moștenite la pointeri/referințe către clasa de bază).
- Realizarea downcasting-ului **în cel puțin o situație** în codul vostru (unde are sens), folosind `dynamic_cast` sau RTTI.

### Excepții

(2p)

- Definiți **minim un tip de excepție custom**, care să extindă clasa `exception` din biblioteca standard.
- Aruncați excepții în **minim 4 funcții/metode diferite** (folosiți tipuri de excepții definite de voi sau cele din biblioteca standard).
- Implementați **minim un bloc** `try...catch` care să prindă o excepție aruncată de voi (cu mențiunea explicită a tipului acesteia) și să o trateze într-un fel (în funcție de specificul erorii).
- Implementați **minim un bloc** `try...catch` care să prindă o excepție, să o proceseze și să re-arunce un alt tip de excepție din blocul `catch`.

### Variabile și metode statice

(1p)

- Definiți o variabilă membru statică în **cel puțin o clasă**.
- Implementați **cel puțin două metode statice** în clasele voastre (din care **cel puțin una** trebuie să acceseze/folosească variabila statică definită la subpunctul anterior).

### Bonus

(2p)

Fiecare cerință bonus valorează **1 punct**.



- **Minim o situație** de moștenire în diamant (trebuie să aibă sens pentru tema voastră). Moștenirea în diamant se referă la a avea o clasă de bază, pe care o moștenesc cu virtual două clase distincte, iar apoi aveți o clasă care moștenește ambele clase intermediare.
- Utilizarea **unei lambda expresii** pentru a parametriza funcționalitatea unei funcții sau a unei clase. Puteți transmite ca parametru sau reține o lambda expresie folosind tipul de date `std::function`.

Se acordă punctaj parțial dacă doar folosiți lambda expresii pentru apelarea unor funcții care deja există în biblioteca standard (de exemplu, `std::sort`), fără să definiți voi o funcție/clasă care să primească un obiect de acest tip.

**Oficiu**

(1p)

## Cerințe etapa 3

**Deadline:** cerințele trebuie implementate până la sfârșitul **săptămânii 12** de facultate (**21 mai 2023**). Pentru a fi punctați, va trebui să le și prezentați la laboratorul din **săptămâna 13**.

### Programare generică

(3p)

- Utilizați **minim o clasă șablon** (template) **definită de voi**. Trebuie să fie parametrizată de **cel puțin un tip de date generic** (cel puțin un typename), care să fie folosit în mod util în interiorul clasei (e.g. pentru a defini un atribut, o metodă etc.).
- Definiți și apelați **minim o funcție șablon** (poate fi funcție liberă sau metodă a unei clase care nu este neapărat generică). Trebuie să fie parametrizată de **cel puțin un tip de date generic** (cel puțin un typename), care să fie folosit în definirea funcției (e.g. parametru, tip de date returnat).
- Definiți **cel puțin o implementare specializată** pentru clasa șablon sau funcția șablon (nu neapărat pentru ambele).

**Referințe utile:** [specializarea claselor/funcțiilor șablon în C++](#).

### Design patterns

(2p)

Identificați **minim 2 object-oriented design patterns diferite** în clasele implementate de voi.

**Observație:** pattern-ul trebuie să existe în codul scris de voi. De exemplu, dacă ați folosit o clasă iterator deja existentă din biblioteca standard, nu înseamnă că ați implementat design pattern-ul *iterator*.

**Referințe utile:** [design patterns](#) cu exemple de cod și când se utilizează, [catalog de design patterns comune](#).

### Biblioteca standard

(2p)

- Utilizarea a **minim două tipuri de date container diferite** din STL.

**Exemple:** [vector](#), [string](#), [array](#), [list](#), [set](#), [map](#) etc.

**Observație:** ar trebui să puteți înlocui toată alocarea dinamică explicită din codul vostru cu clasele container din biblioteca standard, plus clase de tip *smart pointers* (menționate mai jos).

- Utilizarea a **minim două funcții utilitare diferite** din biblioteca standard (funcții libere, nu metode).

**Exemple:** `sort`, `find`, `search`, `all_of/any_of/none_of`, `accumulate`, `fill`, `generate`, `copy`, `reverse`, orice alte funcții din fișierul header din biblioteca standard `<algorithm>`.

## Smart pointers

(2p)

Utilizați în mod corespunzător **minim două tipuri diferite de *smart pointers*** din biblioteca standard.

Prin *smart pointer* ne referim la una dintre clasele `std::reference_wrapper`<sup>3</sup>, `std::unique_ptr`, `std::shared_ptr` sau `std::weak_ptr`.

Puteți să folosiți aceste clase în locul referințelor sau pointerilor obișnuiți din codul vostru.

**Referințe utile:** [avantajele smart pointers](#), [utilizarea smart pointers în C++](#).

## Bonus

(2p)

Fiecare cerință bonus valorează **1 punct**.

- **Utilizarea modificadorului `const`** în mod corect, peste tot unde este posibil și are sens în program (pe date membre, pe metode, la transmiterea parametrilor prin referință etc.).

**Referințe utile:** [const correctness](#).

- Folosirea **unei biblioteci utilitare** din biblioteca standard, în afară de cele cu care am lucrat la laborator. Puteți alege (în funcție de specificul proiectului) dintre:
  - `chrono`: lucru cu date și timp.
  - `regex`: lucru cu expresii regulate (pentru căutarea/înlocuirea de text).
  - `random`: generare de numere aleatoare.
  - `thread`: execuție paralelă a codului.

Pentru a fi punctați, trebuie să **instanțiați** și să **utilizați** cel puțin câteva tipuri de date din biblioteca aleasă. De exemplu, pentru `random`: să inițializați

---

<sup>3</sup>`reference_wrapper` este mai degrabă un *smart reference*, dar se utilizează în mod similar cu un *smart pointer*.

un generator de numere random, să alegeți o distribuție, să generați niște numere aleatoare pe care să le utilizați în aplicație.

**Oficiu**

(1p)