**Project OOP**

**Cerințe proiect (V1):**

Proiectele voastre au teme diferite și nu vreau să vă îngrădesc imaginația sau viziunea în general, dat fiind că acest curs are ca subiect principal arhitectura codului. Dar, pentru claritate, puteți să faceți referința la lista care urmează pentru a ști ce ar trebui să conțină proiectul în forma lui finală. În mare, **obiectivul proiectului este să dezvoltați o soluție de dimensiune medie spre mare** (în raport cu ce ați lucrat până acum la facultate), **în care să folosiți principiile arhitecturii programării orientate pe obiecte, în limbajul C++ (11)**. După alegerea temei, va trebui să abstractizați subiectul pe care l-ați ales și să dezvoltați o funcționalitate bazată pe temă, cu arhitectura de tip OOP.

Sub-obiectivele voastre pe care trebuie să le atingeți în proiect sunt:

* Respectarea coding-guideline-ului (pe care puteți să îl găsiți după descrierea proiectului în acest document)
* Realizarea temei alese până la capăt. Cum ați avut liber la alegerea temei, înafara de cerințele de mai jos, scope-ul proiectului este la latitudinea voastră. Cum am spus un pic mai sus, deși subiectul pe care îl studiem este unul de arhitectură, proiectul vostru trebuie să conțină și o parte de algoritmică în care să dezvoltați o "funcționalitate" (am folosit termenul foarte general de "funcționalitate" pentru că este la latitudinea voastră dacă vreți să faceți ceva care să aibă doar un UI funcțional, un joc, o simulare etc). Este foarte important însă să duceți până la capăt tema pe care ați ales-o indiferent de scope-ul acesteia.

**Este foarte important însă să duceți până la capăt tema pe care ați ales-o indiferent de scope-ul acesteia.**

* **10+ Clase** (nu vreau să tratați acest număr ca un prag, proiectul vostru poate să aibe 20 de clase sau 8, atâta vreme cât puteți să vă justificați alegerea. Numărul a fost ales pentru a arăta mai degraba dimensiunea ideală la care ar trebui să ajungă proiectul vostru)
* Să demonstrați, în mod repetat și variat, principiile de bază a OOP:
  + o **Abstractizare** (natura proiectului în sine vă obliga să folosiți acest concept)
  + o **Moștenire** – Trebuie să realizați una sau mai multe ierarhii de clase organizate într-un mod eficient
  + o **Încapsulare** – Coding-guideline-ul vă constrânge să folosiți încapsularea
  + o **Polimorfism** – În cazul claselor – overriding și folosirea obiectelor derivate ca alt tip de obiecte
* **Supraîncărcare de constructori și destructori**
* **Creerea unor situații în care să se folosească atât obiecte de pe stack cât și de pe heap**
* **Utilizarea de struct împreună cu clase**
* **Clasa care conține ca membru o instanță a unei alte clase**
* **Clasa care să aibe unul sau mai mulți membrii statici**
* **Supraîncărcarea a "+" și "<<"**
* **Folosirea de funcții virtuale**
* **Folosirea Interfețelor**
* **Supraîncărcare funcțiilor**
* **Utilizați metode private în mod rațional.**

**Code Guideline:**

**Versiune C++ - 11**, evitați să folosiți feature din versiuni mai noi, nu că ar fi ceva în neregula propiu-zis dar ar fi bine să rămânem cât mai apropiați de curs.

În general, **fiecare fișier "cpp" trebuie să aibă un fișier "h" asociat**. Evident există excepții (ex: în cazul în care vă definiți un fișier cu funcții "helper").

**Nu includeți headere pe care nu le folosiți.** > Header guards

**Evitați să folosiți "forward declarations"**

**Folosiți Namespace-uri pentru cod** – codul trebuie să fie aproape întodeauna plasat într-un namespace, care la rândul lor trebuie să aibe nume clare și unice.

Când folosiți membrii statici pentru o clăsă, aceștia trebuie să fie asociați clar cu instanțele clasei, nu folosiți clase pentru a grupa funcții statice. Folosiți namespace-uri atunci când vreți să grupați bucăți de cod.

**Plasați variabilele astfel încât să aibe cel mai mic scope posibil.**

**Class:**

**Evitați să apelați metode virtuale în constructori.**

**Struct vs Class**: folosiți struct pentru structuri de date care doar țin date. Folosiți Class pentru structuri de date care necesită interfață. Preferați să folosiți struct în loc de tuple.

**Inheritance**: când folosiți moștenirea, folosiți "public"

**Operator Overloading**: Folosiți operator overloading în situații în care uzul este evident și natural. Definiți acești operatori în același loc cu header-ele/namespace-urile în care sunt folosiți și doar pe tipurile voastre de date.

**Membrii claselor trebuie să fie privați.**

**Ordinea declarării membrilor este public > protected > private.**

**Function/Funcții:**

**Funcțiile trebuie să fie scurte, să aibe pe cât posibil un singur scop (as in, single puropose).** În general o funcție ar trebui să aibe maximum 40 de linii.

**Naming:**

**Numele fișierelor vor fi scrise cu literă mică cu cuvinte despărțite de "\_"** my\_file.h

**Numele variabilelor vor fi scrise cu literă mică cu cuvinte despărțite de "\_" – snake case** int my\_int;

**Numele membrilor dintr-o clasă vor fi snake case dar precedați de "m\_"** m\_my\_member;

**Numele funcțiilor vor fi scrise cu literă mare cu cuvintele alturate, fiecare cu literă mare** MyFunction();

**Metodele private încep cu "\_"** \_PrivateMethod();

**Numele namespace-urilor vor fi scrise cu literă mare pentru fiecare cuvânt și cuvintele despărțite de "\_".** This\_Is\_A\_Namespace{}

**Comentați-vă codul** - Folosiți "/*" & "*/" sau // pentru comentarii. Comentariile trebuie să însoțească majoritatea codul funcțiilor, variabilelor, claselor și membrilor pe care îi scrieți și trebuie să descrie funcționalitatea/scopul codului de preferat în 1 sau 2 rânduri (se pot face excepții). Trebuie comentate și bucăți din interiorul unei funcții (de ex), dacă aceasta conține un algoritm sau o rezolvare mai puțin evidentă.