Laborator 2

1. **`class`** *vs* **`struct;`**
2. *Object x-ray;*
3. *Accessing members;*
4. *Simple composition.*

În C++, decidem foarte clar când să folosim **`struct`** *și când să folosim* **class***:*

- Folosim **`struct`** când avem nevoie de date compuse, în general **simple (date calendaristice, perechi de valori precum puncte 2D/3D)**

- Folosim **class** când datele sunt mai complexe și codul mai profesional. Pe de o parte pentru că vom fi mai stricți si nu oricine va putea, e.g., să citească un CNP cu număr greșit de caractere. Pe de altă parte, este mai standardizat ca în clase să implementăm și funcții membre, constructori.. Care se vor dovedi indispensabile.

Suficientă gargară!..

0. Descărcați editorul CLion. Va fi nevoie să vă înregistrați pe www.jetbrains.com , folosind conturile @s.unibuc.ro de la universitate ca să puteți accesa https://education.github.com/pack .

   Veți avea trial de 30 zile cu pentru noul cont,

   iar mai târziu puteți urma pașii de pe GitHub, unde veți avea nevoie inclusiv de poză la legitimație, care să demonstreze că sunteți studenți etc etc :)

Teorie + exemple

Crearea clasei:

```
class Data {
private:
    // date membre (denumite si campuri)
    int an, luna, zi;
public:
    // datele publice, dar aici vom pune doar funcții
    void afiseaza() {
        cout << an << '.' << luna << '.' << zi << ' ';
    }
    // in partea public:  punem si metode speciale precum
constuctori
    Data() {
        cout << "Constructor gol!" << endl;
        an = luna = zi = 0; // ca sa stim ca am folosit
constructorul gol, punem valori goale aici
    }
    Data(int _an, int _luna, int _zi) {
        an = _an;
        luna = _luna;
        zi = _zi;
    }
    // și getteri/setteri
    int getLuna() {
        return luna;
    }

    void setLuna(int _luna) {
        luna = _luna;
    }
};
```

Utilizarea în cod (funcția main in acest caz) a clasei:

```
// cum folosim clasa Data:
int main() {
   Data d1; // ruleaza codul din constructorul fără parametrii
   Data d2(2022,2,22); // !! ruleaza codul din constructorul cu parametrii, deci 2022 devine anul, 2 devine luna si 22 devina ziua
   Data d3(2022,2,23); // !! ruleaza codul din constructorul cu parametrii, deci 2022 devine anul, 2 devine luna si 23 devina ziua
   // ! putem scrie si    Data d1, d2(2022,2,22);  etc  dar era mai clar pentru exemplu nostru
   //   sa scriem pe cate un rand

   d1.afiseaza();
   d2.afiseaza();
   d3.afiseaza();
   cout << '\n' << '\n';

   // Putem folosi constructorul și în acest mod:
   d3 = Data(30, 4, 2022);
   cout << "Lucrăm în fiecare săptămână, ca nici măcar să nu știm cum ajungem în ";
   d3.afiseaza();
   cout << "..și codăm programe POO la un nivel avansat!" << '\n' << '\n';

   // Dacă în schimb vrem să modificăm un singur câmp, folosim un setter:
   d3.setLuna(5); // până atunci deja devenim maeștrii
   cout << "După apelul d3.setLuna(5), avem d3 = ";
   d3.afiseaza();
   cout << '\n';

   // Și similar, dacă ne dorim valoarea unui singur câmp, folosim un getter:
   int valoareLuna = d2.getLuna();
   cout << "Data d2 conține luna " << valoareLuna << '\n';
   return 0;
}
```

1. **Bilete de avion**

Se cere să citim de la tastatură n bilete de avion.
Despre fiecare bilet stim ca este rezervat pe numele unui calator, și daca biletul are sau nu loc la clasa 1.

Să se implementeze clasa Bilet, constructor, getter + setter (pentru nume calator), dupa care vom rezolva următoarele cerințe:

a) Vom citi cele n bilete de avion. Pentru simplitate vom crea o metodă de citire a unui bilet, cu antetul

```
void citeste()
```

b) Creați o funcție globală cu antetul:

```
void statisticaZboruri(int n, Bilet bilete[])
```

Funcția va primi numărul de bilete, plus un array cu biletele și va afișa câte bilete sunt la clasa I și câte nu sunt la clasa I.

Exemplu:

```
Bilet bilete[] = {Bilet("Popescu Leuraș", true),
                  Bilet("Manolescu Alexandra", true),
                  Bilet("Popescu Mănăila", false)};
statisticaZboruri(3, bilete);
```

⇒

```
Avem 2 bilete la clasa I, dar și 1 bilete la alte clase.
```

2. Filme

Se citesc de la tastatura datele despre n filme. Flecare film are un titlu și un număr de like-uri.

Se cere:

a) Vom citi cele n filme. Pentru simplitate vom crea o metodă de citire a unui film, cu antetul

```
void citeste()
```

b) Creați o funcție globală cu antetul:

```
void afisarePopulare(int n, Film filme[], int k)
```

In care vei afișa cele mai populare k Filme (conform numărului de like-uri)
Funcția va primi numărul de filme, un array cu filme și numărul maxim de filme pe care vrem să le afișăm.

3. Tablouri decorative

După ce v-ați mutat în chirie ați observat nevoie de a adăuga câteva tablouri decorative în locație, care să vă motiveze în același timp. De aceea ați decis să citiți de la tastatură datele celor n tablouri. Fiecare tablou are o înălțime, o lățime, un preț și un mesaj motivațional.

Se cere:

a) Citiți cele n tablouri de la tastatură. De data aceasta avem mai multe date, deci citirea va fi mai clară dacă scriem:

```
class Tablou {
private:
   string mesaj;
   // ...
public:
   void citeste() {
        cout << "Mesaj motivational: ";
        getline(cin, mesaj); // echivalentul a cin.getline() pt tipul string
        // ...
   }
};
```

Și completăm pentru fiecare câmp în parte (înălțime, lățime, preț).

b) Afișați tabloul a cărui mesaj motivațional are număr maxim de cuvinte, dar și câte cuvinte are

4. Cărți

Creați clasa **`Carte`**, împreună cu metodele necesare pentru a rula următorul cod:

```
int main() {
   Carte c;
   c.citeste();
   c.afiseaza();
   // afiseaza titlul, numărul de pagini și pagina curentă, sub forma
   //   Cartea cu titlul "Pamflet" stă deschisă la pagina 29 din 103
   c.setPaginaCurenta(30);
   c.afiseaza();

   c.setPaginaCurenta(c.getPaginaMaxima() + 1);
   // cand rulam metoda de mai sus, ne va afisa mesajul
   // "Nu poți sări la pagina 104 din 103, deoarece nu există!"
}
```

Considerăm că implementați o carte care va reține date despre o carte electronică pe care o citiți, motiv pentru care conține inclusiv câmpul **paginaCurentă,** pe langă titlu și numărul total de pagini.

5. Banca și datele clienților ei

Haideți să vedem diferențele dintre a folosi o clasă în altă clasă și a folosi o singură clasă cu mai multe câmpuri și metode în ea.

Aveți o cerință simplă:

Implementați o clasă Bancă care va avea o listă de clienți. Fiecare client în parte are un *ID unic* și deține o *sumă de bani*. Banca reține *numărul de clienți* și are implementate *funcționalități* care ne permit să afișăm:

- cantitatea totală de bani care se află în bancă
- numărul de clienți care au cel puțin 1000 RON în cont, dar și;
- câți la sută dintre clienți au peste 100 RON în contul lor.

a) Rezolvați această cerință utilizând doua clase: **`Banca, Client`**

b) Descrieți succint (sau implementați dacă considerați necesar) cum ar trebui implementată clasa Banca daca nu am vrea săfolosi clasa Client

!! Pentru tema bonus veți folosi două clase compuse.

Dar ce înseamnă clase compuse?

Dacă ești programator și deja ai un cod funcțional și vrei să-l refolosești, ~~**trânțești o copie a codului**~~ creezi o funcție în care incluzi tot codul.
Să facem același lucru în cazul claselor:

Dacă ai implementat deja o clasa Data (adică dată calendaristică) și funcționează, o putem folosi inclusiv în cadrul altor clase:

```
// Aici aveți un exemplu de clasă compusă:

class Data {
private:
   int an, luna, zi;
public:
   // Începem să folosim CLion
   //
   // Dacă dați Alt+Insert (sau click-dreapta→Generate)
   // când aveți cursorul în interiorul clasei
   //  descoperiți lista cu metode care pot fi generate, inclusiv
Constructor
   Data(int an, int luna, int zi) : an(an), luna(luna), zi(zi) {}
};

class Event {
private:
   Data start, end;
   string name;
public:
   // TODO
};
```

6. Trecutul se repetă

Când trecutul se repetă, să-l facem mai bun!

Rezolvați exercițiul 5 din laboratorul anterior, dar de data aceasta utilizați clase compuse. Dacă ați folosit data trecută, cu atât mai bine :).

7. Căminele și cântatul

Se citesc de la tastatură date despre n cămine. Fiecare cămin are o denumire, un număr de camera, dar și informații despre orele de liniste.

E.g. Căminul D poate avea o lista cu următoarele ore de liniște: 22:00-8:00, 16:00-18:30, dar numărul de intervale poate să varieze, fiind in total k intervale dar știm că în total avem k intervale.

O modalitate destul de simplă de a gestiona aceste date este prin compunere = clasa **`Cămin`** va conține un array cu elemente de tip **`IntervalOrar`**. A doua clasă va conține 4 întregi, utili în a reține ora și minutul de început, respectiv ora și minutul de final.

Se cere:

a) Creați clasele necesare pentru reținerea datelor (date + constructori + getter + setteri pe care ii considerati necesari)
b) Se cere să afișati o listă de 3 cămine astfel încât să maximizați durata de timp zilnică în care puteți “cânta” cu volumul maxim, în oricare dintre cele 3 cămine. Adică veți considera că intervalul 10-11 este deschis pentru cântat doar dacă niciunul dintre cele 3 cămine alese nu conțin vreun interval de liniște care să se intersecteze cu intervalul 10-11.
   Dacă puteți, să afișați și toate intervalele deschise pentru cântat :).