

**Lista temelor, cu toate cerințele pentru a treia lucrare practica.**

- **Cerințe comune tuturor temelor:**

1. utilizarea sabloanelor
2. utilizarea STL
3. utilizarea variabilelor, funcțiilor statice, constantelor și a unei funcții const
4. utilizarea conceptelor de RTTI raportat la template-uri
5. tratarea excepțiilor
6. citirea informațiilor complete a n obiecte, memorarea și afișarea acestora

**Obs. Se pot considera și probleme din fișierul Tema 2 adaptate la cerințele actuale.**

- cerințe generale aplicate fiecărei teme din acest fișier:
  - să se identifice și să se implementeze ierarhia de clase;
  - clasele să conțină constructori, destructori, =, funcție prietena de citire a datelor;
  - clasa de baza să conțină funcție virtuală de afișare, rescrisă în clasele derivate, iar operatorul de citire să fie implementat ca funcție prieten (în clasele derivate să se facă referire la implementarea acestuia în clasa de baza).

**Tema 1.** Firma X are un domeniu de business unde este necesar să se urmărească modul în care clienții plătesc (numerar, cec sau card de credit). Indiferent de modul de plată, firma X știe în ce data s-a efectuat plata și ce sumă a fost primită. Dacă se plătește cu cardul, atunci se cunoaște și numărul cardului de credit. Pentru cash, nu e necesară identificarea clientului care a făcut plata.

**Structura de date:** unordered\_map sau unordered\_set <id\_plata, structura care reține datele plății>

**Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **Gestiune** care să conțină numărul total de plăți de un anumit tip (incrementat automat la adăugarea unei noi chitanțe) și structura de obiecte de tipul plății, alocat dinamic. Să se suprîncarce operatorul += pentru inserarea unei plăți în vector.
- Să se construiască o specializare aferent tipul de plată prin card, care să stocheze și numărul de clienți, împreună cu numele acestora. Specializarea va adapta operatorii menționați în cerințe și operatorul la acest tip de plată.

**Tema 2.** La ora de Biologie, copiii din ciclul gimnazial învață că regnul animal se împarte în 2 grupuri: nevertebrate și vertebrate. La rândul lor, vertebratele se împart în pești, păsări, mamifere și reptile.

**Structura de date:** list<Animal\*>

**Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.

- Să se construiască clasa template **AtlasZoologic** care sa conțină un număr de animale (incrementat automat la adaugarea unei noi file) și structura de obiecte de tipul regnurilor implementate, alocat dinamic. Sa se suprincarce operatorul += pentru inserarea unei fișe de observație a unui animal în vector.
- Să se construiască o specializare pentru tipul **Pești** care sa adapteze operatorii menționați și care sa afișeze, în plus, cati pesti rapitori de lungime mai mare de 1m s-au citit.

**Tema 3.** La facultatea Y studenții intra în sesiune. În regulament e prevăzut că ei sa aibă un anumit număr de examene. Fiecare examen are un număr, incrementat automat, denumirea materiei data și nota la scris. Partialul conține și nota la oral, iar examenul final conține extra-puncte pe un proiect. Dacă partialul nu e luat, atunci se reface la examenul final, altfel, se păstrează nota. Cei care vor să-si mareasca nota, mai dau un quiz, conținând un număr de itemi de tip grila.

**Structura de date:** unordered\_set sau unordered\_map <id\_examen, vector<elev>> (se rețin elevii care nu au trecut examenul cu id-ul id\_examen)

#### **Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **CatalogIndividual** care sa conțină informații despre tipurile de examene date de un student. Clasa conține nr matricol al studentului (incrementat automat la adaugarea unei noi file), nr\_examene și structura de obiecte. Sa se suprincarce operatorul += pentru inserarea unei fișe de observație a unui examen în structura.
- Să se construiască o specializare pentru tipul **Quizz** care sa adapteze operatorii menționați și care sa afișeze, în plus, cate persoane au dat cel puțin 2 quizz-uri.

**Tema 4.** Dintr-un parc auto se poate cumpăra o gama variată de automobile din următoarele clase: MINI (mașina de oraș de mic litraj, de obicei sub 4m lungime), MICA (mașini de oraș, cu spațiu interior mai mare decât MINI și lungime între 3.85 și 4.1), COMPACTA (mașini ușor de folosit atât de oraș cât și la drum lung, de dimensiune 4.2 – 4.5m; modelele vin sub forma de hatchback, combi sau sedan) și MONOVOLUME (automobile sub forma de van ce pot transporta 5-7 persoane). Monovolumele pot și achiziționate atât noi cât și second hand. La cele achiziționate sh se percepe un discount proporțional cu numărul de ani vechime și, în lunile de vară, beneficiază de zile promotionale cu reducere fixa de 10% din preț.

**Structura de date:** set<pair<tip\_automobil, bool nou>> (nou = false pentru cele sh)

#### **Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se adauge campurile și metodele necesare pentru implementarea corecta;
- Să se construiască clasa template **Vanzare** care sa conțină nr total de mașini în stoc (decrementat automat la vanzarea unei mașini), nr de mașini vândute (incrementat automat) și două structuri de obiecte, alocate dinamic, prin care să se gestioneze automobilele din stoc și cele vandute. Sa se suprincarce operatorul -= care sa actualizeze datele din clasa la vanzarea unei mașini.
- Să se construiască o specializare pentru tipul **MONOVOLUM** care sa conțină și să afișeze gestioneze doar MONOVOLUMELE.

**Tema 5.** În luna mai se organizează târgul mașinilor sport, astfel ca pasionații se pot delecta cu modele din clasa COUPE, HOT-HATCH (modele hatchback de clasa mică și compactă cu motoare performante ce au la bază modele obișnuite), CABRIO (modele decapotabile cu acoperiș metalic sau din material textil) și SUPERSPORT (mașini de viteză gen Audi R8, Bugatti Veyron, Lexus LF-A, etc.). O parte din mașinile supersport poate să fie vândută chiar în cadrul expoziției, dacă se tranzacțiază se face cu plata pe loc.

**Structura de date:** vector sau list <pair<mașina, preț>> (se rețin mașinile vândute și prețul de vânzare, dacă mașina nu a fost vândută prețul este -1)

**Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **Expoziție** care să conțină nr total de mașini expuse (incrementat automat) și un vector de obiecte de tipul unei mașini, alocat dinamic.
- Să se construiască o specializare pentru tipul **SUPERSPORT** care să conțină nr total de mașini supersport expuse (decrementat automat la vânzarea unei mașini), nr de mașini vândute (incrementat automat) și doi vectori de pointeri la obiecte de tip mașina supersport, două structuri alocate dinamic, prin care să se gestioneze automobilele din stoc și cele vândute. Să se supraincarce operatorul -= care să actualizeze datele din clasa la vânzarea unei mașini.

**Tema 6.** Se dorește implementarea unei aplicații OOP care să permită gestionarea activității unor farmacii aparținând proprietarului X. Pentru fiecare farmacie se cunoaște cel puțin denumirea, numărul de angajați și profiturile pe fiecare lună. Farmaciile pot fi și online. În acest caz se cunoaște doar adresa web, numărul de vizitatori și discountul utilizat.

**Structura de date:** vector sau list <tuple<web, nr\_vizitatori, discount>> se rețin farmaciile online

**Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **GestionareFarmacii** care să conțină informații despre diversele tipuri de farmacii. Clasa conține indexul farmaciei (incrementat automat la adăugarea unei noi file), id-ul lanțului (constant) și o structură de obiecte, alocată dinamic. Să se supraincarce operatorul += pentru inserarea unei noi farmacii online în structură.
- Să se construiască o specializare pentru tipul **Farmacie\_online** care să conțină și să afișeze doar numărul total de vizitatori ai farmaciilor online.

**Tema 7.** La realizarea unui film (nume, tip, durată), participă o mulțime de persoane: un regizor, actori, personal tehnic etc. Fiecare personal (cnp, nume, nume\_film) are prevăzut prin contract un procent din încasări, drept urmare primește, pentru fiecare film la care participă, o anumită sumă. Actorii, dacă sunt personaje principale, primesc un bonus de 10% din încasări, iar regizorul are prevăzut și o sumă fixă, indiferent de calitatea filmului produs.

**Structura de date:** set<tuple<persoana, film, bonus>>

**Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.

- Sa se construiască clasa template **FirmaDistributie** care sa conțină informații despre filmele realizate într-un an. Clasa conțină numărul total de persoane implicat incrementat automat), numărul de actori (incrementat automat) și o structura de obiecte, alocată dinamic.
- Să se realizeze o specializare pentru tipul **Actor**, care sa conțină numărul actorilor care au fost distribuiți în roluri principale.

**Tema 8** – Se dorește implementarea unei aplicații care sa permita gestionarea conturilor deschise la banca X. Fiecare cont bancar are obligatoriu un detinator, o data a deschiderii lui și un sold. În cazul conturilor de economii, trebuie retinuta si rata dobânzii (care poate fi pe 3 luni, pe 6 luni sau la un an), precum și un istoric al reactualizarii soldurilor de la deschidere și până în prezent. În cazul în care deținătorul optează pentru un cont curent, el beneficiază de un număr de tranzacții gratuite și altele contra cost (de exemplu orice depunere este gratuită, dar retragerea poate sa coste dacă s-a depășit numărul de tranzacții gratuite, sau e făcută de la bancomatele altor bănci; sau orice cumparatura online are un cost, etc.). Simulati cat mai corect activitatea băncii X.

**Structura de date:** unordered\_map sau unordered\_set <id\_cont, list sau vector <operatiuni pe contul id\_cont>>

**Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **GestionareConturi** care sa conțină informații despre banca X. Clasa conține indexul unui cont (incrementat automat la adaugarea unui nou prin supraincercarea operatorului +=) și o structură de date.
- Să se construiască o specializare pentru tipul **Cont\_Economii** care sa afișeze toate conturile de economii care au rata dobânzii la 1 an.

**Tema 9** – Pizzeria X testează un nou soft, dezvoltat în maniera OOP, pentru gestionarea activității sale. Codul propriu-zis contine o clasa abstracta care contine doar metoda virtuala pura de calcul al prețului unui produs, iar din aceasta clasa deriva clasa Pizza. În realizarea oricărui produs intra un anumit număr de ingrediente pentru care se cunosc denumirile, cantitatile și prețul unitar. Prețul unui produs finit este data de prețul ingredientelor plus manopera (o suma constanta fixată de producător). Daca pizza este comandata OnLine, la preț se mai adaugă și 5% din pret la fiecare 10 km parcursi de angajatul care livrează la domiciliu.

**Structura de date:** unordered\_map sau unordered\_set <id\_pizza, list sau vector <ingredient>>

**Cerința suplimentară:**

- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **Meniu** care sa gestioneze tipurie de pizza comercializate. Clasa trebuie sa contina indexul unui produs (incrementat automat la vanzarea unui produs prin supraincercarea operatorului +=) și o structură de date, alocata dinamic.
- Să se construiască o specializare pentru tipul **Comanda\_Online** care sa trateze tipurile de pizza vegetariana într-un document contabil separat din care sa rezulte valoarea totala incasata din vanzarea acestora.

**Tema 10** – Se dorește implementarea unei aplicații care să permită gestionarea clienților și a proprietăților imobiliare în cadrul unei agenții imobiliare nou înființate. Pentru fiecare locuință se cunoaște numele clientului care o închiriază, suprafața utilă și discount-ul (0-10%). La apartamente se cunoaște etajul, iar la case se știe câți metri pătrați are curtea, câte etaje are casa și care este suprafața utilă pe fiecare etaj în parte.

Evident, calculul chiriei se face diferit. Dacă la apartamente se consideră doar formula dată de prețul de închiriere pe metru pătrat \* suprafața utilă, având grijă să se aplice discount unde este cazul, la casa, se adaugă, indiferent de discount, prețul pe metru pătrat de curte \* suprafața acesteia.

**Structura de date:** set<pair<locuința, tip>> unde tip = apartament sau casa

**Cerința suplimentară:**

- Să se adauge toate câmpurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **Gestiune**, conținând structura de obiecte de tipul locuințelor implementate, alocat dinamic, unde indexul fiecărei locuințe este incrementat automat la adăugarea unei noi, prin supraincercarea operatorului +=.
- Să se construiască o specializare pentru tipul **Casa** care să stocheze numărul de case, fiecare cu chiria aferentă și afișează totalul obținut de agenția imobiliară de pe urma acestora.