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

- Cerințe comune tuturor temelor:
  - utilizarea sabloanelor
  - utilizarea STL
  - utilizarea variabilelor, funcțiilor statice, constantelor și a unei functii const
  - utilizarea conceptelor de RTTI (ex: upcast & dynamic_cast)
  - tratarea excepțiilor
  - 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, =, supraincarcare pe >> si << pentru citire si afisare;
  - clasa de baza sa conțină funcție virtuala de afisare si citire, rescrisa în clasele derivate, iar operatorul de citire si afisare să fie implementat ca funcție prieten (în clasele derivate să se facă referire la implementarea acestuia în clasa de baza).

Programul sa ruleze in CODEBLOCKS !!!

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 plata, 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 necesara 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 sa conțină numărul total de plăți de un anumit tip (incrementat automat la adaugarea unei noi chitanțe) și structura de obiecte de tipul plății, alocat dinamic. Sa se supraincarce operatorul += pentru inserarea unei plăti în vector.
- Să se construiască o specializare aferent tipul de plata prin card, care sa stocheze si numărul de clienți, împreuna cu numele acestora. Specializarea va adapta operatorii menționați în cerințe și operatorul la acest tip de plata.

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 supraincarce 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 afiseze, 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 reține 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 supraincarce 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 afiseze, 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 achizitionate 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 supraincarce 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 pasionatii se pot delecta cu modele din clasa COUPE, HOT-HATCH (modele hatckback de clasa mica și compacta cu motoare performante ce au la bază modele obișnuite), CABRIO(modele decapotabile cu acoperiș metalic sau din material textil) și SUPERSPORT(masini de viteza gen Audi R8, Bugatti Veyron, Lexus LF-A, etc.). O parte din masinile supersport poate să fie vânduta chiar în cadrul expozitiei, dacă se tranzacția se face cu plata pe loc.
Structura de date: vector sau list <pair<masina, preț>> (se reține mașinile vândute și prețul de vanzare, daca masina nu a fost vanduta prețul este -1)

Cerința suplimentară:
- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **Expoziție** care sa 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 sa conțină nr total de masini supersport expuse (decrementat automat la vanzarea 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 vandute. Sa se supraincarce operatorul -= care sa actualizeze datele din clasa la vanzarea unei mașini.

Tema 6. Se dorește implementarea unei aplicații OOP care sa permita gestionarea activității unor farmacii apartinand proprietarului X. Pentru fiecare farmacie se cunoaște cel puțin denumirea, numărul de angajați și profiturile pe fiecare luna. 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 reține farmaciile online

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

Tema 7 – 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 unuia nou prin supraincarcarea 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 8 – 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 supraincarcarea 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 9 – Se dorește implementarea unei aplicații care sa permita gestionarea clienților și a proprietăților imobiliare în cadrul unei agenții imobiliare nou infiintate. Pentru fiecare locuință se cunoaște numele clientului care o închiriază, suprafata utila și discount-ul (0-10%). La apartamente se cunoaste etajul, iar la case se stie cati metri patrati are curtea, cate etaje are casa și care este suprafata utila pe fiecare etaj în parte.
Evident, calculul chiriei se face diferit. Dacă la apartamente se considera doar formula data de prețul de închiriere pe metru pătrat * suprafata utila, avand grija sa 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<locuinta, tip>> unde tip = apartament sau casa

Cerința suplimentară:
- Să se adauge toate campurile relevante pentru modelarea acestei probleme.
- Să se construiască clasa template **Gestiune,** continand structura de obiecte de tipul locuintelor implementate, alocat dinamic, unde indexul fiecărei locuințe este incrementat automat la adaugarea uneia noi, prin supraincarcarea operatorului +=.
- Să se construiască o specializare pentru tipul **Casa** care sa stocheze numărul de case, fiecare cu chiria aferentă și afișează totalul obținut de agentia imobiliara de pe urma acestora.

Tema 10 Sa se realizeze o aplicatie C++ de evidenta a articolelor (carti si CD –uri) dintr-o biblioteca studenteasca. Orice articol existent in biblioteca contine o cota si numarul de exemplare existent in biblioteca. Cota articolelor din biblioteca foloseste la cautarea si imprumutarea acestora catre studenti. Fiecare carte are un titlu, iar fiecare revista este emisa intr-o anumita luna.

Sa se implementeze clasele necesare pentru gestionarea imprumuturilor din biblioteca si afisarea situatiei exitente la un moment dat. Un exemplu de test pentru functia main este :

```
main () {
   try {
        CBiblioteca B;
        CArticol *C1 = new CCarte (1000, 10, “Programare in C++”);
        CArticol *C2 = new CCarte (1001, 15, “Programare in Java”);
        CArticol *C3 = new CCD (1002, 8, “Documentatie electronica C++”);
        CArticol *C4 = new CCD (1003, 8, “Documentatie electronica Java”);
        CArticol *C5 = new CCD (1004, 8, “Documentatie electronica Retele”);
        B.addArticol (C1); B.addArticol (C2); B.addArticol (C3);
        B.printArticole ( ); // afiseaza articolele existente, tipul acestora (carte sau cd) si numarul de bucati existente in acest moment
        B.addArticol (C4); B.addArticol (C5);
        B.imprumutaArticol (1001); B.printArticole ( ); B.imprumutaArticol (1002); B.printArticole ( );
        B.restituieArticol(1001); B.printArticole ( );
    } catch (CException *e)
    {
        e->printErrMessage();
        delete e;
    }
}
```

Trebuie sa aveti si un meniu interactiv, testul de mai sus este doar ilustrativ pentru a va face o idee a functionalitatilor.

Tema 11 A) Sa se proiecteze si implementeze obiectual o aplicatie de gestiune a resurselor umane pentru un grup de ***Companii***. Gestionarea se va realiza prin intermediul unui gestionar numit ***HRManager***. Se vor avea in vedere urmatoarele caracteristici :

- Companiile detin o serie de departamente de lucru (ex: "*derulare proiecte"*, "*financiar"*, "*IT"*, ...)
- Companiile au o serie de angajati, distribuiti pe departamente, unii dintre acestia avand functia de manageri pentru alti angajati. Fiecare angajat poate avea cel mult un manager.
- Aplicatia trebuie sa poata gestiona pe langa numele angajatilor si informatii cum ar fi data de angajare in companie si salariul lunar brut al fiecarui angajat.
- Se vor realiza interfete prin care sa se poata adauga/sterge departamente sau angajati din companie precum si companii din gestionarul de companii.
- Aplicatia trebuie sa poata furniza urmatoarele rapoarte:
    - Numarul de angajati ai unui companie/departament dat
    - Cheltuielile salariale lunare totale existente la nivelul unei companii
    - Numele, data de angajare si salariul angajatilor care au functia de manageri pecum si lista de angajati care le sunt subordonati, la nivelul unei companii
    - Lista departamentelor care au minim **n** angajati (n transmis ca parametru).
- Trebuie sa existe posibilitatea de fuzionare a doua companii (se va defini un operator + care realizeaza acest lucru, precum si o metoda la nivelul gestionarului). Departamentele comune vor fuziona intr-unul singur. La fuzionare toti managerii isi pastreaza functia.
- La adaugarea unei companii in gestionar trebuie sa se verifice daca acea companie nu exista deja acolo. O companie este identica cu alta daca are acelasi nume si aceeasi schema de deparatamente si angajati.

B) Pornind de la modelul proiectat mai sus, sa se adauge posibilitatea de lucru cu companii care beneficiaza de scutire de impozit (detin departamente cu scutire de impozit (16%)). Acolo cheltuielile salariale sunt diminuate.

C) Toate problemele si situatiile deosebite se vor trata prin intermediul mecanismului de exceptii. Exemple de probleme: incercarea de adaugare a unei companii in gestionar, in conditiile in care ea deja exista acolo, incercarea de stergere a unei entitati (companie, departament, angajat) care nu exista, alocari nereusite de memorie, etc.

Obs: se vor folosi cat se poate de mult mecanismele ce tin de tehnicile de programare obiectuala : incapsularea, derivarea si polimorfismul bazat pe supraincarare de operatori si functii virtuale.