**Lista temelor**, cu toate cerintele pentru **prima lucrare practica**.

* Cerinte comune tuturor temelor:
* implementare in C++ folosind clase
* **ATENTIE:** functiile pe care le-am numit mai jos *metode* (fie ca sunt supraincarcari de operatori, fie altfel de functii), **pot fi implementate ca functii prieten** in loc de metode ale claselor respective, daca se considera ca aceasta alegere este mai naturala;
* supraincarcarea operatorilor << si >> pentru iesiri si intrari de obiecte, dupa indicatiile de mai jos, astfel incat sa fie permise (si ilustrate in program):
* citirea de la tastatura si dintr-un fisier;
* scrierea pe ecran sau intr-un fisier;
* citirea informațiilor complete a n obiecte, memorarea și afisarea acestora
* Cerinte specifice fiecarei teme:

**Tema 1.** liste de numere intregi reprezentate ca tablouri unidimensionale:

* + metode ale clasei vector de intregi:

- suma tuturor elementelor vectorului;

- gasirea maximului si a pozitiei lui;

- sortarea crescatoare;

- produsul scalar a doi vectori de aceeasi lungime, implementat prin supraincarcarea operatorului \*;

* + sa se creeze o clasa matrice de numere intregi, prietena a clasei vector de intregi, care sa reprezinte matricile de numere intregi, nu neaparat patratice, sub forma unui vector de elemente de tip clasa vector de numere intregi, privite ca vectori linie; mai precis, o matrice cu m linii si n coloane va fi reprezentata ca un vector de m elemente de tip clasa vector, fiecare avand dimensiunea n, elemente care vor reprezenta liniile matricii; sa se supraincarce operatorii << si >> si pentru clasa matrice de intregi, folosind supraincarcarea operatorilor << si >> pentru clasa vector de intregi; sa se creeze o metoda pentru adunarea a doua matrici, care sa supraincarce operatorul +;
  + sa se scrie o functie care sa aiba ca parametri o matrice de intregi A cu m linii si n coloane, un vector de intregi x de dimensiune n privit ca vector coloana, si un vector de intregi y de dimensiune m privit tot ca vector coloana, si care sa verifice daca Ax=y, folosind produsul scalar din clasa vector de intregi;

**Tema 2**. vectori de numere complexe:

* + clasa vector de complecsi sa fie prietena a clasei numar complex;
  + clasa numar complex sa contina functii pentru: supraincarcarea operatorilor << si >>; determinarea modulului unui numar complex; suma a doua numere complexe, implementata prin supraincarcarea operatorului +; produsul a doua numere complexe, implementat prin supraincarcarea operatorului \*;
  + metode ale clasei vector de complecsi: supraincarcarea operatorilor << si >>, care sa foloseasca supraincarcarile acestor operatori din clasa numar complex; determinarea vectorului modulelor, folosind metoda de determinare a modulului din clasa numar complex; sortarea crescatoare dupa module a vectorului de complecsi; suma tuturor elementelor vectorului de complecsi, care sa foloseasca suma a doi complecsi din clasa numar complex; produsul scalar a doi vectori de aceeasi lungime, care sa foloseasca suma si produsul a doi complecsi din clasa numar complex, si sa supraincarce operatorul \*;

**Tema 3.** stive de caractere (implementate dinamic):

* + clasa stiva sa fie prietena a clasei nod si sa contina metode de: push; pop; top (afisarea varfului stivei); afisarea stivei, concomitent cu golirea ei, realizata prin supraincarcarea operatorului <<; supraincarcarea operatorului >>, realizata prin push-uri succesive; inversarea unui sir de caractere folosind o stiva; diferenta a doua stive de caractere, care sa supraincarce operatorul - si sa fie constea in eliminarea din prima stiva a tuturor elementelor de la inceputul acelei prime stive care coincid cu elementele de la inceputul celei de-a doua stive, atat ca informatie (adica valoare, caracterul pe care il contin), cat si ca pozitie in stiva; aceasta ultima metoda va efectua pop-uri succesive pe ambele stive;

**Tema 4.** cozi de caractere (implementate dinamic):

* + clasa coada sa fie prietena a clasei nod si sa contina metode de: push; pop; top (afisarea varfului cozii); afisarea cozii, concomitent cu golirea ei, realizata prin supraincarcarea operatorului <<; supraincarcarea operatorului >>, realizata prin push-uri succesive; concatenarea a doua cozi de caractere, obtinand o alta coada de caractere, implementata prin supraincarcarea operatorului +; diferenta a doua cozi de caractere, care sa supraincarce operatorul - si sa fie constea in eliminarea din prima coada a tuturor elementelor de la inceputul acelei prime cozi care coincid cu elementele de la inceputul celei de-a doua cozi, atat ca informatie (adica valoare, caracterul pe care il contin), cat si ca pozitie in coada; aceasta ultima metoda va efectua pop-uri succesive pe ambele cozi;

**Tema 5.** liste circulare de numere intregi (implementate dinamic):

* + clasa lista sa fie prietena a clasei nod si sa contina metode de: introducere a unui element (unde alege programatorul); supraincarcare a operatorului >>, realizata prin introduceri succesive; supraincarcare a operatorului << pentru afisarea listei; stergere a unui element (de unde alege programatorul); inversare a legaturilor listei; eliminare a elementelor din k in k, unde k este un numar natural nenul citit de la tastatura sau dintr-un fisier, pana ce in lista ramane un singur element, cu afisarea elementelor care se elimina, in ordinea in care se elimina; supraincarcare a operatorului +, care sa efectueze concatenarea a doua liste circulare de intregi, prin ruperea cate unei legaturi a fiecareia si legarea celor doua liste in locurile legaturilor rupte;

**Tema 6.** liste dublu inlantuite de numere reale (implementate dinamic):

* + clasa lista sa fie prietena a clasei nod si sa contina metode de: introducere a unui element (unde alege programatorul); supraincarcare a operatorului >>, realizata prin introduceri succesive; supraincarcare a operatorului << pentru afisarea listei in ambele sensuri, in aceeasi functie; stergere a unui element (de unde alege programatorul); concatenare a doua liste dublu inlantuite, obtinand o alta lista dublu inlantuita, implementata prin supraincarcarea operatorului +;

**Tema 7.** multimi finite de numere intregi reprezentate ca tablouri unidimensionale:

* + clasa multime sa contina metode de: transformare a unui vector in multime, prin eliminarea duplicatelor din respectivul vector; supraincarcare a operatorilor << si >>; reuniune a doua multimi, implementata prin supraincarcarea operatorului +; intersectie a doua multimi, implementata prin supraincarcarea operatorului \*; diferenta dintre doua multimi, implementata prin supraincarcarea operatorului -;
  + sa se creeze o clasa pereche de numere intregi si o clasa multime de perechi de numere intregi reprezentata ca vector de obiecte de tip clasa pereche; clasa multime de perechi sa fie prietena a clasei pereche de intregi; fiecare dintre clasele pereche si multime de perechi sa contina supraincarcari pentru << si >>;
  + sa se scrie o functie care efectueaza produsul cartezian a doua multimi de intregi, date ca instante ale clasei multime de intregi, si il memoreaza intr-o instanta a clasei multime de perechi de intregi;

**Tema 8.** multimi finite de numere intregi reprezentate ca liste inlantuite:

* + clasa multime sa contina metode de: transformare a unei liste inlantuite in multime, prin eliminarea duplicatelor din respectiva lista; supraincarcare a operatorilor << si >>; reuniune a doua multimi, implementata prin supraincarcarea operatorului +; intersectie a doua multimi, implementata prin supraincarcarea operatorului \*; diferenta dintre doua multimi, implementata prin supraincarcarea operatorului -;
  + sa se creeze o clasa pereche de numere intregi si o clasa multime de perechi de numere intregi reprezentata ca lista inlantuita de obiecte de tip clasa pereche; clasa multime de perechi sa fie prietena a clasei pereche de intregi; fiecare dintre clasele pereche si multime de perechi sa contina supraincarcari pentru << si >>;
  + sa se scrie o functie care efectueaza produsul cartezian a doua multimi de intregi, date ca instante ale clasei multime de intregi, si il memoreaza intr-o instanta a clasei multime de perechi de intregi;

**Tema 9.** polinoame reprezentate ca tablouri unidimensionale (prin gradul polinomului si vectorul coeficientilor (coeficientii vor apartine la monoame de grade consecutive), de dimensiune egala cu gradul plus 1, de la cel al termenului liber la cel de grad maxim):

* + fiecare coeficient va fi de tip double;
  + clasa polinom sa contina metode pentru: supraincarcarea lui << si >>; calculul valorii unui polinom intr-un punct (de tip double); suma a doua polinoame, implementata prin supraincarcarea operatorului +; diferenta dintre doua polinoame, implementata prin supraincarcarea operatorului -; produsul a doua polinoame, implementat prin supraincarcarea operatorului \*;
  + sa se creeze o clasa de perechi (numar real,polinom), cu numarul real dat ca fiind de tip double si polinomul de tip clasa polinom, care sa fie clasa prietena a clasei polinom, si sa contina: supraincarcari pentru << si >>, care sa foloseasca supraincarcarile pentru << si >> din clasa polinom; o metoda care sa verifice daca numarul real dat de respectiva data membru este radacina a polinomului dat de cealalta data membru, folosind metoda de calcul a valorii unui polinom intr-un punct din clasa polinom;

**Tema 10.** polinoame reprezentate ca liste inlantuite (ca polinoame rare, prin lista perechilor (coeficient,exponent), ordonata crescator dupa exponent, si nu neaparat cu exponentii consecutivi):

* + toti coeficientii vor fi nenuli si de tip double (ceea ce inseamna ca, in metodele de mai jos, din polinoamele obtinute trebuiesc eliminati coeficientii egali cu 0), iar exponentii corespunzatori vor reprezenta gradele monoamelor date de respectivele perechi;
  + clasa pereche (coeficient,exponent) sa contina metode de supraincarcare pentru << si >>; eventual, daca programatorul considera acest lucru util, poate implementa in clasa pereche metodele de mai jos aplicate monomului dat de respectiva pereche in locul unui polinom, metode pe care sa le apeleze in clasa polinom, acolo unde sunt necesare;
  + clasa polinom sa fie prietena a clasei pereche (coeficient,exponent) si sa contina metode pentru: supraincarcarea lui << si >>, pe baza celor din clasa pereche; calculul valorii unui polinom intr-un punct; suma a doua polinoame, implementata prin supraincarcarea operatorului +; diferenta dintre doua polinoame, implementata prin supraincarcarea operatorului -; produsul a doua polinoame, implementat prin supraincarcarea operatorului \*;

**Tema 11.** matrice de numere complexe reprezentate ca tablouri bidimensionale (nu neaparat ca matrici patratice):

* + clasa numar complex sa contina metode pentru: supraincarcarea operatorilor << si >>; suma a doua numere complexe, implementata prin supraincarcarea operatorului +; produsul a doua numere complexe, implementat prin supraincarcarea operatorului \*;
  + clasa matrice sa fie prietena a clasei numar complex si sa contina metode pentru: supraincarcarea operatorilor << si >>, apelandu-le pe cele din clasa numar complex; suma a doua matrici, implementata prin supraincarcarea operatorului + si pe baza functiei de supraincarcare a lui + din clasa numar complex; produsul a doua matrici, implementat prin supraincarcarea operatorului \* si pe baza functiilor de supraincarcare ale lui + si \* din clasa numar complex; determinantul unei matrici patratice, implementat tot cu ajutorul sumei si produsului de numere complexe (cele care supraincarca + si \* in clasa numar complex); inversa unei matrici patratice cu determinantul nenul, care sa foloseasca functiile descrise anterior;

**Tema 12.** matrice de numere complexe reprezentate ca structuri inlantuite (ca matrici rare si nu neaparat patratice):

* clasa numar complex sa contina metode pentru: supraincarcarea operatorilor << si >>; suma a doua numere complexe, implementata prin supraincarcarea operatorului +; produsul a doua numere complexe, implementat prin supraincarcarea operatorului \*;
* clasa matrice sa fie prietena a clasei numar complex si sa contina metode pentru: supraincarcarea operatorilor << si >>, apelandu-le pe cele din clasa numar complex; suma a doua matrici, implementata prin supraincarcarea operatorului + si pe baza functiei de supraincarcare a lui + din clasa numar complex; produsul a doua matrici, implementat prin supraincarcarea operatorului \* si pe baza functiilor de supraincarcare ale lui + si \* din clasa numar complex; determinantul unei matrici patratice, implementat tot cu ajutorul sumei si produsului de numere complexe (cele care supraincarca + si \* in clasa numar complex); inversa unei matrici patratice cu determinantul nenul, care sa foloseasca functiile descrise anterior;

**Tema 13.** arbori binari de cautare, in reprezentare inlantuita:

* + clasa arbore binar de cautare sa fie prietena a clasei nod, si sa contina metode pentru: inserarea unui element, care sa supraincarce operatorul +, care va fi aplicat aici intre un numar reprezentand elementul nou introdus si un arbore; supraincarcarea operatorului >>, prin inserari succesive; supraincarcarea operatorului <<, care sa efectueze afisarea arborelui in inordine; stergerea unui element; determinarea inaltimii arborelui; afisarea listei frunzelor;

**Tema 14.** arbori oarecare, in reprezentare inlantuita, prin legaturile fiu (catre fiul cel mai din stanga) si frate (catre urmatorul fiu al tatalui, in ordinea fiilor tatalui de la stanga la dreapta):

* + clasa arbore sa fie prietena a clasei nod, si sa contina metode pentru: citirea unui arbore, care sa supraincarce operatorul >>; afisarea unui arbore, care sa supraincarce operatorul <<; citirea si afisarea sa foloseasca acelasi mod de reprezentare a arborelui, care sa arate intreaga structura a arborelui, ceea ce parcurgerile urmatoare nu pot face; parcurgerea pe niveluri, adica in latime; parcurgerea in adancime; determinarea inaltimii arborelui; afisarea listei frunzelor; supraincarcarea operatorului +, care sa efectueze aici legarea de radacina unui arbore a unui alt arbore, astfel incat al doilea arbore sa devina noul subarbore cel mai din dreapta a radacinii primului arbore, obtinand astfel un nou arbore;

**Tema 15.** grafuri neorientate, reprezentate cum doreste programatorul:

* + clasa graf neorientat sa contina metode pentru: citirea grafului, care sa supraincarce operatorul >>; afisarea grafului, care sa supraincarce operatorul <<; citirea si afisarea sa foloseasca acelasi mod de reprezentare a grafului, care sa arate intreaga structura a grafului, ceea ce parcurgerile urmatoare nu pot face; parcurgerea in latime; parcurgerea in adancime; determinarea matricii (existentei) drumurilor; determinarea componentelor conexe nu ca grafuri, ci ca liste de noduri; o metoda care sa determine daca graful este conex, care poate folosi oricare dintre metodele anterioare; o metoda de supraincarcare a operatorului +, care sa determine, din doua grafuri neorientate avand aceeasi multime de noduri, graful cu aceeasi multime de noduri ca si acele doua grafuri, si cu multimea muchiilor egala cu, reuniunea multimilor muchiilor acelor doua grafuri;

**Tema 16.** grafuri ponderate, reprezentate ca grafuri neorientate cu ponderi atasate muchiilor, in ce mod doreste programatorul:

* + clasa graf ponderat sa contina metode pentru: citirea grafului, care sa supraincarce operatorul >>; afisarea grafului, care sa supraincarce operatorul <<; citirea si afisarea sa foloseasca acelasi mod de reprezentare a grafului, care sa arate intreaga structura a grafului; determinarea matricii ponderilor drumurilor cu ponderi minime; determinarea nodurilor intermediare de pe drumul de pondere minima intre doua noduri; determinarea componentelor conexe, nu ca grafuri, ci ca liste de noduri; determinarea arborelui partial de cost minim al componentei conexe a unui nod dat; o metoda care sa determine daca graful este conex, care poate folosi orice metoda descrisa anterior; o metoda de supraincarcare a operatorului \*, care sa determine, din doua grafuri ponderate avand aceeasi multime de noduri, graful ponderat cu aceeasi multime de noduri ca si acele doua grafuri, si cu multimea muchiilor egala cu intersectia multimilor muchiilor acelor doua grafuri, cu fiecare muchie avand ca pondere minimul dintre ponderile muchiilor corespunzatoare din acele doua grafuri;

**Tema 17.** grafuri orientate, reprezentate cum doreste programatorul:

* + clasa graf orientat sa contina metode pentru: citirea grafului, care sa supraincarce operatorul >>; afisarea grafului, care sa supraincarce operatorul <<; citirea si afisarea sa foloseasca acelasi mod de reprezentare a grafului, care sa arate intreaga structura a grafului, ceea ce parcurgerile urmatoare nu pot face; parcurgerea in latime; parcurgerea in adancime; determinarea matricii (existentei) drumurilor; determinarea componentelor tare conexe, nu ca grafuri, ci ca liste de varfuri; o metoda care sa determine daca graful este tare conex, care poate folosi oricare dintre metodele anterioare; o metoda de supraincarcare a operatorului +, care sa determine, din doua grafuri orientate avand aceeasi multime de varfuri, graful orientat cu aceeasi multime de varfuri ca si acele doua grafuri, si cu multimea arcelor egala cu, reuniunea multimilor arcelor acelor doua grafuri (bineinteles, tinand cont de orientarea arcelor);

**Tema 18.** numere intregi mari (reprezentate ca indicator de semn si liste dinamice de cifre, incepand cu cifra unitatilor - vor fi afisate uzual, incepand cu cifra dominanta si incheind cu cifra unitatilor):

* + clasa numar intreg mare sa contina metode pentru supraincarcarea operatorilor << si >> pentru iesiri si intrari, precum si pentru calculul: sumei a doua numere intregi mari, prin supraincarcarea operatorului +, diferentei dintre doua numere intregi mari, prin supraincarcarea operatorului -, produsului dintre doua numere intregi mari, prin supraincarcarea operatorului \*, maximului dintre valorile absolute a doua numere intregi mari;
  + sa se creeze o clasa vector de numere intregi mari, prietena a clasei numar intreg mare, care sa contina metode pentru: supraincarcarea operatorilor << si >>, folosind metodele de supraincare pentru << si >> din clasa numar intreg mare; produs scalar dintre doi vectori de numere intregi mari cu acelasi numar de elemente, care sa supraincarce operatorul \* si sa foloseasca produsul si suma din clasa numar intreg mare; calculul valorii absolute maxime dintr-un vector de numere intregi mari, folosind maximul dintre valorile absolute a doua numere intregi mari preluat din clasa numar intreg mare;

**Tema 19.** numere rationale mari (reprezentate ca perechi de numere intregi mari, fiecare element al unei astfel de perechi fiind reprezentat ca la proiectul 18 de mai sus, elementele unei astfel de perechi reprezentand respectiv numaratorul si numitorul fractiei care defineste numarul rational mare):

* + sa se creeze o clasa numar intreg mare, cu supraincarcari pentru operatorii: << si >> pentru iesiri si intrari; + pentru suma a doua numere intregi mari; \* pentru produsul a doua numere intregi mari; - pentru diferenta a doua numere intregi mari; / si respectiv % pentru catul si restul impartirii intregi a doua numere intregi mari; sa mai contina si o metoda pentru calculul celui mai mare divizor comun al valorilor absolute a doua numere intregi mari (sugestie: sa se aplice aici algoritmul lui Euclid, utilizand metodele pentru determinarea catului si a restului impartirii intregi);
  + clasa numar rational mare sa fie clasa prietena a clasei numar intreg mare, iar metodele ei sa apeleze metodele necesare din clasa numar intreg mare; clasa numar rational mare sa contina metode de supraincarcare pentru operatorii: << si >> pentru iesiri si intrari de obiecte; + pentru suma a doua numere rationale mari; \* pentru produsul a doua numere rationale mari; sa contina si o metoda pentru scrierea unui numar rational mare ca fractie ireductibila, prin impartirea numaratorului si numitorului la cel mai mare divizor comun al valorilor lor absolute;

**Tema 20.** numere reale mari (reprezentate ca perechi de numere intregi mari, astfel incat valoarea unui numar real mare reprezentat astfel sa fie egala cu primul numar intreg mare din pereche inmultit cu 10 la puterea al doilea numar intreg mare din pereche; fiecare element al unei astfel de perechi sa fie reprezentat ca la proiectul 18 de mai sus):

* + clasa numar intreg mare sa contina: supraincarcari pentru operatorii << si >> pentru iesiri si intrari de obiecte; supraincarcari pentru operatorii + si \* pentru adunarea a doua numere intregi mari si respectiv produsul a doua numere intregi mari; o metoda pentru determinarea maximului dintre valorile absolute a doua numere intregi mari; o metoda care: aplicata lui 0 ca obiect al clasei numar intreg mare, sa intoarca 0 si sa nu modifice obiectul 0, iar, aplicata la un numar intreg mare nenul, sa elimine toate zerourile consecutive de la sfarsitul acelui numar intreg mare nenul, adica dintre cifrele cele mai putin semnificative, si sa intoarca numarul de zerouri pe care le-a eliminat (acel numar de zerouri reprezinta puterea naturala maxima a lui 10 care divide acel numar intreg mare, iar obiectul modificat astfel reprezinta catul impartirii obiectului initial la acea putere a lui 10);
  + clasa numar real mare sa fie clasa prietena a clasei numar intreg mare si sa contina urmatoarele metode, care sa apeleze metodele necesare din clasa numar intreg mare: supraincarcari pentru operatorii << si >> pentru iesiri si intrari de obiecte; scrierea unui numar real mare nenul astfel incat primul numar din perechea care il reprezinta sa nu aiba zerouri consecutive la sfarsit, adica la cifrele cel mai putin semnificative; supraincarcari pentru operatorii + si \* pentru adunarea a doua numere reale mari si respectiv produsul a doua numere reale mari; o metoda pentru determinarea maximului dintre valorile absolute a doua numere reale mari.