

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

Cerinte comune tuturor temelor:

- implementare in C++ folosind clase
- **ATENȚIE:** funcțiile pe care le-am numit mai jos *metode* (fie ca sunt supraincari de operatori, fie altfel de funcții), **pot fi implementate ca funcții prieten** in loc de metode ale claselor respective, daca se considera ca aceasta alegere este mai naturala;
- supraincarea operatorilor << si >> pentru iesiri si intrari de obiecte, dupa indicatiile de mai jos, astfel incat sa fie permise (si ilustrate in program):
- sa existe o metoda publica prin care se realizeaza citirea informațiilor complete a n obiecte, memorarea și afisarea acestora.

Cerinte specifice fiecarei teme:

Tema 1. Clasa "Numar_Complex"

- membrii privati pentru partea reala si partea imaginara (double);
 - constructori pentru initializare si copiere;
 - destructor (în cazul alocarii statice, se seteaza dimensiunea vectorului la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
 - metode publice pentru setat/furnizat partea reala si partea imaginara;
 - metoda publica de afisare (sub forma "a", "i*a", "-i*a", "a+i*b", "a-i*b");
 - metoda publica pentru determinarea modulului unui numar complex;
 - suma a doua numere complexe, implementata prin supraincarea op +;
 - produsul a doua numere complexe, implementat prin supraincarea op *;
 - împărțirea a doua numere complexe, implementata prin supraincarea op /.
 - metoda publica **sqr**t pentru furnizarea radicalului de ordinul 2 al unui complex.
- Să se realizeze un program de rezolvare a ecuatiei de ordinul doi cu coeficienti complecsi.

Tema 2. Clasa "Fractie"

- membrii privati pentru numarator si numitor (int);
 - constructori pentru initializare si copiere;
 - destructor (în cazul alocarii statice, se seteaza dimensiunea vectorului la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
 - metode publice pentru setat/furnizat numaratorul si numitorul;
 - metoda publica pentru simplificare;
 - metoda publica de afisare (sub forma "a", "a/b", "-a/b", dupa caz);
 - metode publice pentru suma, diferența, produsul si împărțirea a doua numere raționale, implementate prin supraincarea operatorilor +, -, *, /
 - metode publice pentru inmultirea unui numar rațional cu un numar întreg, realizata prin supraincarea operatorului *.
- Să se realizeze un program de rezolvare a unui sistem de doua ecuatii liniare cu doua necunoscute si coeficienti rationali.

Tema 3. Clasa "Vector" (vector de numere întregi)

- membri privati pentru vectorul propriu-zis si numarul de elemente;
- constructor pentru initializarea cu un numar dat pe toate componentele (primeste ca parametru numarul respectiv si numarul componentelor);
- constructori pentru initializare si copiere;
- destructor (în cazul alocarii statice, se seteaza dimensiunea vectorului la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
- metoda-operator public de atribuire =;
- metoda publica pentru reactualizarea numarului de componente si initializarea componentelor cu un numar dat (primeste ca parametru numarul respectiv si numarul componentelor);
- metoda publica pentru calculul sumei tuturor elementelor vectorului;
- metoda publica pentru găsirea maximului și a pozitiei lui;
- metoda publica pentru sortarea crescătoare a vectorului;
- produsul scalar a doi vectori de aceeasi lungime, implementat prin supraincercarea operatorului *.

Tema 4. Clasa "Vector_Complex"

- clasa este prietena a clasei **Numar_Complex** definita în Tema 1;
- membri privati pentru vectorul propriu-zis si numarul de elemente;
- constructor pentru initializarea cu un numar dat pe toate componentele (primeste ca parametru numarul respectiv si numarul componentelor);
- constructori pentru initializare si copiere;
- destructor (în cazul alocarii statice, se seteaza dimensiunea vectorului la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
- supraincercarea operatorilor << și >> sa utilizeze supraincercarea acestora în cadrul clasei **Numar_Complex**;
- metoda publica pentru determinarea vectorului modulelor, folosind metoda de determinare a modulului din clasa **numar complex**;
- metoda publica pentru sortarea crescatoare dupa module a vectorului;
- metoda publica pentru calculul sumei tuturor elementelor vectorului, care sa utilizeze operatorul + din clasa de numere complexe;
- metoda publica pentru a calcula produsul scalar a doi vectori de aceeasi lungime, care sa foloseasca suma si produsul a doi complexi din clasa **numar complex**, si sa supraincarce operatorul *.

Tema 5. Clasa "Matrice_Complexa"

- clasa este prietena a clasei **Numar_Complex** definita în Tema 1;
- membri privati pentru matricea propriu-zisa, nr linii și nr coloane;
- constructor pentru initializarea cu un numar dat pe toate componentele (primeste ca parametru numarul respectiv, numarul de linii și numărul de coloane);
- constructori pentru initializare si copiere;
- destructor (în cazul alocarii statice, se seteaza dimensiunile matricei la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
- supraincercarea operatorilor << și >> sa utilizeze supraincercarea acestora în cadrul clasei **Numar_Complex**;
- metoda publica pentru calculul sumei a doua matrici, implementata prin supraincercarea operatorului + si pe baza functiei de supraincercare a lui + din clasa **numar complex**;
- metoda publica pentru calculul produsului a doua matrici, implementat prin supraincercarea operatorului * si pe baza functiilor de supraincercare ale lui + si * din clasa **numar complex**.

Tema 6. Clasa "Vectori_de_vectori"

Se considera **Class Vectori_de_vectori { int dim; Vector *v;};**

- clasa este prietena a clasei **Vector** definita în **Tema 3**;
- constructor pentru initializarea cu un numar dat pe toate componentele (primeste ca parametru numarul respectiv si numarul componentelor);
- constructori pentru initializare si copiere;
- destructor (în cazul alocarii statice, se seteaza dimensiunea vectorului la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
- supraincercarea operatorilor << și >> sa utilizeze supraincercarea acestora în cadrul clasei Vector;
- metoda publica prin care se creeaza o matrice de numere întregi cu nr_linii = dim și nr_coloane = maximul dimensiunilor vectorilor declarati, elementele matricei fiind cele declarate în vectori, completandu-se cu zerouri în cazul în care dimensiunea unui vector linie e mai mica decât nr_coloane.
- metoda publica prin care să se calculeze suma a doua obiecte de tip Vectori_de_vectori, realizata prin supraincercarea operatorului + rezultatul fiind dat sub forma de matrice, ca în exemplul de mai jos:

Vector_de_vectori A,B;

A = { { 1,2}	B = { { 4}	A + B = 5 2 0 0
{ 1,2,3} }	{ 4,5,6,2}	5 7 8 2
	{ 3,8} }	3 8 0 0.

Tema 7. Clasa "Stiva_de_caractere" (implementata dinamic)

Se considera **Class Nod{ char info; Nod* next;}**

- constructori de inițializare și parametrizati pentru clasa Nod;
- destructor pentru clasa Nod;

Clasa Stiva_de_caractere are:

- membru privat, un "Nod*" (varful stivei);
- un constructor care initializeaza varful stivei cu NULL;
- un destructor care dezaloca toate elementele stivei;
- metode publice de adaugare a unui element în stiva (**push**), de stergere a unui element (**pop**), de afisare a varfului stivei (**top**) și pentru a testa daca stiva e vida (**isempty**);
- metoda publica de fisarea stivei, concomitent cu golirea ei, realizata prin supraincercarea operatorului <<;
- supraincercarea operatorului >>, realizata prin push-uri succesive;
- metoda publica pentru inversarea unui sir de caractere folosind o stiva;
- metoda publica, realizata prin supraincercarea operatorului -, care sa considere doua stive și sa elimine, concomitent, elementele din ambele stive adaugand caracterul ce are codul ASCII mai mare într-o noua stiva, ca în exemplul de mai jos:

Stiva_de_caractere S1,S2;

S1 = {E,X,A,M,E,N}; S2 = {P,O,O,L,A,B,O,R,A,T,O,R} S1 - S2 = {R,O,T,A,X,O}.

Tema 8. Clasa "Coadă_de_caractere" (implementată dinamic)

Se considera **Class Nod**{ *char info; Nod* next;*}

- constructori de inițializare și parametrizați pentru clasa Nod;
- destructor pentru clasa Nod;

Clasa Coadă_de_caractere are:

- membri privați, "Nod*", "Nod*" (primul și ultimul element al cozii);
- un constructor care inițializează coada cu NULL;
- un destructor care dezalocă toate elementele cozii;
- metode publice de adăugare a unui element în stivă (**push**), de ștergere a unui element (**pop**) și pentru a testa dacă e vidă (**isempty**);
- metoda publică de fisare a cozii, concomitent cu golirea ei, realizată prin supraincercarea operatorului <<;
- supraincercarea operatorului >>, realizată prin push-uri succesive;
- metoda publică pentru concatenarea a două cozi de caractere, obținând o altă coadă de caractere, implementată prin supraincercarea operatorului +;
- metoda publică, realizată prin supraincercarea operatorului -, care să considere două cozi și să elimine, concomitent, elementele din ambele cozi adăugând caracterul ce are codul ASCII mai mare într-o nouă coadă, ca în exemplul de mai jos:

Coadă_de_caractere C1, C2;

C1 = {E, X, A, M, E, N}; C2 = {P, O, O, L, A, B, O, R, A, T, O, R} C1 - C2 = {P, X, O, M, E, N}.

Tema 9. Clasa "Listă_circulară" (implementată dinamic)

Se considera **Class Nod**{ *int info; Nod* next;*}

- constructori de inițializare și parametrizați pentru clasa Nod;
- destructor pentru clasa Nod;

Clasa Listă_circulară are:

- cel puțin un membru privat „Nod*” reprezentând nodul considerat prim;
- metoda publică de adăugare a unui element pe o poziție;
- supraincercare a operatorului >>, realizată prin utilizarea succesivă a metodei declarată anterior;
- metoda publică de ștergere a unui element de pe o poziție;
- metoda publică pentru inversarea legăturilor listei;
- metoda publică care primește parametrul întreg k și realizează eliminarea elementelor listei circulare din k în k până la golirea acesteia (elementele vor fi afișate în ordinea în care sunt eliminate);
- supraincercarea operatorului +, care să efectueze concatenarea a două liste circulare, rezultând într-o nouă listă circulară (ca în exemplul de mai jos).

Listă_circulară L1 = { 1 → 2 → 3 → 1 }, L2 = { 4 → 5 → 6 → 4 }




L1 + L2 = { 1 → 2 → 3 → 4 → 5 → 6 → 1 }




Tema 10. Clasa "Listă_dublu_inlantuită"

Se considera **Class Nod**{ *int info; Nod* prev, next;*}

- constructori de inițializare și parametrizați pentru clasa Nod; 
- destructor pentru clasa Nod; 

Clasa Listă_dublu_inlantuită are:

- doi membri privați „Nod*” reprezentând primul și ultimul element; 
- metoda publică de adăugare a unui element pe o poziție; 
- supraincercare a operatorului >>, realizată prin utilizarea succesivă a metodei declarată anterior; 

- supraincarcare a operatorului << pentru afisarea listei in ambele sensuri, in aceeași funcție; 
- metoda publica de stergere a unui element de pe o poziție; 
- supraincarcarea operatorului +, care sa efectueze concatenarea a doua liste dublu inlantuite, rezultand într-o noua lista dublu inlantuita. 

Tema 11. Clasa "Multime" (multimi finite de numere intregi reprezentate ca tablouri unidimensionale)

- membri privati pentru vectorul propriu si numarul de elemente;
- constructori pentru initializare si copiere;
- destructor (în cazul alocarii statice, se seteaza dimensiunea vectorului la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
- metoda publica pentru transformare a unui vector in multime, prin eliminarea duplicatelor din respectivul vector;
- reuniune a doua multimi, implementata prin supraincarcarea operatorului +;
- intersectie a doua multimi, implementata prin supraincarcarea operatorului *;
- diferenta a doua multimi, implementata prin supraincarcarea operatorului -.

Tema 12. Clasa "Multime_dinamic" (multimi finite de numere intregi reprezentate ca liste inlantuite). Cerintele sunt aceleași ca la Tema 11, adaptate pentru liste alocate dinamic.

Tema 13. Clasa "Polinom" - reprezentat ca tablou unidimensional (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;
- membri privati pentru vectorul propriu si numarul de elemente;
- constructori pentru initializare si copiere;
- destructor (în cazul alocarii statice, se seteaza dimensiunea vectorului la zero, iar în cazul alocarii dinamice, se dezaloca zona de memorie utilizata);
- metoda publica pentru calculul valorii unui polinom într-un punct;
- suma a doua polinoame, implementata prin supraincarcarea operatorului +;
- diferenta a doua polinoame, implementata prin supraincarcarea operatorului -;
- produsul a doua polinoame, implementat prin supraincarcarea operatorului *;

Tema 14. Clasa "Polinom_dinamic" - reprezentat ca lista inlantuita (ca polinoame rare, prin lista perechilor (coeficient,exponent), ordonata crescator dupa exponent, si nu neaparat cu exponentii consecutivi). Cerintele sunt aceleași ca la Tema 13, adaptate pentru liste alocate dinamic.

Tema 15. Clasa "ABC"- arbori binari de cautare, in reprezentare inlantuita.

Se considera **Class Nod{ int info; Nod* st, dr;}**

- constructori de inițializare și parametrizați pentru clasa Nod;
- destructor pentru clasa Nod;

Clasa "ABC" are:

- membru privat de tip „Nod*” reprezentând rădăcina arborelui;
- metode publice pentru inserarea unui element, care sa supraincarce operatorul +, care va fi aplicat între un numar reprezentand elementul nou introdus si un arbore;
- supraincarcarea operatorului >>, prin inserari succesive;
- supraincarcarea operatorului <<, care sa afiseze arborele în toate cele 3 metode (preordine, inordine, postordine);
- metoda publica pentru stergerea unui element;
- metoda publica pentru determinarea inaltimii arborelui;
- metoda publica pentru afisarea listei frunzelor.

Tema 16. Clasa "Graf_neorientat"- reprezentate cum doreste programatorul.

- constructori de inițializare și de copiere;
- destructor;
- metoda publica pentru afisarea grafului, care sa supraincarce operatorul << și sa ilustreze toate modalitatile de reprezentare a grafului;
- 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 17. Clasa "Graf_ponderat"- a grafuri neorientate cu ponderi atasate muchiilor, in ce mod doreste programatorul.

- constructori de inițializare și de copiere;
- destructor;
- metoda publica pentru afisarea grafului, care sa supraincarce operatorul << și sa ilustreze toate modalitatile de reprezentare a grafului;
- parcurgerea in latime;
- parcurgerea in adancime;
- determinarea matricii ponderilor drumurilor cu ponderi minime;
- determinarea nodurilor intermediare de pe drumul de pondere minima intre doua 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 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 celor doua grafuri, cu fiecare muchie avand ca pondere minimul dintre ponderile muchiilor corespunzatoare din acele doua grafuri.

Tema 18. Clasa "Numar_intreg_mare"- 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. Clasa "Numar_rational_mare"- 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 supraincari 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 supraincari 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. Clasa "Numar_real_mare"- 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: supraincari pentru operatorii << si >> pentru iesiri si intrari de obiecte; supraincari 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: supraincari 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; supraincari 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.

Urmatoarele teme sunt preluate integral de la domnul profesor Dragulici, cu acordul verbal al dumnealui!

D1. Definiti clasa "persoana", avand:

- membri privati pentru nume (string), anul nasterii (intreg), sex (caracter);
- sase metode publice pentru setarea/furnizarea informatiilor din cei trei membri privati.

Definiti clasa "baza_de_date" avand

- ca membri privati un vector de pointeri la "persoana" si un intreg reprezentand numarul persoanelor;
- ca metode publice un constructor care initializeaza vectorul cu NULL pe fiecare componenta, un destructor care dezaloca toate "persoanele" pointate de componentele vectorului, o metoda pentru adaugat o "persoana", trei metode pentru eliminat persoanele avand un anumit nume, respectiv an al nasterii, respectiv sex (cele trei metode vor avea acelasi nume, prin supraincari), doua metode pentru afisat persoanele continute in baza de date in ordinea alfabetica a numelor, respectiv crescatoare a varstelor.

Cele doua clase de mai sus pot avea si alti membri/metode, dar se va respecta principiul incapsularii datelor (orice setare/furnizare a informatiilor continute in membrii privati se va face doar prin intermediul unor metode). Pentru cele doua clase se va scrie cate un fisier header si un fisier de implementare.

Scrieti un program care sa gestioneze o baza de date folosind clasele de mai sus. El va afisa un meniu (in mod text), care va oferi functii de adaugat o persoana, eliminat persoanele dupa nume, varsta, sex, afisat persoanele in ordine alfabetica sau dupa varsta. Programul va fi construit ca un "project" continand si fisierele de implementare ale celor doua clase.

D2 Clasa pentru numar complex, avand:

- membrii privati pentru partea reala si partea imaginara (double);
 - constructori pentru initializare si copiere;
 - metode publice pentru setat/furnizat partea reala si partea imaginara;
 - metoda publica pentru citire;
 - metoda publica de afisare (sub forma "a", "i*a", "-i*a", "a+i*b", "a-i*b", dupa caz);
 - operatori friend: + (adunare), +(numarul insusi), - (scadere), - (opus), * (inmultire), / (impartire), +=, -=, *=, /= (cu efectul cunoscut), ^ (ridicare la putere naturala), == (test de egalitate), != (test de neegalitate);
- operatorii binari +, -, *, /, ==, != se scriu in cate trei variante, pentru a putea lucra cu operanzi complex-complex, double-complex, complex-double; operatorii +=, -=, *=, /= se scriu in doua variante, pentru a putea lucra cu operanzi complex-complex, complex-double.
- functii friend: "abs" (furnizeaza modulul), "sqrt" (furnizeaza radicalul de ordinul doi).

Program de rezolvare a ecuatiei de ordinul doi cu coeficienti complecsi.

Pentru clasa "complex" se va scrie un fisier header si un fisier de implementare.

Programul se va scrie ca un project.

D3 Clasa "rational", avand:

- membrii privati pentru numarator si numitor (int);
 - metoda privata pentru simplificare;
 - constructori pentru initializare si copiere;
 - metode publice pentru setat/furnizat numaratorul si numitorul;
 - metoda publica pentru citire;
 - metoda publica de afisare (sub forma "a", "a/b", "-a/b", dupa caz);
 - metoda-operator publica de cast, de la rational la double;
 - operatori friend: + (adunare), +(numarul insusi), - (scadere), - (opus), * (inmultire), / (impartire), +=, -=, *=, /= (cu efectul cunoscut), ^ (ridicare la putere naturala), == (test de egalitate), != (test de neegalitate), <, <=, >, >= (comparatii);
- operatorii binari +, -, *, /, ==, !=, <, <=, >, >= se scriu in cate trei variante, pentru a putea lucra cu operanzi rational-rational, rational-intreg, intreg-rational; operatorii +=, -=, *=, /= se scriu in cate doua variante, pentru a putea lucra cu operanzi rational-rational, rational-intreg;
- functie friend "abs" (furnizeaza modulul).

Numerele rationale vor fi memorate sub forma lor canonica (adica numaratorul si numitorul sa fie prime intre ele iar numitorul sa fie > 0). De aceea, implementarea operatiilor aritmetice se va face astfel incat la sfarsit sa se faca automat si o simplificare;

Program de rezolvare a unui sistem de doua ecuatii liniare cu doua necunoscute si coeficienti rationali.

Pentru clasa "rational" se va scrie un fisier header si un fisier de implementare.

Programul se va scrie ca un project.

D4 Clasa "string", avand:

- membrii privati pentru sirul de caractere si lungimea sa;
- constructori pentru initializare si copiere (sa poata lucra cu alte obiecte "string", cu constante de forma "abc", cu siruri pointate de un "char*", si cu valori de tip "char", privite ca stringuri de un element);
- metode publice pentru setat/furnizat sirul de caractere;
- metode publice pentru citire si afisare;

- metode-operator publice de atribuire =; se scrie in doua variante, pentru a putea lucra cu operanzi string-string, string-caracter;

- metoda-operator publica [] (furnizeaza al n-lea caracter din sir);

- operatori friend: + (concatenare), == (testarea egalitatii), !=(testarea neegalitatii), <, <=, >, >= (comparare lexicografica); operatorii se scriu in trei variante, pentru a putea lucra cu operanzi string-string, string-caracter, caracter-string;

- functie friend "length" care furnizeaza lungimea sirului;

Pentru clasa "string" se va scrie un fisier header si un fisier de implementare.

Implementati functiile si procedurile "pos", "copy", "insert", "delete" de lucru cu stringuri din Turbo Pascal, astfel incat sa lucreze cu obiecte "string". Pentru ansamblul acestor functii se va scrie un fisier header si un fisier de implementare.

Program care citeste doua stringuri s1 si s2 si afisaza stringul obtinut prin eliminarea tuturor aparitiilor lui s2 in s1 ca subcuvant (exemplu: s1="abcbabab", s2="ab" ==> "cb").

D5 Clasa "vector" (vector de double), avand:

- membri privati pentru vectorul propriu-zis si numarul de elemente;

- constructor pentru initializarea cu un numar dat pe toate componentele (primeste ca parametru numarul respectiv si numarul componentelor);

- constructori pentru initializare si copiere;

- metode publice pentru citire si afisare;

- metoda-operator public de atribuire =;

- metoda publica pentru reactualizarea numarului de componente si initializarea componentelor cu un numar dat (primeste ca parametru numarul respectiv si numarul componentelor);

- metoda-operator publica [] (furnizeaza o referinta la al n-lea element din vector); implementarea se va face astfel incat daca v este obiect "vector", i este un intreg si x un real, sa fie posibile constructii ca: "cin>>v[i]", "cout<<v[i]", "v[i]=x", "x=v[i]", cu efectul obisnuit;

- operatori friend: + (in doua variante: suma si vectorul insusi (unar)), - (in doua variante: diferenta si opusul (unar)), * (in doua variante: produsul scalar si inmultirea cu un scalar), +=, -= (cu efectul cunoscut), == (testarea egalitatii), != (testarea neegalitatii), <, <=, >, >= (comparare pe componente), !(daca vectorul are 0 pe toate componentele, furnizeaza intregul 0, altfel furnizeaza intregul 1); operatorii ==, !=, <, <=, >, >= se vor scrie in trei variante, pentru a putea lucra cu operanzi vector-vector, real-vector, vector-real (in ultimele doua cazuri se va compara vectorul dat cu un vector de aceeasi dimensiune care are pe fiecare componenta numarul respectiv);

- functie friend "length" care furnizeaza numarul de elemente.

Actiunile vor fi implementate astfel ca atunci cand nu este posibila operatia (de exemplu adunarea unor vectori de dimensiune diferita) sa se afiseze un mesaj de eroare.

Program care citeste mai multi vectori si calculeaza suma lor si maximul lor.

Pentru clasa "vector" se va scrie un fisier header si un fisier de implementare. Programul se va scrie ca un proiect.

D6 Clasa "matrice" (matrice de double), avand:

- membri privati pentru matricea propriu-zisa, numarul de linii si numarul de coloane;

- constructor pentru initializarea cu un numar dat pe toate componentele (primeste ca parametru numarul respectiv si numarul de linii si de coloane);

- constructori pentru initializare si copiere;

- metode publice pentru citire si afisare;

- metoda-operator public de atribuire =;

- metoda publica pentru reactualizarea numarului de linii si coloane si initializarea componentelor cu un numar dat (primeste ca parametru numarul respectiv, numarul liniilor si al coloanelor);

- metoda-operator publica []; daca matricea are o linie sau o coloana, furnizeaza o referinta la al n-lea element; in caz contrar furnizeaza o matrice linie reprezentand a n-a linie din matricea initiala;

implementarea se va face astfel incat daca a este o matrice, i si j doi intregi iar x un real, sa fie posibile constructii ca: "cin>>a[i][j]", "cout<<a[i][j]", "a[i][j]=x", "x=a[i][j]";

- operatori friend: + (in doua variante: suma si matricea insasi (unar)), - (in doua variante: diferenta si opusa (unar)), * (in doua variante: produsul matricilor si inmultirea cu un scalar), +=, -=, *= (cu efectul cunoscut; *= se va scrie in doua variante, avand in vedere inmultirea si inmultirea cu un scalar), == (testarea egalitatii), != (testarea neegalitatii) <, <=, >, >= (comparare pe componente), !(daca matricea are 0 pe toate componentele, furnizeaza intregul 0, altfel furnizeaza intregul 1), ~(inversa); operatorii ==, !=, <, <=, >, >= se vor scrie in trei variante, pentru a putea lucra cu operanzi matrice-matrice, real-matrice, matrice-real (in ultimele doua cazuri se va compara matricea data cu un vector de aceeasi dimensiune care are pe fiecare componenta numarul respectiv);

- functii friend "nrlinii", "nrcoloane", "nrelemente" care furnizeaza numarul liniilor, coloanelor, respectiv elementelor matricii;

Program pentru rezolvarea unei ecuatii matriciale de forma "A*X+B=0" (A,B,X,0 sunt matrici).

Pentru clasa "matrice" se va scrie un fisier header si un fisier de implementare. Programul se va scrie ca un proiect.

D7 Clasa "polinom" (polinom cu coeficienti double), avand:

- membri privati pentru vectorul de coeficienti si numarul de elemente (sau gradul);
- constructori pentru initializare si copiere (sa poata lucra si cu numere reale (privite ca polinoame de grad 0));

- metode publice pentru citire si afisare;

- metoda-operator public de atribuire =; se scrie in doua variante, pentru a putea lucra cu operanzi polinom-polinom, polinom-real (un real fiind privit ca un polinom de grad 0);

- metoda-operator publica [] (furnizeaza o referinta la coeficientul de grad n din polinom); implementarea se va face astfel incat daca p este obiect "polinom", i este un intreg si x un real, sa fie posibile constructii ca: "cin>>p[i]", "cout<<p[i]", "p[i]=x", "x=p[i]", cu efectul obisnuit;

- operatori friend: + (in doua variante: suma si polinomul insusi (unar)), - (in doua variante: diferenta si opusul (unar)), * (inmultirea), / (catul), % (restul), ^ (ridicarea la putere naturala), +=, -=, *=, /= (cu efectul cunoscut), == (testarea egalitatii), != (testarea neegalitatii), !(daca polinomul este nul furnizeaza intregul 0, altfel furnizeaza intregul 1); operatorii binari +, -, *, /, +=, -=, *=, /=, ==, != se vor scrie in trei variante, pentru a putea lucra cu operanzi polinom-polinom, real-polinom, polinom-real (realul e privit ca polinom de grad 0);

- metoda-operator publica () pentru calculul valorii polinomului intr-un punct;

- functie friend "grad" care furnizeaza gradul.

Program pentru determinarea celui mai mare divizor comun a doua polinoame (algoritmul lui Euclid).

Pentru clasa "polinom" se va scrie un fisier header si un fisier de implementare. Acesta va contine si definitia polinomului "X". Programul se va scrie ca un proiect.

D8 Clasa "obiect", avand:

- ca membri privati: un "void*" (adresa unei zone de memorie, alocata dinamic, continand informatia aferenta obiectului), si un "int" (dimensiunea zonei);

- un constructor de initializare (primeste ca parametru dimensiunea zonei si o aloca, initializand-o cu toti bitii 0) si unul de copiere;

- un destructor, care dezaloca zona alocata dinamic;

- o metoda de citire si una de afisare (implementarea este la latitudinea programatorului);

- operatori friend: ==, !=, <, <=, >, >= (implementarea lor este la latitudinea programatorului).

Pentru clasa "obiect" se va scrie un fisier header si un fisier de implementare.

Structura "element", avand ca membri un "obiect*" si un "element*" (ea va fi folosita mai jos la implementarea unei liste de "obiecte" alocata inlantuit).

Clasa "stiva" (stiva de "obiecte", alocata inlantuit), avand:

- ca membru privat, un "element*" (varful stivei);

- un constructor care initializeaza varful stivei cu NULL;

- un destructor care dezaloca toate elementele stivei;

- metode publice pentru: testarea daca stiva e vida, push, pop.

Pentru clasa "stiva" se va scrie un fisier header si un fisier de implementare. Fisierul de implementare va contine si structura "element". Un fisier header si unul de implementare, referitor la un ansamblu de functii uzuale de lucru cu stive: citirea stivei, afisarea stivei, golirea stivei, copierea unei stive in alta stiva, inversarea ordinii elementelor intr-o stiva; aceste functii vor accesa stivele doar prin intermediul metodelor publice.

Un program care citeste o stiva, apoi o inverseaza, apoi iar o inverseaza, apoi o copiaza in alta stiva, apoi o goleste, iar dupa fiecare operatie afisaza stivele obtinute. Programul se va scrie ca un proiect.

D9 La fel ca la I.8, dar cu cozi (clasa "coada" va contine doi membri privati de tip "element*", pentru varful si baza cozii).

D10 Clasa "indivind", avand:

- membrii privati: i,j (intregi) = pozitia intr-o tabela 20X70;
tip (char) = „numele” speciei ('+' sau '0');
varsta (intreg) = varsta (de la 0 la o valoare maxima fixa aleasa de programator);
energie (double) = energia;
viu (unsigned char) = este 1 daca e viu si 0 daca e mort;

- constructor care primeste ca parametrii pozitia si tipul si initializeaza corespunzator membrii respectivi; varsta se initializeaza cu 0, energia cu o valoare fixa aleasa de programator, viu cu 1;

- metode private:

hraneste = creste energia proprie cu o valoare care este cu atat mai mare cu cat aportul energetic al pozitiilor libere invecinate este mai mare iar varsta este mai aproape de jumatatea varstei maxime (formula este la latitudinea programatorului); aportul energetic al unei pozitii libere in tabel este egal cu 1 + numarul de pozitii libere invecinate cu ea;

inmulteste = alocu un numar de fii (indivizi de acelasi tip) in pozitii libere invecinate (fii indivizi sunt initializati de constructorul lor); numarul de fii alocati depinde crescator de energia proprie si de diferenta intre varsta proprie si jumatatea varstei maxime; pentru fiecare fiu generat, energia proprie scade cu o constanta; formulele sunt la latitudinea programatorului;

ataca = pentru fiecare individ x invecinat si de alt tip, daca energia proprie este mai mare decat energia lui x, din energia proprie se scade energia lui x iar x este omorat (i se aplica metoda "**moare**");

imbatraneste = creste varsta cu 1; scade energie cu o valoare constanta aleasa de programator; daca a atins o varsta maxima aleasa de programator sau daca energia sa a devenit ≤ 0 , individul curent este omorat (i se aplica metoda "**moare**");

moare = viu devine 0;

- metode publice:

actualizare = aplica succesiv metodele: hraneste, inmulteste, ataca, imbatraneste;

esteviu = returneaza 1 daca viu==1 si 0 altfel;

gettip = returneaza tipul.

Program care genereaza aleator o tabela 20X70 de indivizi, majoritatea pozitiilor fiind insa goale, apoi intr-un ciclu, la fiecare iteratie, parcurge indivizii (intr-o anumita ordine), pe cei vii ii actualizeaza, pe cei morti ii elimina din tabela, apoi afisaza tabla. Ciclarea dureaza pana se apasa o tasta sau pana nu mai ramane nici un individ pe tabla.

Propunere: tabla sa fie ea insasi un obiect avand metode pentru initializare aleatoare, afisare, actualizare, test daca mai exista indivizi, alocarea unui nou element pe o pozitie data, eliminarea unui element de pe o pozitie data, etc.