Programare Orientată pe Obiecte

Propuneri de proiecte în C++

1. Sa se definească o clasă generică pentru fracţii cu numărător şi numitor de un tip T neprecizat (parametru), în care operatorii =, ==, operatorii aritmetici, +, - (unar si binar), * şi / să fie supraîncărcaţi pentru operaţiile obişnuite cu fracţii, iar operatorul (tip) de conversie a tipurilor să fie supraîncărcat pentru a efectua conversia unui obiect de tipul T la o fracţie care are ca numitor "unitatea" din tipul T (element neutru la *), care poate fi elementul construit de un anume constructor al clasei T, având argument de tip int, care să dea sens declaraţiei cu iniţializare `T a=1` (ca şi declaraţiei cu iniţializare `T a=0`, din care va rezulta elementul "zero" din tipul T, neutru la +). Se vor da exemple de creare şi utilizare de obiecte pentru diferite tipuri ale numărătorului şi numitorului : int, întregi Gauss, adică numere complexe cu părţile reală şi imaginară de tip int (după definirea acestora ca o clasă separată), etc.
2. Sa se definească o clasă generică pentru polinoame de o nedeterminată cu coeficienţi de un tip neprecizat (parametru), în care operatorii =, ==, operatorii aritmetici, +, - (unar si binar), * şi, eventual /, %, să fie supraîncărcaţi pentru operaţiile obişnuite cu polinoame, iar operatorul (tip) de conversie a tipurilor să fie supraîncărcat pentru a efectua conversia unui obiect de tipul coeficienţilor la un polinom de grad 0 şi invers. Şirul coeficienţilor unui polinom se poate reprezenta ca un vector (adică pointer la tipul coeficienţilor) sau ca o listă simplu/dublu înlănţuită (la alegere). Se vor da exemple de creare şi utilizare de obiecte pentru diferite tipuri ale coeficienţilor: int, float, complex (după definirea acestui tip ca o clasă separată), etc.
3. Idem pentru matrici dreptunghiulare cu elemente de tip neprecizat, cu supraîncărcarea operatorilor =, ==, +, - (unar si binar), *, [ ] (indexare) şi conversie de la un obiect de tipul neprecizat dat la o matrice cu 1 linie şi 1 coloană. O matrice cu m linii şi n coloane se va reprezenta ca un vector (adică pointer la tipul elementelor) format din liniile matricii aşezate succesiv, adică a_{11}, a_{12}, ... a_{1n}, a_{21}, a_{22}, ... a_{2n}, ... a_{m1}, a_{m2}, ..., a_{mn}. Operatorul

[ ] va întoarce un pointer la începutul liniei al cărei număr este parametru, numărându-se începand cu 0, în reprezentarea,dată mai sus, adică, dacă a este un obiect a clasei, a[i] va fi pointer la tipul elementelor având ca valoare adresa elementului $a_{(i+1)1}$ în notaţia de mai sus, iar a[i][j] va fi elementul $a_{(i+1)(j+1)}$ din aceeaşi notaţie, a doua utilizare a operatorului [ ] fiind cea predefinită pentru pointeri la tipul elementelor matricii.

4. Idem pentru liste simplu (sau dublu) înlănţuite cu elemente de tip neprecizat, cu supraîncărcarea operatorilor =, ==, <, <=, >, >= (care întorc 1 sau 0 după cum primul operand este sau nu listă iniţială/finală strictă/nestrictă al celui de al doilea) , + (concatenare de liste), - (binar, care întoarce lista obţinută prin ştergerea listei "scazator" din "descazut", dacă "scăzătorul" este prefix al "descăzutului", altfel întoarce "descăzutul" neschimbat), * (care pentru cei doi operanzi liste întoarce lista elementelor care care ocupă aceeaşi poziţie şi coincid în cele două liste), <<, >> reprezentând deplasările la stânga/ dreapta ale listei invocante (primul operand) cu un numar întreg de poziţii care este parametru (al doilea operand), [ ] (indicarea unui element al unei liste prin numărul său de ordine = indice), conversie de la un obiect de tipul neprecizat dat la o listă cu un element şi conversia inversă de la o listă la primul/ultimul său element. O listă se va reprezenta ca un pointer la primul său element.
5. Idem pentru şiruri de elemente de un tip neprecizat, cu supraîncărcarea operatorilor =, ==, <, <=, >, >= (care întorc 1 sau 0 după cum primul operand este sau nu subşir/supraşir strict/nestrict al celui de al doilea), + (reprezentând concatenarea), - (binar, care întoarce şirul obţinut prin ştergerea şirului "scazator" din "descăzut", dacă "scăzătorul" este prefix al "descăzutului", altfel întoarce "descăzutul" neschimbat), * (care pentru cei doi operanzi şiruri întoarce şirul elementelor care ocupă aceeaşi poziţie şi coincid în cele două şiruri), <<, >> reprezentând deplasările la stânga/ dreapta ale şirului invocant (primul operand) cu un numar întreg de poziţii care este al doilea operand, (aceasta însemnând ştergerea de la începutul/sfârşitul şirului al unui număr de elemente egal cu al doilea operand, eventual ştergerea întregului şir dacă lungimea sa este mai mică sau egală cu al doilea operand), conversia de la un obiect de tipul elementelor la un şir de lungime 1 şi conversia inversă de la un şir la primul său element. Şirurile se vor reprezenta ca vectori (adică pointeri la tipul elementelor), deci operatorul [ ] va fi cel predefinit pentru pointeri, nu trebuie supraîncărcat

.

6. Idem pentru şiruri ordonate de elemente de un tip neprecizat, prevăzut cu o ordine totală (şi cu operatorii relaţionali, operatorii de ordine fiind corespunzători respectivei ordini totale), cu supraîncărcarea operatorilor: =, ==, <, <=, >, >= (operatorii de ordine întorc 1 sau 0 conform ordonării lexicografice a operanzilor); + (reprezentând interclasarea); - (binar, care întoarce şirul obţinut prin scoaterea din "descăzut" ale elementelor şirului "scazator" care sunt în "descăzut"); * (care pentru cei doi operanzi şiruri întoarce şirul ordonat al elementelor comune celor două şiruri, indiferent de poziţie); <<, >> ca la tema 5; conversia de la un obiect de tipul elementelor la un şir de lungime 1 şi conversia inversă de la un şir la primul său element. Se va programa construirea unui şir prin adăugarea succesivă la el (cu operatorul supraîncărcat +) a unor elemente citite, ceea ce revine la ordonarea prin inserţie simplă a elementelor citite. Şirurile se vor reprezenta ca vectori (adică pointeri la tipul elementelor), deci operatorul [ ] va fi cel predefinit pentru pointeri, nu trebuie supraîncărcat .
7. Idem pentru mulţimi finite de elemente de un tip neprecizat, cu supraîncărcarea operatorilor =, ==, <, <=, (incluziunea strictă/nestrictă), >, >= (relaţiile duale); + (reuniunea); * (intersecţia); - (binar, diferenţa de mulţimi); <<, >> (adăugarea/ extragerea unui element, care este al doilea operand, la/dintr-o mulţime, reprezentată de primul operand) şi conversia unui element la mulţimea care îl are ca unic element. Multimile se reprezintă printr-un întreg fără semn care este numărul elementelor (pentru mulţimea vidă acest numar va fi 0) şi printr-un vector (adică pointer la tipul elementelor) care conţine elementele. Metodele de creare şi modificare a mulţimilor vor asigura ca fiecare element să apară o singură dată în acest vector.
8. Idem pentru arbori binari de căutare stricţi cu contor, având chei de un tip neprecizat, prevăzut cu o ordine totală (şi cu operatorii relaţionali, operatorii de ordine fiind corespunzători respectivei ordini totale), cu supraîncărcarea operatorilor indicaţi mai jos.

   =, ==; <, <=, >, >= (operatorii de ordine întorc 1 sau conform ordonării lexicografice a şirurilor ordonate care rezultă prin parcurgerea în inordine a operanzilor, cu repetarea fiecărei chei conform contorului);

   + se supraîncarcă în 3 moduri: a) primul operand cheie, al doilea operand arbore binar de căutare strict cu contor; b) primul operand arbore binar de

căutare strict cu contor, al doilea operand cheie; c) ambii operanzi arbori binari de căutare stricţi cu contor; rezultatul operaţiilor a) b) este arborele obţinut prin inserarea cheii în arbore; rezultatul operaţiei c) este arborele obţinut prin inserarea în primul arbore a cheilor arborelui al doilea, parcurse în inordine, cu repetarea fiecărei chei conform contorului;

- se supraîncarcă în 2 moduri: a) primul operand arbore binar de căutare strict cu contor, al doilea operand cheie; b) ambii operanzi arbori binari de căutare stricţi cu contor; rezultatul operaţiei a) este arborele obţinut prin ştergerea cheii din arbore; rezultatul operaţiei b) este arborele obţinut prin ştergerea din primul arbore a cheilor arborelui al doilea, parcurse în inordine, cu repetarea fiecărei chei conform contorului.

9. Idem pentru ansamble, reprezentate ca vectori, având chei de un tip neprecizat, prevăzut cu o ordine totală (şi cu operatorii relaţionali, operatorii de ordine fiind corespunzători respectivei ordini totale), cu supraîncărcarea operatorilor indicaţi mai jos.

   =, ==;

   <, <=, >, >= (operatorii de ordine întorc 1 sau 0 conform ordonării lexicografice a şirurilor ordonate care rezultă din operanzi);

   + se supraîncarcă în 3 moduri: a) primul operand cheie, al doilea ansamblu b) primul operand ansamblu, al doilea operand cheie; c) ambii operanzi ansamble; rezultatul operaţiilor a) b) este ansamblul obţinut prin inserarea cheii în ansamblu; rezultatul operaţiei c) este ansamblul obţinut prin inserarea în primul ansamblu a cheilor ansamblului al doilea, extrase prin decapitări succesive;

   - se supraîncarcă în 2 moduri: a) primul operand ansamblu, al doilea operand cheie; b) ambii operanzi ansamble; rezultatul operaţiei a) este ansamblul obţinut prin ştergerea unei apariţii a cheii din ansamblu, dacă există (aceasta revine la reasamblarea elementelor rămase după ştergerea cheii din vectorul care reprezintă ansamblul); rezultatul operaţiei b) este ansamblul obţinut prin ştergerea din primul arbore a cheilor arborelui al doilea, extrase prin decapitări succesive.

10. Structură ierarhică de clase privitoare la figuri plane, care conţine o clasă abstractă, din care derivă clase care reprezintă: puncte (date prin perechi de coordonate exprimate în pixeli, perechile fiind date cu ajutorul unei structuri definite separat); segmente orizontale (date prin 3 numere reprezentând ordonata comună şi cele două abscise ale capetelor, exprimate în pixeli);

segmente verticale (date prin 3 numere reprezentând abscisa comună şi cele două ordonate ale capetelor, exprimate în pixeli); dreptunghiuri (date printr-un punct reprezentând vârful din stânga sus şi două numere reprezentând laturile dreptunghiului, exprimate în pixeli). Operatorii +, * sunt supraîncărcaţi cu operaţiile de translaţie, respectiv de scalare (adică modificarea fiecărei coordonate a punctelor figurii prin înmulţirea cu un număr real numit factor de scalare ataşat respectivei coordonate). Ambele operaţii au primul operand o figură, iar al doilea operand o pereche de numere reprezentând vectorul de translaţie, respectiv perechea de factori de scalare, rezultatul fiind o figură. Operaţiile se dau ca metode virtuale pure în clasa abstractă şi se definesc pentru fiecare clasă reprezentând figuri particulare. Se vor scrie metode de afişare a figurilor, prin supradefinirea unei metode virtuale pure din clasa abstractă, utilizându-se funcţii din bibliotecile grafice. Clasa abstractă are o dată membră de tip int reprezentând tipul de figură, iar fiecare constructor al unei clase derivate dă acestui câmp o valoare proprie clasei: 0 pentru puncte, 1 pentru segmente orizontale , 2 pentru segmente verticale , 3 pentru dreptunghiuri.

11. Structură ierarhică de clase privitoare la figuri sau grupuri de figuri, unde structura ierarhică pentru figuri este cea de la tema 11, iar grupurile de figuri formează o clasă separată în care un grup se reprezintă ca un vector (pointer) de pointeri la clasa abstractă pentru figuri, construit prin alocare dinamică de memorie corespunzător numărului de elemente ale grupului, acest număr fiind, alături de vector, o dată membră a clasei. Operatorul + se supraîncarcă în 3 moduri: formarea unui grup din două figuri (funcţie prietenă); reuniunea dintre două grupuri (funcţie prietenă); adăugarea unei figuri la un grup (funcţie membră). Operatorul * se supraîncarcă în 2 moduri: intersecţia a 2 figuri; intersecţia tuturor figurilor dintr-un grup cu toate figurile din alt grup (este vorba de intersecţia contururilor), rezultatul fiind întotdeauna un grup (intersecţia a două segmente este grupul vid sau un grup cu un punct sau cu un segment; a unui segment cu un dreptunghi este grupul vid sau un grup cu un segment sau cu 1 sau 2 puncte; a două dreptunghiuri este grupul vid sau un grup cu un dreptunghi, 1, 2, 3 segmente, un număr de segmente şi puncte, sau 1, 2, 4 puncte; a unui punct cu altă figură este grupul vid sau punctul respectiv). Se vor scrie metode de afişare a figurilor şi grupurilor de figuri, utilizându-se funcţii din bibliotecile grafice.

Cerinţe comune

Toate clasele vor trebui să aibă constructor fără argumente, constructor de copiere, cel puţin încă un constructor, destructor şi metode de citire a datelor de la consolă.

Se vor putea propune şi alte teme de către studenţi, îndeplinind următoarele condiţii:

- cerinţele comune de mai sus;
- supraîncărcarea mai multor funcţii şi operatori, dintre care operatorul de atribuire şi cel puţin încă 2 operatori;
- se utilizează clase generice sau clase abstracte şi moştenire.

La o temă pot lucra până la 3 studenţi, urmând ca examinarea şi verificarea cunoştinţelor să se facă individual.

Implementările unor algoritmi predaţi la cursul Structuri de Date şi Algoritmi care sunt incluse în proiectele de Programare Orientată pe Obiecte pot fi incluse şi în proiectele prezentate la examenul de Structuri de Date şi Algoritmi. O astfele de implementare poate fi inclusă în proiectul de Structuri de Date şi Algoritmi al numai unuia din autorii unui proiect de Programare Orientată pe Obiecte.