Teme de casă
Setul 1 de probleme

1. **Clasă care implementează o secvență ordonată de șiruri de carcatere**. Această clasă memorează șirurile îmtr-o listă de liniară, fiecare element reprezentând un șir. Se va crea suplimentar o clasă pentru reprezentarea elementelor listei.

Funcții membre publice pentru clasa secvență:

a) constructori;
b) destructor;
c) o funcție care determină numărul de șiruri din secvență;
d) o funcție care inserează un nou șir în secvență, astfel încât ea să rămână ordonată;
e) o funcție care elimină un șir din secvență;
f) o funcție care caută un șir în secvență;
g) o funcție iterator care parcurge elementele secvenței; iteratorul returnează elementul curent și va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul;
h) o funcție de afișare pe ecran a secvenței.

Operatori supraîncărcați pentru clasa secvență:

a) operatorul = de atribuire;
b) operatorul + pentru concatenare a doua secvențe; după concatenare, secvența rezultată se va ordona;
c) operatorul + între o secvență și un șir de caractere, care inserează șirul în secvență (secvența rămâne ordonată după inserare);
d) operatori de comparație (<, >, ==, !=) pentru compararea secvențelor; compararea se face pe baza ordinii lexicografice.

Să se proiecteze clasele astfel încât să permită operațiile:

```
Secventa s1;         // secventa vida
s1.Insert("abc");
Sir a("xyz");
s1.Insert(a);
Secventa s2 = s1, s3;
s3 = s1 + s2 + "123";      // s3 contine "123" "abc" "abc" "xyz" "xyz"
Sir b = a;
b.Set("999");
Sir c;               // sir vid
a = c = b + a;       // sirurile a si c contin "999abc"
```

Clasa elementelor din listă se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa secvenței.

2. **Clasă care implementează matrici de șiruri de caractere**. Exemplu de matrice pătrată bidimensională:

```
"aaa"  "bb"
"c"    "ddd"
```

Matricea este caracterizată de dimensiunile proprii (numărul de linii și coloane), precum și de lungimea maximă permisă a șirurilor. Un element al unei asemenea matrici are tipul de date `char*` și este un șir terminat cu caracterul '`\0`'. Elementele matricii sunt memorate într-un tablou.

Funcții membre publice:

a) constructori;
b) destructor;
c) funcții care determină numărul de linii și de coloane al unei matrici;
d) o funcție iterator care parcurge matricea pe linii, iar în cadrul fiecarei linii, element cu element; iteratorul returnează elementul curent din matrice și va fi circular, în sensul că după ultimul element din matrice, va fi apoi poziționat pe primul element;
e) o funcție de afișare pe ecran a matricii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
a) operatorul + care concatenează șirurile din aceeași poziție din cele două matrici;
b) operatorul `()` care permite accesul la un șir prin poziția lui în matrice: `a(i,j)` este un `char*` ce indică spre șirul din linia *i* și coloana *j*;
c) operatori relaționali (<, >, ==, !=) pentru compararea lexicografică a matricilor; elementele matricii se compară pe linii, iar în cadrul liniilor element cu element;

Să se proiecteze clasa astfel încât să permită operațiile:

```
MatSir a1;           // matricea vida de dimensiune 0, 0
```

```
MatSir a2(3, 4);    // matrice cu 3 linii si 4 coloane
MatSir a3(3, 4, TRUE);
    // constructorul citeste de la tastatura sirurile,
    // daca are al treilea argument TRUE;
    // argumentul implicit este FALSE, caz in care
    // sirurile sunt siruri vide ""
MatSir a4 = a3;
a1 = a2 = a3 + a4;
```

3. **Clasă care implementeaza matrici pătrate cu alocare dinamică**. Elementele unei asemenea matrici sunt numere reale.

Funcții membre publice:

a) constructori;
b) destructor;
c) o funcție care determină dimensiunea unei matrici;
d) o funcție iterator care parcurge matricea pe linii, iar în cadrul fiecarei linii, element cu element; iteratorul returnează elementul curent din matrice şi va fi circular, în sensul că după ultimul element din matrice, va fi apoi poziționat pe primul element;
e) o funcție `New` care eliberează memoria ocupată de elementele matricii curente şi alocă o nouă matrice cu o dimensiune specificată;
f) o funcție de afişare pe ecran a matricii.

Operatori supraîncărcați:

a) operatorul `=` de atribuire;
b) operatorii `+`, `-`, `*` pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul `()` care permite accesul la un element din matrice prin indici;
d) operatorul `!` care transformă matricea într-o matrice unitate;
e) operatorul `~` pentru operația de transpunere a matricii.

Să se proiecteze clasa astfel încât să permită operațiile:

```
Mat a;        // matrice de dimensiune 1, 1
Mat b(2);     // matrice de dimensiune 2, 2
Mat c(0);     // se citesc de la tastatura atat dimensiunea,
              // cat si valorile elementelor;
Mat d = c;
a.New(2);     // matricea a are acum dimeniunea 2, 2
a(1, 2) = 1.5;
```

4. **Clasă care implementeaza matrici bidimensionale cu alocare dinamică**. Elementele unei asemenea matrici sunt numere întregi.

Funcții membre publice:

a) constructori;
b) destructor;
c) funcții care determină dimensiunile unei matrici;
d) o funcție iterator care parcurge matricea pe linii, iar în cadrul fiecarei linii, element cu element; iteratorul returnează elementul curent din matrice şi va fi circular, în sensul că după ultimul element din matrice, va fi apoi poziționat pe primul element;
e) o funcție `New` care eliberează memoria ocupată de elementele matricii curente şi alocă o nouă matrice cu dimensiuni specificate;
f) o funcție de afişare pe ecran a matricii.

Operatori supraîncărcați:

a) operatorul `=` de atribuire;
b) operatorii `+`, `-`, `*` pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul `()` care permite accesul la un element din matrice prin indici;

Să se proiecteze clasa astfel încât să permită operațiile:

```
Mat a;        // matrice de dimensiuni 1, 1
Mat b(2, 3);  // matrice de dimensiuni 2, 3
Mat c(0);     // se citesc de la tastatura atat dimensiunile,
              // cat si valorile elementelor;
Mat d = c;
a.New(2, 3);  // matricea a are acum dimeniunile 2, 3
a(1, 2) = 10;
```

5. **Clasă care implementeaza matrici tridimensionale cu alocare dinamica**. Elementele unei asemenea matrici sunt numere întregi, iar pentru memorarea lor internă se va utiliza un singur tablou cu alocare dinamică. Elementele matricii vor fi liniarizate în tabloul dimanic în mod asemănător algoritmului utilizat de compilator. De exemplu, pentru o matrice de dimensiune `a(m, n, k)`, elementul `a(i, j, k)` se va afla în vectorul liniarizat pe poziția următoare:

```
i + m * (j + k * n)
```

Funcții membre publice:

a) constructori;
b) destructor;
c) funcții care determină dimensiunile unei matrici;
d) o funcție iterator care parcurge matricea, ordinea de variație a indicilor unui element `a(i, j, k)` fiind considerată *k*, *j*, *i*; iteratorul returnează elementul curent din matrice şi va fi circular, în sensul că după ultimul element din matrice, va fi apoi poziționat pe primul element;
e) o funcție `New` care eliberează memoria ocupată de elementele matricii curente şi alocă o nouă matrice cu dimensiuni specificate;
f) o funcție de afişare pe ecran a matricii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, - pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul `()` care permite accesul la un element din matrice prin indici;

Să se proiecteze clasa astfel încât să permită operațiile:

```
Mat a;        // matrice de dimensiune 1, 1, 1
Mat b(2, 3, 2);       // matrice de dimensiune 2, 3, 2
Mat c(0);     // se citesc de la tastatura atat dimensiunile,
              // cat si valorile elementelor;
Mat d = c;
a.New(2, 3, 2);       // matricea a are acum dimeniunea 2, 3, 2
a(1, 2, 1) = 10;
```

6. **Clasă care implementeaza matrici pătrate de numere complexe**. Se va crea şi utiliza în plus o clasă asociată numerelor complexe.

Funcții membre publice pentru matrice:

a) constructori;
b) destructor;
c) o funcție care determină dimensiunea unei matrici;
d) o funcție iterator care parcurge matricea pe linii, iar în cadrul fiecarei linii, element cu element; iteratorul returnează elementul curent din matrice şi va fi circular, în sensul că după ultimul element din matrice, va fi apoi poziționat pe primul element;
e) o funcție de afişare pe ecran a matricii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, * pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul `()` care permite accesul la un element din matrice prin indici;

Să se proiecteze clasa matrice astfel încât să permită operațiile:

```
Mat a;        // matrice de dimensiune 1, 1
Mat b(2);     // matrice de dimensiune 2, 2
Mat c(0);     // se citesc de la tastatura atat dimensiunea,
              // cat si valorile elementelor;
Mat d = c;
complex z(2, 3);
a(1, 2) = z;
complex z1 = z;
z1.Re() = 5;
a(1, 3) = z1;
```

Clasa numerelor complexe se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa matrice.

7. **Clasă care implementeaza matrici bidimensionale de numere complexe**. Se va crea şi utiliza în plus o clasă asociată numerelor complexe.

Funcții membre publice pentru matrice:

a) constructori;
b) destructor;

c) funcții care determină dimensiunile unei matrici;
d) o funcție iterator care parcurge matricea pe linii, iar în cadrul fiecarei linii, element cu element; iteratorul returnează elementul curent din matrice şi va fi circular, în sensul că după ultimul element din matrice, va fi apoi poziționat pe primul element;
e) o funcție de afişare pe ecran a matricii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, * pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul () care permite accesul la un element din matrice prin indici;

Să se proiecteze clasa matrice astfel încât să permită operațiile:

```
Mat a;        // matrice de dimensiuni 1, 1
Mat b(2, 3);  // matrice de dimensiuni 2, 3
Mat c(0);     // se citesc de la tastatura atat dimensiunile,
              // cat si valorile elementelor;
Mat d = c;
complex z(2, 3);
a(1, 2) = z;
complex z1 = z;
z1.Re() = 5;
a(1, 3) = z1;
```

Clasa numerelor complexe se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa matrice.

8. **O clasă care implementează matricile rare cu elemente reale**. Matricile rare sunt acele matrici la care majoritatea elementelor sunt zero. În practică, acesta matrici au dimensiuni mari (de exemplu 500×500) şi reprezentarea lor normală necesită o mare cantitate de memorie. O metodă economică de reprezentare a acestora presupune memorarea doar a valorilor nenule, împreună cu poziția lor în matrice.

Definiți o clasă care utilizează o listă simplu legată pentru memorarea elementelor nenule şi a poziției acestora. Pentru reprezentarea elementelor listei se va crea o clasă suplimentară.

Funcții membre publice pentru matrice:

a) constructori;
b) destructor;
c) funcții care determină dimensiunile unei matrici;
d) o funcție iterator care parcurge lista elementelor nenule; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element din listă, va fi apoi poziționat pe primul element;
e) o funcție de afişare pe ecran a matricii (se vor afişa şi elementele nule).

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, * pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul () care permite accesul la un element din matrice prin indici;

Să se proiecteze clasa matrice astfel încât să permită operațiile:

```
MatRara a; //dimensiunile si elementele nenule
             //se citesc de la tastatura
MatRara b(3, 4, 2); //3 linii, 4 coloane, 2 valori nenule
MatRara c(3, 4);    //3 linii, 4 coloane; numarul de elemente
                    //nenule si valorile lor
                    //se citesc de la tastatura
a.Init();    //datele se citesc de la tastatura
MatRara d = b;
a(1, 2) = 3.7;
```

Clasa elementelor listei se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa matrice.

9. **O clasă care implementează matricile rare cu elemente reale**. Matricile rare sunt acele matrici la care majoritatea elementelor sunt zero. În practică, acesta matrici au dimensiuni mari (de exemplu 500×500) şi reprezentarea lor normală necesită o mare cantitate de memorie. O metodă economică de reprezentare a acestora presupune memorarea doar a valorilor nenule, împreună cu poziția lor în matrice.

Definiți o clasă care utilizează un tablou pentru memorarea elementelor nenule şi a poziției acestora. Pentru reprezentarea elementelor nenule şi a poziției lor se va crea o clasă suplimentară.

Funcții membre publice pentru matrice:

a) constructori;
b) destructor;
c) funcții care determină dimensiunile unei matrici;

d) o funcție iterator care parcurge tabloul elementelor nenule; iteratorul returnează elementul curent din tablou şi va fi circular, în sensul că după ultimul element din tablou, va fi apoi poziționat pe primul element;
e) o funcție de afişare pe ecran a matricii (se vor afişa şi elementele nule).

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, * pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul ( ) care permite accesul la un element din matrice prin indici;

Să se proiecteze clasa matrice astfel încât să permită operațiile:

```
MatRara a;   //dimensiunile si elementele nenule
             //se citesc de la tastatura
MatRara b(3, 4, 2); //3 linii, 4 coloane, 2 valori nenule
MatRara c(3, 4);    //3 linii, 4 coloane; numarul de elemente
                    //nenule si valorile lor
                    //se citesc de la tastatura
a.Init();    //datele se citesc de la tastatura
MatRara d = b;
a(1, 2) = 3.7;
```

Clasa valorilor nenule se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa matrice.

10. **O clasă care implementează matricile diagonale cu elemente reale**. Matricile diagonale sunt acele matrici pătrate la care toate elementele din afara diagonalei principale sunt zero. O metodă economică de reprezentare a acestora presupune memorarea doar a valorilor nenule de pe diagonală, împreună cu poziția lor în matrice.
Definiți o clasă care utilizează o listă simplu legată pentru memorarea elementelor nenule şi a poziției acestora. Pentru reprezentarea elementelor listei şi a poziției lor se va crea o clasă suplimentară.
Funcții membre publice pentru matrice:

a) constructori;
b) destructor;
c) funcții care determină dimensiunile unei matrici;
d) o funcție iterator care parcurge lista elementelor nenule; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element din listă, va fi apoi poziționat pe primul element;
e) o funcție de afişare pe ecran a matricii (se vor completa şi elementele nule).

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, * pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul ( ) care permite accesul la un element din matrice prin indici;
d) operatorul ~ pentru transpunerea unei matrici.

Să se proiecteze clasa matrice astfel încât să permită operațiile:

```
Diag a;      //dimensiunile si elementele nenule
             //se citesc de la tastatura
Diag b(3);   //3 linii, 3 coloane;
             //valorile elementelor nenule
             //se citesc de la tastatura
Diag c = b;
a(1, 2) = 3.7;
```

Clasa elementelor listei se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa matrice.

11. **O clasă care implementează matricile diagonale cu elemente reale**. Matricile diagonale sunt acele matrici pătrate la care toate elementele din afara diagonalei principale sunt zero. O metodă economică de reprezentare a acestora presupune memorarea doar vectorului diagonalei principale. Definiți o clasă care utilizează o tablou pentru memorarea diagonalei principale.
Funcții membre publice pentru matrice:

a) constructori;
b) destructor;
c) funcții care determină dimensiunile unei matrici;
d) o funcție iterator care parcurge tabloul elementelor nenule; iteratorul returnează elementul curent din tablou şi va fi circular, în sensul că după ultimul element din tablou, va fi apoi poziționat pe primul element;
e) o funcție de afişare pe ecran a matricii (se vor completa şi elementele nule).

Operatori supraîncărcați:

a) operatorul = de atribuire;

b) operatorii +, -, * pentru operațiile elementare cu matrici, cu verificarea corectitudinii acestora;
c) operatorul ( ) care permite accesul la un element din matrice prin indici;
d) operatorul ~ pentru transpunerea unei matrici.

Să se proiecteze clasa matrice astfel încât să permită operațiile:

```
Diag a;      //dimensiunile si elementele nenule
             //se citesc de la tastatura
Diag b(3);   //3 linii, 3 coloane;
             //valorile elementelor nenule
             //se citesc de la tastatura
Diag c = b;
a(1, 2) = 3.7;
```

Clasa valorilor nenule se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa matrice.

12. **O clasă care implementează vectori de numere complexe**. Un asemenea vector este reprezentat de o pereche de numere complexe, $v = (z_i, z_o)$, unde z_i reprezintă originea vectorului, iar z_o vârful său. Se va crea şi utiliza în plus o clasă asociată numerelor complexe.

Funcții membre publice pentru clasa vector:

a) constructori;
b) funcții care determină originea şi vârful vectorului;
c) funcții care setează originea şi vârful vectorului;
a) o funcție de afişare pe ecran a vectorului.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, - pentru operațiile elementare cu vectori;
d) operatorul ~ pentru calculul modulul unui vector conform relației:
$|v| = ((Im(z_o)-Im(z_l))^2 + (Im(z_o)-Im(z_l))^2)^{1/2}$;
e) operatori de comparare <, >, ==, care compară vectorii pe baza valorii modulelor; de exemplu:
$v_1 < v_2$, dacă $|v_1| < |v_2|$.

Să se proiecteze clasa vector astfel încât să permită operațiile:

```
VectComplex u;       // datele initiale se introduc de la tastatura
VectComplex v(1.1, 2.2, 3.3, 4.4);       // v = ((1.1, 2.2), (3.3, 4.4))
VectComplex w(1.1, 2.2);          // v = ((1.1, 2.2), (0, 0))
complex c1(1, 2), c2(3, 4);
VectComplex v1(c1, c2), v2 = v1;
```

Clasa numerelor complexe se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa vector.

Funcția de test trebuie să fie proiectată astfel încât conțină una din variantele următoare:

- alocarea dinamică a unor obiecte din clasa vector;
- un tablou de obiecte de tip vector.

13. **O clasă care implementează vectori în plan**. Un vector este specificat prin coordonatele punctului origine si cele ale punctului destinatie: $v = ((x_i, y_i), (x_o, y_o))$. Se va crea şi utiliza în plus o clasă asociată punctelor din plan.

Funcții membre publice pentru clasa vector:

f) constructori;
g) funcții care determină originea şi vârful vectorului;
h) funcții care setează originea şi vârful vectorului;
b) o funcție de afişare pe ecran a vectorului.

Operatori supraîncărcați:

c) operatorul = de atribuire;
d) operatorii +, - pentru operațiile elementare cu vectori; de exemplu, suma a doi vectori se poate defini astfel:

```
v1 = ((xi1,yi1), (xo1,yo1)), v2 = ((xi2,yi2), (xo2,yo2))
v3 = v1 + v2 = ((xi3,yi3), (xo3,yo3)), unde:
xi3 = xi1
dx1 = xo1 – xi1, dx2 = xo2 – xi2
xo3 = xi3 + dx1 + dx2
yi3 = yi1
dy1 = yo1 – yi1, dy2 = yo2 – yi2
yo3 = yi3 + dy1 + dy2
```

i) operatorul ~ pentru calculul modulul unui vector conform relației:
$|v| = ((x_o-x_i)^2 + (y_o-y_i)^2)^{1/2}$;
j) operatori de comparare <, >, ==, care compară vectorii pe baza valorii modulelor; de exemplu:
$v_1 < v_2$, dacă $|v_1| < |v_2|$.

Să se proiecteze clasa vector astfel încât să permită operațiile:

```
VectPlan u;  // datele initiale se introduc de la tastatura
VectPlan v(1.1, 2.2, 3.3, 4.4);          // v = ((1.1, 2.2), (3.3, 4.4))
VectPlan w(1.1, 2.2);             // v = ((1.1, 2.2), (0, 0))
Punct p1(1, 2), p2(3, 4);
VectPlan v1(p1, p2), v2 = v1;
```

Clasa punctelor din plan se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa vector.
Funcția de test trebuie să fie proiectată astfel încât conțină una din variantele următoare:

- alocarea dinamică a unor obiecte din clasa vector;
- un tablou de obiecte de tip vector.

14. **O clasă care implementează vectori în spațiul R^n**. Un asemenea vector este specificat tabloul de numere reale:

$v = (x_1, x_2, \ldots, x_n)$.

Clasa vector va memora dimensiunea sa, precum şi tabloul coordonatelor.
Funcții membre publice pentru clasa vector:

a) constructori;
b) funcții care determină dimensiunea vectorului;
c) o funcție iterator care parcurge elementele vectorului; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
c) o funcție de afişare pe ecran a vectorului.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul `[]` pentru accesarea componentelor vectorului;
c) operatorii +, -, * pentru operațiile elementare: suma, diferența şi produsul scalar a doi vectori;
   $u = (x_1, \ldots, x_n)$, $v = (y_1, \ldots, y_n)$
   $u+v = (x_1+y_1, \ldots, x_n+y_n)$
   $u-v = (x_1-y_1, \ldots, x_n-y_n)$
   $u*v = x_1*y_1 + x_2*y_2 + \ldots + x_n*y_n$
d) operatorul ~ pentru calculul modulului unui vector conform relației:
   $|v| = (x_1^2 + \ldots + x_n^2)^{1/2}$;
e) operatori de comparare <, >, ==, care compară vectorii pe baza ordinii lexicografice.

Să se proiecteze clasa vector astfel încât să permită operațiile:

```
VectR u;      // datele initiale se introduc de la tastatura
VectR v(4);   // vector cu dimensiunea 4 neinitializat
VectR w = u;
V[1] = 5.5;
```

Funcția de test trebuie să fie proiectată astfel încât conțină una din variantele următoare:

- alocarea dinamică a unor obiecte din clasa vector;
- un tablou de obiecte de tip vector.

15. **O clasă care implementează vectori în spațiul C^n**. Un asemenea vector este specificat tabloul de numere complexe:

$v = (z_1, z_2, \ldots, z_n)$, $z_i \in C$, i=1, 2, …, n.

Clasa vector va memora dimensiunea sa, precum şi tabloul coordonatelor. Se va crea şi utiliza în plus o clasă asociată numerelor complexe.
Funcții membre publice pentru clasa vector:

a) constructori;
b) funcții care determină dimensiunea vectorului;
c) o funcție iterator care parcurge elementele vectorului; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
d) funcție de afişare pe ecran a vectorului.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul `[]` pentru accesarea componentelor vectorului;
c) operatorii +, -, * pentru operațiile elementare: suma, diferența şi produsul scalar a doi vectori;
   $u = (x_1, \ldots, x_n)$, $v = (y_1, \ldots, y_n)$
   $u+v = (x_1+y_1, \ldots, x_n+y_n)$
   $u-v = (x_1-y_1, \ldots, x_n-y_n)$
   $u*v = x_1*y_1 + x_2*y_2 + \ldots + x_n*y_n$
e) operatorul ~ pentru calculul modulului unui vector conform relației:

$|v| = (x_1^2 + \ldots + x_n^2)^{1/2}$;

f) operatori de comparare <, >, ==, care compară vectorii pe baza ordinii lexicografice; numerele complexe se vor compara pe baza modulului lor.

Să se proiecteze clasa vector astfel încât să permită operațiile:

```
VectC u;      // datele initiale se introduc de la tastatura
VectC v(4);   // vector cu dimensiunea 4 neinitializat
VectC w = u;
Complex z(1.1, 4.4);
V[1] = z;
```

Clasa numerelor complexe se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa vector.

16. **Clasă care implementează o stivă de numere întregi**. Stiva este reprezentată printr-o listă simplu legată a elementelor sale. Se va crea în plus o clasă pentru memorarea elementelor listei.

Funcții membre publice pentru clasa stivă:

a) constructori;
b) destructor;
c) funcție care determină dimensiunea stivei;
d) funcție care determină dacă stiva este vidă;
e) funcțiile specifice *Push* şi *Pop* pentru introducerea şi extragerea unui element;
f) o funcție iterator care parcurge elementele listei; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
g) funcție de afişare pe ecran a stivei.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul + pentru concatenarea elementelor a două stive;
c) operatori de comparare <, >, ==, care compară stivele pe baza dimensiunii lor;
d) operatorul [] pentru accesarea componentelor stivei pe baza poziției lor;

Să se proiecteze clasa stivă astfel încât să permită operațiile:

```
stiva s1, s2;       // stiva vida
stiva s3 = s1, s4;
s4 = s3 = s2 + s1 ;
```

Clasa elementelor se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa stivă.

17. **Clasă care implementează o coadă de numere întregi**. Coada este reprezentată printr-o listă simplu legată a elementelor sale. Se va crea în plus o clasă pentru memorarea elementelor listei.

Funcții membre publice pentru clasa coadă:

a) constructori;
b) destructor;
c) funcție care determină dimensiunea cozii;
d) funcție care determină dacă coada este vidă;
e) funcțiile specifice *Add* şi *Del* pentru introducerea şi extragerea unui element;
f) o funcție iterator care parcurge elementele listei; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
g) funcție de afişare pe ecran a cozii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul + pentru concatenarea elementelor a două cozi;
c) operatori de comparare <, >, ==, care compară cozile pe baza dimensiunii lor;
d) operatorul [] pentru accesarea componentelor cozii pe baza poziției lor;

Să se proiecteze clasa coadă astfel încât să permită operațiile:

```
coada c1, c2;       // coada vida
coada c3 = c1, c4;
c4 = c3 = c2 + c1 ;
```

Clasa elementelor se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa coadă.

18. **Clasă care implementează o stivă de numere întregi**. Stiva este reprezentată printr-un tablou cu alocare dinamică a elementelor, precum şi prin dimensiunea curentă a stivei, numărul total al elemntelor din stivă şi indicele elementului din vârful stivei.

Funcții membre publice pentru clasa stivă:

a) constructori;
b) destructor;

c) funcție care determină dimensiunea stivei;
d) funcție care determină dacă stiva este vidă;
e) funcțiile specifice *Push* şi *Pop* pentru introducerea şi extragerea unui element; pentru funcția *Push*, dacă nu mai sunt elemente disponibile, se va realoca tabloul cu o dimensiune mai mare;
f) o funcție iterator care parcurge elementele tabloului; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
g) funcție de afişare pe ecran a stivei.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul + pentru concatenarea elementelor a două stive;
c) operatorul + între o stivă şi un întreg: `s+n` reprezintă mărirea dimensiunii stivei cu *n* elemente;
d) operatori de comparare <, >, ==, care compară stivele pe baza numărului lor de elemente;
e) operatorul `[]` pentru accesarea componentelor stivei pe baza poziției lor;

Să se proiecteze clasa stivă astfel încât să permită operațiile:

```
stiva s1, s2;       // stiva vida
stiva s3 = s1, s4;
s4 = s3 = s2 + s1 ;
```

19. **Clasă care implementează o coadă de numere întregi**. Coada este reprezentată printr-un tablou cu alocare dinamică a elementelor, precum şi prin dimensiunea curentă a cozii, numărul total al elemntelor din coadă şi indicii primului şi ultimului element.

Funcții membre publice pentru clasa coadă:

a) constructori;
b) destructor;
c) funcție care determină dimensiunea cozii;
d) funcție care determină dacă coada este vidă;
e) funcțiile specifice *Add* şi *Del* pentru introducerea şi extragerea unui element; pentru funcția *Add*, dacă nu mai sunt elemente disponibile, se va realoca tabloul cu o dimensiune mai mare;
f) o funcție iterator care parcurge elementele tabloului; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
g) funcție de afişare pe ecran a cozii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul + pentru concatenarea elementelor a două cozi;
c) operatorul + între o coadă şi un întreg: `s+n` reprezintă mărirea dimensiunii cozii cu *n* elemente;
d) operatori de comparare <, >, ==, care compară cozile pe baza numărului lor de elemente;
e) operatorul `[]` pentru accesarea componentelor cozii pe baza poziției lor;

Să se proiecteze clasa coadă astfel încât să permită operațiile:

```
coada c1, c2;       // coada vida
coada c3 = c1, c4;
c4 = c3 = c2 + c1 ;
```

20. **O clasă care implementează mulțimi de numere întregi**. O mulțime este specificată prin numărul de elemente, precum şi prin tabloul elementelor. Clasa mulțime va utiliza un tablou cu alocare dinamică a elementelor.

Funcții membre publice pentru clasa multime:

a) constructori;
b) destructor;
c) o funcție care determină numărul de elemente din mulțime;
d) o funcție care determină dacă mulțimea este vidă;
e) o funcție care determină dacă un element aparține multimii;
f) o funcție iterator care parcurge elementele mulțimii; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
g) o funcție de afişare pe ecran a mulțimii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, *, pentru operațiile de reuniune, diferență şi intersecție a două mulțimi;
c) operatori de comparare <, >, ==, care compară mulțimile pe baza operației de incluziune;
d) operatorul << pentru introducerea unui element în mulțime; dacă nu mai sunt elemente disponibile în tablou, se va realoca tabloul cu o dimensiune mai mare;

Să se proiecteze clasa mulțime astfel încât să permită operațiile:

```
multime m1;         // multime vida
multime m2(3, 2, 7);      // multimea {3, 2, 7}
multime m3 = m1, m4;
m4 = m3 = m2 + m1 ;
```

21. **O clasă care implementează mulțimi de numere întregi**. O mulțime este specificată printr-o listă liniară a elementelor componente. Se va crea în plus o clasă pentru memorarea elementelor listei.
Funcții membre publice pentru clasa multime:

a) constructori;
b) destructor;
c) o funcție care determină numărul de elemente din mulțime;
d) o funcție care determină dacă mulțimea este vidă;
e) o funcție care determină dacă un element aparține multimii;
f) o funcție iterator care parcurge elementele mulțimii; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
g) o funcție de afişare pe ecran a mulțimii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, *, pentru operațiile de reuniune, diferență şi intersecție a două mulțimi;
c) operatori de comparare <, >, ==, care compară mulțimile pe baza operației de incluziune;
d) operatorul << pentru introducerea unui element în mulțime;

Să se proiecteze clasa mulțime astfel încât să permită operațiile:

```
multime m1;         // multime vida
multime m2(3, 2, 7);      // multimea {3, 2, 7}
multime m3 = m1, m4;
m4 = m3 = m2 + m1 ;
```

Clasa elementelor mulțimii se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa mulțime.

22. **O clasă care implementează mulțimi multiple de numere întregi**. O mulțime multiplă reprezintă o extindere a noțiunii de mulțime, la care elementele nu mai trebuie să fie distincte; de exemplu, {1, 1, 2, 3, 3, 3} reprezintă o mulțime multiplă. Numărul de elemente identice *e* dintr-o mulțime multiplă *M* se notează k(e, M). Sunt permise aceleaşi operații ca şi la mulțimile simple. În plus, pentru o mulțime multiplă se poate determina suportul ei, notat cu `supp`, adică mulțimea simplă ce conține numai elementele distincte din mulțimea multiplă. Operația de incluziune se poate extinde asupra mulțimilor multiple:

$M_1 \subseteq M_2$ dacă $supp(M_1) \subseteq supp(M_2)$ şi $\forall e \in supp(M_1)$, $k(e, M_1) \leq k(e,)$.

Pentru două mulțimi multiple, M_1 şi M_2, se poate extinde noțiunea de diferență M_1-M_2, dacă $M_2 \subseteq M_1$.
O mulțime multiplă este specificată prin numărul de elemente, precum şi prin tabloul elementelor. Clasa mulțime multiplă va utiliza un tablou cu alocare dinamică a elementelor.
Funcții membre publice pentru clasa multime multiplă:

a) constructori;
b) destructor;
c) o funcție care determină numărul de elemente din mulțime;
d) o funcție care determină dacă mulțimea este vidă;
e) o funcție care determină dacă un element aparține multimii;
f) o funcție care determină suportul mulțimii multiple;
g) o funcție care determină k(e, M), pentru un element *e* specificat;
h) o funcție care determină dacă mulțimea multiplă curentă este inclusă în altă mulțime specificată;
i) o funcție iterator care parcurge elementele mulțimii; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
j) o funcție de afişare pe ecran a mulțimii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, *, pentru operațiile de reuniune, diferență şi intersecție a două mulțimi multiple;
c) operatori de comparare <, >, ==, care compară mulțimile pe baza operației de incluziune;
d) operatorul << pentru introducerea unui element în mulțime; dacă nu mai sunt elemente disponibile în tablou, se va realoca tabloul cu o dimensiune mai mare;

Să se proiecteze clasa mulțime multiplă astfel încât să permită operațiile:

```
MultimeMultipla m1;       // multime vida
MultimeMultipla m2(3, 3, 7);     // multimea {3, 3, 7}
```

```
MultimeMultipla m3 = m1, m4;
m4 = m3 = m2 + m1;
```

23. **O clasă care implementează mulțimi multiple de numere întregi**. O mulțime multiplă reprezintă o extindere a noțiunii de mulțime, la care elementele nu mai trebuie să fie distincte; de exemplu, {1, 1, 2, 3, 3, 3} reprezintă o mulțime multiplă. Numărul de elemente identice *e* dintr-o mulțime multiplă *M* se notează k(e, M). Sunt permise aceleaşi operații ca şi la mulțimile simple. În plus, pentru o mulțime multiplă se poate determina suportul ei, notat cu `supp`, adică mulțimea simplă ce conține numai elementele distincte din mulțimea multiplă. Operația de incluziune se poate extinde asupra mulțimilor multiple:

$M_1 \subseteq M_2$ dacă $supp(M_1) \subseteq supp(M_2)$ şi $\forall e \in supp(M_1)$, $k(e, M_1) \leq k(e,)$.

Pentru două mulțimi multiple, M_1 şi M_2, se poate extinde noțiunea de diferență M_1-M_2, dacă $M_2 \subseteq M_1$.
O mulțime multiplă este specificată printr-o listă liniară a elementelor componente. Se va crea în plus o clasă pentru memorarea elementelor listei.
Funcții membre publice pentru clasa multime multiplă:

a) constructori;
b) destructor;
c) o funcție care determină numărul de elemente din mulțime;
d) o funcție care determină dacă mulțimea este vidă;
e) o funcție care determină dacă un element aparține multimii;
f) o funcție care determină suportul mulțimii multiple;
g) o funcție care determină k(e, M), pentru un element *e* specificat;
h) o funcție care determină dacă mulțimea multiplă curentă este inclusă în altă mulțime specificată;
i) o funcție iterator care parcurge elementele mulțimii; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul element;
j) o funcție de afişare pe ecran a mulțimii.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorii +, -, *, pentru operațiile de reuniune, diferență şi intersecție a două mulțimi multiple;
c) operatori de comparare <, >, ==, care compară mulțimile pe baza operației de incluziune;
d) operatorul << pentru introducerea unui element în mulțime;

Să se proiecteze clasa mulțime multiplă astfel încât să permită operațiile:

```
MultimeMultipla m1;       // multime vida
MultimeMultipla m2(3, 3, 7);     // multimea {3, 3, 7}
MultimeMultipla m3 = m1, m4;
m4 = m3 = m2 + m1;
```

Clasa elementelor mulțimii se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa mulțime multiplă.

24. **O clasă care implementează poligoane cu *n* laturi**. Un poligon este reprezentat prin numărul de laturi şi tabloul coordonatelor vârfurilor, specificate în ordine trigonometrică. Se va crea în plus o clasă pentru reprezentarea vârfurilor poligonului.
Funcții membre publice pentru clasa poligon:

a) constructori;
b) o funcție care determină numărul laturilor poligonului;
c) o funcție care determină aria poligonului;
d) o funcție care determină perimetrul poligonului;
e) o funcție care determină centrul de greutate al poligonului;
f) o funcție care determină dacă poligonul este convex;
g) o funcție iterator care parcurge vârfurile poligonului; iteratorul returnează vârful curent şi va fi circular, în sensul că după ultimul vârf, va fi apoi poziționat pe primul;
h) o funcție de afişare pe ecran a principalelor elemente ale poligonului.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul `[]` pentru accesarea vârfurilor poligonului prin poziția lor;
c) (**) operatorul + pentru operația de concatenare a două poligoane; concatenarea a două poligoane este întotdeauna un poligon convex obținut astfel:
   - se determină reuniunea mulțimii vârfurilor celor două poligoane;
   - se determină înfăşurătoarea convexă a mulțimii de puncte rezultată anterior;
   - mulțimea punctelor înfăşurătorii reprezintă vârfurile noului poligon;
d) operatori de comparare <, >, ==, care compară poligoanele în raport de aria lor;

Să se proiecteze clasa poligon astfel încât să permită operaţiile:

```
Poligon p1(5);            // poligon cu 5 laturi
Poligon p2, p3;           // implicit, un poligon cu 0 laturi şi un varf
Poligon p4 = p1;
Punct p(3, 7);
p1[1] = p;
p3 = p2 = p1;
```

Clasa varfurilor se va proiecta astfel încât să permită efectuarea corectă a operaţiilor din clasa poligon.

25. **O clasă care implementează poligoane cu *n* laturi**. Un poligon este reprezentat prin lista liniară cu coordonatelor vârfurilor, specificate în ordine trigonometrică. Se va crea în plus o clasă pentru reprezentarea elementor din listă.

Funcţii membre publice pentru clasa poligon:

a) constructori;
b) destructor;
c) o funcţie care determină numărul laturilor poligonului;
d) o funcţie care determină aria poligonului;
e) o funcţie care determină perimetrul poligonului;
f) o funcţie care determină centrul de greutate al poligonului;
g) o funcţie care determină dacă poligonul este convex;
h) o funcţie iterator care parcurge vârfurile poligonului; iteratorul returnează vârful curent şi va fi circular, în sensul că după ultimul vârf, va fi apoi poziţionat pe primul;
i) o funcţie de afişare pe ecran a principalelor elemente ale poligonului.

Operatori supraâncărcaţi:

a) operatorul = de atribuire;
b) operatorul `[]` pentru accesarea vârfurilor poligonului prin poziţia lor;
c) (**) operatorul + pentru operaţia de concatenare a două poligoane; concatenarea a două poligoane este întotdeauna un poligon convex obţinut astfel:
   - se determină reuniunea mulţimii vârfurilor celor două poligoane;
   - se determină înfăşurătoarea convexă a mulţimii de puncte rezultată anterior;
   - mulţimea punctelor înfăşurătorii reprezintă vârfurile noului poligon;
d) operatori de comparare <, >, ==, care compară poligoanele în raport de aria lor;

Să se proiecteze clasa poligon astfel încât să permită operaţiile:

```
Poligon p1(5);            // poligon cu 5 laturi
Poligon p2, p3;           // implicit, un poligoane cu 0 laturi şi un varf
Poligon p4 = p1;
Punct p(3, 7);
p1[1] = p;
p3 = p2 = p1;
```

Clasa elementelor din listă se va proiecta astfel încât să permită efectuarea corectă a operaţiilor din clasa poligon.

26. **O clasă care implementează polinoame cu coeficienţi reali**. Un polinom este caracterizat prin gradul său şi tabloul coeficienţilor.

Funcţii membre publice pentru clasa polinom:

a) constructori;
b) o funcţie care citeşte coeficienţii polinomului de la tasatatură;
c) o funcţie care determină gradul polinomului;
d) o funcţie care determină aria polinomului;
e) o funcţie care evaluează polinomul într-un punct dat;
f) o funcţie iterator care parcurge coeficienţii polinomului; iteratorul returnează coeficientul curent şi va fi circular, în sensul că după ultimul coeficient, va fi apoi poziţionat pe primul;
g) o funcţie de afişare pe ecran a polinomului.

Operatori supraâncărcaţi:

a) operatorul = de atribuire;
b) operatorul `[]` pentru accesarea coeficienţilor polinomului;
c) Operatorii +, - pentru adunarea şi scăderea polinoamelor;
d) Operatorul ! de derivare a unui polinom;
e) Operatorul ^ între un polinom şi un întreg: `p^n` determină derivata de ordin *n* a polinomului *p*.

Să se proiecteze clasa poligon astfel încât să permită operaţiile:

```
Polinom p1(5);      // polinom de gradul 5
```

```
p1.Coef();          // se citesc coeficientii polinomului
Polinom p2;         // inițializarile pentru grad si coeficienti
                    // se citesc de la tastatura
Polinom p3 = p2;
p1 = p2 = !p3;
```

27. **O clasă care implementează polinoame cu coeficienți reali**. Un polinom este caracterizat prin gradul său şi lista coeficienților. Clasa polinom memorează coeficienții într-o listă liniară. Se va crea suplimentar o clasă pentru reprezentarea elementelor listei de coeficienți.
Funcții membre publice pentru clasa polinom:

a) constructori;
b) destructor;
c) o funcție care citeşte coeficienții polinomului de la tasatatură;
d) o funcție care determină gradul polinomului;
e) o funcție care determină aria polinomului;
f) o funcție care evaluează polinomul într-un punct dat;
g) o funcție iterator care parcurge coeficienții polinomului; iteratorul returnează coeficientul curent şi va fi circular, în sensul că după ultimul coeficient, va fi apoi poziționat pe primul;
h) o funcție de afişare pe ecran a polinomului.

Operatori supraîncărcați:

a) operatorul = de atribuire;
b) operatorul `[]` pentru accesarea coeficienților polinomului;
c) Operatorii +, - pentru adunarea şi scăderea polinoamelor;
d) Operatorul ! de derivare a unui polinom;
e) Operatorul ^ între un polinom şi un întreg: `p^n` determină derivata de ordin *n* a polinomului *p*.

Să se proiecteze clasa poligon astfel încât să permită operațiile:

```
Polinom p1(5);      // polinom de gradul 5
p1.Coef();          // se citesc coeficientii polinomului
Polinom p2;         // inițializarile pentru grad si coeficienti
                    // se citesc de la tastatura
Polinom p3 = p2;
p1 = p2 = !p3;
```

Clasa elementelor din listă se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa polinom.

28. **O clasă care implementează o tabelă de simboluri**. Tabela de simboluri este reprezentată ca o listă liniară ordonată, ale cărei elemente conțin informațiile următoare:

- şirul de caractere reprezentând un simbol dintr-un program sursă;
- tipul simbolului (identificator, cuvânt cheie, operator, constanta, etc.);
- un tablou de întregi reprezentând liniile din program unde se află simbolul;

Interfața clasei trebuie să conțină funcții membre şi operatori supraîncărcați care să permită operațiile:

- crearea şi distrugerea tabelei de simboluri;
- afişarea tabelei;
- adăugarea unui nou simbol în tabelă (sau o noua linie în tabloul de linii, dacă simbolul există deja);
- căutarea unui simbol în tabelă (după şirul de caractere) şi afişarea liniilor din program unde apare un anumit simbol.

Simbolurile din listă sunt ordonate după şirul de caractere asociat fiecărui simbol. Se va crea şi utiliza suplimentar o altă clasă pentru reprezentarea elementelor din lista asociată tabelei de simboluri.
Să se proiecteze clasa tabelei de simboluri şi clasa simbolurilor, astfel încât să permită operațiile:

```
TabSimb t;          // tabela t are lista de simboluri vida
t.Add(IDENT, "v1", 5);    // se adauga un nou simbol in tabela,
                          // sau doar linia 5
Simbol s;           // un simbol neinitializat
s.SetType(INTCONSTANT);   // se seteaza tipul simbolului: constanta intreaga
s.SetVal(5);              // se seteaza valoarea constantei
s.AddLine(10);            // se adauga o nuoa linie la simbol
t = t + s;          // se adauga simbolul in tabela, sau doar linia 10
Simbol s1(OPER, "<=", 11);// un simbol initializat
t = t + s1;         // se adauga simbolul in tabela, sau doar linia 11
Simbol s2 = s1;
s2 = s2 + 15;       // se adauga o noua linie la simbol
t = t + s2;
t.Print("v1");
```

Clasa simbol se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa tabelei de simboluri.

29. **O clasă care implementează o tabelă asociativă** între nume (şiruri de caractere) şi valori întregi. Unui nume i se pune în corespondență un număr, iar pentru aceasta tabela este reprezentată printr-o listă liniară de elemente, fiecare element fiind format dintr-un şir de caractere şi o valoare întragă. Lista este ordonată după numele elementelor. Se va crea suplimentar o clasă pentru reprezentarea elementelor listei.
Funcții membre publice pentru clasa tabelei:

a) constructori;
b) destructor;
c) o funcție care determină numărul de elemente din tabelă;
d) o funcție care determină numărul asociat unui nume;
e) o funcție iterator care elementele tabelei; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul;
f) o funcție de afişare pe ecran a tabelei.

Operatori supraâncărcați pentru clasa tabelei:

a) operatorul = de atribuire;
b) operatorul + între o tabelă şi un element pentru adăugarea elementului în tabelă; dacă numele există deja, atunci se face suma între valoarea existentă şi noua valoare;
c) operatorul - intre o tabelă si un şi sir de caractere pentru eliminarea unui element din tabelă;
d) operatorul [] de acces în tabelă după nume, care returează valoarea corespunzătoare;
e) operatorul + între două tabele pentru concatenarea acestora.

Să se proiecteze clasa tabelei asociative şi clasa elementelor, astfel încât să permită operațiile:

```
Tabela t;           // tabela fara elemente
Elem e(“v1”, 5);    // element initializat
t = t + e;          // se adauga elementul (“v1”, 5)
Elem e1;            // element neinitializat
e1.SetNume(“v2”);   // initializare nume
e1.SetVal(7);       // initializare valoare
t = t + e1;         // se adauga elementul (“v2”, 7)
e1 = e1 + 4;        // e1 are valoarea 11
t = t + e1;         // nu se adauga un nou element; se modifica elementul
                    // (“v2”, 7) la (“v2”, 11)
e1 = e1 – 2;        // e1 are valoarea 9
t = t + e1;         // se modifica elementul (“v2”, 11) la (“v2”, 9)
e = e1 – 3;         // e are valoarea 6, dar numele ramane tot “v1”
t = t + e;          // se modifica elementul (“v1”, 5) la (“v1”, 6)
Tabela t1 = t;
t1 = t1 - “v2”;     // se elimina elementul (“v2”, 9)
t1[“v1”] = 9;       // se modifica elementul (“v1”, 6) la (“v1”, 9)
```

Clasa elementelor se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa tabelei asociative.

30. **O clasă care implementează o tabelă asociativă** între nume (şiruri de caractere) şi valori întregi. Unui nume i se pune în corespondență un număr, iar pentru aceasta tabela este reprezentată printr-un tablou cu alocare dinamică, fiecare element al tabloului fiind format dintr-un şir de caractere şi o valoare întragă. Se va crea suplimentar o clasă pentru reprezentarea elementelor tabloului.
Funcții membre publice pentru clasa tabelei asociative:

a) constructori;
b) destructor;
c) o funcție care determină numărul de elemente din tabelă;
d) o funcție care determină numărul asociat unui nume;
e) o funcție iterator care elementele tabelei; iteratorul returnează elementul curent şi va fi circular, în sensul că după ultimul element, va fi apoi poziționat pe primul;
f) o funcție de afişare pe ecran a tabelei.

Operatori supraâncărcați pentru clasa tabelei:

a) operatorul = de atribuire;
b) operatorul + între o tabelă şi un element pentru adăugarea elementului în tabelă; dacă numele există deja, atunci se face suma între valoarea existentă şi noua valoare; dacă tabelul nu mai are elemente libere, se va face o realocare cu o dimensiune mai mare;
c) operatorul - intre o tabelă si un şi sir de caractere pentru eliminarea unui element din tabelă;
d) operatorul [] de acces în tabelă după nume, care returează valoarea corespunzătoare;

e) operatorul + între două tabele pentru concatenarea acestora.

Să se proiecteze clasa tabelei asociative și clasa elementelor, astfel încât să permită operațiile:

```
Tabela t0;          // tabela fara elemente
Tabele t(10);       // tabela cu 10 elemente
Elem e(“v1”, 5);    // element initializat
t = t + e;          // se adauga elementul (“v1”, 5)
Elem e1;            // element neinitializat
e1.SetNume(“v2”);   // initializare nume
e1.SetVal(7);       // initializare valoare
t = t + e1;         // se adauga elementul (“v2”, 7)
e1 = e1 + 4;        // e1 are valoarea 11
t = t + e1;         // nu se adauga un nou element; se modifica elementul
                    // (“v2”, 7) la (“v2”, 11)
e1 = e1 – 2;        // e1 are valoarea 9
t = t + e1;         // se modifica elementul (“v2”, 11) la (“v2”, 9)
e = e1 – 3;         // e are valoarea 6, dar numele ramane tot “v1”
t = t + e;          // se modifica elementul (“v1”, 5) la (“v1”, 6)
Tabela t1 = t;
t1 = t1 - “v2”;     // se elimina elementul (“v2”, 9)
t1[“v1”] = 9;       // se modifica elementul (“v1”, 6) la (“v1”, 9)
```

Clasa elementelor se va proiecta astfel încât să permită efectuarea corectă a operațiilor din clasa tabelei asociative.