Feladat

Valósítsuk meg a diagonális mátrixtípust (amelynek négyzetes mátrixai csak a főátlójukban tartalmazhatnak nullától különböző számot)! Ilyenkor elegendő csak a főátló elemeit reprezentálni egy sorozatban. Implementáljuk a mátrix i-edik sorának j-edik elemét megváltoztató illetve azt lekérdező műveletet, valamint az összeadás és szorzás műveleteket! Ne feledkezzünk meg a megfelelő beolvasó és kiíró műveletekről sem!

Diagonális mátrix típus

A feladat lényege egy felhasználói típusnak a diagonális mátrix típusnak a megvalósítása.

Típusérték-halmaz[1]

Olyan számokat (ebben az esetben egész számokat: $\mathbb{Z}$) tartalmazó négyzetes mátrixokkal akarunk dolgozni, amelyek csak a főátlójukban tartalmazhatnak nullától különböző elemeket. E mátrixoknak lényeges paramétere a méretük ($n\in\mathbb{N}$).

Formálisan: $Diag(n) = \{ a\in \mathbb{Z}^{n\times n} \mid \{ n\in\mathbb{N} \wedge \forall i,j\in[1..n]: i\neq j \rightarrow a[i,j]=0 \}$

Típus-műveletek[2]

1. Lekérdezés
A mátrix i-edik sorának j-edik pozícióján ($i,j\in[1..n]$) álló érték kiolvasása: $e:=a[i,j]$.

Formálisan:
$$A = \underset{a}{Diag(n)} \times \underset{i}{\mathbb{Z}} \times \underset{j}{\mathbb{Z}} \times \underset{e}{\mathbb{Z}}$$
$$Q = (a=a' \wedge i=i' \wedge j=j' \wedge i,j\in[1..n])$$
$$R = (Q \wedge e=a[i,j])$$

Egy mátrix elemét visszaadó művelettel kapcsolatban arra érdemes felfigyelni, hogy ez a műveletek csak $i=j$ esetén igényelnek tényleges tevékenységet, hiszen egyébként a mátrix eleme nulla.

2. Felülírás
A mátrix i-edik sorának j-edik pozíciójára ($i,j\in[1..n]$) új érték beírása: $a[i,j]:=e$. A főátlón kívüli elemeket nem szabad felülírni, azaz $i=j$.

Formálisan:
$$A = \underset{a}{Diag(n)} \times \underset{i}{\mathbb{Z}} \times \underset{j}{\mathbb{Z}} \times \underset{e}{\mathbb{Z}}$$
$$Q = (e=e' \wedge a=a' \wedge i=i' \wedge j=j' \wedge i,j\in[1..n] \wedge i=j)$$
$$R = (e=e' \wedge i=i' \wedge j=j' \wedge a[i,j]=e \wedge \forall k,l\in[1..n]: (k\neq i \vee l\neq j)\rightarrow a[k,l]=a'[k,l])$$

Egy mátrix elemét megváltoztató művelettel kapcsolatban arra érdemes felfigyelni, hogy ez a műveletek csak $i=j$ esetén igényelnek tényleges tevékenységet, hiszen egyébként a mátrix eleme nulla.

---

[1] A típusérték-halmazt kétféleképpen is le lehet írni: szövegesen és formálisan. Elég csak az egyik formát használni.
[2] A típusműveletek leírására is kétféle definíciót használok: egy informálist és egy formálist. Elég csak az egyik formát használni.

3. Összeadás
Két mátrix összeadása: $c:=a+b$. Az összeadásban szereplő mátrixok azonos méretűek.

Formálisan:
$$A = \underset{a}{Diag(n)} \times \underset{b}{Diag(n)} \times \underset{c}{Diag(n)}$$
$$Q = (a=a' \wedge b=b')$$
$$R = (Q \wedge \forall i,j \in [1..n]: c[i,j]= a[i,j] + b[i,j])$$

Az implementáció szempontjából hasznos már itt megjegyezni azt, hogy diagonális mátrixok esetén a fenti művelet jóval egyszerűbben is megfogalmazható: $\forall i \in [1..n]$: $c[i,i]= a[i,i] + b[i,i]$ és $\forall i,j \in [1..n]$: $i \neq j \rightarrow c[i,j]=0$.

4. Szorzás
Két mátrix összeadása: $c:=a*b$. Az összeadásban szereplő mátrixok azonos méretűek.

Formálisan:
$$A = \underset{a}{Diag(n)} \times \underset{b}{Diag(n)} \times \underset{c}{Diag(n)}$$
$$Q = (a=a' \wedge b=b')$$
$$R = (Q \wedge \forall i,j \in [1..n]: c[i,j]= \Sigma_{k=1..n}\, a[i,k] * b[k,j])$$

Az implementáció szempontjából hasznos már itt megjegyezni azt, hogy diagonális mátrixok esetén a fenti művelet jóval egyszerűbben is megfogalmazható: $\forall i \in [1..n]$: $c[i,i]= a[i,i]*b[i,i]$ és $\forall i,j \in [1..n]$: $i \neq j \rightarrow c[i,j]=0$.

Reprezentáció[3]

Az $n\times n$-es diagonális mátrixnak csak a főátlóját kell ábrázolni.

$$\begin{matrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ 0 & 0 & a_{33} & \dots & 0 \\ 0 & 0 & 0 & \dots & a_{nn} \end{matrix} \qquad \leftrightarrow \qquad a_{11} \quad a_{22} \quad a_{33} \quad a_{nn}$$

Ehhez egy 0-tól $n-1$-ig indexelt egydimenziós tömböt (v) használhatunk. Ezt a tömböt felhasználva a diagonális mátrix bármelyik elemét meghatározhatjuk az alábbi képlet alapján:

$$a[i,j] = \begin{cases} v[i-1] & ha \quad i = j \\ 0 & ha \quad i \neq j \end{cases}$$

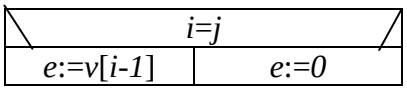
Formálisan, ha v tömbbel ábrázolt a mátrixot $\rho(v)$-vel jelölhetjük, ahol $\rho : \mathbb{Z}^n \rightarrow Diag(n)$ és $\rho(v) \in \mathbb{Z}^{n\times n}$ és $\forall i \in [1..n]$: $\rho(v) \in [i,i]=v[i-1]$ és $\forall i,j \in [1..n]$: $i \neq j \rightarrow \rho(v) \in [i,j]=0$

[3] A reprezentációt többféleképpen is le lehet írni: szövegesen és formálisan. Elég csak az egyik formát használni.

Implementáció[4]

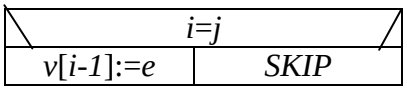
1. Lekérdezés

A *v* tömbbel ábrázolt *a* mátrix *i*-edik sorának *j*-edik elemét visszaadó *e*:=*a*[*i*,*j*] értékadás programmal implementálható feltéve, hogy $1 \le i \le n$, ahol *n* a mátrix mérete:



2. Felülírás

A *v* tömbbel ábrázolt *a* mátrix *i*-edik sorának *j*-edik elemét megváltoztató *a*[*i*,*j*]:=*e* értékadás az alábbi programmal implementálható feltéve, hogy $1 \le i \le n$, ahol *n* a mátrix mérete.



3. Összeadás

A *v* tömbbel ábrázolt *a* mátrix és a *t* tömbbel ábrázolt *b* mátrix összege az *u* tömbbel ábrázolt *c* mátrixba kerül, ha az alábbi programot elkészítjük. A végrehajtás előtt ellenőrizni kell, hogy mindhárom mátrix, pontosabban az őket reprezentáló tömb azonos méretű-e.

$$\forall i \in [0..n-1]: u[i] := v[i] + t[i]$$

4. Szorzás

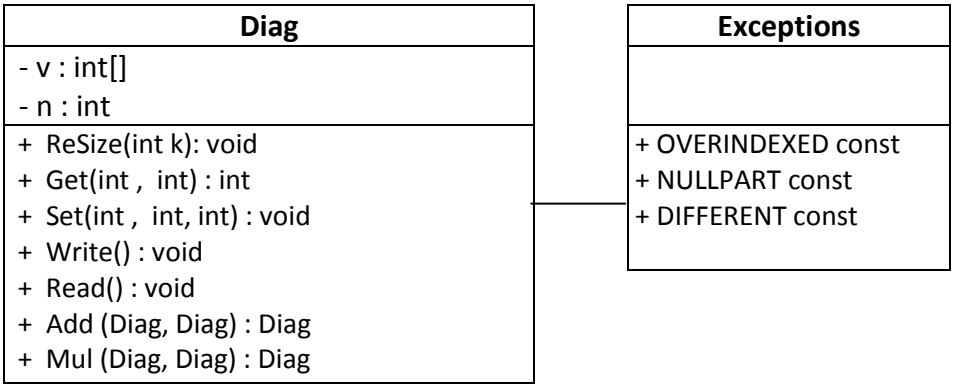
A *v* tömbbel ábrázolt *a* mátrix és a *t* tömbbel ábrázolt *b* mátrix szorzata az *u* tömbbel ábrázolt *c* mátrixba kerül, ha az alábbi programot elkészítjük. A végrehajtás előtt ellenőrizni kell, hogy mindhárom mátrix, pontosabban az őket reprezentáló tömb azonos méretű-e.

$$\forall i \in [0..n-1]: u[i] := v[i] * t[i]$$

[4] A műveletek implementálásánál az a fontos, hogy egy programot adjunk meg, amit azonban nemcsak struktogrammal lehet leírni.

Osztály[5]

A diagonális mátrixok típusát egy osztály segítségével valósítjuk meg.



Az osztály privát részében definiáljuk azt a tömböt (ez majd egy dinamikus helyfoglalású tömb lesz a C++ nyelvben), amely a főátlóbeli elemek tárolására szolgál, valamint ezen elemek számát, azaz a mátrix dimenzióját.

Az osztály publikus metódusai nemcsak a specifikációban körvonalazott műveletek (egy elem lekérdezése, egy elem felülírása, mátrix kiírása, mátrix beolvasása, két mátrix összeadása és szorzása) lesznek. A műveleteket a C++ nyelvben operátor felüldefiniálással implementáljuk. A lekérdezés és a felülírás műveletei nem a teljes értékadást, hanem csak az értékadás jobb illetve baloldalán megjelenő (mátrix adott elemére történő) hivatkozást adják meg. Az összeadás, a szorzás, a kiírás és a beolvasás műveleteket külső barát függvényként valósítjuk meg.

A hibakezelésre három kivételt definiálunk. Az `OVERINDEXED` a helytelenül megadott sor és oszlopindexek esetén váltódik ki a mátrix elemeit lekérdező és felülíró műveletekben. A `NULLPART` kivétel a főátlón kívüli elemek felülírásakor aktivizálódik. A `DIFFERENT` kivétel a különböző méretű mátrixok esetén váltódik ki az értékadás, az összeadás és a szorzás műveletekben.

A teljes osztály-definíciót a `diag.h` fejállományban helyezzük el.

[5] Az itt következő leírásból elég csak az osztálydiagramot megadni.

Tesztelési terv

Megvalósított műveletek tesztelése (fekete doboz tesztelés)

1) Különféle méretű mátrixok létrehozása, feltöltése és kiírása.
   a) 0, 1, 2, 5 dimenziójú mátrix
   b) Az *a*+*b* illetve *a***b* kifejezés kiíratása
2) Mátrix adott pozíciójú értékének lekérdezése és megváltoztatása. (hibás pozíció megadása)
   a) Diagonálisra eső elem lekérdezése és megváltoztatása
   b) Diagonálison kívüli elem lekérdezése és megváltoztatása
   c) Illegális index megadása, 0 dimenziós mátrix indexelése
3) Új mátrix létrehozása meglevő mátrix alapján, majd kiírása.
   a) új mátrix létrehozása, majd az új és a régi megváltoztatása, mindkettő kiírása.
4) Mátrix-értékadás kipróbálása (különböző méretű mátrixokra is).
   a) a = a
   b) a = b = c (azonos méretű mátrixokra)
   c) a = b (azonos és eltérő méretű mátrixokra)
5) A *c*:=*a*+*b* mátrixösszeadás kipróbálása.
   a) Eltérő méretű mátrixokkal (az *a* és *b* mérete különbözik, a *c* és *a* mérete különbözik)
   b) Kommutatívitás ellenőrzése (a + b == b + a)
   c) Asszociatívitás ellenőrzése (a + b + c == (a + b) + c == a + (b + c))
   d) Null elem vizsgálata (a + 0 == a, ahol 0 a null mátrix)
6) A *c*:=*a***b* mátrixszorzás kipróbálása.
   a) Eltérő méretű mátrixokkal. (az *a* és *b* mérete különbözik, a *c* és *a* mérete különbözik)
   b) Kommutatívitás ellenőrzése (a * b == b * a)
   c) Asszociatívitás ellenőrzése (a * b * c == (a * b) * c == a * (b * c))
   d) Null elem vizsgálata (a * 0 == 0, ahol 0 a null mátrix)
   e) Egység elem vizsgálata (a * 1 == a, ahol 1 az egység mátrix)

Tesztesetek a kód alapján (fehér doboz tesztelés)

1. Extrém méretű (-1, 0, 1, 1000) mátrix létrehozása.
2. Kivételek generálása és elkapása.

Melléklet: C++ kód[6]

```
#ifndef DIAG_H
#define DIAG_H
#include <iostream>

class Diag {
  public:
      enum Exceptions{OVERINDEXED,NULLPART, DIFFERENT};

       Diag(){v = NULL; n = 0;}
       ~Diag(){if (v!=NULL) delete[] v;}
      Diag(int k);
      Diag(const Diag& a);
      Diag& operator=(const Diag& a);

      void Size(int k);

      int  operator()(int i, int j) const;
      int& operator()(int i, int j);

      friend Diag operator+ (const Diag& a, const Diag& b);
      friend Diag operator* (const Diag& a, const Diag& b);
      friend std::istream& operator>>
                            (std::istream& s, Diag& a);
      friend std::ostream& operator<<
                            (std::ostream& s, const Diag& a);
  private:
      int* v;
      int  n;
};

#endif
```

A metódusok megvalósítása a diag.cpp forrásállományba kerül. Ennek elején helyezzük el a `#include "diad.h"`, `#include <iostream>`, `#include <iomanip>` direktívákat, valamint a **`using namespace`** `std;` utasítást.

1. Konstruktor

Tevékenység: A konstruktor létrehoz egy *n*×*n*-es diagonális mátrixot, azaz lefoglalja annak főátlóbeli elemeit tartalmazó *n* hosszú tömböt, és feltölti csupa nulla értékkel.
Bemenő adatok: mátrix mérete (`k:int`)
Kimenő adatok: új null mátrix (`Diag`)
Definíció:

```
Diag::Diag(int k)
{
    n = k;
    v = new int[n];
    for(int i=0; i<n; ++i) v[i] = 0;
}
```

[6] A jól kommentezett programkódot nem kell a dokumentációhoz csatolni.

2. Másoló konstruktor

Tevékenység: A konstruktor létrehoz egy *n*×*n*-es diagonális mátrixot, azaz automatikusan lefoglalja annak főátlóbeli elemeit tartalmazó *n* hosszú tömböt, és átmásolja a megadott `a` mátrixbeli értékkel.
Bemenő adatok: egy mátrix (`a:Diag`)
Kimenő adatok: új mátrix (`Diag`)
Definíció:

```
Diag::Diag(const Diag& a)
{
    n = a.n;
    v = new int[n];
    for(int i=0; i<n; ++i)  v[i] = a.v[i];
}
```

3. Értékadás operátor

Tevékenység: Az operátor értékül adja az értékadás jobboldalán adott mátrixot a baloldalán álló (az operátor által alapértelmezett) mátrixnak. Ha a két mátrix nem ugyanaz és azonos méretűek, akkor az alapértelmezett mátrix főátlóját ábrázoló tömbbe másolja az adott mátrix főátlóját ábrázoló tömb értékeit. Eltérő méretű mátrixok esetén `DIFFERENT` kivételt dob.
Bemenő adatok: mátrix (`a:Diag`)
Kimenő adatok: alapértelmezett mátrix (`Diag`)

Definíció:

```
Diag& Diag::operator=(const Diag& a)
{
    if(n!=a.n) throw DIFFERENT;

    if(this==&a) return *this;
    for(int i=0; i<n; ++i) v[i] = a.v[i];
    return *this;
}
```

4. Méret megadás

Tevékenység: Egy üres konstruktorral létrehozott diagonális mátrixot *n*×*n*-essé alakít, azaz lefoglalja annak főátlóbeli elemeit tartalmazó *n* hosszú tömböt, és feltölti csupa nulla értékkel.
Bemenő adatok: mátrix mérete (`k:int`)
Kimenő adatok: új null mátrix (`Diag`)
Definíció:

```
void Diag::Size(int k)
{
    if (n==k) return;
    if(v!=NULL) delete[] v;
    if( n!=k && k>0) v = new int[k];
    n = k;
    for(int i=0; i<n; ++i) v[i] = 0;
}
```

5. Lekérdezése egy elemnek

Tevékenység: Az operátor segítségével az alapértelmezett mátrix *i*-edik sorának j-edik elemének értékét kapjuk vissza.
Bemenő adatok: az alapértelmezett mátrix (`Diag`)
sor és oszlop indexek (`i,j:int`)
Kimenő adatok: a mátrix egy elemének értéke
Definíció:

```
int Diag::operator()(int i, int j) const
{
    if(i>=n || i<0 || j>=n || j<0 ) throw OVERINDEXED;
    if(i==j) return v[i];
    else return 0;
}
```

6. Hivatkozás egy elemre

Tevékenység: Az operátor segítségével az alapértelmezett mátrix *i*-edik sorának j-edik elemére történő hivatkozást kapjuk vissza, amennyiben az a főátlóra esik.
Bemenő adatok: az alapértelmezett mátrix (`Diag`)
sor és oszlop indexek (`i,j:int`)
Kimenő adatok: hivatkozás az alapértelmezett mátrix egy elemére
Definíció:

```
int& Diag::operator()(int i, int j)
{
    if(i>=n || i<0 || j>=n || j<0 ) throw OVERINDEXED;
    if(i==j) return v[i];
    else throw NULLPART;
}
```

7. Összeadás

Tevékenység: A barát operátor létrehoz egy, a bemenő mátrixok méretével azonos, eredmény mátrixot (`c`), amelynek főátlóját ábrázoló tömböt a két adott mátrixot ábrázoló tömbök összeadásával tölt ki. `DIFFERENT` kivételt dob, ha a bemenő mátrixok eltérő méretűek.
Bemenő adatok: két mátrix (`a,b:Diag`)
Kimenő adatok: eredmény mátrix (`Diag`)
Definíció:

```
Diag operator+(const Diag& a ,const Diag& b)
{
    if(a.n!=b.n) throw Diag::DIFFERENT;
    Diag c(a.n);
    for(int i=0; i<c.n; ++i)  c.v[i]=a.v[i]+b.v[i];
    return c;
}
```

8. Szorzás

Tevékenység: A barát operátor létrehoz egy, a bemenő mátrixok méretével azonos, eredmény mátrixot (`c`), amelynek főátlóját ábrázoló tömböt a két adott

mátrixot ábrázoló tömbök elemenkénti összeszorzásával tölt ki. `DIFFERENT` kivételt dob, ha a bemenő mátrixok eltérő méretűek.

Bemenő adatok: két mátrix (`a,b:Diag`)
Kimenő adatok: eredmény mátrix (`Diag`)
Definíció:

```
Diag operator*(const Diag& a ,const Diag& b)
{
    if(a.n!=b.n) throw Diag::DIFFERENT;
    Diag c(a.n);
    for(int i=0; i<c.n; ++i) c.v[i]=a.v[i]*b.v[i];
    return c;
}
```

9. Beolvasás

Tevékenység: Az operátor a standard bemenetről bekéri az adott diagonális mátrix főátlójának elemeit.
Be/kimenő adat: standard bemeneti folyam (`istream`)
Bemenő adat: mátrix (`a:Diag`)
Definíció:

```
istream& operator>>(istream& s, Diag& a)
{
    for(int i=0; i<a.n; ++i) {
        cout << "[" << i << "," << i << "]=";
        s >> a(i,i);
    }
    return s;
}
```

10. Kiírás

Tevékenység: Az operátor a standard kimenetre írja az adott mátrix összes elemét.
Be/kimenő adat: standard bemeneti folyam (`ostream`)
Bemenő adat: mátrix (`a:Diag`)
Definíció:

```
ostream& operator<<(ostream& s, const Diag& a)
{
    for(int i=0; i<a.n; ++i){
        for(int j=0; j<a.n; ++j)
            s << setw(5) << a(i,j);
        s << endl;
    }
    return s;
}
```

Tesztkörnyezet

A fekete doboz teszteseteket egy részét lefedő C++ kód: `main.cpp`

```
#include "diag.h"
#include <iostream>
using namespace std;

class Menu{
  public:
    Menu(){a.Size(3);}
    void Run();

  private:
    Diag a;

    void MenuWrite();
    void Get();
    void Set();
    void Sum();
    void Mul();
};

int main()
{
    Menu m;
    m.Run();
    return 0;
}

void Menu::Run()
{
    int c = 0;
    do{
        MenuWrite();
        cin >> c;
        switch(c){
           case 1: Get(); break;
           case 2: Set(); break;
           case 3: cin >> a ; break;
           case 4: cout << a; break;
           case 5: Sum(); break;
           case 6: Mul(); break;
        }
    }while(c!=0);
}

void Menu::MenuWrite()
{
     cout << endl << endl;
     cout << " 1. -Matrix feltoltese" << endl;
     cout << " 2. -Matrix adott elemenek lekerdezese" << endl;
     cout << " 3. -Matrix adott elemenek felulirasa" << endl;
     cout << " 4. -Matrix kiirasa" << endl;
```

```
     cout << " 5. -Matrixhoz egy masik matrix hozzaadasa" << endl;
     cout << " 6. -Matrix szorzasa egy masik matrixszal" << endl;
     cout << " 0. -Kilepes" << endl;
}

void Menu::Get()
{
    int i,j;
    cout << "Adja meg a sor indexet:"; cin >> i;
    cout << "Adja meg az oszlop indexet:"; cin >> j;
    cout << "a[" << i << "," << j << "]= "
    try{
       cout << a(i,j);
    }catch(Diag::Exceptions ex){
        switch (ex){
           case Diag::OVERINDEXED :
              cout << "Túlindexelés!" << endl; break;
           case Diag::NULLPART :
              cout << 0 << endl; break;

        }
    }
}

void Menu::Set()
{
    int i,j;
    cout << "Adja meg a sor indexet:"; cin >> i;
    cout << "Adja meg az oszlop indexet:"; cin >> j;
    try{
        cout << "Adja meg az értéket:"; cin >> a(i,j);
    }catch(Diag::Exceptions ex){
        switch (ex){
           case Diag::OVERINDEXED :
              cout << "Túlindexelés!" << endl; break;
           case Diag::NULLPART :
              cout << "Diagonálison kívül eső elem nem írható!\n";
              break;
    }
}

void Menu::Sum()
{
     cout << "Második mátrix:" << endl;
     Diag b(3);
     cin >> b;
     cout << "a+b=" << endl << a+b << endl;
}

void Menu::Mul()
{
     cout << "Második mátrix:" << endl;
     Diag b(3);
     cin >> b;
     cout << "a*b=" << endl << a*b << endl;
}
```