

Objektum elvű alkalmazások fejlesztése 1. beadandó

Berezvai Dániel <http://elte.3ice.hu/>

Feladat

```
//A típust egy osztály segítségével valósítsa meg.
//Az összes megvalósítandó típus, azonos típusú elemek összetett
adatszerkezetű
//gyűjteményét írja le, amelyet dinamikusan lefoglalt tömbben kell
elhelyezni,
//ezért az osztályban meg kell valósítani a másoló konstruktort és
az értékadás
//operátort is. Ahol a feladat szövege nem definiálja, az elemi
típus az egész
//számok típusa. (Ne alkalmazzon sablonokat!)

//Egy osztály szolgáltatásainak (összes metódusának) bemutatásához
olyan
//főprogramot kell készíteni, amelyik egy menü segítségével teszi
lehetővé a
//metódusok tetszőleges sorrendben történő kipróbálását. A főprogram
//példányosítson egy objektumot, amelyre a menüpontok közvetítésével
lehessen
//meghívni az egyes metódusokat. Természetesen szükség lehet minden
tevékenység
//után az objektum állapotának kiírására vagy egy az objektum
állapotát kiíró
//külön menüpontra. Ha vannak olyan metódusok (esetleg barát
függvények),
//amelyek több objektum közötti műveleteket valósítanak meg, a
főprogram több
//objektum létrehozására és azok állapotának kiírására is adjon
lehetőséget.

//3ICE: I chose the second task.

//(2) Valósítsa meg az egész számokat tartalmazó felsőháromszög
mátrixtípust
//(a mátrixok a főátlójuk alatt csak nullát tartalmaznak)! Ilyenkor
elegendő
//csak a főátló és afeletti elemeket reprezentálni egy sorozatban,
amelyet egy
//dinamikus helyfoglalású tömbben helyezünk el. Implementálja
önálló
//metódusként a mátrix i-edik sorának j-edik elemét visszaadó
műveletet,
//valamint hatékony összeadás és szorzás műveleteket, továbbá a
mátrix
//(négyzetes alakú) kiírását, és végül a másoló konstruktort és az
//értékadás operátort!
```

Egész számokat tartalmazó felsőháromszög mátrix típus

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

Típusérték-halmaz

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

Típus-műveletek

1. Lekérdezés

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

Egy mátrix elemét visszaadó művelettel kapcsolatban arra érdemes figyelni, 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 aluli elemeket nem szabad felülírni, azaz $i \geq j$.

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

3. Összeadás

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

4. Szorzás

Két mátrix összeszorozása: $c := a * b$. A szorzásban szereplő mátrixok azonos méretűek.

Reprezentáció

Az $n \times n$ -es diagonális felsőháromszög mátrixnak csak a főátlóját és felső háromszögét kell ábrázolni.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ 0 & 0 & \ddots & \\ 0 & 0 & 0 & a_{ii} & \dots \\ 0 & 0 & 0 & 0 & \ddots \\ 0 & 0 & 0 & 0 & 0 & a_{nn} \end{bmatrix}$$

Ehhez egy 0-tól k -ig indexelt egydimenziós tömböt (v) használhatunk, ahol:

$$k = n * \frac{n + 1}{2} - 1$$

Ezt a tömböt felhasználva a mátrix bármelyik nemnulla elemét meghatározhatjuk az alábbi logika alapján:

$$[1 \ 2 \ 3 \ 4 \ 5 \ 6] \mapsto \begin{bmatrix} 1 & 2 & 3 \\ & 4 & 5 \\ & & 6 \end{bmatrix}$$

Implementáció

1. Lekérdezés

A v tömbbel ábrázolt a mátrix i -edik sorának j -edik elemét visszaadó $e := a[i, j]$ értékadás programmal implementálható.

2. Felülírás

A v tömbbel ábrázolt a mátrix i -edik sorának j -edik elemét megváltoztató $a[i, j] := e$ értékadás programmal implementálható.

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.

4. Szorzás

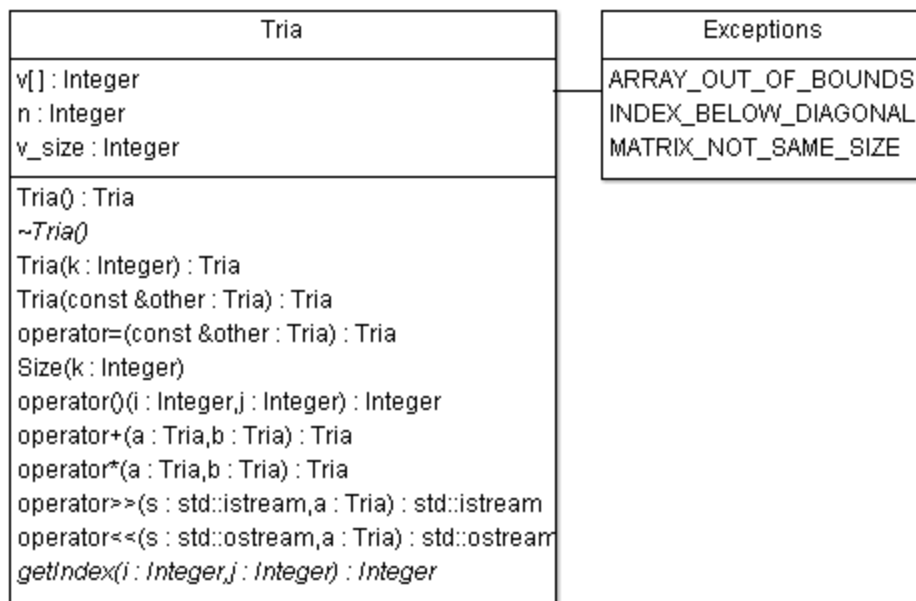
A v tömbbel ábrázolt a mátrix és a t tömbbel ábrázolt b mátrix szorzata az u tömbbel ábrázolt c mátrixba kerül, ha az alábbi programot elkészítjük. A végrehajtás előtt ellenőrizni kell, hogy mindhárom mátrix, pontosabban az őket reprezentáló tömb azonos méretű-e.

→ Tria.cpp

Osztály

A diagonális felsőháromszög mátrixok típusát egy osztály segítségével valósítjuk meg.

UML osztálydiagram



→ UpperTriangularMatrix.zargo

→ UpperTriangularMatrix_ClassDiagram.png

Megjegyzések

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

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 `ARRAY_OUT_OF_BOUNDS` a helytelenül megadott sor és oszlopindexek esetén váltódik ki a mátrix elemeit lekérdező és felülíró műveletekben. Az `INDEX_BELOW_DIAGONAL` kivétel a főátlón aluli elemek felülírásakor aktivizálódik. A `MATRIX_NOT_SAME_SIZE` kivétel a különböző méretű mátrixok esetén váltódik ki az értékadás, az összeadás és a szorzás műveletekben.

A teljes osztály-definíciót a diag.h fejlécfájlban helyezzük el.

→ Tria.h

Tesztelési terv

```
?: Print this help message.  
r: Reset matrices (3 x 3 with values 11 12 13 22 23 33)  
o: Output Matrices  
+: Matrix Add  
*: Matrix Multiply  
a: Custom Matrix a (Example usage: a 3 11 12 13 22 23 33)  
b: Custom Matrix b (E.g.: b 1 1 1)  
g: Get Matrix A element (Example: g 1 1)  
s: Set Matrix A element (Usage: s 3 3 100)  
G: Get Matrix B element (G 1 2 -1)  
S: Set Matrix B element (S 1 1 1)  
x: Swap Matrix A with B  
0: Exit
```

Megvalósított műveletek tesztelése (fekete doboz tesztelés)

- 1) Különböző méretű mátrixok létrehozása, feltöltése és kiírása.
 - a) 0, 1, 2, 5 dimenziójú mátrix
 - b) Az $a+b$ illetve $a*b$ kifejezés kiírása
- 2) Mátrix adott pozíciójú értékének lekérdezése és megváltoztatása. (hibás pozíció megadása)
 - a) Diagonálisra eső elem lekérdezése és megváltoztatása
 - b) Sarokba eső elem lekérdezése és megváltoztatása
 - c) Diagonális alatti elem lekérdezése és megváltoztatása
 - d) Illegális index megadása, 0 dimenziós mátrix indexelése
- 3) Új mátrix létrehozása meglévő mátrix alapján, majd kiírása.
 - a. új mátrix létrehozása, majd az új és a régi megváltoztatása, mindkettő kiírása.
- 4) Mátrix-értékkadás 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) Kommutativitás ellenőrzése ($a + b == b + a$)
 - c) Null elem vizsgálata ($a + 0 == a$, ahol 0 a null mátrix)

- 6) A $c := a * b$ mátrixszorzás kipróbálása.
 - a) Kommutativitás ellenőrzése ($a * b == b * a$)
 - b) Asszociativitás ellenőrzése ($a * b * c == (a * b) * c == a * (b * c)$)
 - c) Null elem vizsgálata ($a * 0 == 0$, ahol 0 a null mátrix)
 - d) 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

A projekt háromféleképpen is működik:

- 1) Visual Studio 2010 (→UpperTriangularMatrix.sln)
- 2) CodeBlocks 12 (→UpperTriangularMatrix.cbp)
- 3) Parancssor Linux (`g++ -o UpperTriangularMatrix Tria.cpp main.cpp`)
vagy Windows alatt
(`gcc -o UpperTriangularMatrix Tria.cpp main.cpp -lstdc++ -enable-auto-import`)

Tria.h

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

class Tria {
public:
    //3ICE: These gave me a lot of trouble.
    //VS likes to crash with unhelpful error 0x... codes instead of saying
    ARRAY_OUT_OF_BOUNDS
    enum Exceptions {ARRAY_OUT_OF_BOUNDS, INDEX_BELOW_DIAGONAL,
MATRIX_NOT_SAME_SIZE};
    //3ICE: ctor
    Tria();
    //3ICE: dtor
    virtual ~Tria();
    //3ICE: ctor + size
    Tria ( int k );
    //3ICE: copy ctor
    Tria ( const Tria& other );
    //3ICE: a=b, b=a both work fine
    Tria& operator= ( const Tria& other );
    //3ICE: resize
    void Size ( int k );
    //3ICE: Index
    int operator() ( int i, int j ) const;
    //3ICE: Index with write access
    int& operator() ( int i, int j );
```

```

//3ICE: MAtrix add
friend Tria operator+ ( const Tria& a, const Tria& b );
//3ICE: Matri mul
friend Tria operator* ( const Tria& a, const Tria& b );
//3ICE: read whole matrix in one go
friend std::istream& operator>> ( std::istream& s, Tria& a );
//3ICE: print entire matrix
friend std::ostream& operator<< ( std::ostream& s, const Tria& a );
private:
//3ICE: Compress-decompress 2D matrix with lots of zeros into 1D array
with no 0s.
int getIndex ( int i, int j ) const;

//3ICE: Efficient compressed array, not storing any 0s.
int* v;

//3ICE: Size n×n
int n;

//3ICE: Compressed size of v (Almost 2x efficiency!)
int v_size;
};
#endif // TRIA_H

```

Tria.cpp

```

#include "Tria.h"
#include <iomanip>

using namespace std;
Tria::Tria() {
    v = new int[0];
    v_size = 0;
    n = 0;
}
Tria::Tria ( int k ) {
    n = k;
    v_size = k * ( k + 1 ) / 2;
    v = new int[v_size];

    for ( int i = 0; i < v_size; ++i ) { v[i] = 0; }
}

Tria::Tria ( const Tria& other ) {
    n = other.n;
    v_size = other.v_size;
    v = new int[v_size];

    for ( int i = 0; i < v_size; ++i ) { v[i] = other.v[i]; }
}

//3ICE: What does rhs mean? "right-hand-side"
Tria& Tria::operator= ( const Tria& rhs ) {
    if ( n != rhs.n ) { throw MATRIX_NOT_SAME_SIZE; }

    if ( this == &rhs ) { return *this; } // handle self assignment

    for ( int i = 0; i < v_size; ++i ) { v[i] = rhs.v[i]; }

    return *this;
}

```

```

Tria::~~Tria() {
    if ( v != NULL ) {
        delete[] v;
    }
}

//3ICE: I don't know how else I'm supposed to get a usable variable.
//Constructor returns a pointer but Size returns an object?

//Tria a; a.Size ( 3 ); //good

//Tria* a = new Tria ( 3 ); //Bad

//3ICE: Downright ugly:
//x->operator() ( 1, 1 ) = 1;
//cout << x->operator() ( 1, 1 ) << endl;

void Tria::Size ( int k ) {
    //if ( n == k ) { return; }
    if ( v != NULL ) { delete[] v; }

    n = k;
    v_size = k * ( k + 1 ) / 2;
    v = new int[v_size];

    for ( int i = 0; i < v_size; ++i ) { v[i] = 0; }
}

//3ICE: I must verify that it works... Example 3x3:
//| 1 1 | 1 2 | 1 3 |
//| 2 1 | 2 2 | 2 3 |
//| 3 1 | 3 2 | 3 3 |
//
//3ICE: Indices:
//0 1 2
// 3 4
//   5
//
//3ICE: Tests:
//offset=0 step=3
//
//i=1 j=1
//k=1 0+1-1=1
//
//
//i=1 j=2
//k=1 0+2-1=2
//
//i=1 j=3 fenti alapján adott
//
//i=2 j=1 i>j ==>0
//
//i=2 j=2
//k=1
//offset=3, step=2
//3+2-2=3
//
//i=2 j=3
//k=1
//offset=3, step=2

```



```

//3+3-2=4
//
//
//i=3 j=3
//k=1
//offset=3, step=2
//k=2
//offset=5, step=1
//
//5+3-3=5
int Tria::getIndex ( int i, int j ) const {
    int offset = 0;
    int offset_step = n;

    for ( int k = 1; k < i; ++k ) {
        offset += offset_step--;
    }

    return offset + j - i;
}

//3ICE: These two operator() functions behave differently.
int Tria::operator() ( int i, int j ) const {
    if ( i > n || i < 1 || j > n || j < 1 ) { throw ARRAY_OUT_OF_BOUNDS; }

    //3ICE: Read access is allowed...
    //| 1 1 | 1 2 | 1 3 |
    //| 0  | 2 2 | 2 3 |
    //| 0  | 0  | 3 3 |
    if ( i > j ) { return 0; }

    return v[getIndex ( i, j )];
}

//3ICE: These two operator() functions behave differently.
int& Tria::operator() ( int i, int j ) {
    if ( i > n || i < 1 || j > n || j < 1 ) { throw ARRAY_OUT_OF_BOUNDS; }

    //3ICE: Write access is not allowed...
    //| 1 1 | 1 2 | 1 3 |
    //| ERR | 2 2 | 2 3 |
    //| ERR | ERR | 3 3 |
    //3ICE: Can't return 0 or null, the user would try to edit it. Must throw
    error.
    if ( i > j ) {
        cout << "int& Tria::operator() ( int i, int j ) called with " << i <<
        ", " << j
            << "!" << endl << "Throwing INDEX_BELOW_DIAGONAL..." << endl;
        throw INDEX_BELOW_DIAGONAL;
    }

    return v[getIndex ( i, j )];
}

Tria operator+ ( const Tria& a, const Tria& b ) {
    if ( a.n != b.n ) { throw Tria::MATRIX_NOT_SAME_SIZE; }

    Tria c ( a.n );

    for ( int i = 0; i < c.v_size; ++i ) { c.v[i] = a.v[i] + b.v[i]; }
}

```

```

    return c;
}

//3ICE: Steal with caution, this could be heavily optimized!
Tria operator* ( const Tria& a, const Tria& b ) {
    if ( a.n != b.n ) { throw Tria::MATRIX_NOT_SAME_SIZE; }

    Tria c ( a.n );

    //3ICE: Storing the matrix in a compressed vector, only to uncompress it
    by
    //using operator() inside the for loop... Dreadful! But it works and it's
    //already way past midnight.

    for ( int i = 1; i <= c.n; ++i ) {
        c ( i, i ) = a ( i, i ) * b ( i, i );
    }

    //3ICE: Full matrix multiplication would look like this:
    //for ( int i = 0; i < a.n; i++ ) {
    //    for ( int j = 0; j < a.n; j++ ) {
    //        for ( int k = 0; k < a.n; k++ ) {
    //            c ( i, j ) = c ( i, j ) + a ( i, k ) * b ( k, j );
    //        }
    //    }
    //}
    //}
    //}
    //3ICE: And the proper result for example given in main.cpp would be:
    // 11  70 171
    //  0  88 248
    //  0   0 198
    return c;
}

istream& operator>> ( istream& s, Tria& a ) {
    for ( int i = 1; i <= a.n; ++i )
        for ( int j = 1; j <= a.n; ++j )
            if ( i <= j ) {
                cout << "[" << i << ", " << j << "]" = "
                s >> a ( i, j );
            }

    return s;
}

ostream& operator<< ( ostream& s, const Tria& a ) {
    for ( int i = 1; i <= a.n; ++i ) {
        for ( int j = 1; j <= a.n; ++j ) {
            s << setw ( 5 ) << a ( i, j );
        }

        s << endl;
    }

    return s;
}

//Full disclosure: I helped someone with programming.
//(Compiler error messages and warnings, syntax errors,
//hard to grasp keywords like virtual, const, &, *, etc.)
//In exchange, he helped me with the mathy bits.

```

```
//(I don't like Upper Triangular Matrix multiplication, for example)
//I can implement it once I'm told how the mathematicians do it,
//but I CBA to look it up myself and make sense of the symbols. --3ICE
```

main.cpp

```
#include "Tria.h"
#include <iostream>
#include <iomanip>

using namespace std;

//A típust egy osztály segítségével valósítsa meg.
//Az összes megvalósítandó típus, azonos típusú elemek összetett
adatszerkezetű
//gyűjteményét írja le, amelyet dinamikusan lefoglalt tömbben kell
elhelyezni,
//ezért az osztályban meg kell valósítani a másoló konstruktort és az
értékadás
//operátort is. Ahol a feladat szövege nem definiálja, az elemi típus az
egész
//számok típusa. (Ne alkalmazzon sablonokat!)

//Egy osztály szolgáltatásainak (összes metódusának) bemutatásához olyan
//főprogramot kell készíteni, amelyik egy menü segítségével teszi lehetővé
a
//metódusok tetszőleges sorrendben történő kipróbálását. A főprogram
//példányosítson egy objektumot, amelyre a menüpontok közvetítésével
lehessen
//meghívni az egyes metódusokat. Természetesen szükség lehet minden
tevékenység
//után az objektum állapotának kiírására vagy egy az objektum állapotát
kiíró
//külön menüpontra. Ha vannak olyan metódusok (esetleg barát függvények),
//amelyek több objektum közötti műveleteket valósítanak meg, a főprogram
több
//objektum létrehozására és azok állapotának kiírására is adjon
lehetőséget.

//3ICE: I chose the second task.

//(2) Valósítsa meg az egész számokat tartalmazó felsőháromszög
mátrixtípust
//(a mátrixok a főátlójuk alatt csak nullát tartalmaznak)! Ilyenkor
elegendő
//csak a főátló és afeletti elemeket reprezentálni egy sorozatban, amelyet
egy
//dinamikus helyfoglalású tömbben helyezünk el. Implementálja önálló
//metódusként a mátrix i-edik sorának j-edik elemét visszaadó műveletet,
//valamint hatékony összeadás és szorzás műveleteket, továbbá a mátrix
//(négyzetes alakú) kiírását, és végül a másoló konstruktort és az
//értékadás operátort!

Tria a, b;

void Menu() {
    cout << endl << endl;
    cout << " ? : Print this help message." << endl;
    cout << " r : Reset matrices (3 x 3 with values 11 12 13 22 23 33)" <<
endl;
    cout << " o : Output Matrices" << endl;
```

```

    cout << " +: Matrix Add" << endl;
    cout << " *: Matrix Multiply" << endl;
    cout << " a: Custom Matrix a (Example usage: a 3 11 12 13 22 23 33)" <<
endl;
    cout << " b: Custom Matrix b (E.g.: b 1 1 1)" << endl;
    cout << " g: Get Matrix A element (Example: g 1 1)" << endl;
    cout << " s: Set Matrix A element (Usage: s 3 3 100)" << endl;
    cout << " G: Get Matrix B element (G 1 2 -1)" << endl;
    cout << " S: Set Matrix B element (S 1 1 1)" << endl;
    cout << " x: Swap Matrix A with B" << endl;
    cout << " 0: Exit" << endl;
}

void Reset() {
    a.Size ( 3 );
    b.Size ( 3 );
    a ( 1, 1 ) = 11;
    a ( 1, 2 ) = 12;
    a ( 1, 3 ) = 13;
    a ( 2, 2 ) = 22;
    a ( 2, 3 ) = 23;
    a ( 3, 3 ) = 33;
    b ( 1, 1 ) = 1;
    b ( 1, 2 ) = 2;
    b ( 1, 3 ) = 3;
    b ( 2, 2 ) = 4;
    b ( 2, 3 ) = 5;
    b ( 3, 3 ) = 6;
}

void Get ( Tria &x ) {
    int i, j;
    cout << "Row index: "; cin >> i;
    cout << "Column index: "; cin >> j;
    cout << "The element at [" << i << ", " << j << "] is ";

    try {
        cout << x ( i, j );
    }

    catch ( Tria::Exceptions ex ) {
        switch ( ex ) {
            case Tria::ARRAY_OUT_OF_BOUNDS :
                cout << "Array out of bounds!" << endl; break;

            case Tria::INDEX_BELOW_DIAGONAL :
                cout << 0 << endl; break;
        }
    }
}

void Set ( Tria &x ) {
    int i, j;
    cout << "Desired row index: "; cin >> i;
    cout << "Desired column index: "; cin >> j;

    try {
        cout << "Value: "; cin >> x ( i, j );
    }

    catch ( Tria::Exceptions ex ) {

```

```

        switch ( ex ) {
            case Tria::ARRAY_OUT_OF_BOUNDS :
                cout << "Array out of bounds!" << endl;
                break;

            case Tria::INDEX_BELOW_DIAGONAL :
                cout << "You can't write below the diagonal!" << endl;
                break;
        }
    }
}

void Sum() {
    try {
        cout << "A + B = " << endl << a + b << endl;
    }

    catch ( Tria::Exceptions ex ) {
        switch ( ex ) {
            case Tria::MATRIX_NOT_SAME_SIZE :
                cout << "The two matrices are not the same size!" << endl;
                break;
        }
    }
}

void Mul() {
    try {
        cout << "A * B = " << endl << a * b << endl;
    }

    catch ( Tria::Exceptions ex ) {
        switch ( ex ) {
            case Tria::MATRIX_NOT_SAME_SIZE :
                cout << "The two matrices are not the same size!" << endl;
                break;
        }
    }
}

void Swap() {
    Tria c ( a );
    a = b;
    b = c;
}

int main() {
    cout << "=====" << endl;
    cout << "3ICE's Upper Triangular Matrix" << endl;
    cout << "=====" << endl << endl;
    Reset();
    char c = 0;
    int size = 3;
    Menu();

    do {
        cin >> c;

        switch ( c ) {
            case '?':
                Menu();

```

```

        break;

    case 'r':
        cout << "Resetting matrices (3 x 3 matrices with values 11 12 13 22
23 33 and values 1 2 3 4 5 6)"
            << endl;
        Reset();
        break;

    case 'o':
        cout << "Outputting both matrices" << endl;
        cout << "A = " << endl << a << endl << endl << "B = " << endl << b;
        break;

    case '+':
        cout << "Adding matrices" << endl;
        Sum();
        break;

    case '*':
        cout << "Multiplying matrices" << endl;
        Mul();
        break;

    case 'a':
        cout << "Custom Matrix a" << endl;
        cout << "Size: "; cin >> size;
        a.Size ( size );
        cin >> a;
        break;

    case 'b':
        cout << "Custom Matrix b" << endl;
        cout << "Size: "; cin >> size;
        b.Size ( size );
        cin >> b;
        break;

    case 'g':
        cout << "Getting Matrix A element" << endl;
        Get ( a );
        break;

    case 's':
        cout << "Setting Matrix A element" << endl;
        Set ( a );
        break;

    case 'G':
        cout << "Getting Matrix B element" << endl;
        Get ( b );
        break;

    case 'S':
        cout << "Setting Matrix B element" << endl;
        Set ( b );
        break;

    case 'x':
        cout << "Swapping matrices" << endl;
        Swap();

```

```
        break;
    }
}
while ( c != '0' );

cout << "Goodbye!" << endl;
////3ICE: Because visual Studio doesn't do this automatically:
//int noClose;
//cin >> noClose;
return 0;
}
```

Berezvai Dániel <http://elte.3ice.hu/>