# OOP Documentation 1st assignment

**Name:** Giorgi Gulbatashvili
**Date:** 01.04.2023
**Neptun code:** W9NJYS
**Group:** 10

# Task

Implement the block matrix type which contains integers. These are square matrices that can contain nonzero entries only in two blocks on their main diagonal. Let the size of the first and second blocks be b1 and b2, where 1≤b1,b2≤n-1 and b1+b2=n (in the example, b1=2 and b2=4). Don't store the zero entries. Store only the entries that can be nonzero in a sequence or two smaller matrices. Implement as methods: getting the entry located at index (i, j), adding and multiplying two matrices, and printing the matrix (in a square shape).

| | | | | | |
|---|---|---|---|---|---|
| x | x | 0 | 0 | 0 | 0 |
| x | x | 0 | 0 | 0 | 0 |
| 0 | 0 | x | x | x | x |
| 0 | 0 | x | x | x | x |
| 0 | 0 | x | x | x | x |
| 0 | 0 | x | x | x | x |

# Block matrix type

## Set of values

b1 and b2 are sizes of blocks 1 and 2 ,respectively, inside the matrix. Sum of b1 & b2 equals dimension of matrix or n. For simplicity , Two notations (b1+b2) as in ( $\mathbb{Z}^{(b1+b2)\,x\,(b1+b2)}$) and n as in ($\mathbb{Z}^{(n)\,x\,(n)}$) are used interchangeably throughout the documentation.

$$BM(\mathbb{Z}^{(n)\,x\,(n)}) = \{ a \in \mathbb{Z}^{(b1+b2)\,x\,(b1+b2)} \mid \forall\ i, j \in [1..n] : \neg((i \le b1 \wedge j \le b1) \vee (i > b1 \wedge j > b1)) \rightarrow a[i,j] = 0\}$$

## Operations

### Getting the entry

Getting the entry of the *i* th column and *j* th row ( I, j ∈[1..n]): e:=a[I, j]. This operation needs action only if (i ≤ b1∧ j ≤ b1) ∨ (i > b1∧ j > b1) otherwise it gives output zero.

Formally:

A = (a : BM ( $\mathbb{Z}^{(b1+b2)\,x\,(b1+b2)}$), i : $\mathbb{Z}$ , j : $\mathbb{Z}$, e : $\mathbb{Z}$)

Pre = (a = a' $\wedge$ i = I' $\wedge$ j = j' $\wedge$ i, j $\in$ [1..n]) )
Post = (Pre $\wedge$ e := a[i, j])

### Setting the entry

Setting the entry of the *i* th column and *j* th row ( I, j $\in$[1..n]): a [i, j]:=e. Entries outside the block 1 and block 2 cannot be modified. In other words if ¬ ((i ≤ b1$\wedge$ j ≤ b1) $\vee$ (i > b1$\wedge$ j > b1)) it gives an error since we want to modify a zero entry. Only one element of the matrix is modified, as per call of the method, rest stays the same.

Formally:

A = (a : BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$), i : $\mathbb{Z}$ , j : $\mathbb{Z}$, e : $\mathbb{Z}$)
Pre = (a = a'$\wedge$ i=i' $\wedge$ j = j' $\wedge$ e=e' $\wedge$ i, j $\in$ [1..n] $\wedge$ ((i ≤ b1$\wedge$ j ≤ b1)$\vee$(i > b1$\wedge$ j > b1)))
Post = (i = i' $\wedge$ j = j' $\wedge$ e=e' $\wedge$ a[i, j]:= e $\wedge$ $\forall$ k, l $\in$ [1..n]: k ≠ i $\wedge$ l ≠ j a [k, l] =a'[k, l] )

### Sum of two matrices

Sum of two matrices. c := a + b. Matrixes have the same size. Size of block 1 (b1) is same for every a, b, c matrix. Size of block 2 (b2) is same for every a, b, c matrix.

Formally:

A = ( a: BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$),
b: BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$),
c: BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$))
Pre = (a = a' $\wedge$ b = b')
Post = (Pre $\wedge$ $\forall$ i, j $\in$ [1..n] : ((i ≤ b1$\wedge$ j ≤ b1)$\vee$(i > b1$\wedge$ j > b1)) ⇒ c [I, j] = a[i, j] + b[i, j]
$\wedge$ $\forall$ i, j $\in$ [1..n] : ¬((i ≤ b1$\wedge$ j ≤ b1)$\vee$(i > b1$\wedge$ j > b1)) ⇒ c [I, j] = 0)

### Multiplying two matrices

multiplication of two matrices. c:= a * b. Matrixes have the same size. Size of block 1 (b1) is same for every a, b, c matrix. Size of block 2 (b2) is same for every a, b, c matrix.
Formally:

A = ( a: BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$),
b: BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$),
c: BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$))
Pre = (a = a' $\wedge$ b = b')
Post = (Pre
$\wedge$ $\forall$ i, j$\in$[1..n] : ((i ≤ b1$\wedge$ j ≤ b1)$\vee$(i > b1$\wedge$ j >b1))→

c [I, j] = $\sum_{k=1..n}$ a[i, k] * b[k, j]
cond(i,k)* $\wedge$ cond (k,j)**

$\wedge$ $\forall$ i, j $\in$ [1..n] : ¬((i ≤ b1$\wedge$ j ≤ b1)$\vee$(i > b1$\wedge$ j > b1)) → c [I, j] = 0))

### Printing the matrix (in a square shape).

Formally:

A = ( a: BM ( $\mathbb{Z}^{(b1 + b2) \times (b1 + b2)}$)
Pre = (a = a')
Post = (Pre $\wedge$ $\forall$ i, j $\in$ [1..n]: ((i ≤ b1$\wedge$ j ≤ b1) $\vee$ (i > b1$\wedge$ j > b1)) → write(a[i , j] $\wedge$
$\forall$ i, j $\in$ [1..n]: ¬ ((i ≤ b1$\wedge$ j ≤ b1) $\vee$ (i > b1$\wedge$ j > b1)) → write(0))

*((i ≤ b1∧ k ≤ b1)∨(i > b1∧ k >b1))  **((k ≤ b1∧ j ≤ b1)∨(k > b1∧ j >b1))

# Representation

For (b1+b2) x (b1+b2) matrix object, with fields b1(size of block 1) and b2(size of block 2), only those [i , j] elements of a matrix has to be stored for which ((i ≤ b1∧ j ≤ b1) ∨ (i > b1∧ j > b1)) is true. One dimensional List<> in C# can be used for representing every nonzero element(elements which are in block 1 and block 2) of matrix.

$$a = \begin{matrix} a_1 & a_2 & 0 & 0 & 0 \\ a_3 & a_4 & 0 & 0 & 0 \\ 0 & 0 & a_5 & a_6 & a_7 \\ 0 & 0 & a_8 & a_9 & a_{10} \\ 0 & 0 & a_{11} & a_{12} & a_{13} \end{matrix} \leftrightarrow v = \langle a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13} \rangle$$

v : $\mathbb{Z}^s$
b1 : $\mathbb{N}$
b2 : $\mathbb{N}$
s : $\mathbb{N}$
n : $\mathbb{N}$

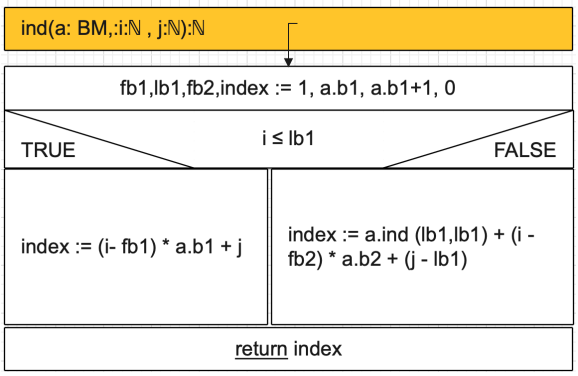
(Aliases in the code: v = vec, b1 = size_b1, b2 = size_b2, s = length, n = size. For simplicity notations presented above will be used throughout the documentation )

(inv: n≥0)
(inv: 1≤ b1)
(inv: b2 ≤n-1)
(inv: b1+ b2 =n)
(inv: (b1 * b1 ) + (b2 * b2 ) =s )

Respective **v** list index for matrix element with indexes *i* and *j* can be generated using method **ind** which takes two arguments, (indexes of matrix) *i*, *j*. Method **ind** uses objects fields b1 and b2 for calculating **index**.

example of using **ind** method:

a[i, j] = if ((i ≤ b1∧ j ≤ b1) ∨ (i > b1∧ j > b1)) then v[ind(i, j )] else 0.




**fb1**(*index Of first Raw & Column Of block 1*),

**lb1** (*index Of Last Raw & Column Of block 1*),

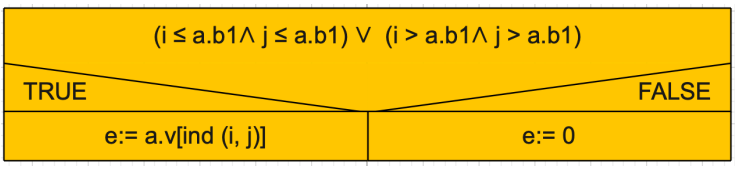
**fb2**(*index Of first Raw & Column Of block 2*)

(fb1, lb1, fb2 variables were used to make transition to 0 indexing easy,) :

# Implementation

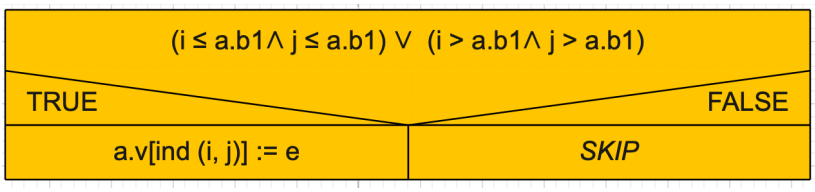
## Getting the entry

Getting the entry of the *i* th column and *j* th row (i, j ∈ [1..n]) e:=a[i, j] can be implemented as



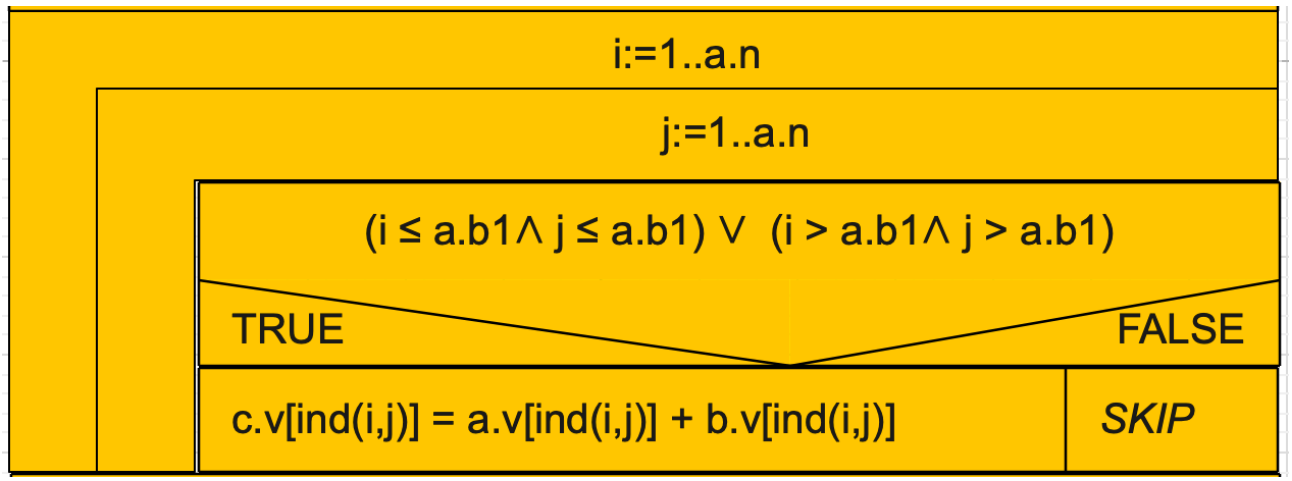

## Setting the entry

Setting the entry of the i th column and j th row (i, j ∈ [1..n]) a[i, j]:= e can be implemented as



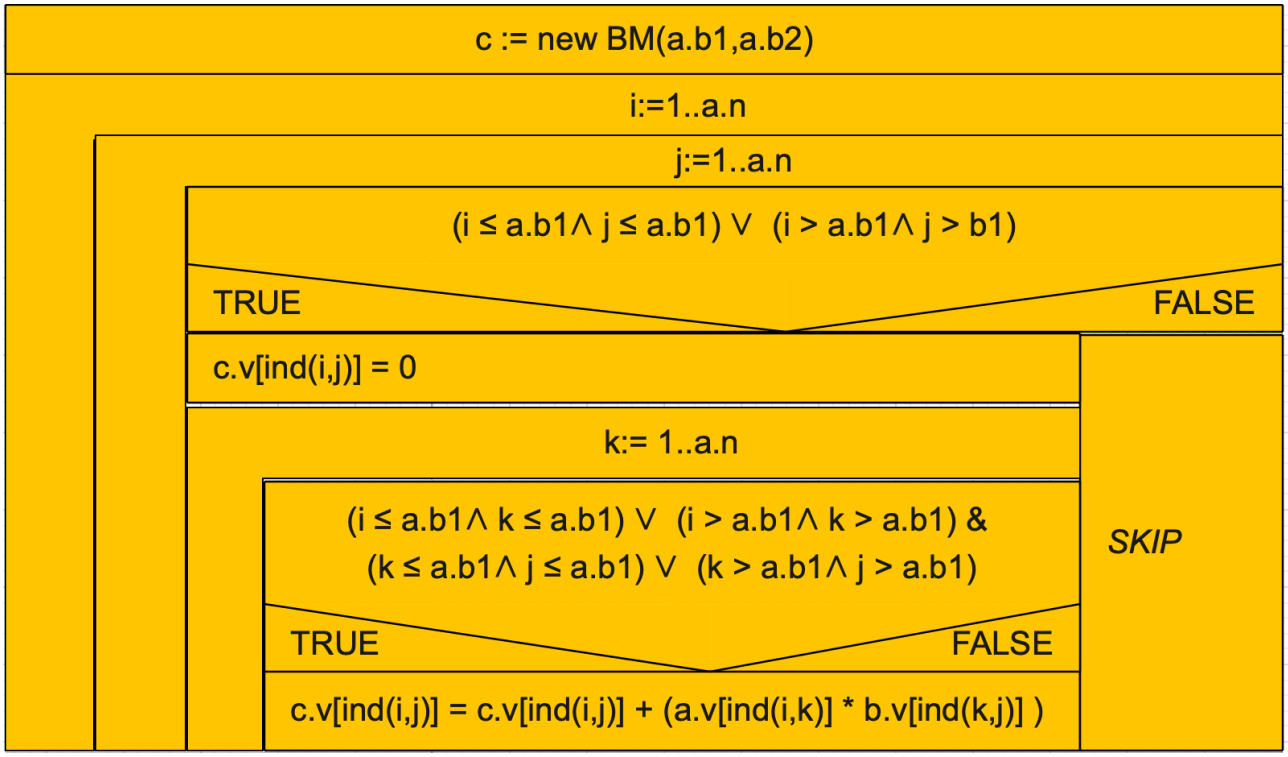

## Sum of two matrices

The sum of matrices a and b goes to matrix c where matrixes have the same size. Size of block 1 (b1) is same for every a, b, c matrix. Size of block 2 (b2) is same for every a, b, c matrix.



## Multiplying two matrices

The product of matrices a and b goes to matrix c where matrixes have the same size. Size of block 1 (b1) is same for every a, b, c matrix. Size of block 2 (b2) is same for every a, b, c matrix.



# Testing

## Testing the operations

### (White box testing)

1) Creating matrices of different size.Testing constructors with variable argument list.
   a) Check that constructor with wrong block size parameter(where size_b1 <1|| size_b1 > 100 || size_b2 <0 || size_b2 >100), throws the exceptions.
   b) check constructor without 0,1,2,3 parameters, during each checking check values inside the blocks, check that exceptions are thrown in case of accessing wrong elements.
2) Test Copy constructor
   a) Creating matrix b based on matrix a, comparing the entries of the two matrices. Then, changing one of the matrices and comparing the entries of the two matrices.

### (Black box testing)

3) Getting and setting an entry
   a) Getting and setting an entry in the block matrix.
      I. Middle positions
      II. First and last indexes
   b) Getting and setting an entry outside the diagonal
4) Comparison of matrices a and b
   a) Executing command a=a for matrix a.
   b) Executing command b=a for matrices a and b (with and without same size), comparing the entries of the two matrices. Then, changing one of the matrices and comparing the entries of the two matrices.
5) Sum of two matrices, command c:=a+b.
   a) With matrices of different block sizes (sizes of block one differ, or sizes of block 2 differ)
   b) Checking the commutativity (a + b == b + a)
   c) Checking the associativity (a + b + c == (a + b) + c == a + (b + c))
   d) Checking the neutral element (a + 0 == a, where 0 is the null matrix)
6) Multiplication of two matrices, command c:=a*b.
   a) With matrices of different size (size of a and b differs, size of c and a differs)
   b) Checking the commutativity (a * b == b * a)
   c) Checking the associativity (a * b * c == (a * b) * c == a * (b * c))
   d) Checking the neutral element (a * 0 == 0, where 0 is the null matrix)
   e) Checking the identity element (a * 1 == a, where 1 is the identity matrix)