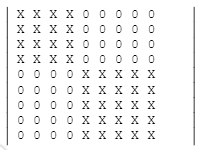
**Documentation for the first assignment:**

G31R6T

li zhipeng

**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=4 and b2=5). 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). 

**Idea**:

Define the variables b1, b2 to be the size of the two blocks b1 = 4, b2 = 5, a represents this matrix, the matrix size is 9, when accessing the element a [i] [j], judge the range of i, j, divided into the following Special case:

* If either i or j does not belong to the range of 0 to 8, it means an access error

2. If i <b1 && && j <b1, it means that a [i] [j] visits block 1, and returns \_b1v [i] [j], like a [1] [1] = \_b1v [1] [1]

3. If i>= b1 && j>= b1, it means that a [i] [j] visits block 2, and returns \_b2v [i-b1] [j-b1], like a [4] [4] = \_b2v [ 0] [0]

4. Otherwise returns 0, which means it is not in two blocks

**Set of values:**

Set(n) = { a ∈ℤ n×n , i,j[1..n]: c →a[i,j]=0 }

Set(c)= {b1,b2∈a l i<b1 && j<b1 , i>=b2 && j>=b2}

**Operations:**

**(**When adding or multiplying, it is the addition and multiplication of the two blocks corresponding to the two matrices.

Pass in parameters i, j when getting or setting, use the above method to return a[i][j]**)**

* Getting an entry Getting the entry of the ith column and jth row (i,j[1..n]): e:=a[i,j]. Formally:

A : Diag(n) × ℤ × ℤ × ℤ

a i j e

Pre = ( a=a’ i=i’ j=j’ i,j[1..n] )

Post = ( Pre e=a[i,j] )

This operation if it is outside this range(b1,b2), it will be rounded to zero.

* Setting an entry Setting the entry of the ith column and jth row (i,j[1..n]): a[i,j]:=e.

Formally:

A = Diag(n) × ℤ × ℤ × ℤ× ℤ × ℤ

a i j e v c

Pre = ( e=e’ a=a’ i=i’ j=j’ i,j[1..n] i=j )

Post = (e=e’ i=i’ j=j’ a[i,j],v[i.j],c[i,j]=e k,l[1..n]: (i<b1 && j<b1 i>=b2 && j>=b2)→ a[i,j]=v(b1\*b1),a[I,j]=c(b2\*b2 )

This operation needs determine whether there is a required block1 or 2, otherwise it gives an error if we want to modify a zero entry.

* **Sum**

Sum of two matrices: z:=b1+b2. The matrices have the same size.

Formally:

A = Diag(n) × Diag(n) × Diag(n)

a b z

Pre = ( a=a’ b=b’)

Post = ( Pre i,j[1..n]: z[i,j]=v( b1\*b1)[i,j] +c( b2\*b2)[i,j] )

In case of 2 square matrices there is an easier version: i[1..n]: z[i,j]= v[i,j] + c[i,j] és i,j[1..n]: (i<b1 && j<b1 i>=b2 && j>=b2)→ z[i,j]=0.

* **Multiplication**

Multiplication of two matrices: z:=v\*c. The 2 matrices have the same size. Formally:

A = Diag(n) × Diag(n) × Diag(n)

a b z

Pre = ( a=a’ b=b’)

Post = ( Pre i,j[1..n]: z[i,j],v[i,j],c[i,j]= k=1..n v[i,j] \* c[i,j])

In case of diagonal matrices there is an easier version: i[1..n]: z[i,j]= v[i,j]\*c[i,j] és i,j[1..n]: (i<b1 && j<b1 i>=b2 && j>=b2)→ z[i,j]=0.

**Representation:**

Only in two blocks on their main diagonal of the n×n.m×m.

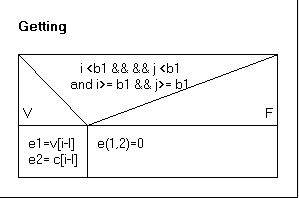
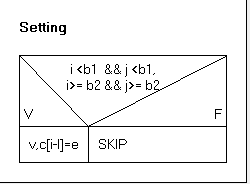
Need 2 two-dimensional array, get a square matrix v(b1\*b1), c=(b2\*b2)

𝑎[𝑖,𝑗] = { 𝑣[𝑖] 𝑖𝑓 i <b1 && j <b1

c[i] if i>=b2,j>=b2

0 𝑖𝑓 𝑖,j ≠ b1,b2 }

**Implementation:**

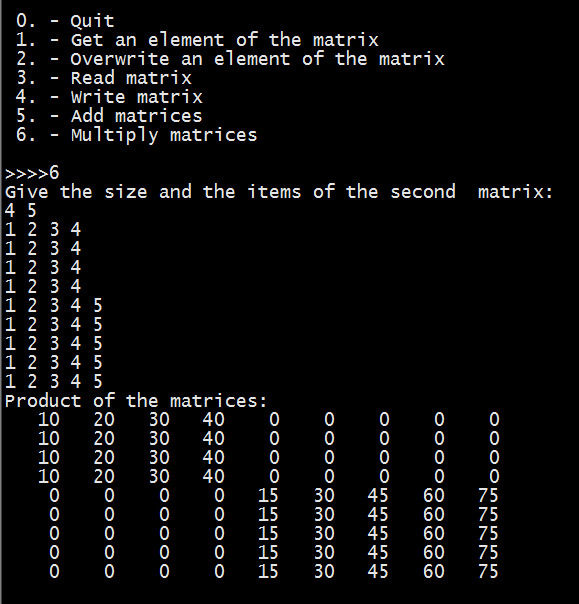
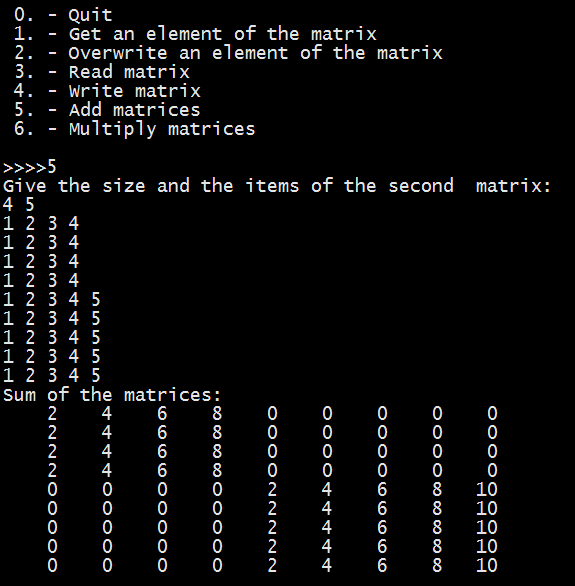
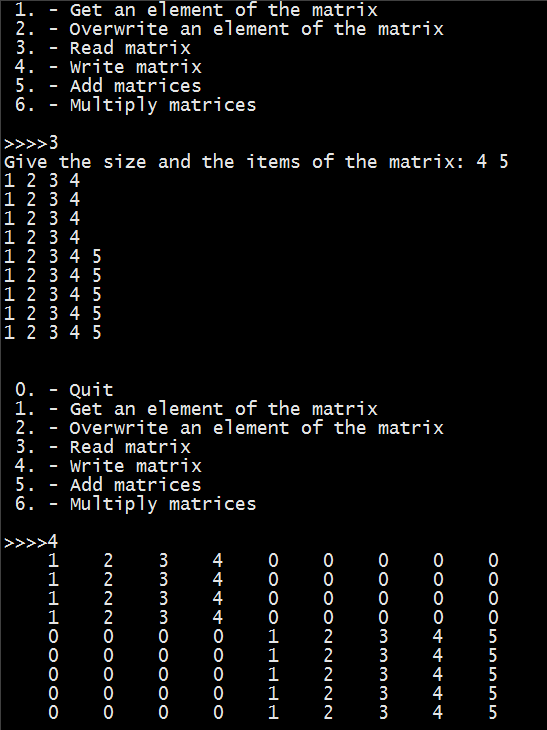
* Getting an entry Getting the entry of the ith column and jth row (i,j[1..n]) e:=a[i,j] where the matrix is represented by b1,b2,1i,jn, and n stands for the size of the matrix can be implemented as
* 
* Setting an entry Setting the entry of the ith column and jth row (i,j[1..n]) a[i,j]:=e where the matrix is represented by v,c,1b1,b2n, and n stands for the size of the matrix can be implemented as
* 

3.Sum The sum of matrices a and b (represented by arrays t and u) goes to matrix c (represented by array z), where all of the arrays have to have the same size.

i,j[0..n-1]: z[i,j]:= v[i,j] + c[i,j]

4. Multiplication The product of matrices a and b (represented by arrays t and u) goes to matrix c (represented by array z), where all of the arrays have to have the same size. i,j[0..n-1]: z[i,j]:= v[i，j] \* c[i,j]

**Testing**

Testing the operations (black box testing)

1) Creating, reading, and writing matrices of different size. a) 0, 1, 2, 5-size matrix

2) Getting and setting an entry a) Getting and setting an entry in the diagonal b) Getting and setting an entry outside the diagonal c) Illegal index, indexing a 0-size matrix

3) 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.

* Assignment operator

a) Executing command b1=b2 for matrices b1 and b2 (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.

b) Executing command c=b2=b1 for matrices b1, b2, and c (with and without same size), comparing the entries of the three matrices. Then, changing one of the matrices and comparing the entries of the three matrices.

c) Executing command a=a for matrix a.

* Sum of two matrices, command z:=c+v.

a) With matrices of different size (size of b1and b2 differs, size of c and a differs)

b) Checking the commutativity (c + v== c + v)

c) Checking the associativity (c + v + z == (v + c) + z == c + (v+ z))

d) Checking the neutral element (c+ 0 == c, v +0==v, where 0 is the null matrix)

* Multiplication of two matrices, command z:=v\*c.

a) With matrices of different size (size of b1and b2 differs, size of c and b1 differs)

b) Checking the commutativity (v \* c == c \* v)

c) Checking the associativity (c\* v\* z == (v \* c) \* z ==v \* (z \* c))

d) Checking the neutral element (v\* 0 == 0, c\*0==0, where 0 is the null matrix)

e) Checking the identity element (c\* 1 == c, v\*1==v,where 1 is the identity matrix)

Testing based on the code (white box testing) 1. Creating an extreme-size matrix (-1, 0, 1, 1000).

2. Generating and catching exceptions.