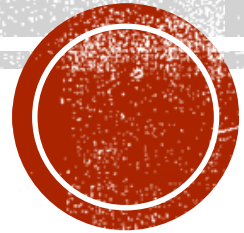
Array of two dimensions

Matrix



CHAPTER HEADLINES

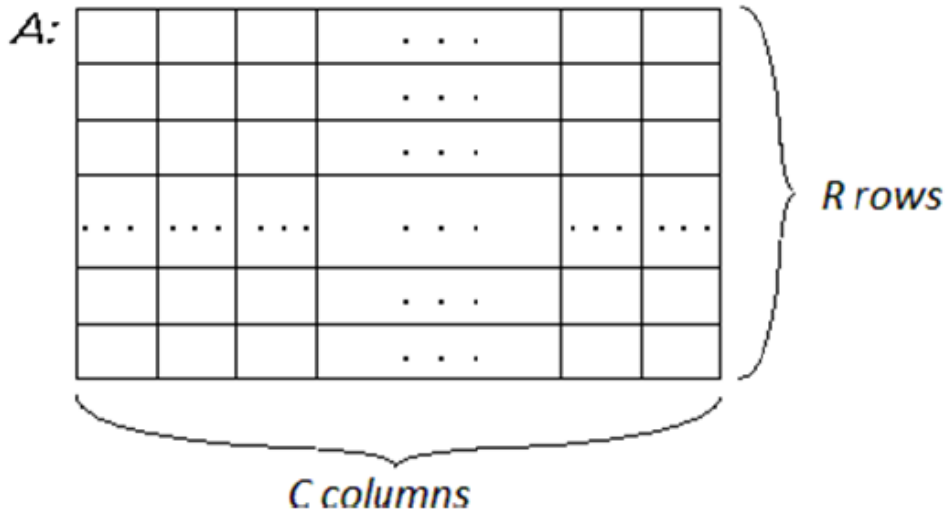
- Part 1
  - Definition
  - Declaration
  - Initialization
  - Storage
  - Automatic reservation
- Part 2
  - Access to components
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - Exercises

CHAPTER HEADLINES

- **Part 1**
  - **Definition**
  - Declaration
  - Initialization
  - Storage
  - Automatic reservation
- Part 2
  - Access to components
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - Exercises

DEFINITION

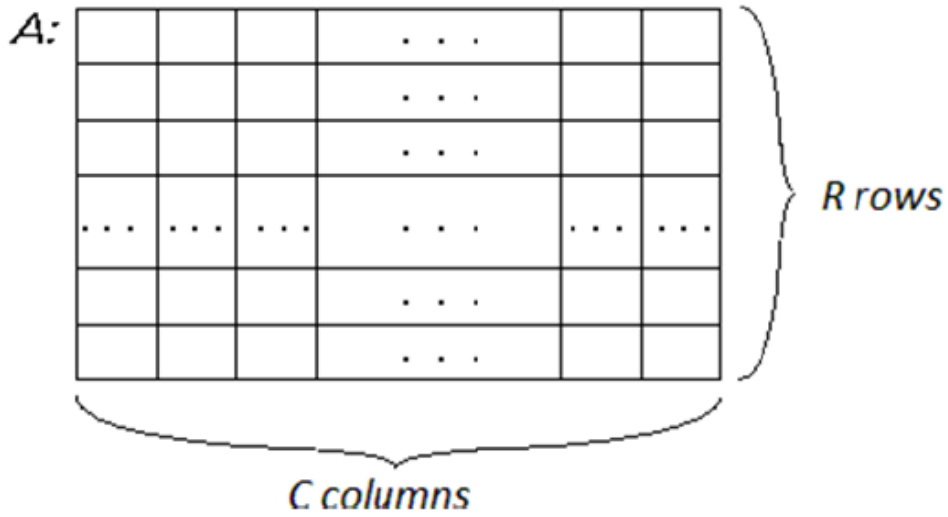
A two-dimensional array A has R rows and C columns It is called also a Matrix.



- ✓ R is the number of rows in the array A
- ✓ C is the number of columns in the array A.

DEFINITION

A two-dimensional array A has R rows and C columns It is called also a Matrix.



- ✓ R is the number of rows in the array A
- ✓ C is the number of columns in the array A.

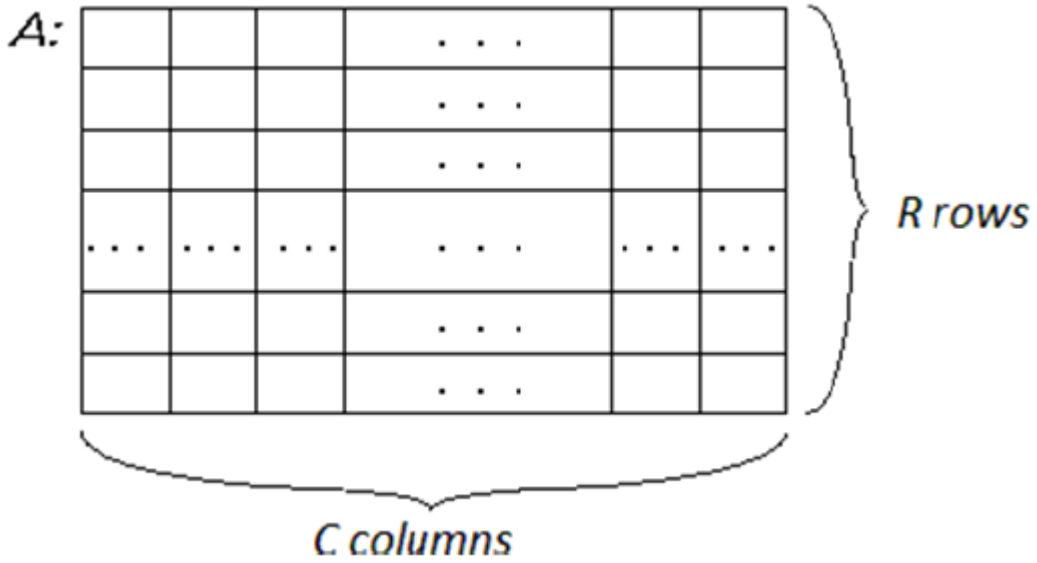
The number of elements/components in the Array is R*C.

For example, if A has 3 rows (i.e. R=3) and 4 columns (i.e. C=4), then the number of elements/components in the Matrix A is 12 elements (i.e. 3*4),

All the elements in the Matrix/array of two dimensions **have the same type**.

DEFINITION

- A two dimensional array is said to be square if the number of rows is equal to the number of columns (i.e. R=C).
- An array A of two dimensions (R rows and C columns) can be interpreted as an array (uni-dimensional) of dimension R, where each component of this unidimensional array is a (one-dimensional) array of dimension C.



DEFINITION

- **Why do we need an array of two dimensions?**
  - An array A of two dimensions can be used to store a set of data depending on each other.

    Example: store the grades of 20 students relevant to 7 courses.

DEFINITION

- **Why do we need an array of two dimensions?**
  - *An array A of two dimensions can be used to store a set of data depending on each other.*

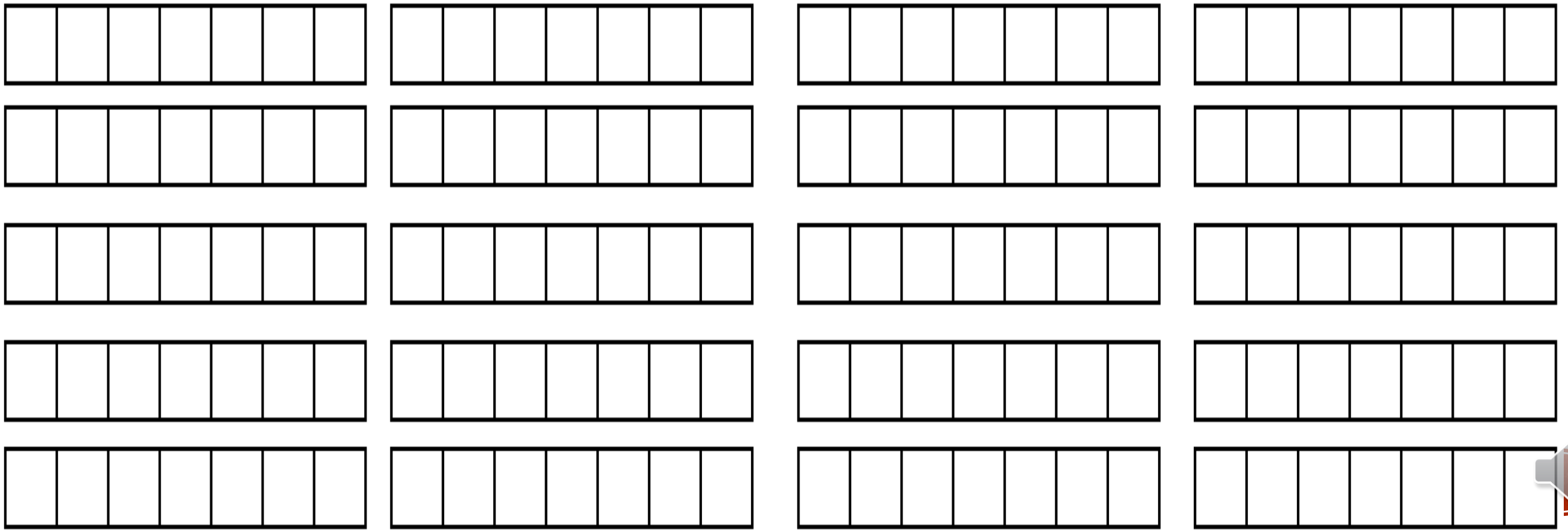
  Example: store the grades of 20 students relevant to 7 courses.

  - *For each student we needs a unidimensional array of 7 elements (size 7) to store the 7 grades of this student.*
  - *for 20 students, we need 20 one-dimensional arrays.*

DEFINITION

- **Why do we need an array of two dimensions?**
  - An array A of two dimensions can be used to store a set of data depending on each other.

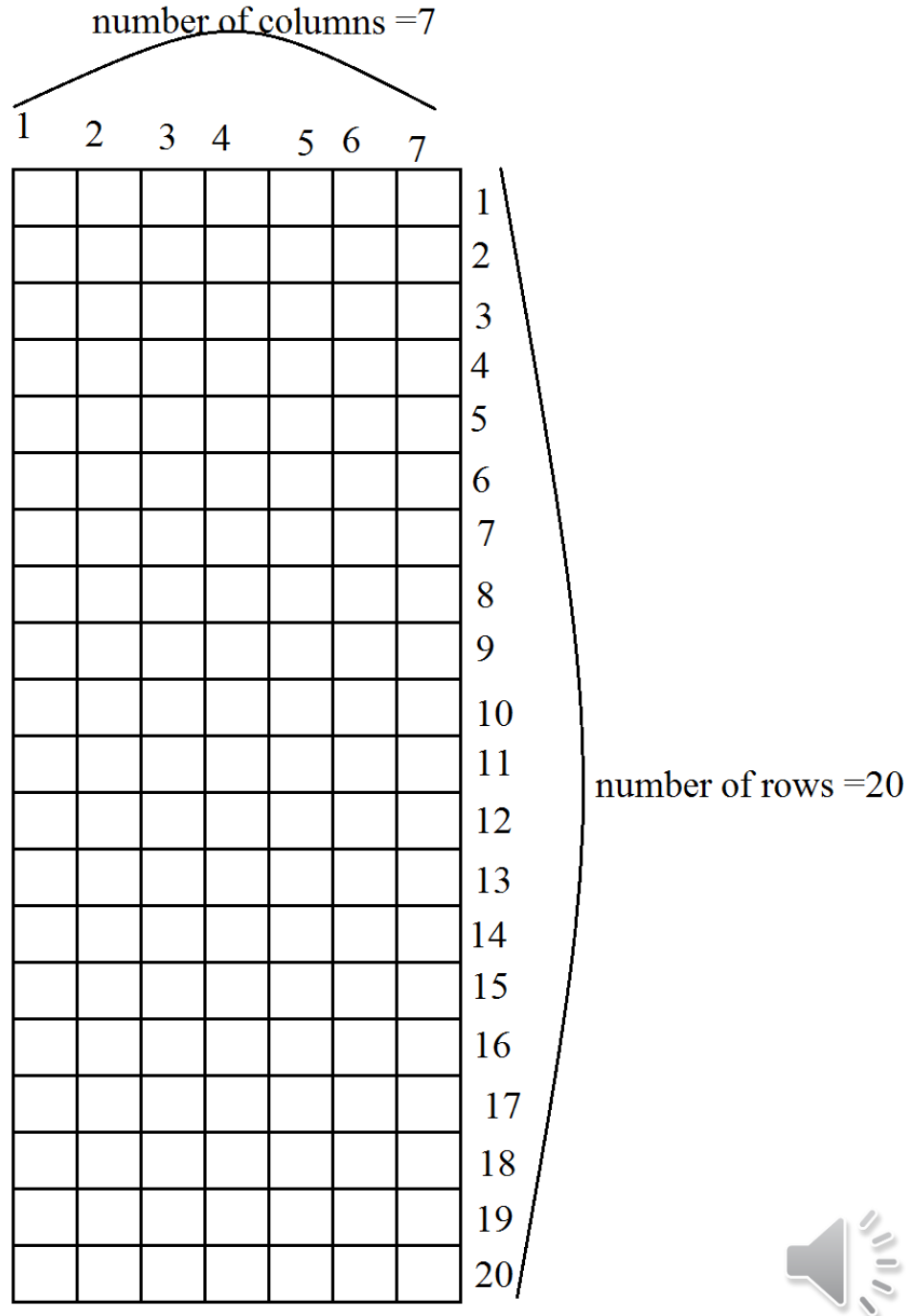
    Example: store the grades of 20 students relevant to 7 courses.



DEFINITION

- **Solution:** we use a matrix of dimension 20 rows and 7 columns.

- **One structured variable (matrix) instead of 20 structured variables (vector/array of one dimension).**



CHAPTER HEADLINES

- **Part 1**
  - Definition
  - **Declaration**
  - Initialization
  - Storage
  - Automatic reservation
- Part 2
  - Access to components
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - Exercises

DECLARATION

- <Type> <MatrixName> [<Rows Dimension >] [<Columns Dimension >] ;
  - Example:

    int A[20][7]; float B[100][20]; int C[10][15] ; char D[30][100];

DECLARATION

- **<Type> <MatrixName> [<Rows Dimension >] [<Columns Dimension >] ;**
  - Example:

    int A[20][7]; float B[100][20]; int C[10][15] ; char D[30][100];

- **Type :** could be any predefined type: int, float, double, short, char, bool, long,… it determines the type of the components/elements of the matrix.

DECLARATION

- **<Type> <MatrixName> [<Rows Dimension >] [<Columns Dimension >] ;**
  - Example:

    int A[20][7]; float B[100][20]; int C[10][15] ; char D[30][100];

- **Type :** could be any predefined type: int, float, double, short, char, bool, long,… it determines the type of the components/elements of the matrix.
- **MatrixName** is the name of the matrix (same conditions):
  - ✓Does not start with numbers, exp: int 1A[10][25]; ERROR
  - ✓Does not contain punctuations characters (space, comma, semi-colon, …) exp: int grades of students[20][10]; ERROR

DECLARATION

- **<Type> <MatrixName> [<Rows Dimension >] [<Columns Dimension >] ;**
  - Example:

    int A[20][7]; float B[100][20]; int C[10][15] ; char D[30][100];

- **Type :** could be any predefined type: int, float, double, short, char, bool, long,… it determines the type of the components/elements of the matrix.
- **MatrixName** is the name of the matrix (same conditions):
  - ✓Does not start with numbers, exp: int 1A[10][25]; ERROR
  - ✓Does not contain punctuations characters (space, comma, semi-colon, …) exp: int grades of students[20][10]; ERROR
  - **<Rows Dimension > and <Columns Dimension >,** they shall be:
    - ✓ Integer number only, exp: int A[20.5][13.6]; ERROR,
    - ✓Positive number Only exp: float B [-15][2]; ERROR
    - ✓Constant value, int A[x][y]; ERROR

CHAPTER HEADLINES

- **Part 1**
  - Definition
  - Declaration
  - **Initialization**
  - Storage
  - Automatic reservation
- Part 2
  - Access to components
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - Exercises

INITIALIZATION

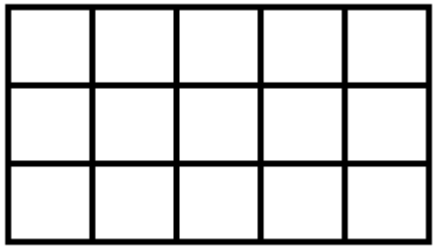
INITIALIZATION

- When declaring an array, we can initialize the components of the array, by indicating the list of the respective values between braces. Inside the list, the components of each row of the array are once again enclosed in braces.
- Example:
  - int A[3] [5] ={ { 0 ,10 ,20 ,30 ,40} , {10 ,11 ,12 ,13 ,14} ,{ 1 ,12 ,23 ,34 ,45 } };
  - Float B[3][2]={{-1.05, -1.10}, {86e-5,87e-5},{-12.5E4, 12.3E4}};

INITIALIZATION

- To improve the readability of the programs, we can indicate the components in several lines. Examples:
  - int A[3] [5] ={ { 0 ,10 ,20 ,30 ,40} ,
    {10 ,11 ,12 ,13 ,14} ,
    { 1 ,12 ,23 ,34 ,45 } };
  - Float B[3][2]={{-1.05, -1.10},
    {86e-5,87e-5},
    {-12.5E4, 12.3E4}};

A



- During initialization, the values are assigned line by line (row by row) from left to right.

INITIALIZATION

- To improve the readability of the programs, we can indicate the components in several lines. Examples:
  - int A[3] [5] ={ { 0 ,10 ,20 ,30 ,40} ,
    {10 ,11 ,12 ,13 ,14} ,
    { 1 ,12 ,23 ,34 ,45 } };

A

0	10	20	30	40
10	11	12	13	14
1	12	23	34	45

  - Float B[3][2]={{-1.05, -1.10},
    {86e-5,87e-5},
    {-12.5E4, 12.3E4}};

B

-1.05	-1.10
86e-5	87e-5
12.5e4	12.3e4

- During initialization, the values are assigned line by line (row by row) from left to right.

INITIALIZATION

- During initialization, the values are assigned line by line from left to right.
- We do not necessarily have to indicate all the values: **Missing values will be initialized by zero**. It is however forbidden to indicate a number values for an array that exceeds its size/dimension.

EXP1:

```
int C[4][4] ={{1, 0, 0, 0}
              {1, 1, 0, 0}
              {1, 1, 1, 0}
              {1, 1, 1, 1}};
```

C:

1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

EXP2:

```
int C[4][4] ={{1, 1, 1, 1}};
```

C:

1	1	1	1
0	0	0	0
0	0	0	0
0	0	0	0

EXP3:

```
int C[4][4]= {1}, {1}, {1}, {1}};
```

C:

1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0

EXP4:

```
int C[4][4] ={{1, 1, 1, 1, 1}};
```

ERROR !

CHAPTER HEADLINES

- **Part 1**
  - Definition
  - Declaration
  - Initialization
  - **Storage**
  - Automatic reservation
- Part 2
  - Access to components
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - Exercises

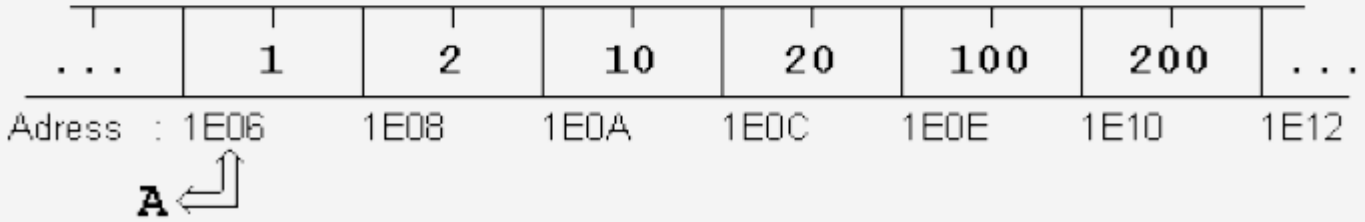
STORAGE

- As with one-dimensional arrays, the name of a array is the representative of the address of the first array element (i.e. the address of the first row of the matrix).

STORAGE

- As with one-dimensional arrays, the name of a array is the representative of the address of the first array element (i.e. the address of the first row of the matrix).
- The components of a two-dimensional array are stored line by line in memory.

```
short A[3][2] = {{1,   2  },
                 {10,  20 },
                 {100, 200}};
```



- An array of R and C dimensions, consisting of components each of which needs M bytes, will occupy R * C * M bytes in memory.

CHAPTER HEADLINES

- **Part 1**
  - Definition
  - Declaration
  - Initialization
  - Storage
  - **Automatic reservation**
- Part 2
  - Access to components
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - Exercises

AUTOMATIC RESERVATION

- If the **number of rows R** is not specified explicitly during initialization, the computer automatically reserves the necessary number of bytes.
  - Examples:

```
int A[][10] = {{ 0,10,20,30,40,50,60,70,80,90},
               {10,11,12,13,14,15,16,17,18,19},
               { 1,12,23,34,45,56,67,78,89,90}};
==> reservation of 3*10*4 = 120 bytes

float B[][2] = {{-1.05,   -1.10  },
                {86e-5,    87e-5 },
                {-12.5E4, -12.3E4}};
==> reservation of 3*2*6 = 36 bytes/ octets
```

```
int C[][4] = {{1, 0, 0, 0}
              {1, 1, 0, 0}
              {1, 1, 1, 0}
              {1, 1, 1, 1}};
```

C:

1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

⇒ reservation of 4*4*4=64bytes

AUTOMATIC RESERVATION

- The **number of Columns C** should be specified explicitly during initialization, otherwise we get an **error message**.
  - Examples:

```
int A[4][]={{1,2},{3,4},{5,6},{7,8}}; ERROR

float A[][]={{1,2,3},{4,5,6}}; ERROR
```

CHAPTER HEADLINES

- **Part 1**
  - Definition
  - Declaration
  - Initialization
  - Storage
  - Automatic reservation
- **Part 2**
  - **Access to components**
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - Exercises

ACCESS TO COMPONENTS

- Access to a two-dimensional array in C

<MatrixName>[<row >][<column>]

- The elements of an array of dimensions R and C are as follows:

A[0][0]	A[0][1]	A[0][2]	. . .	A[0][C–1]
A[1][0]	A[1][1]	A[1][2]	. . .	A[1][C–1]
A[2][0]	A[2][1]	A[2][2]	. . .	A[2][C–1]
. . .	. . .	. . .	. . .	. . .
A[R–1][0]	A[R–1][1]	A[R–1][2]	. . .	A[R–1][C–1]

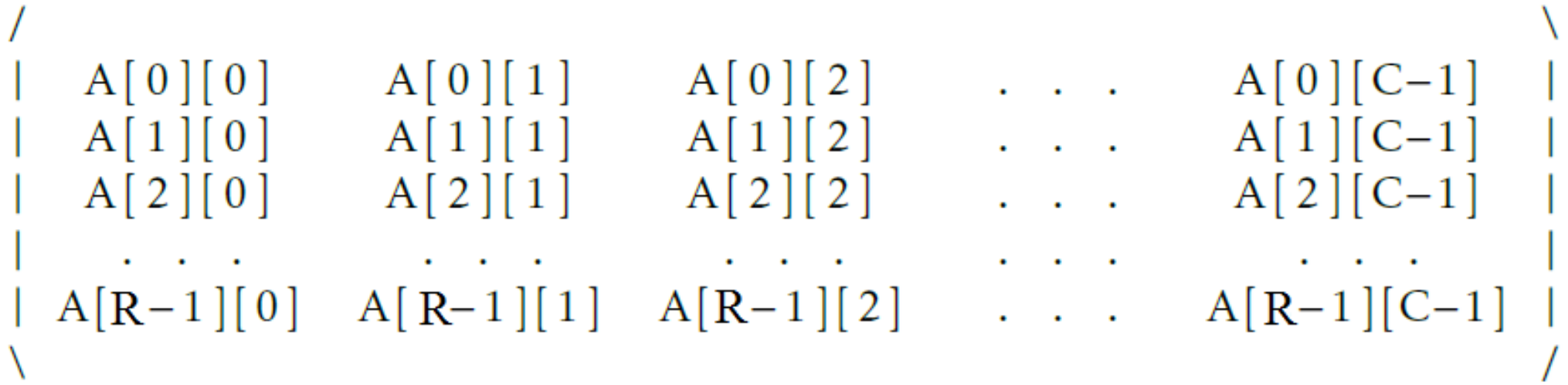
ACCESS TO COMPONENTS

- In C, for a Matrix A having R rows and C column:
  - ✓The row index in the matrix vary from 0 to R-1, and the column index from 0 to C-1.
  - ✓The first component in the matrix is A[0][0]
  - ✓The last component in the matrix is A[R-1][C-1]
  - ✓The component of the Nth row and Mth column is noted: A[N-1][M-1]

$$\begin{pmatrix} A[0][0] & A[0][1] & A[0][2] & \cdots & A[0][C-1] \\ A[1][0] & A[1][1] & A[1][2] & \cdots & A[1][C-1] \\ A[2][0] & A[2][1] & A[2][2] & \cdots & A[2][C-1] \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ A[R-1][0] & A[R-1][1] & A[R-1][2] & \cdots & A[R-1][C-1] \end{pmatrix}$$

ACCESS TO COMPONENTS

- Access to an element of an matrix A can be done by A[i][j], where i and j are the correponding row and column value of the required element.



ACCESS TO COMPONENTS

- Access to an element of an matrix A can be done by A[i][j], where i and j are the correponding row and column value of the required element.
- The value of *i* should be ≥ 0 and ≤R-1 (where R is the row dimension of the matrix A), and the value of *j* should be ≥ 0 and ≤C-1 (where C is the column dimension of the matrix A).
- The values of *i* and *j* could be:
  1. **Constante values**

Example :

```
int A[4][3]={{1,0,1},{1,2,-2},{0,2,6},{-1,1,4}};
A[0][0]=-30;
A[1][0]= A[0][2]+2*A[2][2];
cout<<A[1][0];
```

A

1	0	1
1	2	-2
0	2	6
-1	1	4

ACCESS TO COMPONENTS

- Access to an element of an matrix A can be done by A[i][j], where i and j are the correponding row and column value of the required element.
- The value of *i* should be ≥ 0 and ≤R-1 (where R is the row dimension of the matrix A), and the value of *j* should be ≥ 0 and ≤C-1 (where R is the row dimension of the matrix A).
- The values of *i* and *j* could be:
  1. Constante values
  2. **Variables**

Example :

```
int A[4][3]={{1,0,1},{1,2,-2},{0,2,6},{-1,1,4}};
int i=0,j=1;
A[i][j]=-30;
cout<<A[i][j];
```

A

1	0	1
1	2	-2
0	2	6
-1	1	4

ACCESS TO COMPONENTS

- Access to an element of an matrix A can be done by A[i][j], where i and j are the correponding row and column value of the required element.
- The value of *i* should be ≥ 0 and ≤R-1 (where R is the row dimension of the matrix A), and the value of *j* should be ≥ 0 and ≤C-1 (where R is the row dimension of the matrix A).
- The values of *i* and *j* could be:
  1. Constante values
  2. Variables
  3. **Expression**

Example :

```
int A[4][3]={{1,0,1},{1,2,-2},{0,2,6}, {-1,1,4}};
int i=0,j=1;
A[i+1][j]=-30; i=j*2;
A[i+1][j-1]= A[i+1][j]+2*A[j+1][i+1];
 cout<<A[i+1][j-1];
```

ACCESS TO COMPONENTS

- The values of *i* and *j* should be of **integer** types (integer types: int, short, char, long), but not float, double, bool,....

Example :

```
int A[4][3]={{1,0,1},{1,2,-2},{0,2,6}, {-1,1,4}};
float i=0,j=1;
A[i+1][j]=-30; ERROR
i=j*2;
A[i+1][j-1]= A[i+1][j]+2*A[j+1][i+1]; ERROR
 cout<<A[i+1][j-1]; ERROR
```

CHAPTER HEADLINES

- **Part 1**
  - Definition
  - Declaration
  - Initialization
  - Storage
  - Automatic reservation
- **Part 2**
  - Access to components
  - **Displaying the content of an array**
  - Assignment with values from keyboard
- Part 3
  - Exercises

DISPLAYING THE CONTENT OF A MATRIX

- When working with two-dimensional arrays, we will use two indices (i.e: i and j), with two for structures, nested loops, to browse the rows and columns of the arrays.

DISPLAYING THE CONTENT OF A MATRIX

- When working with two-dimensional arrays, we will use two indices (i.e: i and j), and the for structure, often nested, to browse the rows and columns of the arrays.

```
#include<iostream>
using namespace std;
int main()
{
    int A[5][3]={{1,2,3},{4,5,6},{7},{8}}, i,j;
    for(i=0;i<5;i++)
    {
        for(j=0;j<3;j++)
            cout<<A[i][j]<<",";
        cout<<"\n";
    }
    cout<<"end";
return 0;
```

```
1,2,3,
4,5,6,
7,0,0,
8,0,0,
0,0,0,
end
--------------------------------
Process exited after 8.103 seconds with return value 0
Press any key to continue . . .
```

CHAPTER HEADLINES

- **Part 1**
  - Definition
  - Declaration
  - Initialization
  - Storage
  - Automatic reservation
- **Part 2**
  - Access to components
  - Displaying the content of an array
  - **Assignment with values from keyboard**
- Part 3
  - Exercises

ASSIGNMENT WITH VALUES FROM KEYBOARD

- When working with two-dimensional arrays, we will use two indices (i.e: i and j), with two for structures, nested loops, to browse the rows and columns of the arrays.

ASSIGNMENT WITH VALUES FROM

```
int main()
{
    int A[5][3],i,j;
    //assign values from keyboard
    for(i=0;i<5;i++)
        for(j=0;j<3;j++)
        {
            cout<<" enter value of A["<<i<<"]["<<j<<"]:";
            cin>>A[i][j];
        }
    //Display the values of the elements
    for(i=0;i<5;i++)
    {
        for(j=0;j<3;j++)
            cout<<A[i][j]<<"\t";
    cout<<"\n";
    }
return 0;
}
```

```
 enter value of A[0][0]:14
 enter value of A[0][1]:12
 enter value of A[0][2]:0
 enter value of A[1][0]:-6
 enter value of A[1][1]:5
 enter value of A[1][2]:9
 enter value of A[2][0]:-2
 enter value of A[2][1]:1
 enter value of A[2][2]:3
 enter value of A[3][0]:4
 enter value of A[3][1]:5
 enter value of A[3][2]:8
 enter value of A[4][0]:10
 enter value of A[4][1]:13
 enter value of A[4][2]:15
14      12      0
-6      5       9
-2      1       3
4       5       8
10      13      15

--------------------------------
Process exited after 27.24 seconds with return value 0
Press any key to continue . . .
```

CHAPTER HEADLINES

- **Part 1**
  - Definition
  - Declaration
  - Initialization
  - Storage
  - Automatic reservation
- Part 2
  - Access to components
  - Displaying the content of an array
  - Assignment with values from keyboard
- Part 3
  - **Exercises**

EXERCISE 1:

Write a program that reads the R and C dimensions of a two-dimensional array T of type int (maximum dimensions: 50 rows and 50 columns). Fill in the array with values entered on the keyboard and display the array and the sum of all its elements.

```cpp
#include<iostream>
using namespace std;
int main()
{
    int T[50][50], i,j,R,C,S;
    do
    {
        cout<<" enter the number of rows 0<R<=50: ";
        cin>>R;
    }while(R<=0 || R>50);
    do
    {
        cout<<" enter the number of columns 0<C<=50: ";
        cin>>C;
    }while(C<=0 || C>50);

    for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
            cout<<"enter A["<<i<<"]["<<j<<"]: ";
            cin>>T[i][j];
        }
    cout<<" the values of the array are :\n";
    for(S=0,i=0;i<R;i++)
    {
        for(j=0;j<C;j++)
        {
            S=S+T[i][j];
            cout<<T[i][j]<<"\t";
        }
        cout<<"\n";
    }
    cout<<"Sum of all the components = "<<S;
return 0;
}
```

```
enter A[0][1]: 1
enter A[0][2]: 12
enter A[1][0]: -6
enter A[1][1]: 54
enter A[1][2]: -2
enter A[2][0]: 35
enter A[2][1]: -4
enter A[2][2]: -22
enter A[3][0]: 36
enter A[3][1]: 18
enter A[3][2]: 24
enter A[4][0]: 23
enter A[4][1]: -25
enter A[4][2]: 26
 the values of the array are :
15      1       12
-6      54      -2
35      -4      -22
36      18      24
23      -25     26
Sum of all the components = 185
--------------------------------
Process exited after 72.83 seconds with return value 0
Press any key to continue . . .
```

```
 enter the number of rows 0<R<=50: 3
 enter the number of columns 0<C<=50: 2
enter A[0][0]: 1
enter A[0][1]: 0
enter A[1][0]: 0
enter A[1][1]: -1
enter A[2][0]: 1
enter A[2][1]: 1
 the values of the array are :
1       0
0       -1
1       1
Sum of all the components = 2
--------------------------------
Process exited after 26.32 seconds with return value 0
Press any key to continue . . .
```

EXERCISE 2:

Write a program that reads the R and C dimensions of a two-dimensional array T of type int (maximum dimensions: 50 rows and 50 columns). Fill in the array with values entered on the keyboard and display the array and the sum of each row of the array.

```cpp
#include<iostream>
using namespace std;
int main()
{
    int T[50][50], i,j,R,C,S;
    do
    {
        cout<<" enter the number of rows 0<R<=50: ";
        cin>>R;
    }while(R<=0 || R>50);
    do
    {
        cout<<" enter the number of columns 0<C<=50: ";
        cin>>C;
    }while(C<=0 || C>50);

    for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
            cout<<"enter A["<<i<<"]["<<j<<"]: ";
            cin>>T[i][j];
        }
    cout<<" the values of the array are :\n";

    for(i=0;i<R;i++)
    {
        for(S=0,j=0;j<C;j++)
        {
            S=S+T[i][j];
            cout<<T[i][j]<<"\t";
        }
        cout<<"Sum of the row = "<<S <<"\n";
    }

return 0;
}
```

```
 enter the number of rows 0<R<=50: 3
 enter the number of columns 0<C<=50: 4
enter A[0][0]: 1
enter A[0][1]: 2
enter A[0][2]: 3
enter A[0][3]: 4
enter A[1][0]: 5
enter A[1][1]: 6
enter A[1][2]: -1
enter A[1][3]: -2
enter A[2][0]: -3
enter A[2][1]: -4
enter A[2][2]: -5
enter A[2][3]: -6
 the values of the array are :
1       2       3       4       Sum of the row = 10
5       6       -1      -2      Sum of the row = 8
-3      -4      -5      -6      Sum of the row = -18

--------------------------------
Process exited after 28.97 seconds with return value 0
Press any key to continue . . .
```

EXERCISE 3

Write a program that reads the R and C dimensions of a two-dimensional array T of type int (maximum dimensions: 50 rows and 50 columns). Fill in the array with **positive values** entered on the keyboard and display the array.

EXERCISE 3

Write a program that reads the R and C dimensions of a two-dimensional array T of type int (maximum dimensions: 50 rows and 50 columns). Fill in the array with **positive values** entered on the keyboard and display the array.

What changes are required to fill in the array with positive values only?

```
for(i=0;i<R;i++)
    for(j=0;j<C;j++)
    {
        cout<<"enter A["<<i<<"]["<<j<<"]: ";
        cin>>T[i][j];
    }
```

EXERCISE 3

Write a program that reads the R and C dimensions of a two-dimensional array T of type int (maximum dimensions: 50 rows and 50 columns). Fill in the array with **positive values** entered on the keyboard and display the array.

Answer: We need to use do-while.

```
for(i=0;i<R;i++)
    for(j=0;j<C;j++)
    {
        do
        {
            cout<<"enter A["<<i<<"]["<<j<<"]: ";
            cin>>T[i][j];
        }while(T[i][j]<0);
    }
```

EXERCISE 3: SOLUTION

```
#include<iostream>
using namespace std;
int main()
{
    int T[50][50], i,j,R,C,S;
    do
    {
        cout<<" enter the number of rows 0<R<=50: ";
        cin>>R;
    }while(R<=0 || R>50);
    do
    {
        cout<<" enter the number of columns 0<C<=50: ";
        cin>>C;
    }while(C<=0 || C>50);
    for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
            do
            {
                cout<<"enter A["<<i<<"]["<<j<<"]: ";
                cin>>T[i][j];
            }while(T[i][j]<0);
        }
   cout<<" the values of the array are :\n";
   for(i=0;i<R;i++)
   {
       for(j=0;j<C;j++)
           cout<<T[i][j]<<"\t";
       cout<<"\n";
   }
return 0;
}
```

```
 enter the number of rows 0<R<=50: 3
 enter the number of columns 0<C<=50: 3
enter A[0][0]: -2
enter A[0][0]: -4
enter A[0][0]: -45
enter A[0][0]: 0
enter A[0][1]: 2
enter A[0][2]: 16
enter A[1][0]: -4
enter A[1][0]: -3
enter A[1][0]: 12
enter A[1][1]: 14
enter A[1][2]: 0
enter A[2][0]: 1
enter A[2][1]: 2
enter A[2][2]: 3
 the values of the array are :
0       2       16
12      14      0
1       2       3

--------------------------------
Process exited after 31.48 seconds with return value 0
Press any key to continue . . .
```

EXERCISE 4

Write a program that zeros the elements of the main diagonal of the square matrix T

```cpp
#include<iostream>
using namespace std;
int main()
{
    int T[5][5]={{1,2,3,4,5},
                 {6,7,8,9,10},
                  {11,12,13,14,15},
                  { 16,17,18,19,20},
                  {21,22,23,24,25}},i,j;
    cout<<" the values of the array are :\n";
    for(i=0;i<5;i++)
    {
        for(j=0;j<5;j++)
            cout<<T[i][j]<<"\t";
        cout<<"\n";
    }
    for(i=0;i<5;i++)
        T[i][i]=0;
    cout<<"\n after change, the values of the array are :\n";
    for(i=0;i<5;i++)
    {
        for(j=0;j<5;j++)
            cout<<T[i][j]<<"\t";
        cout<<"\n";
    }
return 0;
}
```

```
 the values of the array are :
1       2       3       4       5
6       7       8       9       10
11      12      13      14      15
16      17      18      19      20
21      22      23      24      25

 after change, the values of the array are :
0       2       3       4       5
6       0       8       9       10
11      12      0       14      15
16      17      18      0       20
21      22      23      24      0

--------------------------------
Process exited after 7.466 seconds with return value 0
Press any key to continue . . .
```

EXERCISE 5

Write a program that constructs and displays a unitary square matrix U of dimension N. A unitary matrix is a matrix, such that:

$$u_{ij} = \begin{cases} 1, if\ i = j \\ 0, if\ i \neq j \end{cases}$$

```
int main()
{
    int U[50][50]={0},N,i,j;
    do
    {
        cout<<" enter the dimension N of the unitary square matrix: ";
        cin>>N;
    }while(N<=0 ||N>50);
    for(i=0;i<N;i++)
        U[i][i]=1;
    cout<<"unitary square matrix components values are :\n";
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
            cout<<U[i][j]<<"\t";
        cout<<"\n";
    }
return 0;
}
```

```
 enter the dimension N of the unitary square matrix: 4
unitary square matrix components values are :
1       0       0       0
0       1       0       0
0       0       1       0
0       0       0       1

--------------------------------
Process exited after 1.489 seconds with return value 0
Press any key to continue . . .
```

EXERCISE 6

Write a program that reads the R and C dimensions of a two-dimensional array T of type float (maximum dimensions: 50 rows and 50 columns). Fill in the array with values entered on the keyboard, search the maximum values stored in the array. Display the array and the maximum value.

```cpp
#include<iostream>
using namespace std;
int main()
{
    int T[50][50], i,j,R,C,max;
    do
    {
        cout<<" enter the number of rows 0<R<=50: ";
        cin>>R;
    }while(R<=0 || R>50);
    do
    {
        cout<<" enter the number of columns 0<C<=50: ";
        cin>>C;
    }while(C<=0 || C>50);
    for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
            cout<<"enter A["<<i<<"]["<<j<<"]: ";
            cin>>T[i][j];
        }
    cout<<" array components are: \n";
    for(i=0;i<R;i++)
    {
        for(j=0;j<C;j++)
            cout<<T[i][j]<<"\t";
        cout<<"\n";
    }
    max=T[0][0];
    for(i=0;i<R;i++)
    {
        for(j=0;j<C;j++)
            if(max<T[i][j])
                max=T[i][j];
    }
    cout<<"max = "<<max;
return 0;
}
```

```
 enter the number of rows 0<R<=50: 3
 enter the number of columns 0<C<=50: 4
enter A[0][0]: 1
enter A[0][1]: 2
enter A[0][2]: 3
enter A[0][3]: 4
enter A[1][0]: -2
enter A[1][1]: 16
enter A[1][2]: 5
enter A[1][3]: 24
enter A[2][0]: 13
enter A[2][1]: 17
enter A[2][2]: 12
enter A[2][3]: -25
 array components are:
1       2       3       4
-2      16      5       24
13      17      12      -25
max = 24
--------------------------------
Process exited after 26 seconds with return value 0
Press any key to continue . . .
```

EXERCISE 7:

- By multiplying a matrix A of dimensions R and C with a matrix B of dimensions C and P, a matrix MULT of dimensions R and P is obtained: A (R, C) * B (C, P) = MULT (R, P). The multiplication of two matrices is done by multiplying the components of the two matrices rows by columns:

$$MULT_{ij} = (\sum_{k=0}^{C-1} a_{ik} \times b_{kj})$$

Write a program that performs the multiplication of two matrices A and B. The result of the multiplication will be stored in a third matrix MULT which will then be displayed.

Recap:

$$\begin{pmatrix} a & b & c \\ e & f & g \\ h & i & j \\ h & l & m \end{pmatrix} \times \begin{pmatrix} p & q \\ r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a \times p + b \times r + c \times t & a \times q + b \times s + c \times u \\ e \times p + f \times r + g \times t & e \times q + f \times s + g \times u \\ h \times p + i \times r + j \times t & h \times q + i \times s + j \times u \\ k \times p + l \times r + m \times t & k \times q + l \times s + m \times u \end{pmatrix}$$

EXERCISE 7:

Write a program that performs the multiplication of two matrices A and B. The result of the multiplication will be stored in a third matrix MULT which will then be displayed.

$$MULT_{ij} = (\sum_{k=0}^{C-1} a_{ik} \times b_{kj})$$

```cpp
#include<iostream>
using namespace std;
int main()
{
    int A[3][4]={{1,2,3},{4,5,6},{5,-5}}, B[4][2]={{0,1},{1,-1},{0,1}},MULT[3][2]={0},i,j,k;
    cout<<"A components are: \n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
            cout<<A[i][j]<<"\t";
        cout<<"\n";
    }
    cout<<"B components are: \n";
    for(i=0;i<4;i++)
    {
        for(j=0;j<2;j++)
            cout<<B[i][j]<<"\t";
        cout<<"\n";
    }
    for(i=0;i<3;i++)
        for(j=0;j<2;j++)
            for(k=0;k<4;k++)
                MULT[i][j]=MULT[i][j]+A[i][k]*B[k][j];
    cout<<"MULTI array components are: \n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<2;j++)
            cout<<MULT[i][j]<<"\t";
        cout<<"\n";
    }
return 0;
}
```

```
A components are:
1       2       3       0
4       5       6       0
5       -5      0       0
B components are:
0       1
1       -1
0       1
0       0
MULTI array components are:
2       2
5       5
-5      10

--------------------------------
Process exited after 1.007 seconds with return value 0
Press any key to continue . . .
```