PROGRAMMING LANGUAGES

Laboratory 9

MATRICES
(part II)

I. FUNDAMENTALS

Initializing arrays

Matrices, like simple variables, can be initialized. Global arrays are initialized by their definitions. Static and automatic arrays are initialized by their statements.

Note: In “C” language, an initialization is done by constant expression.

Global and static arrays are initialized at compilation. Therefore, their elements have the values that were established at the initialization phase.
The automatic arrays are initialized in execution phase, every time is called the function they are declared in. There are some versions of “C” language in which cannot initialize automatic arrays.

- A one-dimensional array (vector) is initialized by a construction like:

```
type name[val] = { ec0, ec1, ec2, ..., eci };
```

or:

```
static type name[val] = { ec0, ec1, ec2, ..., eci };
```

if the matrix is static.

By val, ec0, ec1, ... eci we mean constant expressions.

Note:
a. It is necessary that i <= val - 1. If i >= val, there is an error.

b. If i < val - 1, then:
name[i+1], name[1+2], ..., name[val-1]
remain uninitialized.

The uninitialized elements of an global (or static) array have zero baselines by default. The uninitialized elements of an automatically array (local variable) have undefined initial values.
When initialize all the elements of a one-dimensional array, you can omit the term in square brackets which defines the number of array elements.

Therefore the expressions:

```
type name[] = { ec0, ec1, ec2, ..., ecn };
```

and:

```
static type name[] = { ec0, ec1, ec2, ..., ecn };
```

are correct, and in both cases the matrices have n+1 elements, and the element name[i] is initialized to eci.
If the initialization expression has a type different from the matrix type, then its value is converted to the array type before being assigned to the corresponding element.

<u>Note</u>: In a statement of one-dimensional array, ones can omit the expression that defines the number of array elements in the following cases:

1. Declaration contains constant expression that initializes each element of the array.
2. The statement refers to an one-dimensional array which is a formal parameter:

```
void function(int n, double tab[])
```

3. Statement of an extern one-dimensional matrix:

```
extern int tab[];
```

Examples:

1. `int sequence[5] = { 1, 6, 3, 9, 7};`

2. `int sequence[] = { 1, 6, 3, 9, 7};`
   This definition is identical to the previous one.

3. `double tab[10] = { 9, 3, 4 };`
   The first three elements of the matrix *tab* are initialized with:
   tab[0] = 9
   tab[1] = 3
   tab[2] = 4

The remaining elements of the array will have the initial value 0 if *tab* is a global array, or an unpredictable value if the array is an automatic one.
Since the char arrays are often used, it was introduced a simplification for the initialization of type of arrays.

Thus, for char arrays, you can use one of the following forms:

```
char name[val] = sir;
```

or:

```
static char name[val] = sir;
```

where ***sir*** is a sequence of characters delimited by quotation marks.

Thus, the elements of *name* array are initialized with the values of ASCII code of each character and after the last character there will be automatically attached the NULL character (\0).

- Multidimensional arrays can be initialized by using analogous construction such as:

```
type name[n][m] =
      {
        { ec11, ec12, ..., ec1m },
        { ec21, ec22, ..., ec2m },
        ....
        { ecn1, ecn2, ..., ecnm }
      };
```

where: n, m, ecij (i=1,2,...,n si j=1,2,...,m) – are constant expressions.

The number of constant expressions may be lower than m in any of the braces corresponding to the n rows of the two-dimensional array. For static arrays, the statement is preceded by the keyword static.

For the static arrays, the declaration is preceded by the key word *static*.

Note: In a multidimensional array declaration only the constant term in the first square brackets may be omitted. It may be omitted in one of the following cases:

1.The statements or definitions containing initialization;

2.The statements of function parameters;

3. Declarations of extern.

<u>Examples</u>:

```
1. int mat[3][4] =
      {
        { 3, 4, 5, 2 },
        { 5, 8, 1, 3 },
        { 1, 9, 3, 2 }
      };
```

The matrix will be initialized as follows:

```
mat[0][0]=3  mat[0][1]=4  mat[0][2]=5  mat[0][3]=2
mat[1][0]=5  mat[1][1]=8  mat[1][2]=1  mat[1][3]=3
mat[2][0]=1  mat[2][1]=9  mat[2][2]=3  mat[2][3]=2
```

2. int mat[][4] =
```
        {
          { 3, 4, 5, 2 },
          { 5, 8, 1, 3 },
          { 1, 9, 3, 2 }
        };
```

This statement is identical in effect to that of the previous example.

3. float t[][3] =
```
        {
          { 0, 1 },
          { -3 },
          { 1, 2, 4}
        };
```

II. ASSIGNMENT WORKFLOW

1. Make a matrix multiplication program that uses the following matrices:

$$TAB = \begin{bmatrix} -1 & 0 & 1 & -1 \\ -1 & 0 & 1 & 2 \\ 10 & 20 & 40 & 30 \end{bmatrix} \quad \text{and} \quad MC = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

A possible solution:

```
# include<stdio.h>

void main()

{
double tab[3][4]=
      { {-1, 0, 1, -1},
        {-1, 0, 1, 2},
        {10, 20, 40, 30}
      };
double mc[4]={1,2,3,4};
double rez[3];
int i, j;

for(i=0; i<3; i++)
      {
            rez[i]=0;
            for(j=0; j<4; j++)
                rez[i] += tab[i][j]*mc[j];
            printf("\n rez[%d]=%g", i, rez[i]);
           }
      getch();
      }
```

2. Modify the above program so that the matrices to be defined as global variables. Delete expressions that can be omitted from the definition of matrices.

3. Modify the above program so as to display the matrices and the matrix multiplication result.

4. Make a program that performs a multiplication of a matrix m x n with a column matrix n x 1 column and then display the result. Matrix elements (of type double) will be introduced from the keyboard.

5. In the above program extract the matrix multiplication function from the body of function "main" (by using procedural programming).

III. SOLUTIONS:

2. L9_2.C

```
# include<stdio.h>

double tab[][4]=
        { {-1, 0, 1, -1},
          {-1, 0, 1, 2},
          {10, 20, 40, 30}
        };
double mc[]={1,2,3,4};
double rez[3];

void main()

{

int i, j;

for(i=0; i<3; i++)
        {
          rez[i]=0;
          for(j=0; j<4; j++)
                rez[i] += tab[i][j]*mc[j];
          printf("\n rez[%d]=%g", i, rez[i]);
         }
getch();
}
```

3. L9_3.C

```
# include<stdio.h>

double tab[3][4]=
        { {-1, 0, 1, -1},
          {-1, 0, 1, 2},
          {10, 20, 40, 30}
        };
double mc[4]={1,2,3,4};
double rez[3];


void main()

{

int i, j;
printf("\n\n The result of matrix multiplication \n\nTAB=");
for(i=0;i<3;i++)
      {
        for(j=0; j<4; j++)
                printf("\t%g", tab[i][j]);
        printf("\n");
      }
printf("\n with column matrix \n\nMC=");
for(j=0; j<4; j++)
        printf("\t%g\n",mc[j]);
printf("\n is \n\nREZ=");
for(i=0; i<3; i++)
        {
          rez[i]=0;
          for(j=0; j<4; j++)
                rez[i] += tab[i][j]*mc[j];
          printf("\t%g", rez[i]);
         }
printf("\n\n");
getch();
}
```

4. L9_4.C

```
# include<stdio.h>
double tab[20][20];
double mc[20];
double rez[20];



void main()
{
int i, j, m, n;

printf("\n Enter the number of lines in the first matrix: ");
scanf("%d",&m);
printf("\n Enter the number of columns in the first matrix: ");
scanf("%d",&n);
printf("\n Enter matrix elements: \n");
for(i=0; i<m; i++)
        {
          for(j=0; j<n; j++)
             {
              printf("\ttab[%d][%d]=", i, j);
              scanf("%lf",&tab[i][j]);
             }
          printf("\n");
         }

printf("\n Enter column matrix elements:\n");

for(j=0; j<n; j++)
        {
          printf("\tmc[%d]=", j);
          scanf("%lf",&mc[j]);
        }

printf("\n The result of multiplying is:\n");

for(i=0; i<m; i++)
        {
          rez[i]=0;
          for(j=0; j<n; j++)
                rez[i] += tab[i][j]*mc[j];
          printf("\t rez[%d]=%g", i, rez[i]);
         }
getch();
}
```

5. L9_5.C

```
# include<stdio.h>

double tab[20][20];
double mc[20];
double rez[20];


void matrez(int m, int n, double m1[][],
                double m2[], double prod[]);


void main()

{
int i, j, m, n;

printf("\n Enter the number of lines in the first matrix: ");
scanf("%d",&m);
printf("\n Enter the number of columns in the first matrix: ");
scanf("%d",&n);
printf("\n Enter matrix elements: \n");

for(i=0; i<m; i++)
        {
          for(j=0; j<n; j++)
             {
              printf("\ttab[%d][%d]=", i, j);
              scanf("%lf",&tab[i][j]);
             }
          printf("\n");
         }
printf("\n Enter column matrix elements:\n");

for(j=0; j<n; j++)
        {
          printf("\tmc[%d]=", j);
          scanf("%lf",&mc[j]);
        }

printf("\n The result of multiplying is:\n");
matrez(m, n, tab, mc, rez);

printf("\nREZ = [\t");
for(i=0; i<m; i++)
        printf("%g\t", rez[i]);
printf("]\n");
getch();
}

void matrez(int m, int n, double m1[20][20], double m2[20],
                        double prod[20])
{
 int i, j;
 for(i=0; i<m; i++)
        {
          prod[i]=0;
          for(j=0; j<n; j++)
                prod[i] += m1[i][j]*m2[j];
         }
}
```