

Cairo University Faculty of Computers and Information Computer Science Department



Programming-1 CS112 2018/2019

Multi-dimensional arrays

Dr. Amin Allam

1 2D arrays

Arrays can have multiple dimensions. A 2D array is an array of 1D arrays. For example, suppose that a business man is interested in the sales of 4 devices (phones, cameras, keyboards, monitors) during each of the last 3 months.

```
int a[3][4]; // An array of 3 arrays, a[i] is the sales of 4 devices for month i

// a[3][4] represents a matrix with 3 rows and 4 columns

a[0][0]=5; a[0][1]=7; // He sold 5 phones and 7 cameras in the first month

a[1][2]=4; a[1][3]=8; // He sold 4 keyboards and 8 monitors in the second month

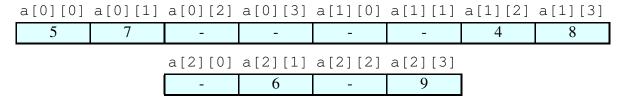
a[2][1]=6; a[2][3]=9; // He sold 6 phones and 9 monitors in the third month

// ... etc
```

Logically, the variables represent the following matrix which have 3 rows and 4 columns:

a[0][0]=5	a[0][1]=7	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]=4	a[1][3]=8
a[2][0]	a[2][1]=6	a[2][2]	a[2][3]=9

Physically, the variables will be *contiguously* allocated in memory as follows:



Note that int a[3][4] is a 2D-array which contains 3 1D-arrays: a[0], a[1] and a[2]. Each 1D-array of them is composed of 4 integers. a[0] integers come first in memory, a[1] integers come after, and a[2] integers come last.

2 Simulating 2D arrays

2D arrays can be simulated by 1D arrays using the following function:

The following examples uses a 1D array b[12] that simulates the 2D array a[3][4] of the previous example.

```
int b[3*4];  // An array of 12 elements simulates a 2D array of 3 rows and 4 columns
b[Ind(0,0,4)]=5; b[Ind(0,1,4)]=7; // b[0]=1; b[1]=7;
b[Ind(1,2,4)]=4; b[Ind(1,3,4)]=8; // b[6]=4; b[7]=8;
b[Ind(2,1,4)]=6; b[Ind(2,3,4)]=9; // b[9]=6; b[11]=9;
// ... etc
```

The variables will be *contiguously* allocated in memory as follows:

_ k	[0]c	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]
	5	7	-	-	-	-	4	8
			b[8]	b[9]	b[10]	b[11]		
			-	6	-	9		

3 Multi-dimensional arrays

Arrays can have any number of dimensions. For example, int a[3][4][2] is a 3D-array which contains 3 2D-arrays: a[0], a[1] and a[2]. Each 2D-array is composed of 4 1D-arrays. For example, the 2D-array a[0] is composed of the 4 1D-arrays: a[0][0], a[0][1] and a[0][2]. Each 1D-array is composed of 2 integers.

A 2D array can be initialized as follows:

```
1 int a[3][4]={{11,12,13,14},{21,22,23,24},{31,32,33,34}};
```

a[0][0]=11	a[0][1]=12	a[0][2]=13	a[0][3]=14
a[1][0]=21	a[1][1]=22	a[1][2]=23	a[1][3]=24
a[2][0]=31	a[2][1]=32	a[2][2]=33	a[2][3]=34

It is possible to leave the first bracket empty in initialization and when passing the array as parameter to a function. However, for any multi-dimensional array, it is not possible to leave any other bracket other than the first one empty, because the compiler internally simulates them as 1D-arrays as we did in the previous section (the Ind() function needs to know the number of columns).

```
char a[][7]={"sea", "desert", "air"}; // Array of 3 strings
// Each string contains at most 6 chars
```

A 3D array can be initialized as follows:

4 Examples

1. Write a function that takes a 3x5 matrix and a 5x3 matrix that hold variables of type double and assigns the second matrix to the transpose of the first one.

2. Write a function that takes a 3x3 matrix that holds variables of type int and returns true only if the matrix is symmetric.

```
bool IsSymmetric(const int M[3][3])
1
2
  {
3
      int i, j;
      for(i=0; i<3; i++)</pre>
4
5
          for(j=0; j<i; j++)</pre>
6
             if (M[i][j]!=M[j][i])
7
                 return false;
8
      return true;
9
```

Note that in the previous examples we used the const modifier to indicate that the function is not going to modify the values of M[][]. This improves code readability. Also, any attempt to modify values of M[][] inside this function will cause a compiler error.