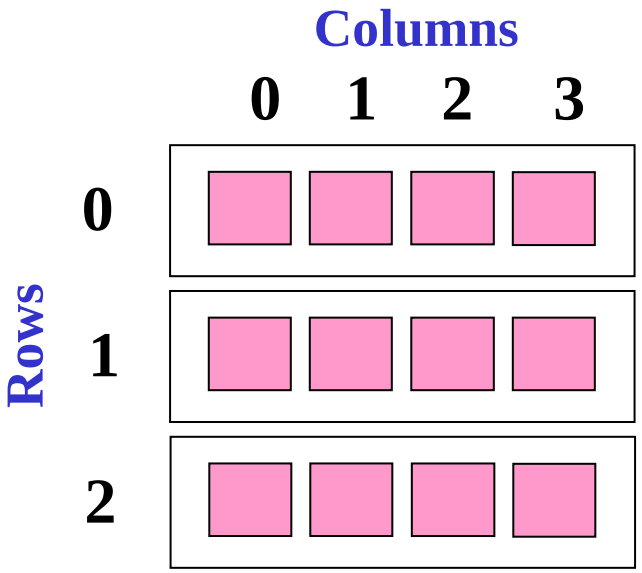
Lecture 10:

The Matrix: 2-dimensional Arrays

Prof. Shervin Shirmohammadi
University of Ottawa

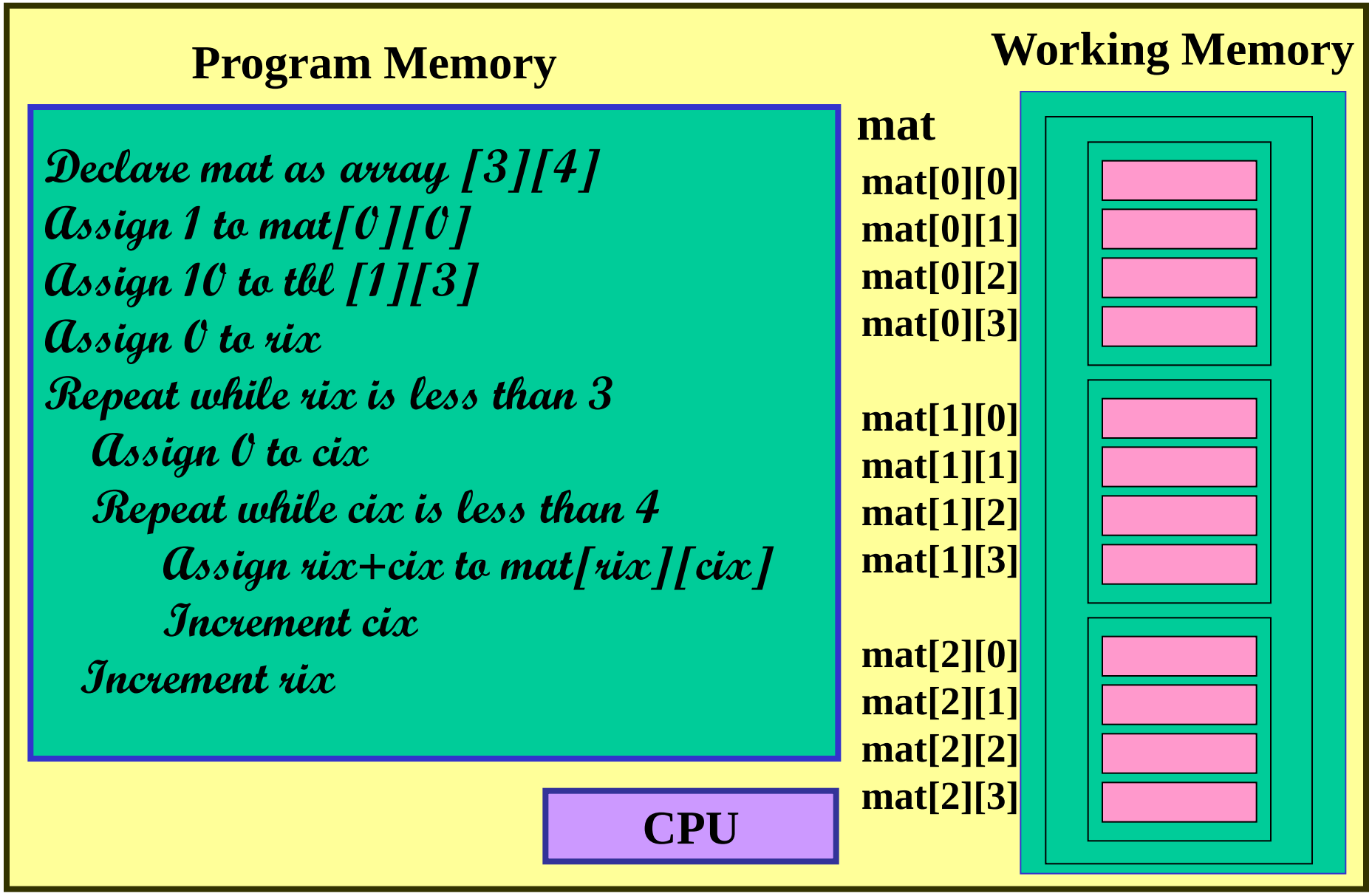
The 2 dimensional array – the matrix

- The 2 dimensional array, a matrix, is essentially an array of arrays
  - Each row in the matrix is a 1-dimensional array
  - The elements in the same positions of the row arrays make up the matrix columns.



- The matrix, like the 1-D array, occupies contiguous memory locations.

The matrix in memory



Example

- Write a program that stores the marks (in percentage, for example 80, 95, etc.) of 200 students in 7 courses. Then, for the first 3 courses, assign them to 80, and for the remaining 4 courses, assign them to 75. Finally, print course number 2 and number 6 for the first 50 students.

The matrix – an array of arrays

- The two dimensional array is often used to store data arranged in a row-column form, i.e. as matrices.
  - Note the organization in memory – it occupies contiguous space just like the array
- A name (like the case of the array) is used to reference the matrix
  - Like the case of the array, the name corresponds to the address of the first element in the matrix.
- The elements in a two dimensional array are accessed by citing the name of the array followed by **two** indices, each inserted between [ ] and [ ].
  - The first index specifies the row and the second index specifies the column.
- What do you think the name of a matrix with a single index represents?
  - E.g. mat[1]

Two Dimensional Arrays in C

- The ANSI standard for C specifies that a C compiler must support at least **12 dimensions** (this means 12 indices)!
- A two dimensional array `a` of `int`'s, having 3 rows and 4 columns (called a 3×4 Matrix) is visualized as:

Matrix `a`

	col 0	col 1	col 2	col3
row 0	2	-1	11	8
row 1	0	4	-5	-9
row 2	3	10	-7	-13

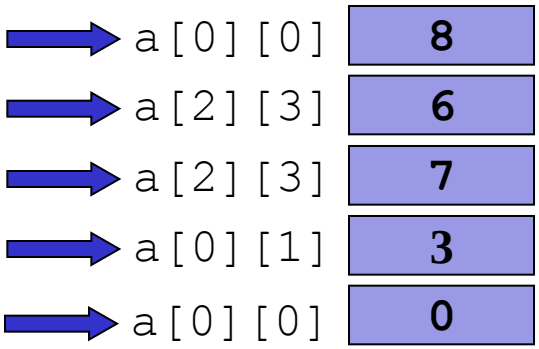
- **In C arrays are always indexed from 0 on.**
  - ➔ The first row is row 0 and the first column is column 0.
- In reference to the array `a` of `int`'s on the previous slide:
  - element `a[0][0]` contains `2`,
  - element `a[2][3]` contains `-13`,
  - element `a[1][2]` contains `-5`.

2	**-1**	**11**	**8**
0	**4**	**-5**	**-9**
3	**10**	**-7**	**-13**

- An element of a two dimensional array is used in the same manner as a variable.
  - It can be used to the left of an assignment operator to store a value in the element.
  - It can be used in an arithmetic or logical expression.

Assume that `a` is an array of `int`'s

E.g.:
```
a[0][0] = 5 + 3;
a[2][3] = a[0][1-1] - 2;
a[2][3]++;
a[0][1] = a[2][3] / 2;
a[0][0] = (a[0][0] <= a[0][1]);
```



- The elements of an array can also be used as an argument in a `printf` or a `scanf`:

E.g.:
```
printf("%d\n", a[0][0]);
scanf("%d", &a[0][1]);
```

Definition and Initialization of Two Dimensional Arrays

- A two dimensional array is defined in a declaration; the declaration specifies:
  - the name of the array,
  - the type of data stored in the array, and
  - the size of the array (number of rows and number of columns).

    E.g.:
    ```
    int a[3][4];
    float y[5][5];
    ```
    Thus, `a` is defined as an array of `int`'s having 3 rows and 4 columns and `y` is an array of `float`'s having 5 rows and 5 columns.
- A two dimensional array can be initialized using nested loops:
  ```
  int a[3][4], row, col;
  for (row=0; row < 3; row++)
     for (col=0; col < 4; col++)
        a[row][col] = 0;
  ```
  - Variables of type `int` called `i` and `j` are also commonly used for accessing the rows and columns of a two dimensional array, respectively.

- We can also initialize an array in the declaration in which it is defined.
  - The initial values are listed row-wise, where each row is grouped using { }; the rows are then separated by commas, enclosed in { } and assigned to the array definition.

    E.g.: `int a[3][4]={{2,-1,11,8},{0,4,-5,-9},{3,10,-7,-13}};`

    row 0 row 1 row 2
  - If the { } delimiting the rows are omitted, then the compiler still initializes row-wise starting from the first row to the last.

    E.g.: `int a[3][4]={2,-1,11,8,0,4,-5,-9,3,10,-7,-13};`

    The above stores exactly the same initial values in the array as in the previous example but it is not as readable.
- If values are missing in the initialization list, then the missing values will be assumed to be zero.

E.g.:

```
int a[3][4]={0};
int a[3][4]={{2,-1,11},{0,4,-5},{3,10,-7}};
int a[3][4]={2,-1,11,8,0,4,-5,-9};
```

In the first case the entire array is set to zero.

In the second case, the last column is set to zero.

In the third case, the last row is set to zero.

- The number of rows of a two dimensional array can also be defined via the initialization list.
  - If the number of rows of a two dimensional array is not stated explicitly, but the declaration contains an initialization list, then the number of rows of the array is taken as being equal to the number of rows in the initialization list.

    E.g.: `int a[][4]={{2,-1,11,8},{0,4,-5,-9},{3,10,-7,-13}};`

    row 0 row 1 row 2

    Thus `a` is a two dimensional array having 3 rows and 4 columns.
  - It is usually not good practice to omit the number of rows when declaring a two dimensional array!
- The use of descriptive symbolic constants in the definition of two dimensional arrays and in their manipulation is **strongly** encouraged for the same reasons as in the case of the one dimensional arrays.

E.g.:

```
#define NBR_STUDENTS 3
#define NBR_TESTS 4
        ⋮
int a[NBR_STUDENTS][NBR_TESTS], i, j;
for(i=0; i < NBR_STUDENTS; i++)
        for(j=0; j < NBR_TESTS; j++)
                a[i][j]=0;
```

Traversing 2D arrays

- As in the case of a one dimensional array, good programming style and consistency when writing loops that traverse a two-dimensional array will help avoid errors generated by going beyond the upper and lower row and column limits. For example, use symbolic constants and logical expressions similar to:

```
for(row=0; row < NBR_ROWS; row++)
   for(col=0; col < NBR_COLS; col++)
```

Example

- Write a program that prompts the user to enter the following grades table for a class of 10 students and store them in an array.

Student	Lab	Midterm	Final	Assignments	Total
Alex	20	23	19	22	???
Fred	19	22	22	17	???
Maria	16	21	24	19	???
John	25	14	25	22	???

Example

```
// enter grades for all the class
Declare grades[4][5]
Declare i and j
Repeat while i is less than 4
   print "enter the grades of a student " i+1
   Repeat while j is less than 5
            print "enter grade" j+1
            Assign entered grade to grades[i][j]
            Increment j
  assign 0 to grades[i][4]
  Increment i

// compute the total grade for each student
Repeat while i is less than 4
   Repeat while j is less than 4
            add grade[i][j] to grades[i][4]
            Increment j
   Increment i
```