2D Arrays and Matrices

Imagine……

- Class of **5** students
- Each student is enrolled in **3** subjects CSE115, ENG101, EEE101
- Store the marks of all students in all three subjects…

Solution (kind of!)……

- int m1[3] = `{ 78, 83, 82 };`
- int m2[3] = `{ 90, 88, 94 };`
- int m3[3] = `{ 71, 73, 78 };`
- int m4[3] = `{ 97, 96, 95 };`
- int m5[3] = `{ 89, 93, 90 };`

What if we have 40 students in the class?

Efficient solution…

- Store this information in a two-dimensional array
- First dimension: which student 0, 1, 2, 3 or 4
- Second dimension: which subject 0, 1, or 2

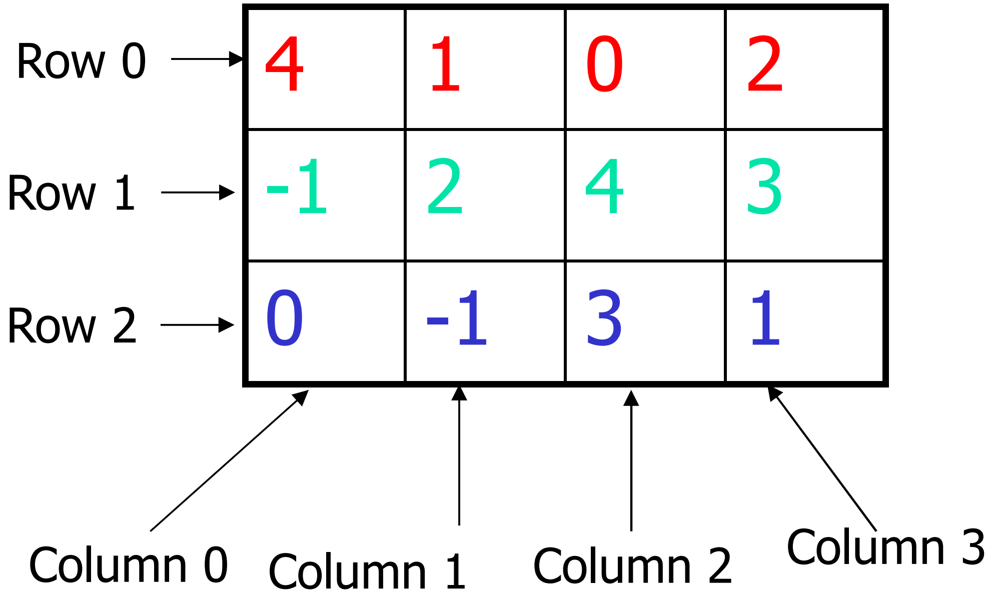
Pictorially

	0	1	**2**
0	78	83	82
1	90	88	94
2	71	73	78
3	97	96	**95**
4	89	93	90

In general a 2D-array

```
datatype array_name[row_size][column_size];
```

```
int matrix[3][4];
```

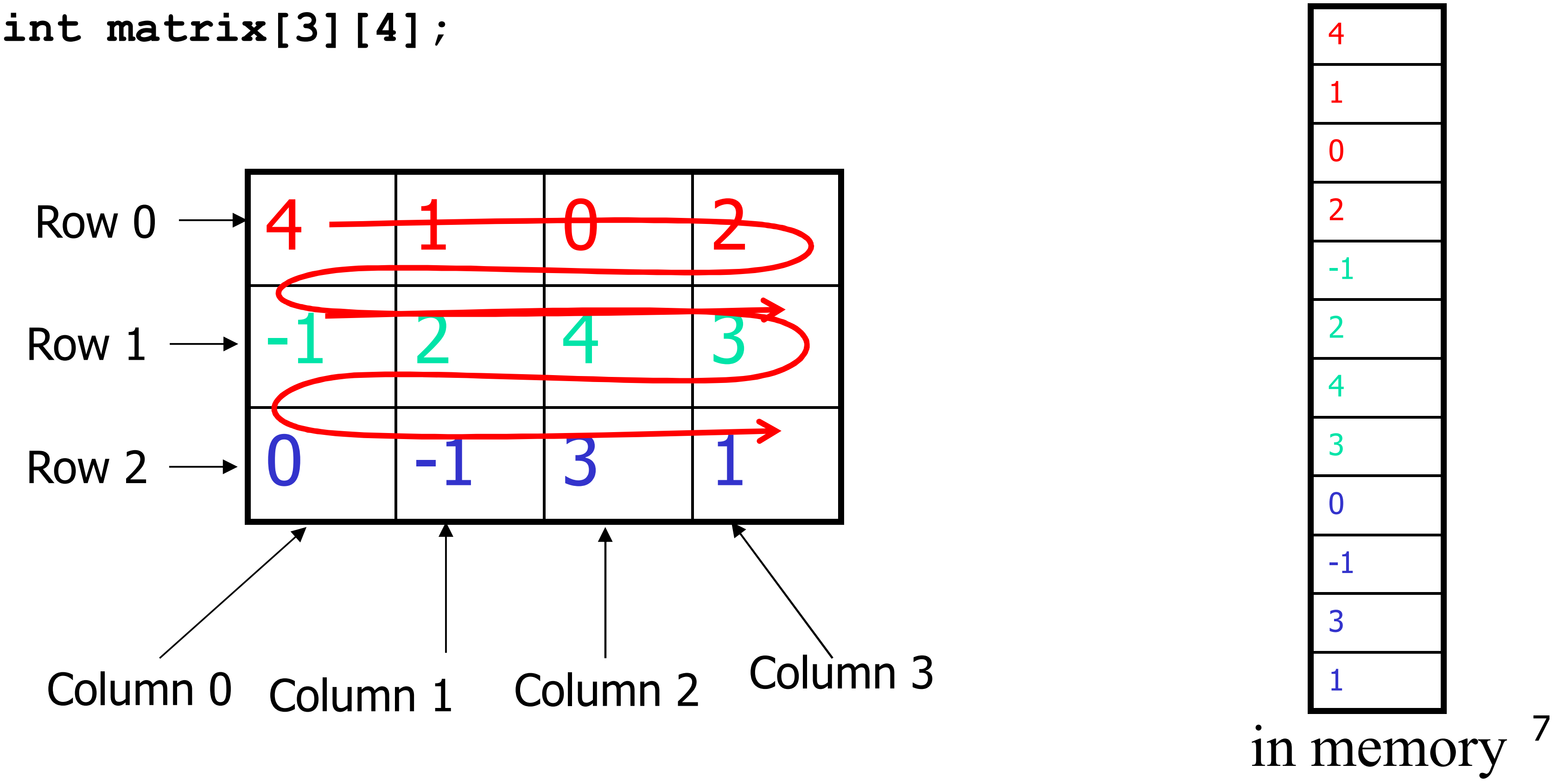


	Column 0	Column 1	Column 2	Column 3
Row 0	4	1	0	2
Row 1	-1	2	4	3
Row 2	0	-1	3	1

In general a 2D-array

```
datatype array_name[row_size][column_size];
```

```
int matrix[3][4];
```



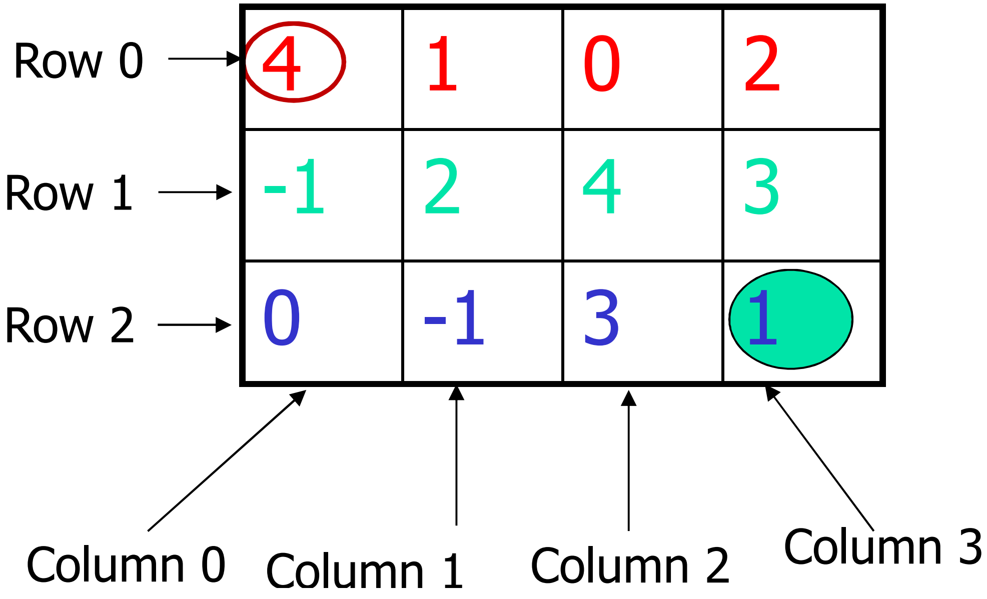
	Column 0	Column 1	Column 2	Column 3
Row 0	4	1	0	2
Row 1	-1	2	4	3
Row 2	0	-1	3	1

in memory
4
1
0
2
-1
2
4
3
0
-1
3
1

Accessing Array Elements

```
int matrix[3][4];
```

- `matrix` has 12 integer elements
- `matrix[0][0]` element in first row, first column
- `matrix[2][3]` element in last row, last column



	Column 0	Column 1	Column 2	Column 3
Row 0	4	1	0	2
Row 1	-1	2	4	3
Row 2	0	-1	3	1

Initialization (1st way)
Initialize when you declare

```
int x[4][4] = {   {2, 3, 7, 2},
                  {7, 4, 5, 9},
                  {5, 1, 6, -3},
                  {2, 5, -1, 3}};
int x[][4] = {    {2, 3, 7, 2},
                  {7, 4, 5, 9},
                  {5, 1, 6, -3},
                  {2, 5, -1, 3}};
```

Initialization (2nd way) Using assignment operator

```
int i, j, matrix[3][4];
for (i=0; i<3; i++)
   for (j=0; j<4; j++)
     matrix[i][j] = i;
```

$$\begin{array}{ccc|c|c|c|c|} & & & \multicolumn{4}{c}{\xrightarrow{\quad j \quad}} \\ & & & 0 & 1 & 2 & 3 \\ \hline & \big\downarrow & 0 & 0 & 0 & 0 & 0 \\ \hline i & & 1 & 1 & 1 & 1 & 1 \\ \hline & & 2 & 2 & 2 & 2 & 2 \\ \hline \end{array}$$

```
matrix[i][j] = j;
```

$$\begin{array}{ccc|c|c|c|c|} & & & \multicolumn{4}{c}{\xrightarrow{\quad j \quad}} \\ & & & 0 & 1 & 2 & 3 \\ \hline & \big\downarrow & 0 & 0 & 1 & 2 & 3 \\ \hline i & & 1 & 0 & 1 & 2 & 3 \\ \hline & & 2 & 0 & 1 & 2 & 3 \\ \hline \end{array}$$

Exercise

- Write the nested loop to initialize a 2D array as follow

0	1	2
1	2	3
2	3	4
3	4	5

```
int i, j, x[4][3];
for(i=0; i<4; i++)
   for(j=0; j<3; j++)
      x[i][j] = i+j;
```

Initialization (3rd way)
By taking input from user

```
int i, j, matrix[3][4];

for (i=0; i<3; i++)
   for (j=0; j<4; j++){
        scanf("%d", &matrix[i][j]);
}
```

Showing content of a 2-Dim Array

```
for (i=0; i<4; i++){
   for (j=0; j<3; j++){
       printf("%d ",m[i][j]);
   }
      printf("\n");
}
```

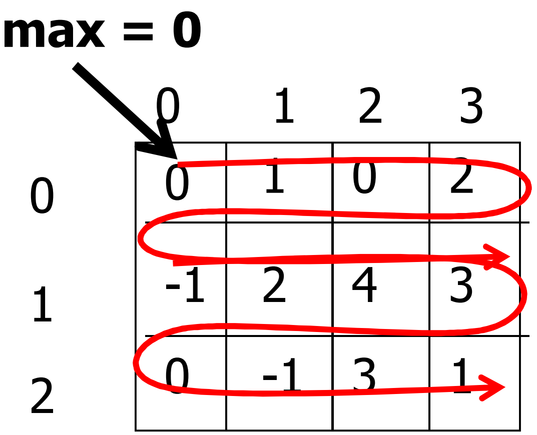
0	1	2
1	2	3
2	3	4
3	4	5

Computations on 2D arrays

Max in 2D

- Find the maximum of *int matrix[3][4]*

```
int max = matrix[0][0];
for (i=0; i<3; i++)
   for (j=0; j<4; j++){
      if (matrix[i][j] > max){
            max = matrix[i][j];
      }
}
```



max = 0

	0	1	2	3
0	0	1	0	2
1	-1	2	4	3
2	0	-1	3	1

Find a value in 2D

- Find the number of times *x* appears in *int matrix[3][4]*

```
 int count = 0;
 for (i=0; i<3; i++)
   for (j=0; j<4; j++){
     if (matrix[i][j] == x)
         count = count + 1;
}
```

	0	1	2	3
0	0	1	0	2
1	-1	2	4	3
2	0	-1	3	1

Matrix sum

- Compute the addition of two matrices

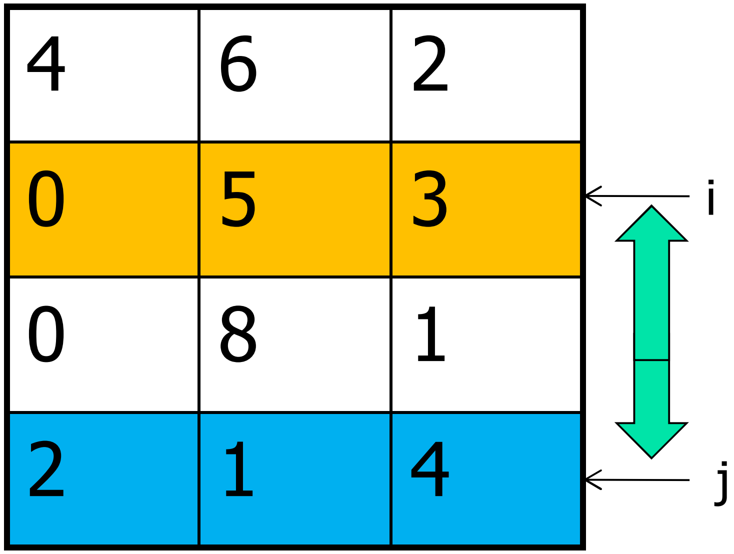
$$\begin{array}{c|c|c|c|c|}
 & 0 & 1 & 2 & 3 \\ \hline
0 & 0 & 1 & 0 & 2 \\ \hline
1 & -1 & 2 & 4 & 3 \\ \hline
2 & 0 & -1 & 3 & 1 \\ \hline
\end{array}
+
\begin{array}{c|c|c|c|c|}
 & 0 & 1 & 2 & 3 \\ \hline
0 & 3 & -1 & 3 & 1 \\ \hline
1 & 1 & 4 & 2 & 0 \\ \hline
2 & 2 & 1 & 1 & 3 \\ \hline
\end{array}
=
\begin{array}{c|c|c|c|c|}
 & 0 & 1 & 2 & 3 \\ \hline
0 & 3 & 0 & 3 & 3 \\ \hline
1 & 0 & 6 & 6 & 3 \\ \hline
2 & 2 & 0 & 4 & 4 \\ \hline
\end{array}$$

solution

```
int matrix1[3][4],
    matrix2[3][4],
    sum[3][4];
// initialize matrix1 and matrix2

for (i=0; i<3; i++)
   for (j=0; j<4; j++)
        sum[i][j]= matrix1[i][j]+matrix2[i][j];
```

Exchange Two Rows



4	6	2
0	5	3
0	8	1
2	1	4

```
for (k=0; k<3; k++){
        t = a[i][k];
        a[i][k] = a[j][k];
        a[j][k] = t;
}
```

4	6	2
2	1	4
0	8	1
0	5	3

Transpose

a

1	5	3
4	2	6
7	9	8

b

1	4	7
5	2	9
3	6	8

```
 int N = 3;
 int a[N][N],b[N][N];
/*  Transfer values to the
    transpose matrix.   */
for(i=0; i<N; i++){
   for(j=0; j<N; j++) {
        b[j][i] = a[i][j];
   }
}
```

In-place Transpose
(you can not use b array)

a

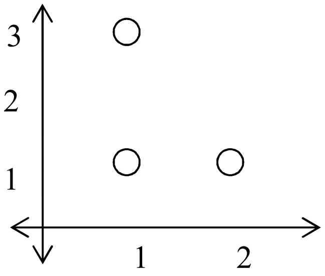
1	5	3
4	2	6
7	9	8

b

1	4	7
5	2	9
3	6	8

```
 int N = 3,t;
 int a[N][N],b[N][N];
/*  Transfer values to the
    transpose matrix.   */
for(i=0; i<N; i++){
   for(j=i+1; j<N; j++) {
        t = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = t;
   }
}
```

Distance between two points



- Imagine a point in a 2-dimentional space (each point is represented by (x,y)) and we store 3 points in an array declared by **double p[3][2]** (you can think that **p[i][0]** is the x value and **p[i][1]** is the y value of ith point).
- We are now interested in finding the closest two points.
  For example, if **p[3][2] = {{1,1},**
  {2,1},
  {1,3} };
  Then we will say that points (1,1) and (2,1) are the closest two points.
- Write a program to do that:

```
double p[3][2] = {{1,1},
                   {2,1},
                   {1,3}};
double d1,d2,d3,dx,dy;
dx = p[0][0]-p[1][0];
dy = p[0][1]-p[1][1];
d1 = dx*dx + dy*dy;
dx = p[0][0]-p[2][0];
dy = p[0][1]-p[2][1];
d2 = dx*dx + dy*dy;
dx = p[1][0]-p[2][0];
dy = p[1][1]-p[2][1];
d3 = dx*dx + dy*dy;
min = (d1<d2)? d1: d2;
min = (min < d3? Min:d3;
```