Week 5 - 2D Arrays

Daniel Alyoshin

CSCA48 - TUT002

February 6, 2026

Content Covered so Far

- C programming
- Memory model
- Arrays and strings
- Pointers
- Maybe CDTs, ADTs, & linked lists

2D Arrays

```c
int main()
{
    char tiny_2D_array[2][3];
}
```

2D Arrays

```
int main()
{
    char tiny_2D_array[2][3];
}
```

$$\begin{bmatrix} A & B & C \\ D & E & F \end{bmatrix}$$

2D Arrays

Storage:

- Stored similarly to 1D arrays as a set of consecutive boxes in memory.

2D Arrays

Storage:

- Stored similarly to 1D arrays as a set of consecutive boxes in memory.
- Stored in **row-major** order - 1st row stored in its entirety followed by 2nd etc.

2D Arrays

Storage:

- Stored similarly to 1D arrays as a set of consecutive boxes in memory.
- Stored in **row-major** order - 1st row stored in its entirety followed by 2nd etc.

Access:

- Access row 1 and column 2 is written as `tiny_2D_array[1][2]`

2D Arrays

Storage:

- Stored similarly to 1D arrays as a set of consecutive boxes in memory.
- Stored in **row-major** order - 1st row stored in its entirety followed by 2nd etc.

Access:

- Access row 1 and column 2 is written as `tiny_2D_array[1][2]`
- Generalized: `array_name[row][column]`

Practice

Provide the memory model for the following code:

```
int main()
{
    double matrix[3][2];
}
```

Assume we fill the matrix with the following values:

2.2	3.3
4.4	5.5
6.6	7.7

2D Array Access with Pointers

Equation for offset: `(row * #_of_columns) + column`

Example: access element at index `[2][0]` in the matrix we just created.

```
double *ptr;
ptr = &matrix[0][0];
*(ptr + (2*2) + 0);
```

Watch out!

```
void proc_2D_array(int array[][]);
```

The above will not work as it did with 1D arrays since the compiler needs to know the second dimension (# of columns) to perform indexing.

Watch out!

```
void proc_2D_array(int array[][]);
```

The above will not work as it did with 1D arrays since the compiler needs to know the second dimension (# of columns) to perform indexing.

Solutions:

1. `void proc_2D_array(int array[2][3]);` (limits you to arrays of this size)

Watch out!

```
void proc_2D_array(int array[][]);
```

The above will not work as it did with 1D arrays since the compiler needs to know the second dimension (# of columns) to perform indexing.

Solutions:

1. `void proc_2D_array(int array[2][3]);` (limits you to arrays of this size)
2. `void proc_2D_array(int *array, int rows, int columns);` (must use pointer + offset for access)

Watch out!

```
void proc_2D_array(int array[][]);
```

The above will not work as it did with 1D arrays since the compiler needs to know the second dimension (# of columns) to perform indexing.

Solutions:

1. `void proc_2D_array(int array[2][3]);` (limits you to arrays of this size)
2. `void proc_2D_array(int *array, int rows, int columns);` (must use pointer + offset for access)
3. `void proc_2D_array(int rows, int columns, int array[rows][columns]);` (should use array notation for access)

Bonus Task

Create a function taking as input:

- a pointer to a 2D array of doubles
- the index of a row
- the index of a column
- a value to be stored at the given indices

Ensure the function inserts the value into the correct entry using only pointer + offset.

The function must work with a **2D array of any size**.

Reminders

- Each student to submit their own file (picture or scan of your work) which can be the same as that submitted by members of their group.
- Ensure your file is named **exactly** as instructed on Quercus.
- You are **not** supposed to keep working on this, you will not be marked on correctness so as long as you submit work that shows you were engaged during this tutorial you will get full marks.