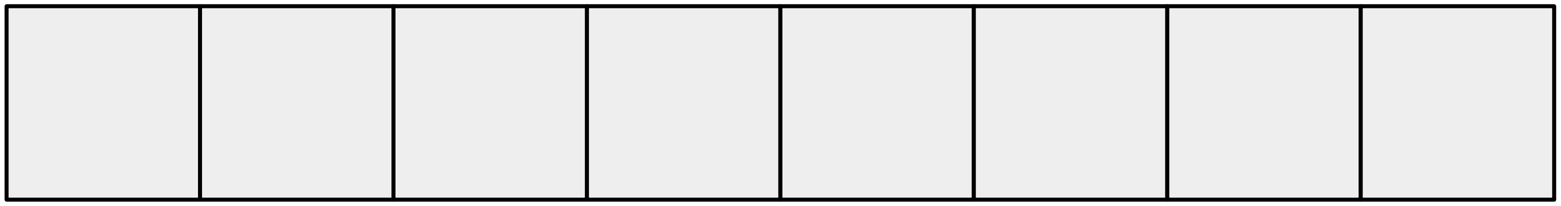
C 2D Arrays

Adapted from materials by Dr. Carrier

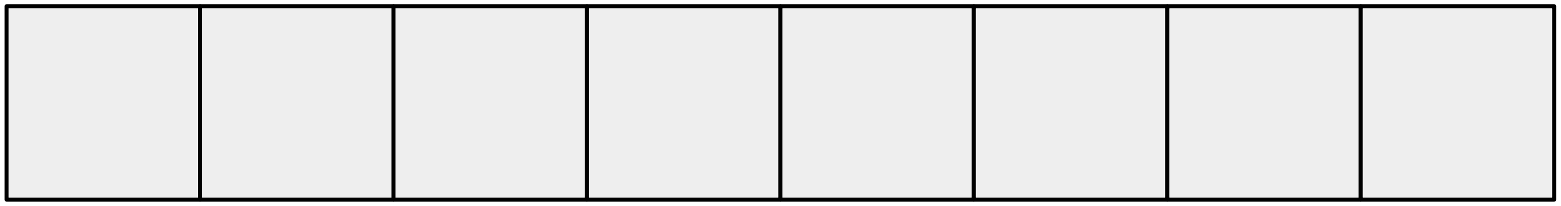
Expanding beyond 1D

We’ve talked about arrays:

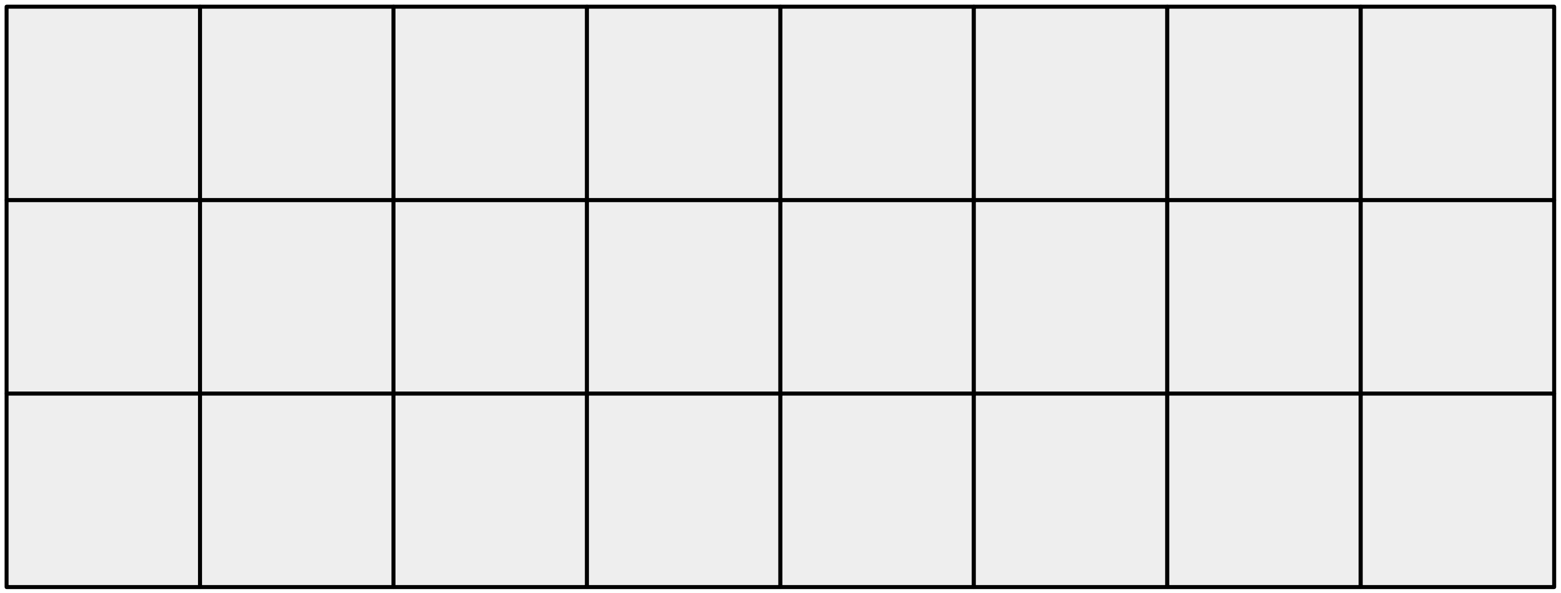


Expanding beyond 1D

We’ve talked about arrays:

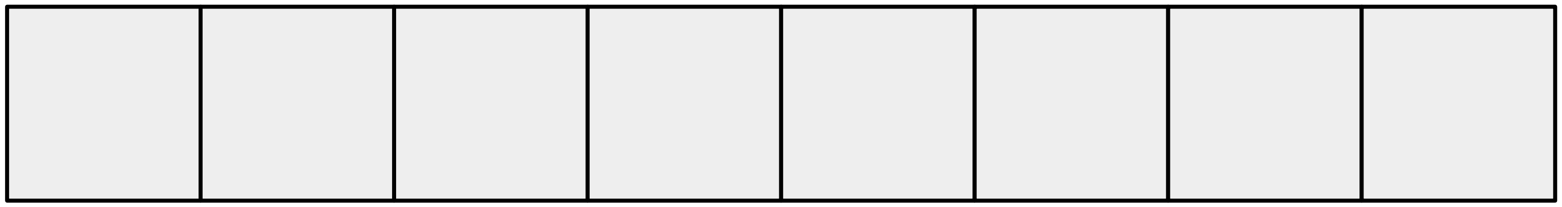


What’s this called?

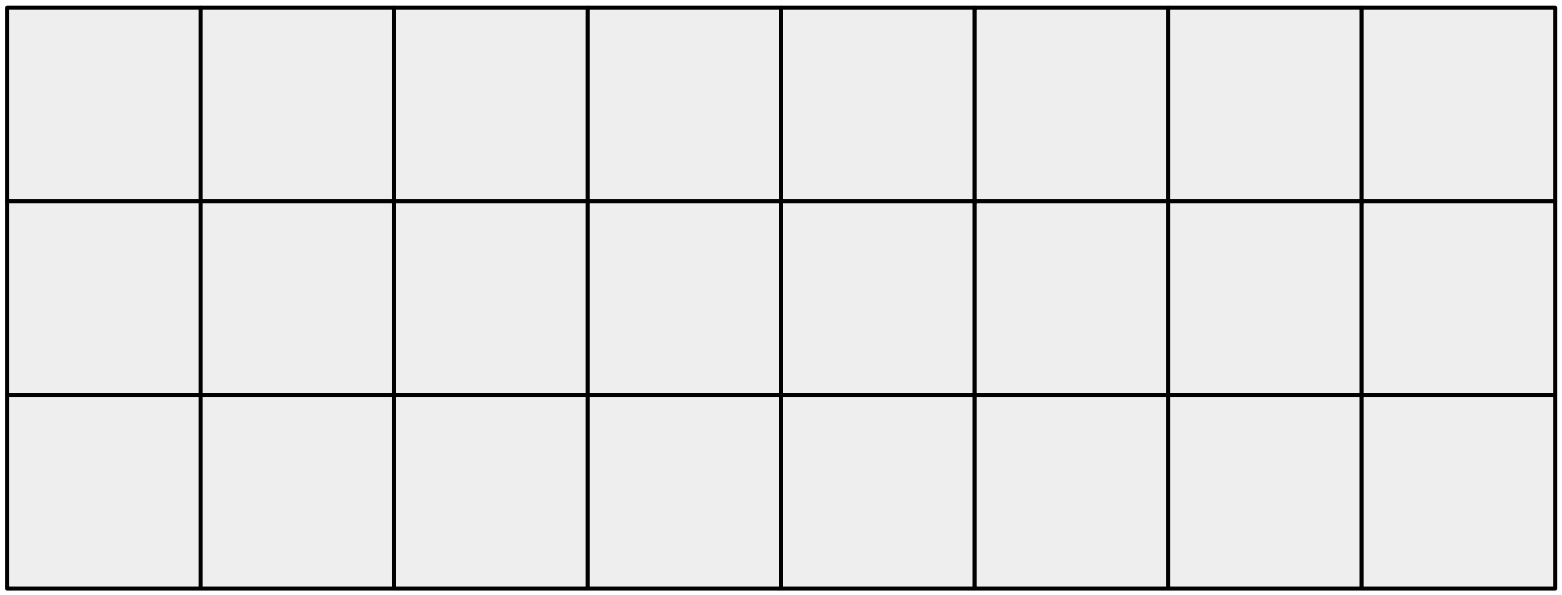


Expanding beyond 1D

We’ve talked about arrays:



What’s this called? A matrix!



Local vars

We can declare a 2D array like this:

```
int mat[2][3];
```

What are the dimensions?

Local vars

We can declare a 2D array like this:

```
int mat[2][3];
```

What are the dimensions? 2 rows, 3 cols (row-major)

Local vars

We can declare a 2D array like this:

```
int mat[2][3];
```

What are the dimensions? 2 rows, 3 cols (row-major)

Where does this memory live?

Local vars

We can declare a 2D array like this:

```
int mat[2][3];
```

What are the dimensions? 2 rows, 3 cols (row-major)

Where does this memory live? On the stack!

Local vars

We can declare a 2D array like this:

```
int mat[2][3];
```

What are the dimensions? 2 rows, 3 cols (row-major)

Where does this memory live? On the stack!

It’s one contiguous block of memory

Local vars

We can declare a 2D array like this:

```
int mat[2][3];
```

What are the dimensions? 2 rows, 3 cols (row-major)

Where does this memory live? On the stack!

It’s one contiguous block of memory

What does mat[1][0] refer to?

0	1	2
3	4	5

Local vars

We can declare a 2D array like this:

```
int mat[2][3];
```

What are the dimensions? 2 rows, 3 cols (row-major)

Where does this memory live? On the stack!

It’s one contiguous block of memory

What does mat[1][0] refer to? 3

0	1	2
3	4	5

Dynamic allocation, the easy way

There are multiple ways to dynamically allocate a matrix

Here, we’ll walk through the easiest

Dynamic allocation, the easy way

What does a matrix look like in memory? (on board)

Dynamic allocation, the easy way

What does a matrix look like in memory? (on board)

(It’s really just a long array)

Dynamic allocation, the easy way

```
int* mat = (int*)malloc(w * h * sizeof(int));
```

How to access the c-th column of the r-th row?

Dynamic allocation, the easy way

```
int* mat = (int*)malloc(w * h * sizeof(int));
```

How to access the c-th column of the r-th row?

```
mat[r * w + c]
```

Dynamic allocation, the easy way

```
int* mat = (int*)malloc(w * h * sizeof(int));
```

How to access the c-th column of the r-th row?

```
mat[r * w + c]
```

This is a cross-language way to handle multi-dimensional arrays

Dynamic allocation, the easy way

```
int* mat = (int*)malloc(w * h * sizeof(int));
```

How to access the c-th column of the r-th row?

```
mat[r * w + c]
```

This is a cross-language way to handle multi-dimensional arrays

Can’t use `[ ][ ]` notation, though

What if I want [ ][ ] syntax?

Alternative #1

Allocating individual arrays

Alternative #1

Allocating individual arrays

```
int** mat = (int**) malloc(h * sizeof(int*));
```

Alternative #1

Allocating individual arrays

```
int** mat = (int**) malloc(h * sizeof(int*));
for(int i = 0; i < h; i++){
   mat[i] = (int*) malloc(w * sizeof(int));
}
```

Alternative #1

Allocating individual arrays

```
int** mat = (int**) malloc(h * sizeof(int*));
for(int i = 0; i < h; i++){
  mat[i] = (int*) malloc(w * sizeof(int));
}
```

Can use `[][]`, but no longer contiguous.

Alternative #2

Assigning pointers into an array

Alternative #2

Assigning pointers into an array

```
int* arr = (int*)malloc(w * h * sizeof(int));
```

Alternative #2

Assigning pointers into an array

```
int* arr = (int*)malloc(w * h * sizeof(int));
int** mat = (int**) malloc(h * sizeof(int*));
```

Alternative #2

Assigning pointers into an array

```
int* arr = (int*)malloc(w * h * sizeof(int));

int** mat = (int**) malloc(h * sizeof(int*));
for(int i = 0; i < h; i++){

  mat[i] = arr + i * w;

}
```

Alternative #2

Assigning pointers into an array

```
int* arr = (int*)malloc(w * h * sizeof(int));
int** mat = (int**) malloc(h * sizeof(int*));
for(int i = 0; i < h; i++){
   mat[i] = arr + i * w;
}
```

Can use `[ ][ ]`, AND is contiguous