C - MULTIDIMENSIONAL ARRAYS

2D ARRAYS

- Example: `int mat[2][3];`
  - 2 rows, 3 columns
- Can initialize when declaring:
  - `int mat[2][3] = {{1,3,4}, {8,2,5}};`
- Access elements: `mat[row_num][colnum]`
- Decleared on the stack
- One contiguous block of memory underneath

2D ARRAYS (CONT.)

- `mat[i][j]` = an element of the 2d array
- `mat` = address of first element
- `mat[i] = &mat[i]` = address to first element of row i
- `mat+i = *(mat+i)` = ^address to first element of row i
- Think of `mat[i]` as a pointer to an array (row)
- Doing pointer arithmetic on `mat` advances by the whole length of the subarrays

2D ARRAYS (CONT.)

- Can also use pointer arithmetic to access into certain col
- `*(mat[i]+j) = mat[i][j]`
- `*(*(mat+i) + j) = mat[i][j]`
  - recall `*(mat+i)` is memory address to start of row i
  - advance this by j cols, then dereference to get actual value stored
- What about `mat+i + j`?

2D ARRAYS - DYNAMIC ALLOCATION

```
int *mat = (int *) malloc(nrows*ncols*sizeof(int));
```

- One contiguous block of memory
- Can't use [][] notation:
- Can use [] notation
- Pointer arithmetic to handle rows and columns

2D ARRAYS - DYNAMIC ALLOCATION

```
int **mat = (int **) malloc(nrows*sizeof(int *));
for (int i=0; i<nrows; i++) {
    *(mat+i) = (int *) malloc(ncols*sizeof(int));
}
```

- Could also use `mat[i]` inside the loop
- Can use [][] notation now
- No longer one contiguous block of memory

2D ARRAYS - DYNAMIC ALLOCATION

```
int *A = (int *) malloc(nrows*ncols*sizeof(int));
int **mat = (int **) malloc(nrows*sizeof(int *));
for (int i=0; i<nrows; i++) {
    mat[i] = A + i*nrows;
}
```

- Allows use of [][] notation
- Meomory for actual entries is contiguous