APS 105 Lecture 20

Last lecture (before midterm revision lectures):

Dynamic memory allocation

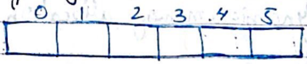
Today: Recap on dynamic memory allocation and introduce 2D arrays.

- ON DEMO -

```
int numOfStudents;
int *marksArray;  // This will be a pointer to the 1st element in
                  // the array.
printf("Enter # of marks");
scanf ("%d", &numOfStudents);

// Recap: CANNOT do int marksArray [numOfStudents];
// since size of this array is fixed/known at compile-time

marksArray = (int *) malloc (numOfStudents * sizeof(int));
points to 1st element   need to     returns              # of bytes
in array                typecast    void*
                                    to 1st byte in allocated memory
printf("Enter marks: ");

for (int i = 0; i < numOfStudents; i++)
            scanf(" %d", marksArray + i);
                         or &marksArray[i]
free (marksArray); // returns memory space to Operating System to reuse
marksArray = NULL; // To make sure if you use it, it gives Seg. fault!
```

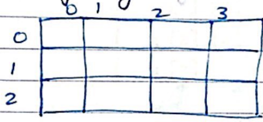
We've learned 1D arrays

```
int myArray [6];
```



Can we have more dimensions? 2D arrays:

```
                ↙rows  ↙columns
int myArray [3] [4];
```



<u>Initialization</u>:

```
int myArray [3][4] = {
                       {1, 2, 3, 4},
                       {5, 6, 7, 8},
                       {9, 10, 11, 12}
                     };
```

3 rows
4 columns

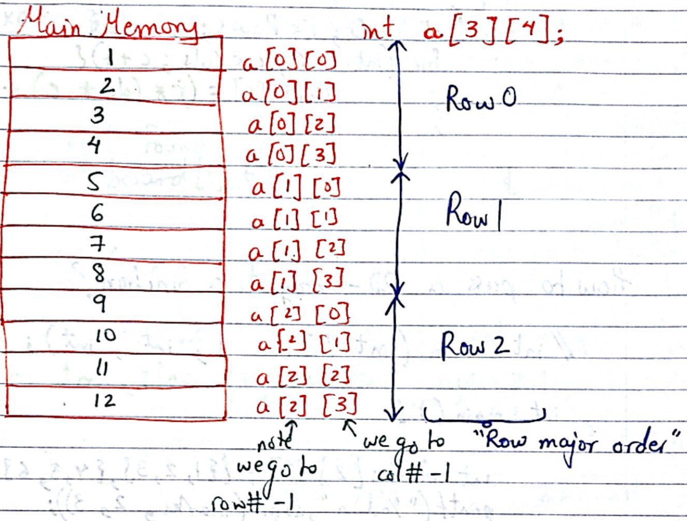
Alternative (messier):

```
int myArray [3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                      11, 12};
```

it will fill the memory locations row by row.

Meaning: Arrays are stored in "<u>row major</u>" order in memory.

Recall: memory is byte addressable
each element in int array is stored in 4 bytes



Main Memory

int a[3][4];

1	a[0][0]	Row 0
2	a[0][1]	
3	a[0][2]	
4	a[0][3]	
5	a[1][0]	Row 1
6	a[1][1]	
7	a[1][2]	
8	a[1][3]	
9	a[2][0]	Row 2
10	a[2][1]	
11	a[2][2]	
12	a[2][3]	

note we go to row# -1

we go to col# -1

"Row major order"

What is the address of i, j element of a (i.e. &a[i][j])?

= address of a[0][0] + sizeof(int)(i * num of columns in total + j)

* Cannot initialize a 2D array like this: int myArray[3][4] = {0};

To initialize elements of a 2D-array → nested for loop.

```
const int Rows = 3;
Const int Cols = 4;
int       a [Rows][Cols];
```

```
for (int r=0; r< Rows; r++){
    for (int c=0; c< Cols; c++){
            a[r][c] = (r* Cols + c) + 1;
    }

}
```

How to pass a 2D-array to a function?

```
// int sum (int [][  ], int , int);
int sum (int rows, int cols, int m [][cols]);
int main () {

    int marks [2] [3] = {{1, 2, 3}, {4, 5, 6}};
    printf("%d\n", sum (marks, 2, 3));

    return 0;
}
```

it is optional to put rows

need to have cols 1st

as mentioned before, whenever an array is passed to a function, it is passed as a pointer. There is no way to find its size, without passing its size to function.

```
//int sum (int marks [ ] [cols ], int rows, int cols){
int sum (int rows, int cols, int marks [ ][cols]) {
            int sum= 0;

            for (int r=0; r<rows; r++){

                for (int c=0; c< cols; c++){
                        sum+= marks [r][c];
                }
            }
            return sum;
}
```

correct way!

cols must appear before usage in marks[] [cols]

this does r* num of. columns + c

MUST be passed with array

Example: Find 3 consective horizontal 1's in a 6x6 array.

0	1	1	0	0	0
0	1	0	1	1	0
0	1	1	1	0	0
0	0	1	1	1	0
0	0	0	1	1	0
1	1	1	0	1	1

```
int board[6][6];

for ( introw=0; row<6 ;row++) {
    for(int col=0; col<6 ; col++){
        int count=0;
        for (int step =0; step+col <6 && step<3;
                    step++) {
            if (board[row][col+step] ==1)
                    count++;

            if (count == 3)
                printf("Found at row %d
                        col %d",
                        row, col);
        }

    }
}
```