APS 105 Lecture 21

Last lecture: 2D array (initialization and pass to a function)

Today: Continue with 2D arrays and cover multi-dimensional arrays

Recap: 2D array initialization:

If we will declare only & initialize later:

```
int arr [2] [3];
```

If we will declare and initialize:

```
int arr [2][3] = {{1, 2, 3}, {4, 5, 6}};
```

OR

```
int arr [ ] [3] = {{1, 2, 3}, {4, 5, 6}};
```

row size will be known from

OR

```
int arr [2][3] = {1, 2, 3, 4, 5, 6};
```

Pass 2D-array to a function, just like 1D array, we pass pointer to 1st element, and size of array

-DEMO-

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

int main (void){

    int marks [2] [3];

    for (int r=0; r<2; r++){
        for (int c=0; c < 3; c++) {
            marks[r][c]= r*3+c+1;
        }
    }
    printf("sum is %d\n", sum (2, 3, marks));
    return 0;
}
int sum (int rows, int cols, int marks [][cols]) {   ← very imp.

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

marks[r][c] → to get to marks [r][c], # of cols is required here to do the following
& marks [0][0] + r * # of cols + c

In multi-dimensional array:

3D → int book [page][row][col];

4D → int shelf [book][page][row][col];

again 1st dimension is not important to identify when passing an array to a function or when declaring & initializing; however other dimensions MUST be identified.

e.g. Function prototype

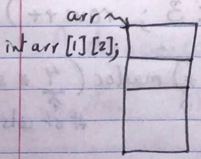
```
int f(double shelf[][page][row][col],
      int book, int page, int row, int col);
```

Dynamic Allocation of 2D-arrays:

array identifier in 1D array is pointer to 1st element in array.
Same with 2D arrays, but for dynamic memory allocation of 2D-arrays, array identifier points to a pointer, that points to a pointer pointing at the 1st element of the array

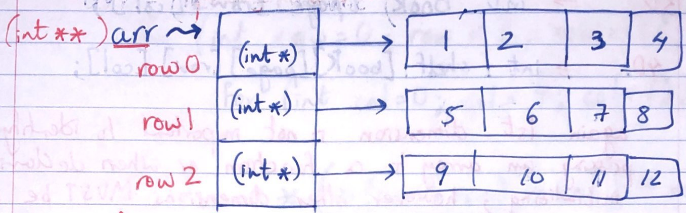
fixed size array

int arr [1][2];



Dynamically allocated 2D-array:

(Next page)



Array of pointers.
Each pointer represents
an array of 1D.
Each pointer is pointing
to the 1st element of its row.

So to dynamically allocate 2D array:

```
int ** arr;

#of rows   arr = (int **)malloc( 3 * size of (int*) );
                             ↑
                          # of rows
```

//Each row is an array. For each row I need to allocate 4 colums. Loop over each row

```
for (int row= 0 ; row < 3 ; row ++) {

     *(arr + row) = (int *) malloc ( 4 * size of(int))
     pointer to 1D                   ↑
     Array of integers            # of cols
          or
     arr [row]
}
```

Lets fill in the 2D - dynamically allocated array

```
for (int row = 0; row < 3; row++) {
    for (int col = 0; col < 4; col++) {
```

$$\underbrace{*\left(\underbrace{*(arr + row)}_{\substack{\text{pointer to a}\\ \text{1D-array}\\ \text{of integers}\\ \text{or}\\ arr[row]}} + col\right)}_{arr[row][col]} = row * cols + col + 1;$$

```
    }
}
```

Lets free the 2D array (Reverse the order of allocation):

```
// For each row, free the allocated 1D array
for (int row = 0; row < 3; row++)
```

$$free\left(\underbrace{*(arr + row)}_{\substack{\text{or}\\ arr[row]}}\right);$$

```
free(arr);
```

A dynamically allocated array can be passed to a function like this: int sum(int rows, int cols, int **arr);