

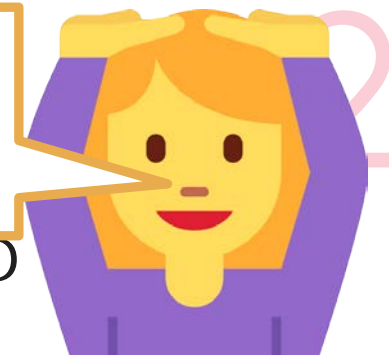
Arrays as pointers

ESC101: Fundamentals of Computing

Nisheeth

Array of Pointers

Can use **array of char pointers** to store many (i.e., an array of) strings



An array of (say char) pointers can be created in two

- Use a static array declaration `char *ptrArr[3];` and initialize each of the 3 pointers `ptrArr[0]`, `ptrArr[1]`, and `ptrArr[2]` using `malloc` or as static arrays

Just like **name of array** is a **pointer**

- Use a dynamic array declaration as a **pointer**

Name of array of pointers is also a **pointer of pointer**

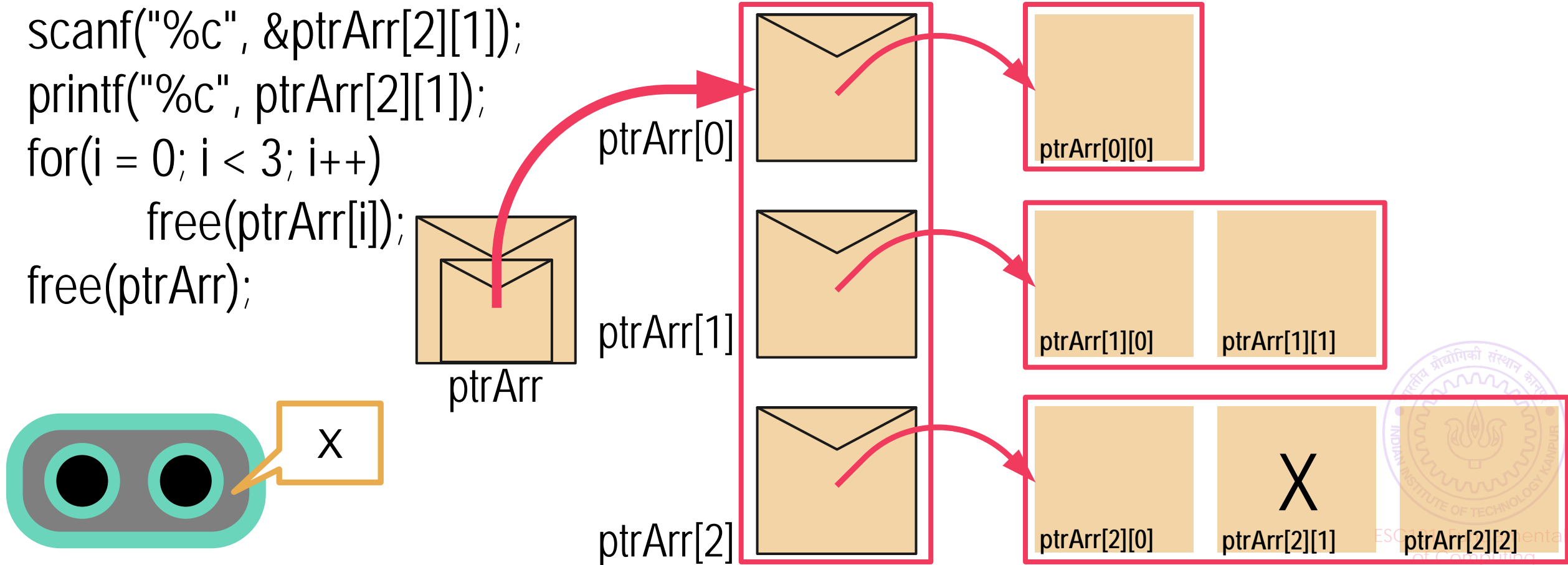
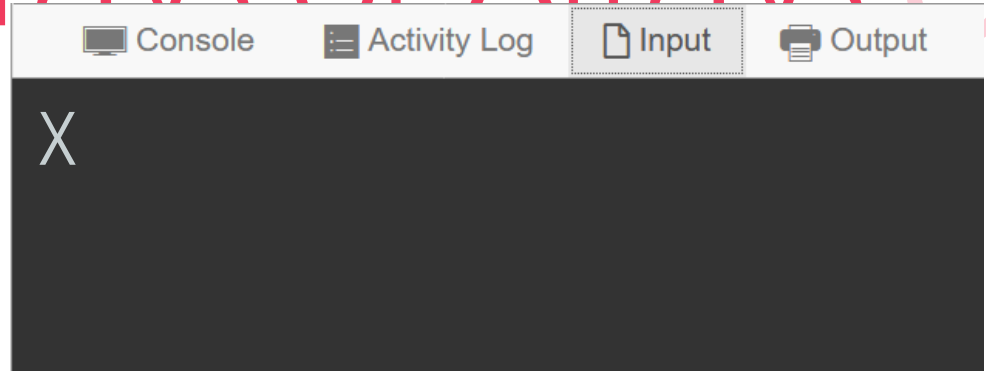
```
char **ptrArr = (char **)malloc(3*sizeof(char *)),
```

and then initialize each of the 3 pointers `ptrArr[0]`, `ptrArr[1]`, and `ptrArr[2]` using `malloc` or as static arrays



Array of Pointers → Arrays of Arrays 3

```
char **ptrArr = (char**)malloc(3*sizeof(char*));  
for(i = 0; i < 3; i++)  
    ptrArr[i] = (char*)malloc((i+1)*sizeof(char));  
scanf("%c", &ptrArr[2][1]);  
printf("%c", ptrArr[2][1]);  
for(i = 0; i < 3; i++)  
    free(ptrArr[i]);  
free(ptrArr);
```



Accessing Elements in Array of Pointers/Arrays

4

Rest assured, the same rules apply as do with pointers

```
char *ptrArr[3], str[3];
```

```
for(i = 0; i < 3; i++)
```

```
    ptrArr[i] = (char*)malloc((i+1)*sizeof(char));
```

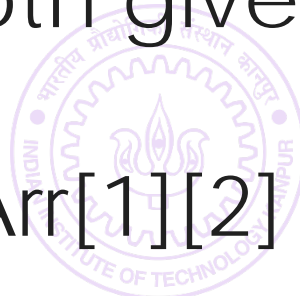
ptrArr[0], ptrArr[1], ptrArr[2] are all arrays of chars

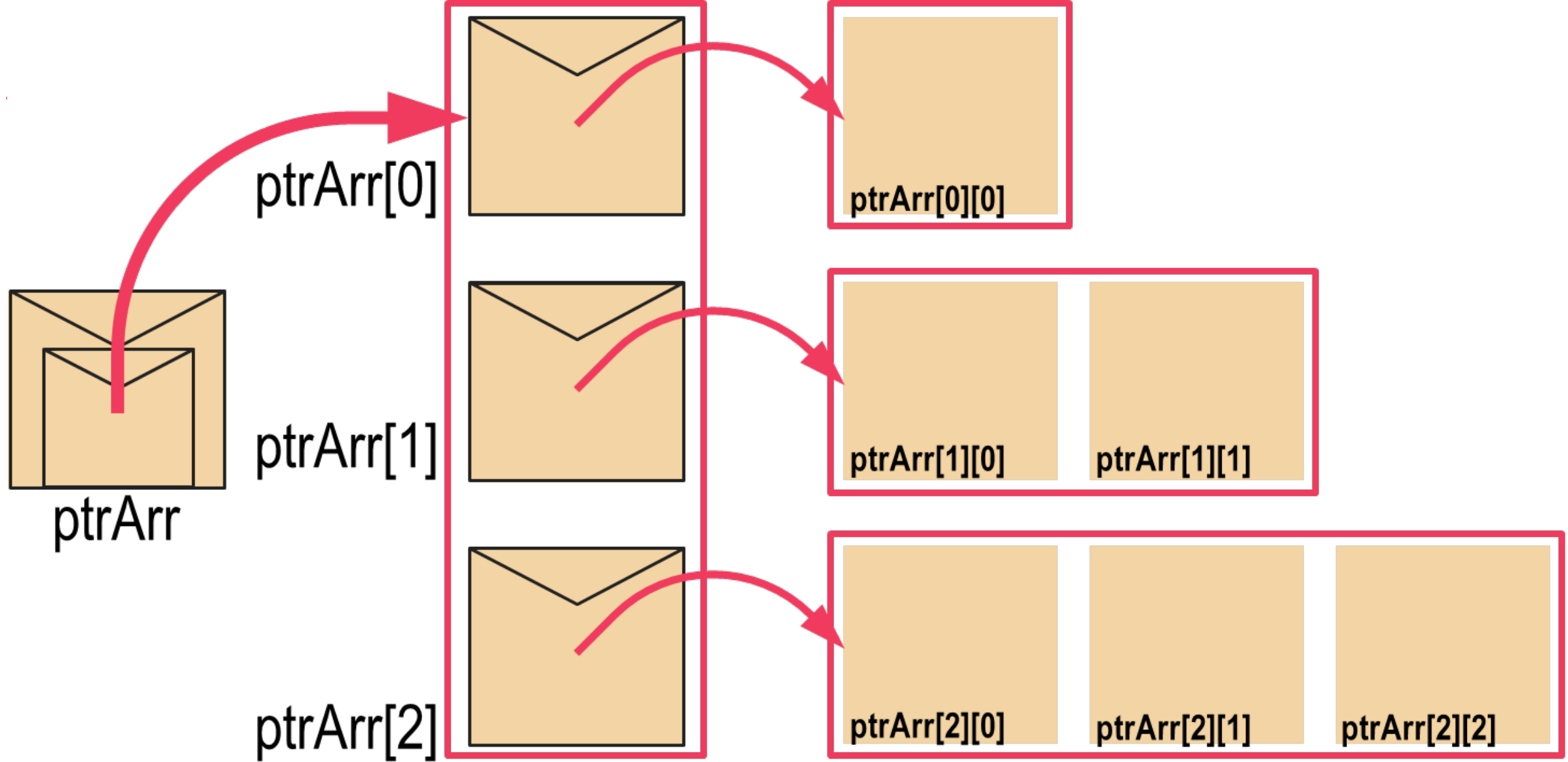
How to access individual elements of these arrays?

Two ways to access index 2 element of str: str[2], *(str+2)

Apply exact same rule : ptrArr[2][2], *(ptrArr[2]+2) both give index 2 element of the array ptrArr[2]

Note that ptrArr[1] does not have 3 elements so ptrArr[1][2] may cause segfault!





We can access index 2 of the third array in many ways: `ptrArr[2][2]`, `*(ptrArr[2] + 2)`, `*(*(ptrArr + 2) + 2)`, `(*(ptrArr + 2))[2]`

2D Arrays: Revisited (Pointer's view) 6

```
int mat[3][5]; // note: 2D array name mat is also a pointer to pointer (int **)
```

Declares a matrix (2D array) with 3 rows 5 columns

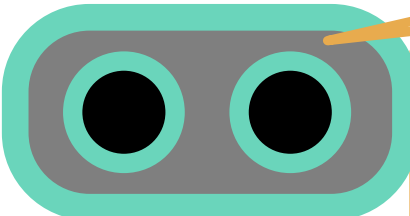
Rows numbered 0, 1, 2. Columns numbered 0, 1, 2, 3, 4

Element at row-index i and column-index j is an int variable

Can access it using several ways

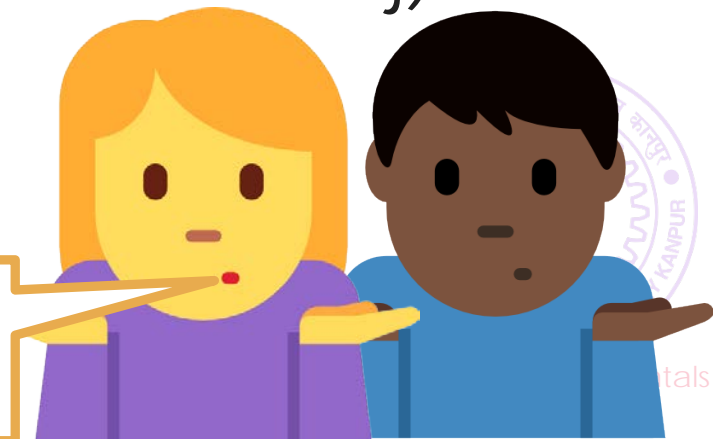
`mat[i][j]`, `*(mat[i] + j)`, `*(*(mat + i) + j)`, `(*(mat + i))[j]`

Careful! `** (mat + i + j) ≠ *(* (mat + i) + j) ≠ *(*mat + i + j)`



Not that much actually – let me show you the differences

This looks exactly like the way we access an array of pointers/arrays – what is the difference?



2D arrays vs Array of pointers

7

2D ARRAYS

Number of elements in each row is the same

All elements of 2D array are located contiguously in memory

Easier to initialize

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

Very convenient 😊

ARRAY OF POINTERS

Different arrays can have different number of elements – more flexibility

Elements of a single array are contiguous but different arrays could be located far off in memory

Have to be initialized element by element

More power, responsibility



Memory layout of 2D arrays

```
char str[3][4] = {"Hi", "Ok", "Bye"};
```

Location of the str pointer not shown

First all elements of row 0 stored in continuous sequence

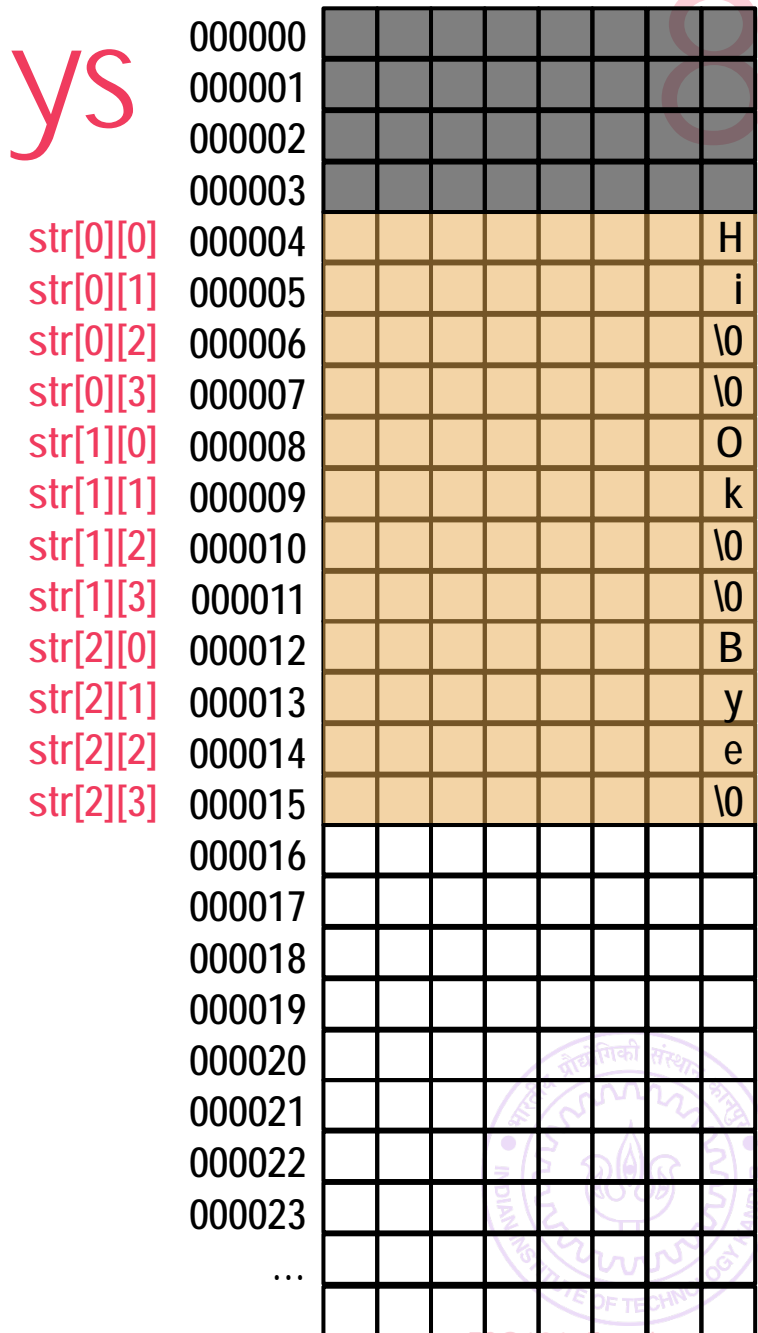
Then without breaking sequence, all elements of row 1 stored and so on

```
char* ptr = *str; // ptr points to str[0][0]
```

```
ptr += 4; // ptr now points to str[1][0]
```

```
ptr += 4; // ptr now points to str[2][0]
```

```
ptr += 1; // ptr now points to str[2][1]
```



Layout of array of pointers

```
char **str = (char**)malloc(3*sizeof(char*));
```

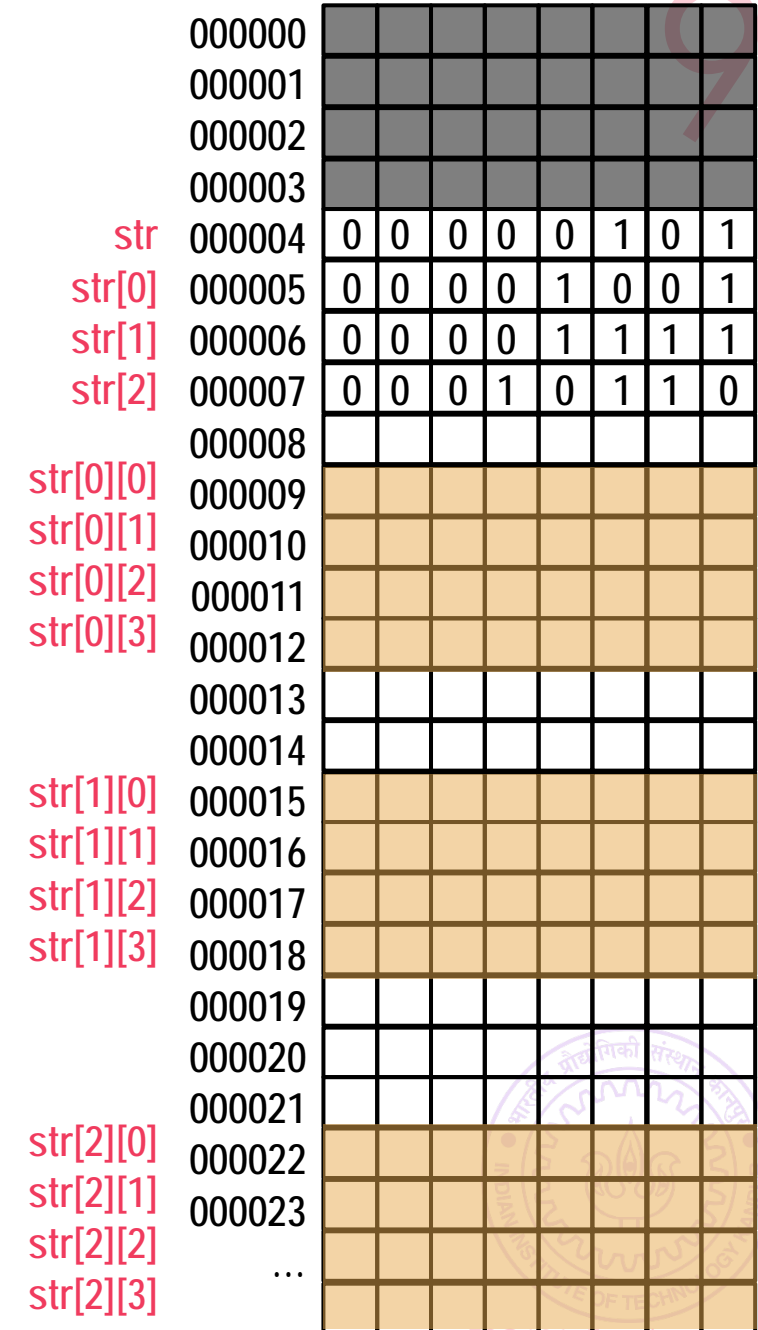
```
str[0] = (char*)malloc(4*sizeof(char));
```

```
str[1] = (char*)malloc(4*sizeof(char));
```

```
str[2] = (char*)malloc(4*sizeof(char));
```

Element within a single array always stored in sequence

Different arrays may be stored far away from each other



Summary

10

Arrays are just an application of pointers

In any dimensions, one can access arbitrary array elements with pointer math

Dynamically allocated arrays of pointers are a much more general data structure

- Multidimensional arrays emerge as a special case

- Other data structures also emerge as special cases, as we will see when we discuss structures

Whenever you get a problem where the size of the input arrays are not fixed, you have to use dynamic allocation

