→ Character array, behaves differently than a normal array.

→ It works as a container for a string (ending with \0)

→ char* : Character address ⇒ cout ; we don't get the address

instead, cout prints byte-by-byte until a character 0 is found.

→ ways to take as input in char array.
→ output (cout) prints the entire string.
→ Palindrome, length, append, reverse

init: char str[] = "welcome"; ✓ valid

assign: Str = "Coding"; ✗ not possible

→ char arrays can only be initialized using a string literal ("——")

→ char arrays can't be assigned a string literal.

→ once you initialize/declare a char[], you can only modify it by going to each index.

there exists an inbuilt string class (data-type).

→ we use string instead of char[], it's more easy to use.

→ behind-the-scenes (implicitly) strings themselves use char[].

Multi-dimensional arrays

↓

2D array: an array of (arrays) → maybe of any basic data type

⇒ int, char, float, bool, double

array of int
array of double
array of char
array of float
array of bool

↳ each index of the 2D array is an array itself.

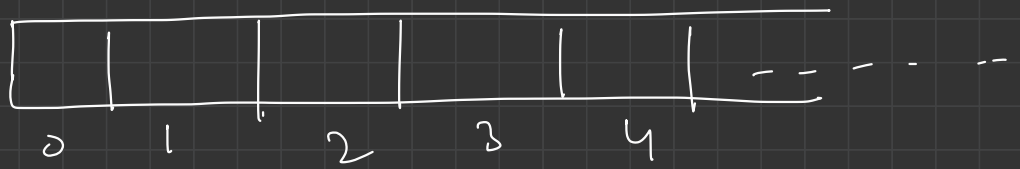
↓

just like normal array.

→ Array is a collection of same data type.

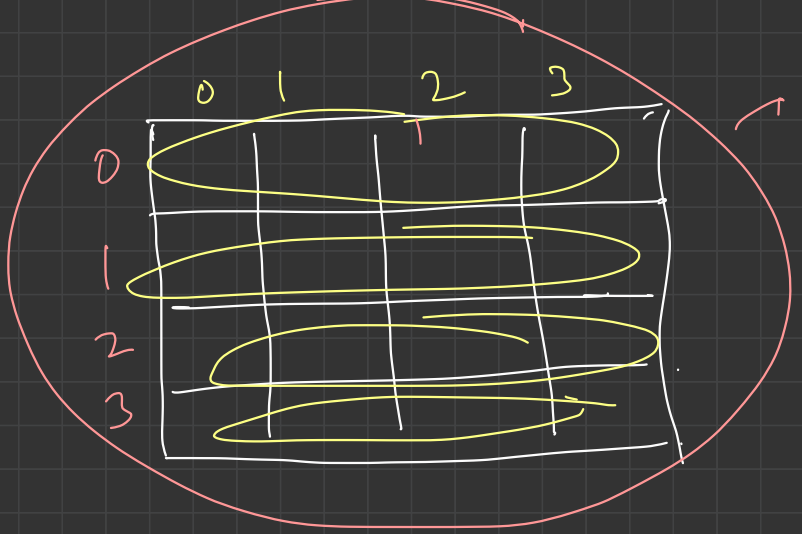
→ no mixture.

1D array



2) array

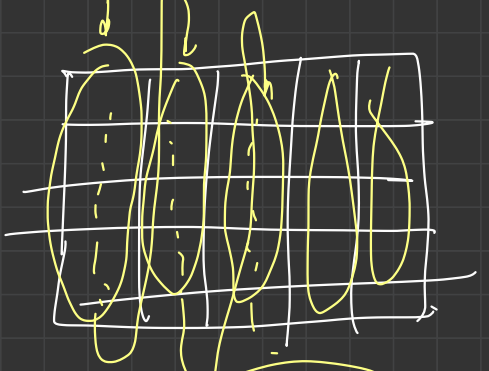
matrix.



↑ an array of 4 arrays

array stores data in consecutive locations.

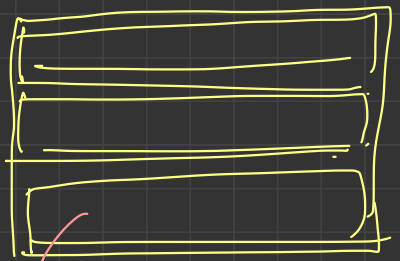
row-major form

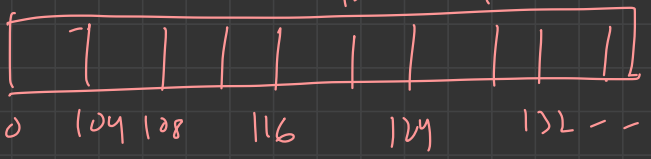


array of arrays
cols

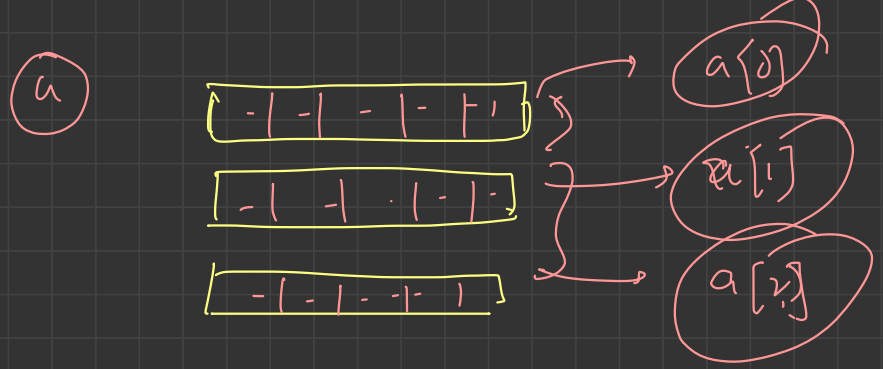
col-major form

but very uncommon



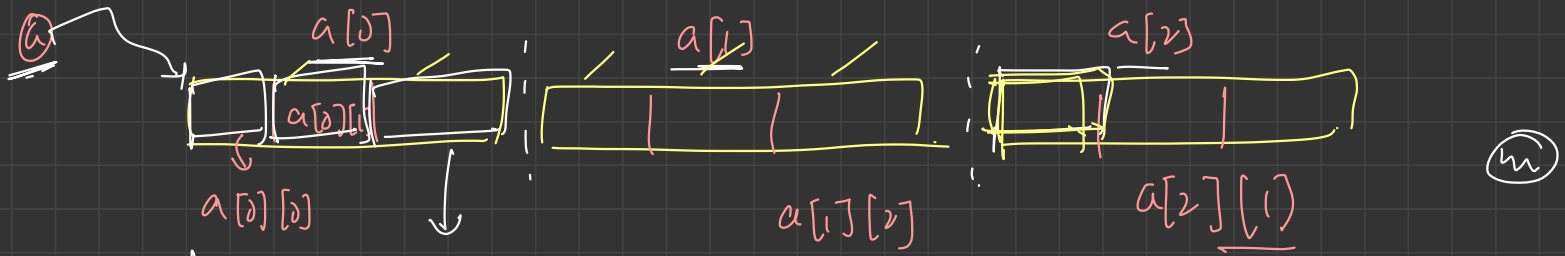


→ these are continuous locations in memory



randomly ??

Same reason, buckets are stored in continuous locations.



m sized chunks divided

$n \times m$

n parts each of size m

a [i][j] ??

arr[i] = *(arr+i)

it was actually address arithmetic

a [i][j]

*(a+i+j) → this would be wrong

a[2][0]

*(a+2+0)

size of each array = second parameter

a[2][0] = *(a + 2x3 + 0)

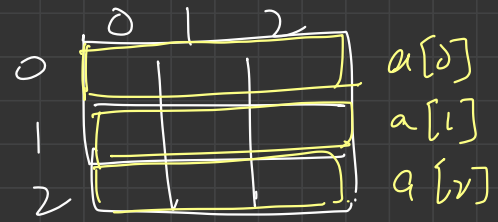
*(a+6)

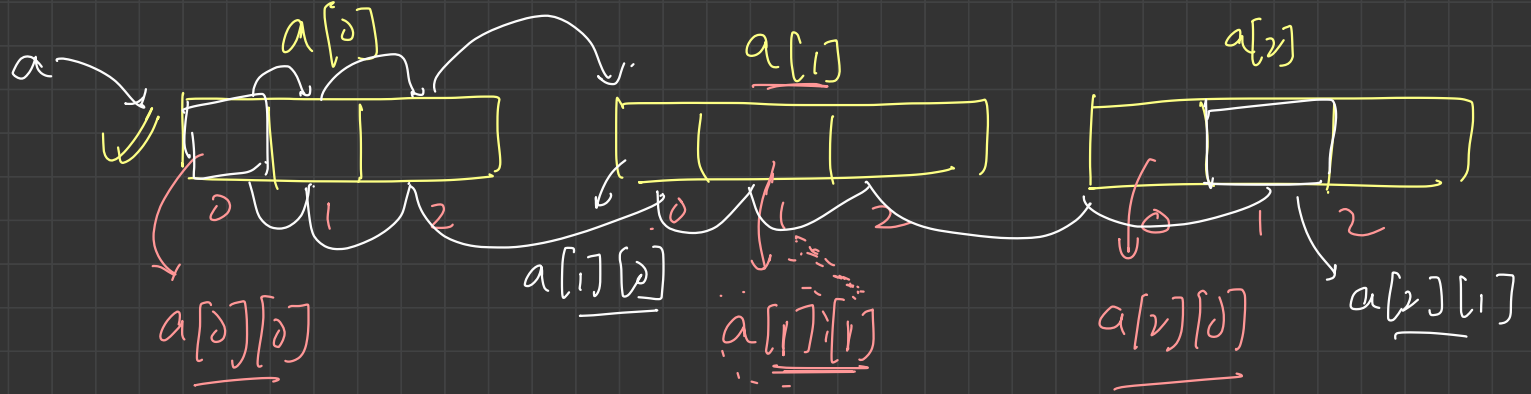
no. of cols

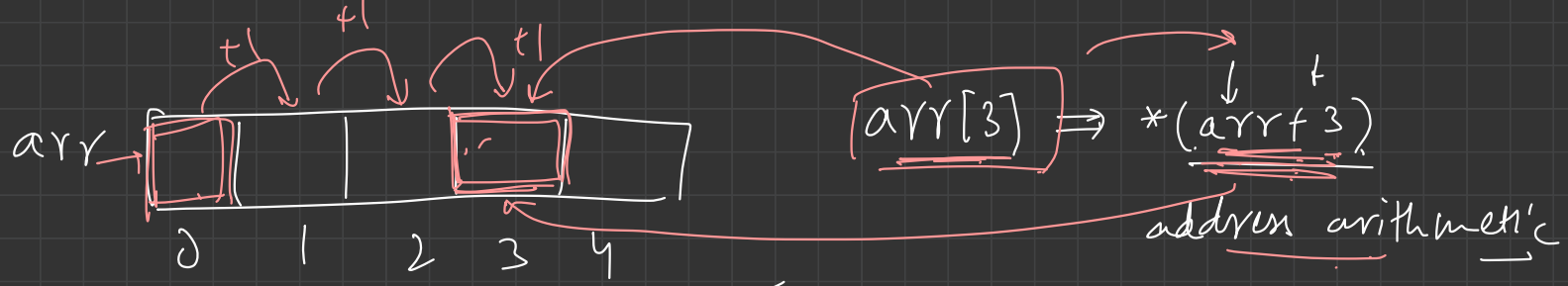
a[i][j] ⇒ *(a + i.m + j)

a[3][3]

n=3 : rows
m=3 : cols







2D

$a[2][1] \Rightarrow *(a+2+1)$??

$= *(a+3)$

this is not the way

$a[2][1] \Rightarrow *(a + 2 \times m + 1)$ $\quad m = 3$

$*(a + 2 \times 3 + 1)$

$*(a+7)$

2 rows entirely skip

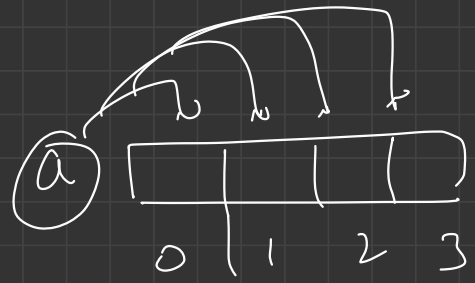
$2 \times m$ buckets

$a[i][j] = *(a + i \times m + j)$

Conceptually correct, but not actually ??

We can say arr is a pointer to the array or in case of 2D arrays, its a row pointer.

$$a[i][j] = *(*(a+i)+j)$$

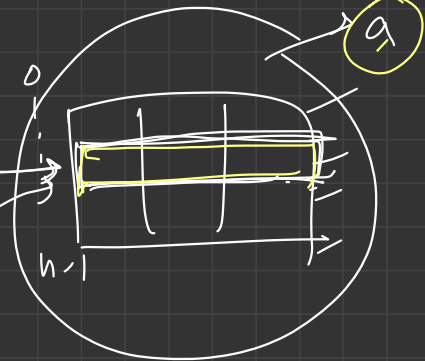


a → a pointer to a row pointer

1D array ⇒ $arr[i] = *(arr+i)$

2D array ⇒ $a[i][j] = *((*(a+i))+j)$

$\Rightarrow *(*(a+i)+j)$



a[i] → ith array / row

$a[k][j] = *(*(a+k)+j)$.

Pointer to an array: `int (*p)[5]`

Array of pointers: `int *p[5]`.

Q. Given a matrix, determine which row or column has the largest sum?