Arrays

- Data Structure
- Finite number of elements. – Declaration.
- ~~Ordered set~~
- Contiguous memory locations.
- Homogenous
- Random Access
- ~~Indexed~~ INDEXED.

a.

10	20	30	40	50	9
0	1	2	3	4	5

int type.

a[3] = 40

base address = 1000.

size

10	20	'a'

X

10	10.5

X

- ~~for~~ chunk or block of memory = divided into equal & small blocks that contain values

1. Variable.

→ collection of variables holding values of same type & for ~~so~~ specific purpose

Types of Array.

- 1D
  - Seq. of elements arranged in - single row
- multi-dimension
  - 2D
  - 3D

1D Array

Declare

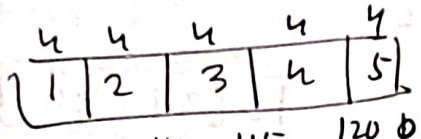
```
int arr[5]
```

Data type name [N]
↳ No. of elements

→ Compiler will allocate = 5 × sizeof(int)

~~5 × 4 = 20.~~

5 × 4 = 20 bytes will be allocated



base address

2D Array

```
int arr[2][3]
        ↑   ↑ col
        row
```

Ex:- Table, Excel, Computer screen.

int arr [3][4]

	col1	col2	col3	col4
	0	1	2	3
0 Row1	00	01	02	03
1 Row2	10	11	12	13
2 Row1	20	21	22	23

3 × 4

Note:- Indexing may not always start from 0

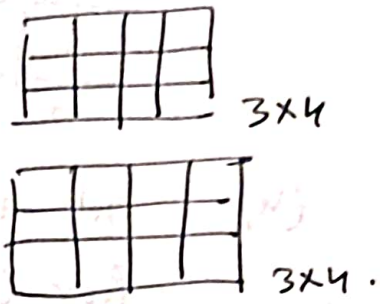
C, C++, Python, Java. ⇒ Indexing starts from 0
: etc

Fortran, Ada → Non-Zero based indexing

3D-Array.

```
int arr [2] [3] [4]
        ↑    ↑    ↑
        |    row  col
   no. of arrays.
```

Array #	Index	
1	0	- 3×4.
2	1	- 3×4



Do we have arrays in Python?

(i) `my_list = [1, 2, 3, 4]` # List = Similar to array

Lists support dynamic re-sizing.

(ii) import array

`my_arr = array.array('i', [1, 2, 3, 4])`.

(iii) import numpy as np.

`my_np_arr = np.array([1, 2, 3, 4])`.

* Show Arrays in Python [C] GitHub.

Different ways to Declare an Array

C-standpoint

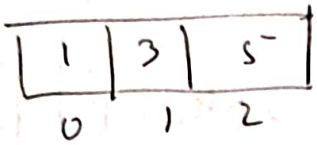
(1) `int arr[5] = {1, 2, 5, 4, 67};`

(2) `int arr[] = {1, 2, 5, 4, 67, 92}`

- you can add as many as elements you want.

(3) `int arr[3];`

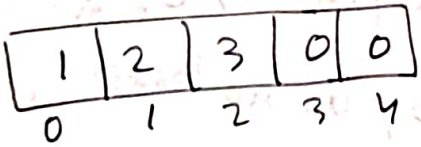
`arr[0] = 1`
`arr[1] = 3`
`arr[2] = 5`



(4)
```
int arr[5];
for (i=0; i<5; i++)
{
    scanf("%d", &a[i]);
}
```

Q:- What if the no. of elements initialized with is < length specified?

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



Q: I want all elements to be initialized with 0.

```
int arr[10];
for (i=0; i<10; i++)
{
    arr[i] = 0
}
```

Bad
Don't do this

Prefer this

`int arr[10] = {0};`

`Int arr[10] = { }` X

legal array declarations

int arr[5] → ✓
int arr[5+5] → ✓
int arr[5*3] → ✓
int arr[-5] → ✗

```
int num;
int arr[num=21/3]; ✓
```

Rules

(1) The number of elements ≤ len(arr).

The rest of the values will be filled with 0.

(2) {} // Don't leave flower brackets empty.

Designated initializers

let say, you want an array

```
int arr[10] = {1, 0, 0, 0, 0, 2, 3, 0, 0, 0};
```

Specific Position = 0.

```
int arr[10] = { [0] = 1,  [5] = 2;  [6] = 3};
                  ↑
                index
```

This is called Designated initialization.

But if you give

```
int arr[] { [0] = 1, [5] = 2, [6] = 3}
```

len(arr) = 7 → assigned by compiler.

. Show print array

.. reversal of array programs

Address of an element in an array

. 1D array

. 2D "

. 3D "

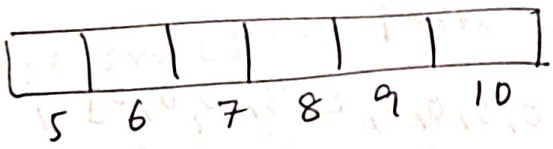
But before, we get into this, we need to look at another concept

int arr [5..10] = {0}

Dedicated ~~Designated~~ indexing.

lower bound (lb) = 5

upper bound (ub) = 10.



How many elements = 6. [10 - 5 + 1]

- no. of elements = ~~(~~(ub - lb + 1)
- If there is no ~~designated~~ dedicated indexing, default lb = 0
- int arr [-3..5] ← legal

it is not to be confused with

int arr [-3]

Ex:-

arr

100	102	103	95
0	1	2	3

base addr = 100.

What is the addr (arr[2]) = ?

addr(arr[2]) = B.A + No. of elements preceeding × Size(Data-type)

addr(arr[0]) = 100

addr(arr[1]) = 100 + 4 = 104

addr(arr[2]) = 104 + 4 = 108

100 + 2×4 = 108. → representation.

Ingredients to find out the address at a given index

(1) index

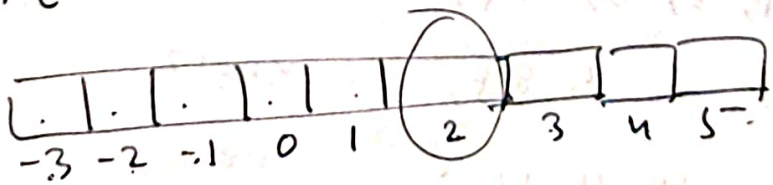
(2) Base Address.

(3) No. of elements preceeding → (i - 0).

(i - i0) → Declaration of the array.

i0 = -3.

int arr[-3..5]



2 - (-3) = 5

(4) Size(Datatype) = 4 - int, ~~unicode~~
2 - int → older computers

- UTF-8 : 1-4 bytes
- UTF-16 : 2-4 bytes
- UTF-32 : 4 bytes

General Formula

addr(arr) = BA + (No. of elements preceding) × Size (Data type)

1D Array

addr (arr[i]) = BA + $(i - i_0)$ × size (Data type).

Ex:- You are given a 1D array

- int A[L..U]
- L = 5 U = 10
- Size (int) = 4
- Base addr = 1000.
- Find A[8] = ?

addr (A[8]) = 1000 + (8 − 5) × 4

= 1000 + 12 = **1012**

How to declare a 2D array

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

You may write like this as well, but prefer not

~~int arr[3][4] = { {1, 2, 3, 4},~~

you could write

int arr [3] [4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }.

Not preferred.

. Show arr.2.c. file in Github.

arr.	c1 0	c2 1	c3 2	c4 3	
0	00	01	02	03	— Row 1
1	10	11	12	13	— Row 2
2	20	21	22	23	— Row 3

index map

base address = 100

what is the address of a [2] [2] ?

addr (arr) = B.A + No. of elements preceding × size (D.T).

We need a formula for this.

Let say if we have an arr [100] [135] ?

Note

Declaration

int arr [3] [4]

forms: int arr [3..5] [6..9] ← Legal case

Actual values → different

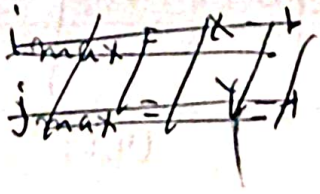
int arr [X] [Y] → Declaration

Total elements = X * Y

. Actual values

arr [i] [j]

X



2D Address

$(i - X_0) \times C + (j - Y_0)$

i, j = index of the value | arr[2][2]

X_0, Y_0 = 0th index = (0, 0) by default

C = no. of columns. = Y

Q. For arr[2][2]

$(2-0) \times 4 + (2-0) = 2 \times 4 + 2 = 10$

$C = 4$

This is where we find the array declaration

2×4 = Total no. of elements traversed covering 2 rows

Q:) You are given a 2D array

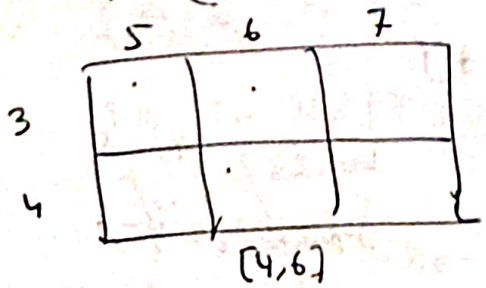
- int mat[3..4][5..7]
- Base address = 1000
- Find the address of mat[4][6]

int mat[3..4][5..7]

$X_0 = 3 \quad Y_0 = 5$

$x = (4-3) + 1 = 2$

$Y = (7-5) + 1 = 3 = C$



$i = 4 \quad j = 6$.

No. of elements preceeding mat[4][6]

$= (i - X_0) \times C + (j - Y_0)$

$= (4-3) \times 3 + (6-5)$

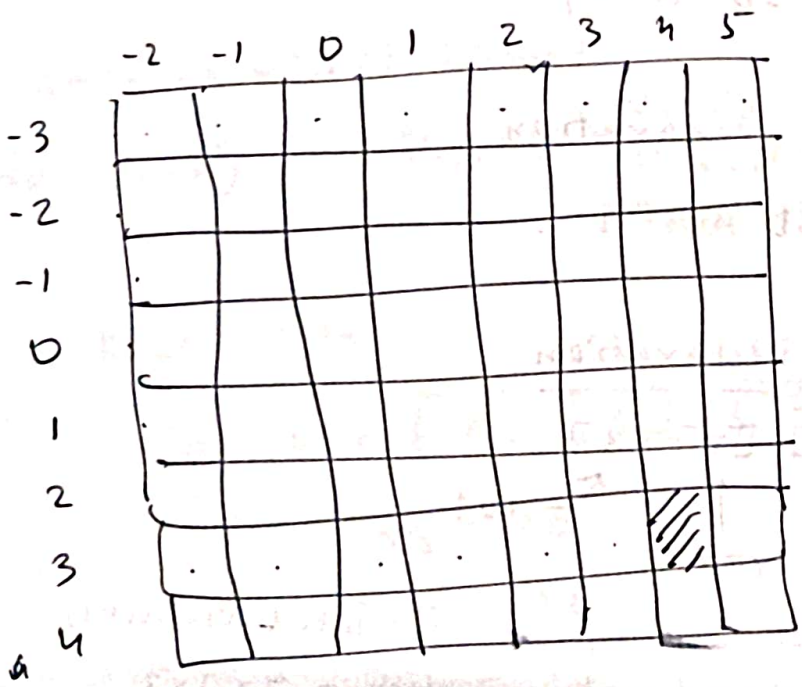
$= 1 \times 3 + 1 = 4$

addr (mat[4][6]) = B.A + 4×4.

$= 1000 + 16 = 1016$.

Q:- int arr[-3..4][-2..5]

- stored in row-major order.
- ~~Each~~ B.A = 1000.
- Compute the address arr[3][4] & also draw the ~~dig~~ diagram.



$X_0 = -3 \quad Y_0 = -2$

$X = 4 - (-3) + 1 = 8$

$Y = 5 - (-2) + 1 = 8$
$= C$

$i = 3$

$j = 4$.

no. of elements preceeding arr[3][4]

$(i - X_0) \times C + (j - Y_0)$

$= (3 - (-3)) \times 8 + (4 - (-2))$

$= 6 \times 8 + 6 \quad = 48 + 6 = 54$.

2D Array Address

$$(i - i_0) \times C + (j - j_0)$$

C = no. of columns
i_0 = 0 for default
j_0 = 0 " "

addr (arr[3][4]) = 1000 + 54 × 4

H.W.

int arr[15..25][70..90]

Find the address of arr[24][89]

BA = 1000 Size of int = 4 bytes Ans = 1832

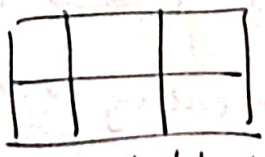
3D Array

- Declaration of 3D array
  - show code
- Visualize the declaration
- Address of 3D array.

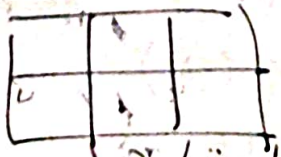
Declaration

int arr[2][2][3]

- [2] ← no. of arrays
- [2] ↑ row
- [3] ← col.



1 / index = 0
~~start~~



2 / index = 1

Total elements
= 2 × 2 × 3
= 12

```
int arr [2] [2] [3] =
{   // array 1
    {   {1, 2, 3},  - row 1
        {4, 5, 6}.  - row 2
    },
  // array 2
    {
        {7, 8, 9},  - row 1
        {10, 11, 12}.  - row 2
    }
}
```

. Now to show the code.

How to find the address

~~addr (arr [i] [j] [k])~~

addr (arr) = B.A + No. of elements preceding that element × size (Data type).

Declaration

```
int arr [X] [Y] [Z]
```

$X = X_{UB} - X_0$

$Y = Y_{UB} - Y_0$

$Z = Z_{UB} - Z_0$

element = arr [i] [j] [k]

No. of elements preceding $= (i - X_0) \times (Y \times Z) + (j - Y_0) \times Z + (k - Z_0)$

$= (i - x_0) \times (Y \times Z) + (j - Y_0) \times Z + (K - Z_0)$

ex:- Given a 3D array initialized as

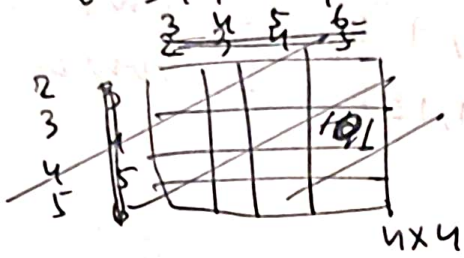
Base Address = 1000

a[5..10] [2..5] [3..6]

→ Find the location of a[8][4][5] ?

- rows = 5 - 2 + 1 = 4

cols = 6 - 3 + 1 = 4



→ nxt pg

5 — (4x4) = 16

6 — (4x4) = 16 } = 48

7 — (4x4) = 16

8 — (4x4)

9 — (4x4)

10 — (4x4)

Total no. of arrays = 10 - 5 + 1 = 6

Total no. of elements = 6 x (4x4) = 96

int a[6][4][4]

X = 6 Y = 4 Z = 4.

$x_0 = 5$ $Y_0 = 2$ $z_0 = 3$.

→ a[8][4][5]

i = 8 j = 4 K = 5

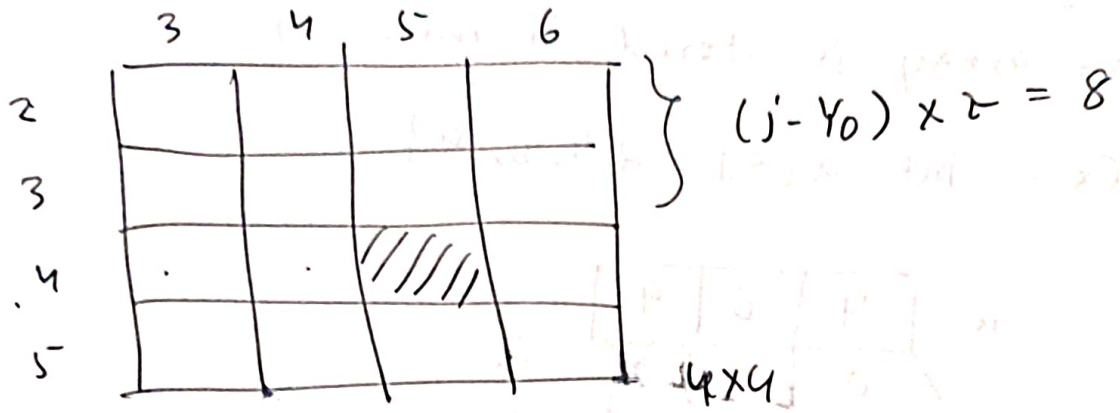
$(i - x_0) \times (Y \times Z)$ = (8-5) x (4x4)

= 3 x 16 = 48

(204)

$(j - Y_0) \times z = (4-2) \times 4 = 2 \times 4 = 8$.

$(k - z_0) = 5 - 3 = 2$



~~8th index array~~

8th index array.

No. of elements

$= 48 + 8 + 2 = 58$

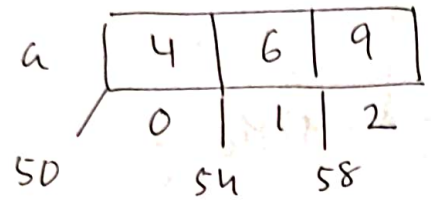
addr(a[8][4][5]) = 1000 + 58 × 4.

= 1232

Row-major order [C/C++ style]
Col-major order [FORTRAN]

① Let us understand how the values of an array is stored in memory.

Ex:- int a[3] = {4, 6, 9}



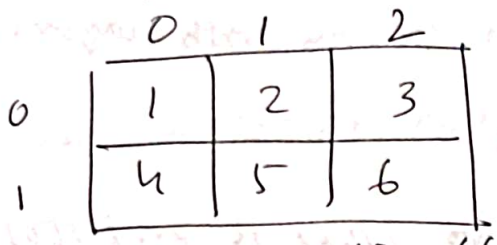
1 byte = 8 bits.

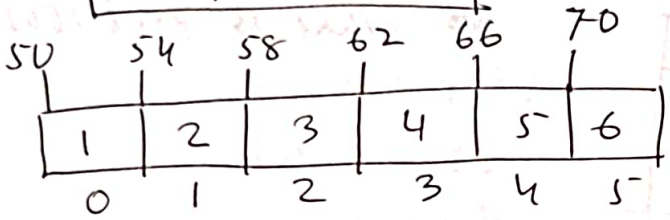
What is the max value that can be represented in 1 byte

- Unsigned = $2^{32} - 1$
- Signed = -2^{31} to $2^{31} - 1$

Let's look at a 2D Array

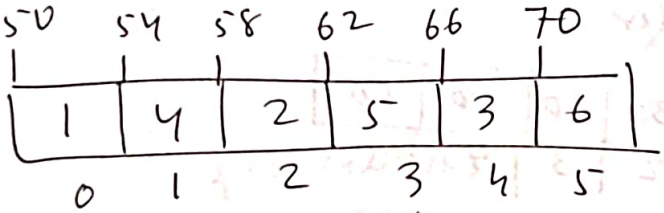
```
int arr[2][3] =
    { {1, 2, 3},
      {4, 5, 6}
    }
```





=> row-major order.

~~for~~



=> col major order.

How do we find the address of an element in col-major order

~~row-major~~

For 2D array, for

row major order

$= (i - X_0) \times C + (j - Y_0)$.

C = no. of cols.

col-major order

$= (j - Y_0) \times R + (i - X_0)$.

R = no of rows

(P.T.O)

Let's look at an example

ex:- int arr [3][2] =

```
{
    {10, 20},
    {30, 40},
    {50, 60}
}
```

- Base address = 100
- Find the addr (arr [1][0]) in both layouts.

- Step 1 : visualize this

	0	1
0	10	20
1	30	40
2	50	60

- What is arr [1][0] = 30

(a) row major order =

10	20	30	40	50	60
0	1	2	3	4	5

100 104 108 112 116 120.

addr (arr [1][0]) = 108.

let's verify using the formula.

$(i - x_0) \times c + (j - y_0)$. = no. of elements before arr [1][0]

$i = 1 \quad j = 0 \quad x_0 = 0 \quad y_0 = 0 \quad c = 2$

$\Rightarrow (1-0) \times 2 + (0-0) = 2$

$\therefore$ addr (arr [1][0]) = $100 + 2 \times 4 = 108$

(b) col major order

10	30	50	20	40	60
0	1	2	3	4	5

100 104

Formula = $(j - y_0) \times R + (i - x_0)$

$= (0-0) \times R + (1-0) = 1$

Address = $100 + 1 \times 4 = 104$.

How to find the address

~~① col-major order - 2D array~~

② col-major order - 3D array.

addr (A[i][j][k])

$= (k - z_0) \cdot (X \cdot Y) + (j - Y_0) \cdot X + (i - X_0)$

ex:-

```
int A[2][3][4]
= {
    {
      {1, 2, 3, 4},
      {5, 6, 7, 8},
      {9, 10, 11, 12},
    },
    {
      {13, 14, 15, 16},
      {17, 18, 19, 20},
      {21, 22, 23, 24},
    }
}
```

Find the address of A[1][1][2]

Visually input

- In column major order, how the values are going to displayed

In each col.

1, 13, 5, 17, 9, 21
2, 14, 6, 18, 10, 22
3, 15, 7, 19, 11, 23
4, 16, 8, 20, 12, 24

show the code.

[0] i

j \ k	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

[1]

	0	1	2	3
0	13	14	15	16
1	17	18	19	20
2	21	22	23	24

the loop in the code

~~for (int k = 0; k~~

int A[2][3][4]

Technically $i = 2$, $j = 3$, $k = 4$

```
for (int k=0; k<4; k++)  // fastest varying
    for (int j=0; j<3; j++)
        for (int i=0; i<2; i++)  // slowest varying
            print ( &A[i][j][k])
```

formula of the address

$$(i - X_0) + (j - Y_0) \cdot X + (k - Z_0)(X \cdot Y)$$

- $(i - X_0)$ → least contributes to the address
- $(k - Z_0)(X \cdot Y)$ → contributes the most and fastest varying

---

Topics (Next)

(1) Lower Triangular matrix - address
(2) Find the 2nd largest number in an array
(3) Two pointer

Topics (Next)

1. lower triangular matrix
2. 2nd largest number in the array
3. Two pointer Technique
4. max Sum - Subarray problem
5. Sliding Window technique
6. Remove Dupliate elements in a sorted array.

(210)

Topics

1. Address of lower triangular matrix

- What is lower triangular matrix?
- Address
- Let's take a matri

VDO

* WE STOP HERE
* DETOUR - UDEMY COURSE