Notebook
Pg. No.
Date / /

Section-6

Array Representations

Introduction to Array :-

Arrays :-

int x = 10 ⟶ x | 10 | 100|101

↓

Scalar variable

Array is a collection of ~~elements~~ similar data types.

int A[5];

↓

meter variable

⟶ A

0	1	2	3	4
		15		

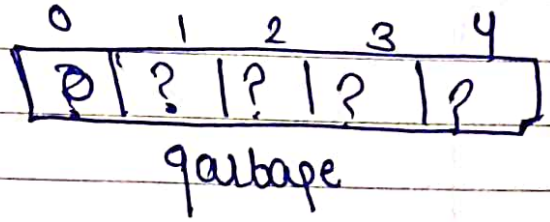
200|1 202|3 204|5 206|7 208|9

A[2] = 15

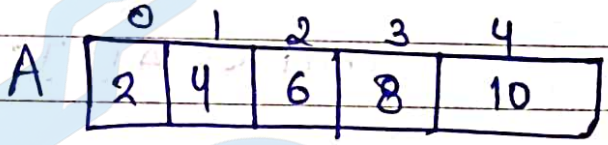
Notebook
Pg. No.
Date / /

• Declaration of Arrays :-

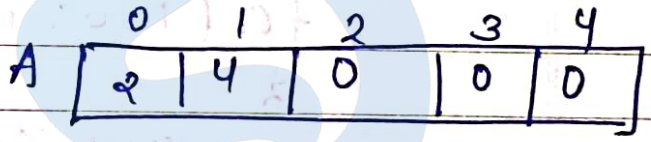
① `int A[5];`



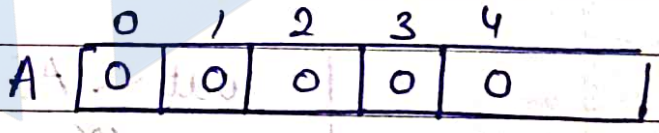
② `int A[5] = {2,4,6,8,10}`



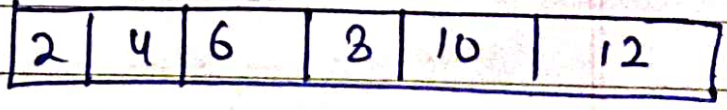
③ `int A[5] = {2,4}`



④ `int A[5] = {0}`

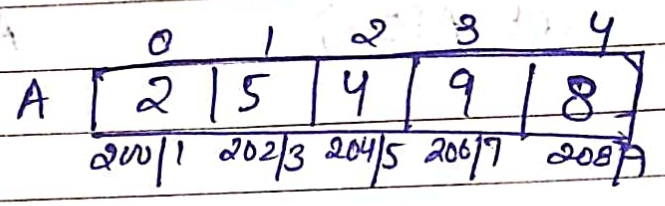


⑤ `int A[] = {2,4,6,8,10,12}`



Notebook
Pg. No. ____
Date ____ / /

int A[5] = {2, 5, 4, 9, 8}



cout << A[0];

cout << A[1];

```
for( i=0; i<5; i++)
{
    cout << A[i];
}
```

cout << A[2];
or
cout << 2[A];

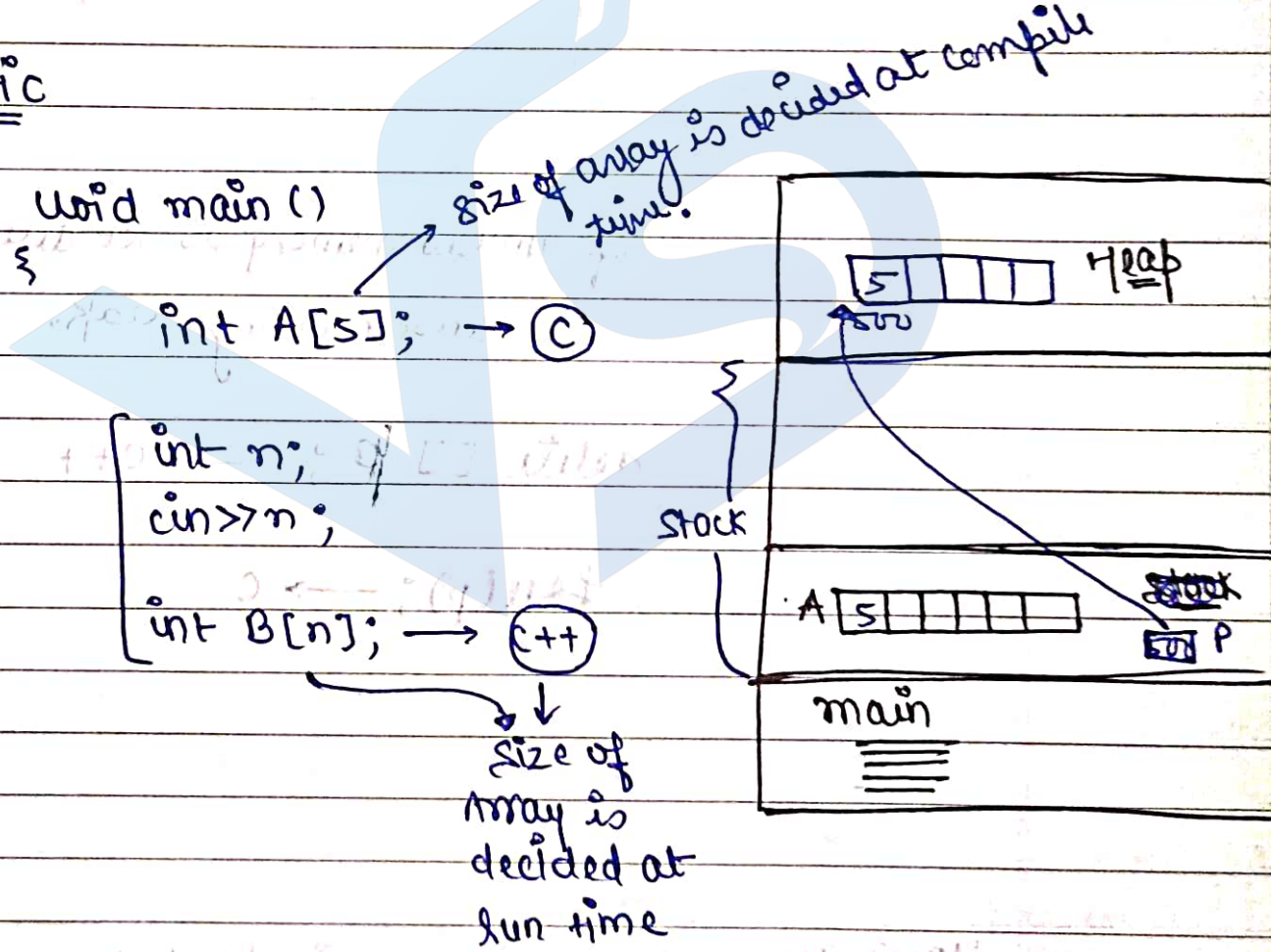
cout << *(A+2)

Static vs Dynamic Array.

Static array means size of array is static.

Dynamic array means size of array is dynamic.

Static



For Accessing Heap, we need pointer

Dynamic Array

```
void main()
{
    int A[5];
    int *p;

    p = new int[5];  ⟶ C++

    p = (int*)malloc(5*sizeof(int));  ⟶ C
```

If unused memory is not deleted then it cause memory leak.

```
    delete [] p;  ⟶ C++

    free(p);  ⟶ C

}
```

How access the array in heap :-

```
int A[5];
int *p;

p = new int[5];
```

A[0] = 5;

P[0] = 5;

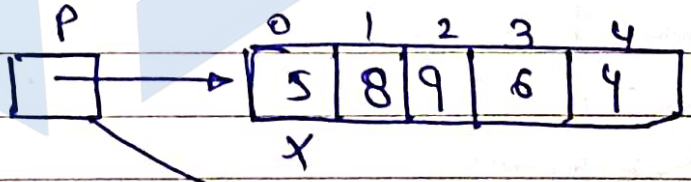
↑

It acts as Heap array name.

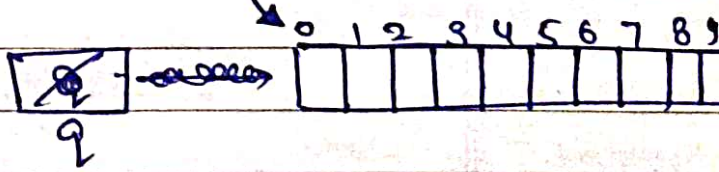
Stack Array size can be Resize.

How to Increase Array size :-

```
int *p = new int[5];
```




```
int *q = new int[10]
```



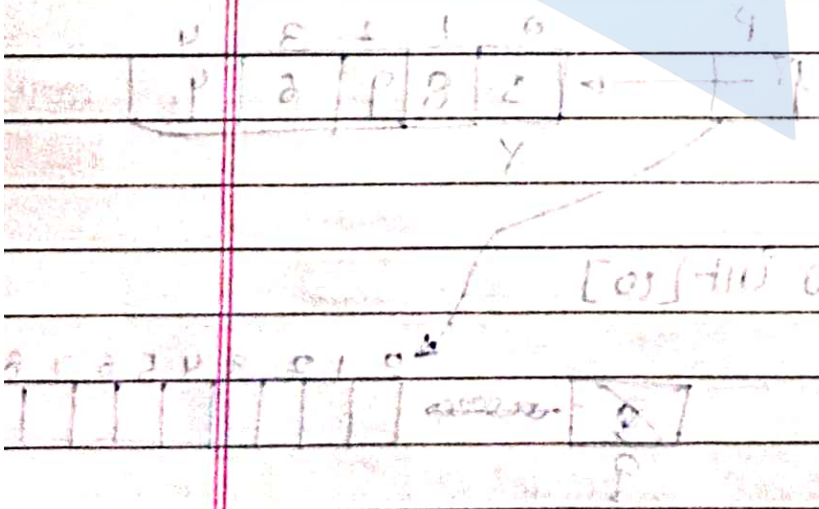

```
for(i=0; i<5; i++)
{
    q[i] = p[i];
}
```

delete[] p;

p = q

q = NULL;

Array is not grown because memory size is contigneous.



Notebook
Pg. No. ____
Date ____ / /

2D Arrays :-

①

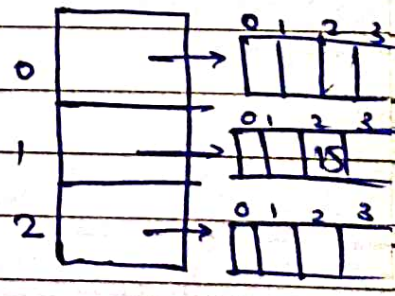
int A.[3][4]; = {{1,2,3,4}, {2,4,6,8}, {3,5,7,9}};

A	0	1	2	3
0	201/1	2/3	4/5	6/7
1	8/9	10/11	12/13 (15)	14/15
2	16/17	18/19	20/21	22/23

A[1][2] = 15 ;

②

int *A[3];



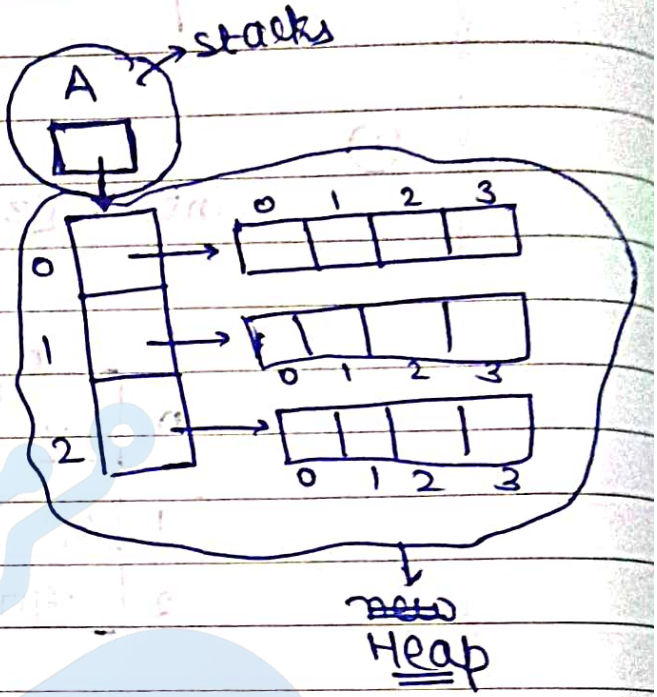
A[0] = new int[4];

A[1] = new int[4];

A[2] = new int[4];

A[1][2] = 15;

③ int **A;



A = new int*[3];

A[0] = new int[4];

A[1] = new int[4];

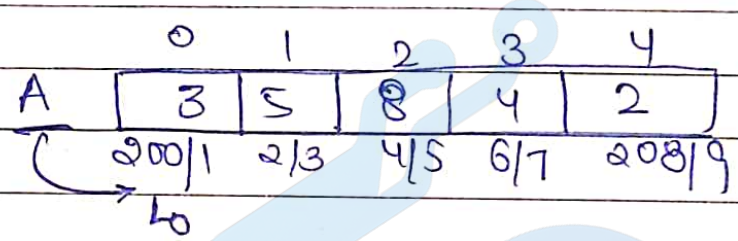
A[2] = new int[4];

```
for (i=0 ; i<3 ; i++)
{
    For (j=0 ; j<4 ; j++)
    {
        A[i][j] = x;
    }
}
```

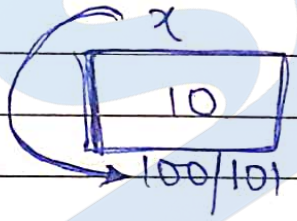
Array Representation by compiles

Arrays in compiles

```
int A[5] = {3, 5, 8, 4, 2};
```




```
int x = 10;
```




```
A[3] = 10;
```

↳ Compiles needs Address (A[3])

Address(A[3]) = 200 + 3*2

= 206

Address(A[3]) = Lo + 3 x size of (int)

Address(A[i]) = Lo + i * w ← size of data type

Base Address, index

Pg. No.
Date / /

int A[1..... 5]

A[3] = 10;

Address(A[3]) = 200 + (3-1) * 2

= 204

Address(A[3]) = Lo + (i-1) * w

Row Major formula in 2D Arrays :-

int A[3][4]; A[m][n]

	0	1	2	3	4	5	6	7	8	9	10	11
A	a_{00}	a_{01}	a_{02}	a_{03}	a_{10}	a_{11}	a_{12}	a_{13}	a_{20}	a_{21}	a_{22}	a_{23}
	0/1	2/3	4/5	6/7	8/9	10/11	12/13	14/15	16/17	18/19	20/21	22/23
	Row 0				Row 1				Row 2			

Row Major Mapping →

A	0	1	2	3
0	a_{00}	a_{01}	a_{02}	a_{03}
1	a_{10}	a_{11}	a_{12}	a_{13}
2	a_{20}	a_{21}	a_{22}	a_{23}

A[1][2] = 10;

Address(A[1][2]) = 200 + [4*1 + 2] * 2

= 200 + 12 ⇒ 212

Notebook
Pg. No.
Date / /

Two types of Mapping :-

- Row-major Mapping
- column-Major Mapping

Address(A[2][3]) = 200 + (2*4 + 3)*2

= 200 + 22

= 202

for Row Major

$$\text{Address}(A[i][j]) = Lo + [i * n + j] * w$$

int A[1....3][1.....4]

$$\text{Address}(A[i][j]) = Lo + [(i-1) * n + (j-1)] * w$$

↓

costlier and take more time as compare to above formula.

Notebook
Pg. No.
Date / /

Column Major formula for 2D Arrays :-

	0	1	2	3
0	a_{00}	a_{01}	a_{02}	a_{03}
1	a_{10}	a_{11}	a_{12}	a_{13}
2	a_{20}	a_{21}	a_{22}	a_{23}

0	1	2	3	4	5	6	7	8	9	10	11
a_{00}	a_{10}	a_{20}	a_{01}	a_{11}	a_{21}	a_{02}	a_{12}	a_{22}	a_{03}	a_{13}	a_{23}
200/1	2/3	4/5	6/7	8/9	10/11	12/13	14/15	16/17	18/19	20/21	22/23
	col 0			col 1		col 2			col 3		

LO

No. of col. Base

$$\text{Address}(A[1][2]) = 200 + (2*3 + 1)*2$$

(total element in column)

$$= 214$$

$$\text{Address}(A[1][3]) = 200 + (3*3 + 1)*2$$

(No. of element in column) (size of int)

$$= 200 + (9+1)*2$$

$$= 200 + 20$$

$$= 220$$

formula for Column Major → A[m][n]

$$Address(A[i][j]) = L_0 + [j * m + i] * w$$

right-left

$$Address(A[i][j]) = L_0 + [i * n + j] * w$$

left-right

Row Major → C/C++ follow row-major

Formulas for nD Arrays :-

Array in compiless

4D

Type A[d1][d2][d3][d4]

$$Addr(A[i_1][i_2][i_3][i_4]) = L_0 + [i_1 \times d_2 \times d_3 \times d_4 + i_2 \times d_3 \times d_4 + i_3 \times d_4 + i_4] * w$$

→ Row Major formula for 4d Array

$$= L_0 + \sum_{p=1}^{n} \left[i_p * \prod_{q=p+1}^{n} d_q \right] * w$$

$$Addr(A[i_1][i_2][i_3][i_4]) = L_0 + [i_4 \times d_3 \times d_2 \times d_1 + i_3 \times d_2 \times d_1 + i_2 \times d_1 + i_1] * w$$

→ Column Major formula for 4d Array

~~Addr~~

$$= L_0 + \sum_{p=n}^{1} \left[i_p * \prod_{q=p-1}^{1} d_q \right] * w$$

General formulae

$$Addr[A[i]...[in]) = Lo + \sum_{p=1}^{n} \left[ip * \bigwedge_{q=p+1}^{n} dq \right] * w$$

↳ Row Major formulae for nd Array

~~Analyzing~~

Analyzing the Row Major formulae for nD Array :-

* Multiplication :-

4D → 3+2+1

5D → 4+3+2+1

nD → n-1+n-2+----3+2+1

$$= \frac{n(n-1)}{2}$$

$$O(n^2)$$

Notebook
Pg. No.
Date / /

Horner's Rule

$$i_4 + i_3 x d_4 + i_2 x d_3 x d_4 + i_1 x d_2 x d_3 x d_4$$

$$i_4 + d_4 * [i_3 + i_2 * d_3 + i_1 * d_2 * d_3]$$

$$i_4 + d_4 * [i_3 + d_3 (i_2 + i_1 * d_2)]$$

for 4D → 3 Multiplication

5D → 4 Multiplication

nD = n-1

O(n)

By taking common, we can reduce the time complexity of Row Major formulae.

Column Major :-

$$\text{Addr}([i_1]\ldots[i_n]) = L_0 + \sum_{p=n}^{1} \left[i_p * \prod_{q=p-1}^{1} d_q \right] * w$$

↳ ~~Row~~ column major formulae for nD array

Formulae for 3D Array :-

int A[l][m][n];

Row-Major :-

Addr[A[i][j][k]) = Lo + [i x m x n + j x n + k] * w

left → right

~~Colum~~

Column Major :-

Addr[A[i][j][k]) = Lo + [k x m x l + j x l + i] * w

right - left

~~int A[m][n]~~

~~int A[l][m][n]~~

Quiz :-

- Let A be two-dimensional array declared as follows:

A: array[1.....10][1.....15] of integers

Assuming that each integer takes one memory location. The array is stored in row-major order and the first element of the array is stored at location 100, what is the address of the element A[i][j]?

→

$L_0 = 100$ $\quad\quad$ L[10][15]

$w = 1$

$$\text{Address}(A[i][j]) = L_0 + [(i-1)*n + (j-1)]*w$$

$$= 100 + [(i-1)*15 + j - 1]*1$$

$$= 100 + 15i - 15 + j - 1$$

$$= 15i + j + 84$$

• what is the output of the following c code? Assume that the address of x is 2000 (in decimal) and an integer requires four bytes of memory.

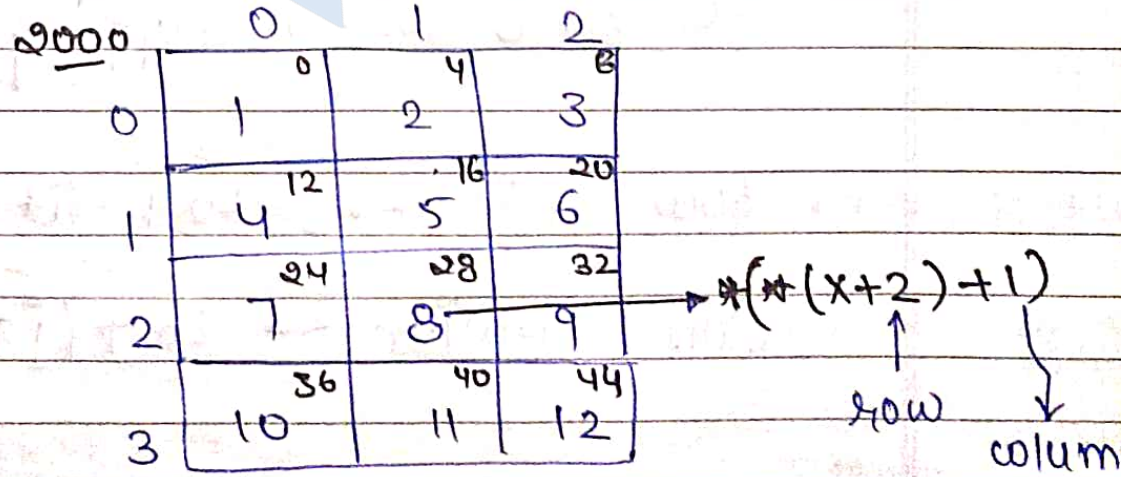
```
int main()
{
    unsigned int x[4][3] = { {1,2,3}, {4,5,6},
                             {7,8,9}, {10,11,12}};

    printf("%u,%u,%u", x+3, *(x+3), *(x+2)+3);

}
```

→ sizeof(int) = 4

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



x+3 = 2036

*(x+3) = 2036

*(x+2)+3 = 2036

Notebook
Pg. No.
Date / /

- If A and B are two matrices, for Multiplying A x B which if the following representation will be efficient.

a) A in Row-major and B in Column-major Representations.

b) A in Column-major and B is Row-Major Representation.

c) Both A and B in Row Major Representation.

d) Independent of Representation

→ A and B are matrices

for A x B which mapping if efficient

Row Major → Lo + (i*n + j)*w

Column Major → Lo + (j*m + i)*w

(d) Independent of Representation

Notebook
Pg. No.
Date / /

- if an 3D Array is declared in C language as follows ?

```
X[?][?][?]
```

find data type and dimensions of Array if compiler performs following intermediate operations for finding Address of any location X[i][j][K] ?

```
t0 = i * 1024      → m*n*w = 1024
                     m = 1024/32 ⇒ 32
t1 = j * 32        → n*w = 32  → n = 8
t2 = K * 4         → w = 4
t3 = t0 + t1
t4 = t3 + t2
t5 = X[t4]
```

(Assume int takes 2 bytes, float takes 4 bytes)

→ type X[l][m][n]

Address(X[i][j][K]) = L0 + (i*m*n + j*n* + K)*w

= L0 + i*m*n*w + j*n*w + K*w ——— (1)

w = 4
n = 8
m = 32

from (1) above

~~Address (x[i][j][k]) =~~

float [x [?] [32] [[illegible]]

↓

Anything