0x100 0x101 ⋯   0x109 0x10A 0x10B ⋯ 0x10E
0x110 0x111

Pointer arithmetic :-

$$\text{int} * P, \quad ^{100} * q;$$

int $* p;$

$p = p + 5;$

0x100   100 + 5 (sizeof (int))

$q = p + 5; \quad \rightarrow 120$

size of (int) = 4

printf ("%d", q-p);

⑤

$P = 0x100 \rightarrow 100$

$q = 0x114 \rightarrow 120$

E ← 15
10 ← 16
11 ← 17
⋮
14 ← 18

$$q - p = \frac{120 - 100}{\text{sizeof (int)}} = ⑤$$

Arrays in C

- Sequential collection of elements of same data type.
- As they are stored sequentially in memory, it gives the advantage to access any element in constant time using the index.

For a type **T**, **T[size]** is the type, **"array of size elements of type T"**.

The elements are indexed from 0 to size-1.

For example:
```
int arr[2];      // an array of two integers: arr[0], arr[1]
char* p[4];      // an array of four pointers to char: p[0], p[1], p[2], p[3]
```

An array can be initialized by a list of values: ✔
- int arr[] = {4, 3, 2, 1}; ✔
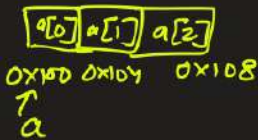- char p[] = {'x', 'y', 'z'}; ✔

Example: Correct or incorrect:
1. int arr1[3] = {1, 2, 3, 4̌}; ✔
2. int arr2[6] = {1, 2, 3, 4}; ✔
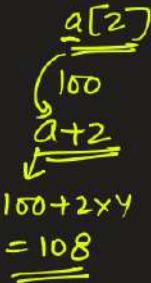3. int arr3[6] = arr2; ✗

(Handwritten annotations:)

$int \ \underline{x, y, z};$

boxes labeled x, y, z

x — $0x\,200$
y — $0x\,384$
z — $0xAAC$

$int \ \underline{a}\,[3];$

boxes: a[0] | a[1] | a[2]

$0x100 \quad 0x104 \quad 0x108$
↑
a

$i^{th} \rightarrow a[i-1]$
n elements
$a[0] \rightarrow a[n-1]$

$a[2]$
100
$a+2$
$100 + 2 \times 4$
$= 108$

$\&a[2] = 100 + 2 \times 4$
3rd element — $(i-1)$ — size of (int)
i^{th}

$[] \ () ✔$

char. a, b, c, d ;

char * p [4] ;

p is an array of 4 character pointers

a ['a'] 1B   b ['b'] 1B   c ['c'] 1B   d ['d'] 1B
0x200   0x300   0x400   0x500

$$\text{sizeof (pointer)} = 8$$

p | 0x200 | 0x300 | 0x400 | 0x500 |
  100   108   116   124

P[0] = &a, P[1] = &b, P[2] = &c, P[3] = &d

* p[2] = 'c'
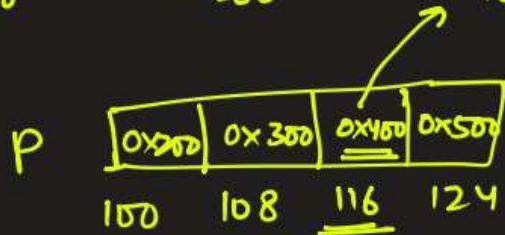
```c
int main() {
    int a[5] = {100,200,300,400,500,600};
    a[500] = 8000;
    printf("%d \n", a[500]);
    return 0;
}
```

// output : 8000

Pointers and Arrays

The name of an array can be used as a pointer to its initial elements,

```c
int v[] = {1, 2, 3, 4};
int* p1 = v;            // pointer to initial element
int* p2 = &v[0];        // pointer to initial element
int* p3 = v+4;
```

```c
#include<stdio.h>
void main(){
    int arr[5] = {1, 2, 3, 4, 5};
    int*p = arr;
    for(int i = 0; i< 5; i++)
        printf("%d  ",*(p+i));
}
```

12 345

P P+1 P+2 P+3 P+4
| . 1 | 2 | 3 | 4 | 5 |
100 104 108 112 116
arr

P 100 int *

*p, *(p+1)
*(p+2) *(p+3)
*(p+4)

int * 116
P3
88

int * 100
P2
88

int int int int
v: | 1 | 2 | 3 | 4 | G |
100 104 108 112 116

v: 100

sizeof (int)=4B
int * 100
P1
8B

*(p2+2) = 3

*(p3 +(p1- p2)-2) = ③

*(p3-2) = 3

int a [5] = {1, 2, 3, 4, 5}

$$a[2] \equiv *(a+2)$$
$$a[i] \equiv *(a+i)$$

printf ("%d", a[4]);
↓ → 5
*(a+4)

a a+1 a+2 a+3 a+4

| 1 | 2 | 3 | 4 | 5 |
100 104 108 112 116

a [100]

7

$$a[0] = *(a+0)$$
$$= *a$$

a[3] = 7

*(a+3) = 7

&3 = 108

&(* 108)

&a[2] = &(* (a+2))
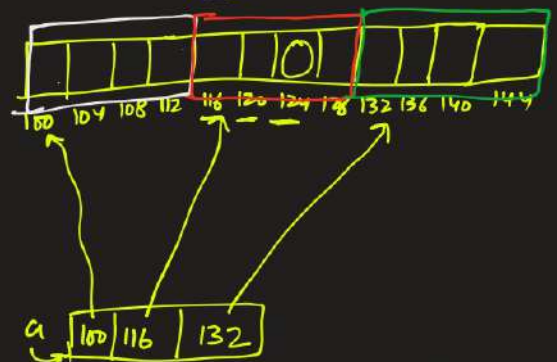= 108 ↑ ↑

= a+2 = 108
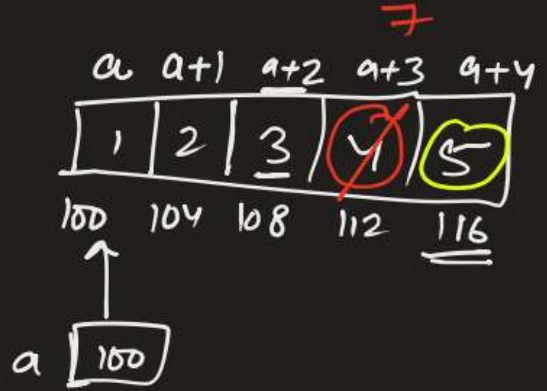
printf ("%d", a[i]);

Multidimensional Arrays

Arrays of arrays are the multidimensional arrays

```
int marr[3][5];    // 3 arrays with 5 elements each
```

```
int arr[3][2] = {{0,1}, {2,3}, {4,5}};
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++)
        printf("%d", arr[i][j]);
}
```

int a[3][4];
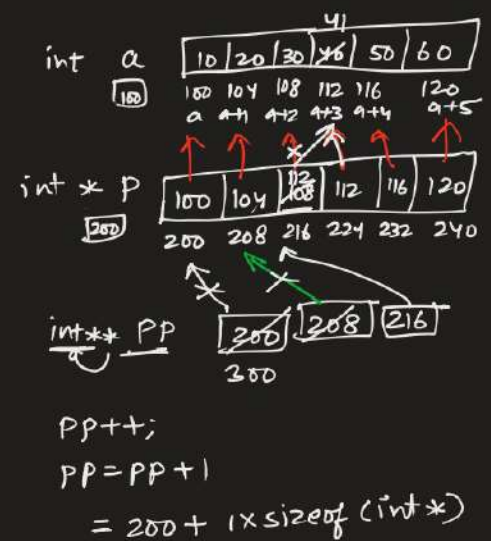
```
eg:    int main() {
          int a[] = { 10,20,30,40,50,60};
          int *p[] = {a, a+1, a+2, a+3, a+4, a+5};
          int ** pp = p;

          pp++;                        ①      ①      ⑳
          printf ("%d, %d, %d", pp-p, *pp-a, ** pp);

          *pp++;                       ②      ②      ㉚
          printf ("%d, %d, %d", pp-p, * pp-a, ** pp);

          ++*pp;                       ②      ③      ㊵
          printf ("%d, %d, %d", pp-p, *pp-a, ** pp);

          ++**pp;                      ②      ③      ㊶
          printf ("%d, %d, %d", pp-p, *pp-a, **pp);

          return 0;
       }
```

int   a | 10 | 20 | 30 | 40 | 50 | 60 |
[100]   100 104 108 112 116   120
            a   a+1 a+2 a+3 a+4 a+5

int * P | 100 | 104 | 108 | 112 | 116 | 120 |
[200]   200   208   216   224   232   240

int** PP | 200 | 208 | 216 |
            300

PP++;
PP = PP + 1
   = 200 + 1 × sizeof (int *)

                    right to left
post inc/dec > pre inc/de = *

Example (GATE CS 2024 Set 2)

What is the output of the following C program?

```c
#include < stdio.h >
int main() {
double a[2]={20.0, 25.0}, *p, *q;
p = a;
q = p + 1;
printf("%d,%d", (int)(q - p), (int)(*q - *p));
return 0;}
```
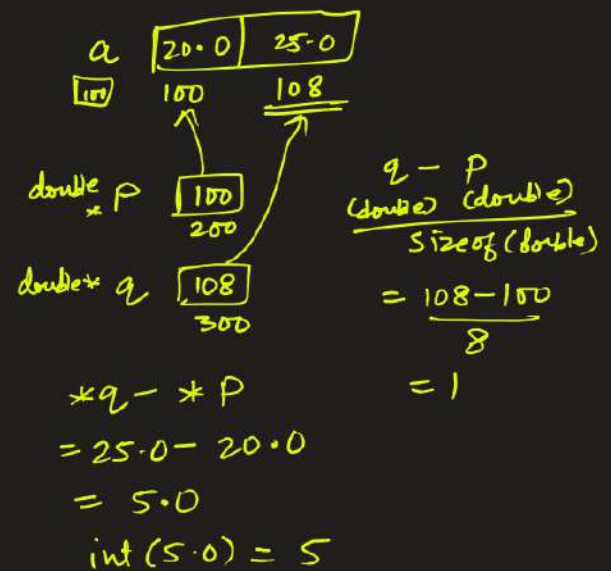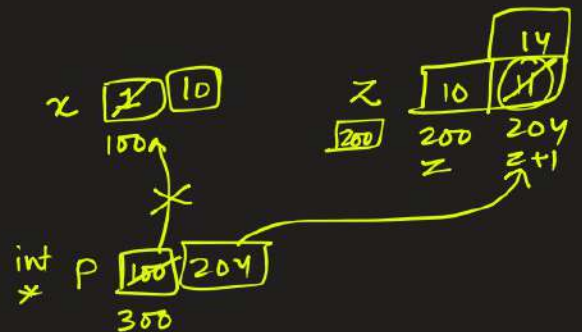
A 4,8

B 1,5

C 8,5

D 1,8

a | 20.0 | 25.0 |
int 100   108

double *p [100]
        200

double* q [108]
         300

$*q - *p$
$= 25.0 - 20.0$
$= 5.0$
$int(5.0) = 5$

$\dfrac{q - p}{(double)\ (double)}$
$\dfrac{}{size\ of\ (double)}$

$= \dfrac{108 - 100}{8}$

$= 1$

What is printed by the following ANSI C program?

```
#include < stdio.h >
int main(int argc, char *argv[])
{
int x = 1, z[2] = {10, 11};
int *p=NULL; p=&x;
*p=10;
p =&z[1];
*(&z[0]+1)+=3;
printf("%d, %d, %d \n",x,z[0],z[1]); return    0;
}
```

10, 10, 14

A 1, 10, 11

B 1, 10, 14

C 10, 14, 11

D 10, 10, 14

x [₁] 10
100↑

int
*   P [100] 204
300

&z[0]

200+1
↓
*204

z [10] [14 / ₁₁]
[200] 200   204
z   z+1

&z[1]
= &(*(z+1))
= z+1

Multidimensional Arrays
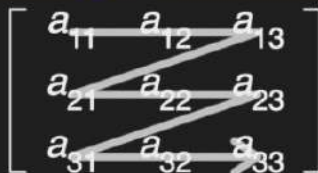
Arrays of arrays are the multidimensional arrays

```
int marr[3][5];    // 3 arrays with 5 elements each

int arr[3][2] = {{0,1}, {2,3}, {4,5}};
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++)
        printf("%d, ", arr[i][j]);
}
```
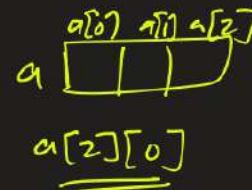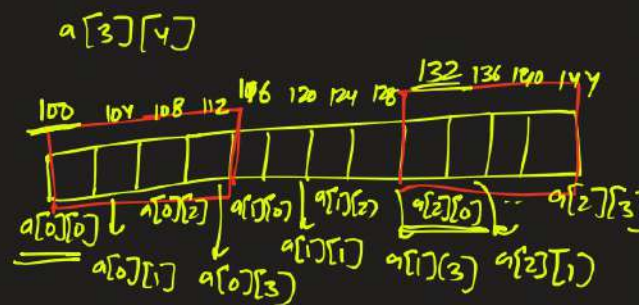
→ rows
→ columns
→ 0, 1, 2, 3, 4, 5

There are two methods of storing multidimensional arrays:

Row major order: The elements are arranged consecutively along the row.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Address	0	1	2	3	4	5	6	7	8
Element	a_{11}	a_{12}	a_{13}	a_{21}	a_{22}	a_{23}	a_{31}	a_{32}	a_{33}

Address[row][column] = Base address + row*num_columns*element_size + column*element_size

a[3][4]

100 104 108 112 116 120 124 128   132 136 140 144

a[0][0]   a[0][2]   a[0][0]   a[1][2]   a[2][0]   a[2][3]
a[0][1] a[0][3]   a[1][1] a[1][3] a[2][1]

a[0] a[1] a[2]
a

a[2][0]

col0 col1   100 104 108 112 116 120

row 0   0   0 1 2 3 4 5

row 1   2   3

row 2   4 5

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

rows = 3

columns = 4
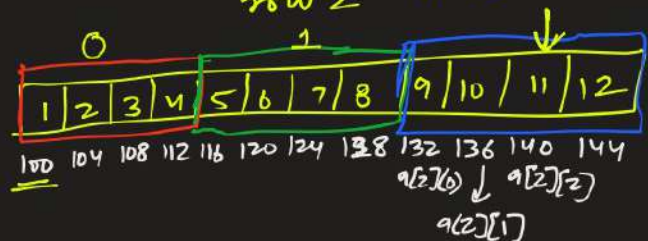
every row contained 4 columns

Base address (BA) = 100

address a[2][2] = ?

2 rows completely passed
in the 3rd row, 2 elements passed and we are at 3rd element

Total elements passed = 2 × 4 + 2 = 10
                          ↑   ↑
                     # columns   # columns passed in 3rd row
          # rows passed

	col 0	col 1	col 2	col 3
a: ✓ row 0	1	2	3	4
✓ row 1	5	6	7	8
row 2	9	10	11	12

a:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

100 104 108 112 116 120 124 128 132 136 140 144

a[2][0]   ↓ a[2][2]
        a[2][1]

---

address of a[2][2] = 100 + 10 x 4
            = 140

0 elements -> Base Address = 100 + 0 x 4 = 100
1 elements -> base address = 100 + 1 x 4 = 104
2 elements -> base address = 100 + 2 x 4 = 108
.
.
.
10 elements -> base address = 100 + 10 x 4 = 140

Example (GATE CS 2002):

row-major order

Consider the following declaration of a 'two-dimensional array in C:
char a[100][100];

100 rows, 100 columns

Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of a[40][50] is

a) 4040
b) 4050
c) 5040
d) 5050

$$\text{address of } a[40][50] = BA + (40 \times 100 + 50) * 1$$
$$= 0 + 4000 + 50$$
$$= 4050$$

Column major order: The elements are arranged consecutively along the column.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Address	0	1	2	3	4	5	6	7	8
Element	a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	a_{32}	a_{13}	a_{23}	a_{33}

Address[row][column] = Base address + column*num_rows*element_size + row*element_size

int a[3][4] = {{1, 2, 3, 4}{5, 6, 7, 8}{9, 10, 11, 12}},   assume CMO

rows   columns

Find the address of   a[2][2]
                          row column

elements passed

= 2 columns * 3

+ 2 rows in colum '2'

= 8

∴ Address of a[2][2] = 100 + 8 × 4

= 132

col 0   col 1   col 2   col 3

row 0   1   2   3   4
row 1   5   6   7   8
row 2   9   10   11   12

col 0   col 1   col 2   col 3

a: | 1 | 5 | 9 | 2 | 6 | 10 | 3 | 7 | 11 | 4 | 8 | 12 |
    100 104 108 112 116 120 124 128 132 136 140 144
    0,0 1,0 2,0 0,1 1,1 2,1 0,2 1,2 2,2 0,3 1,3 2,3
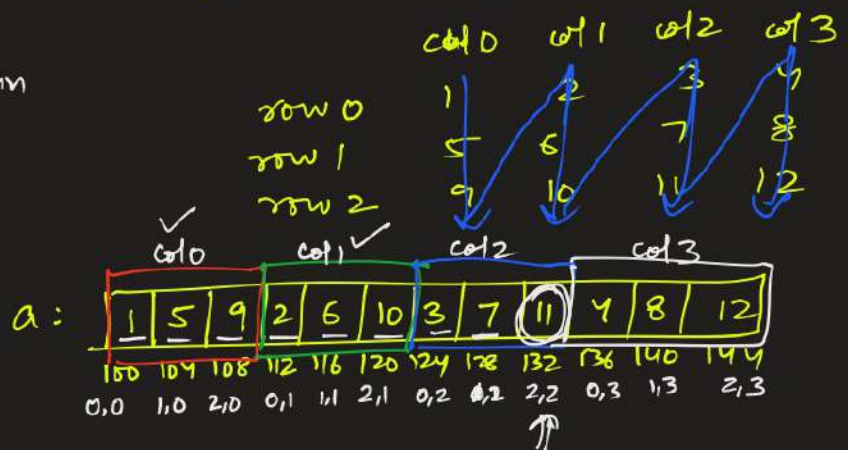
Example [H/W]

Consider a two-dimensional array A with 3 rows and 4 columns stored in memory in column-major order. Given the base address of the array A is 2000, and indices i = 1 (row index) and j = 2 (column index), what is the address of the element A[1][2]? (Assume the array stores integers which hold 4 B of memory)