ARRAYS

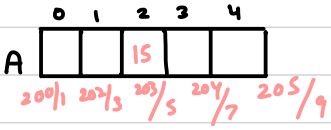
Scalar

```
int x = 10;
```

2 bytes

vector

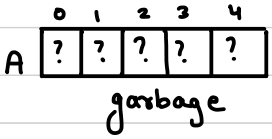
```
int A[5];
A[2] = 15;
```



10 bytes

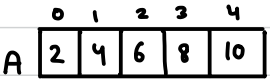
DECLARATION OF ARRAYS

① `int A[5];`

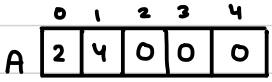


garbage

② `int A[5] = {2,4,6,8,10};`

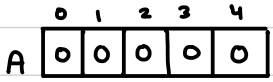


③ `int A[5] = {2,4};`

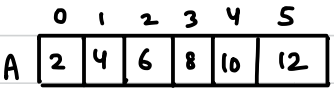


→ Rest of the elements get automatically initialized by 0.

④ `int A[5] = {0};`

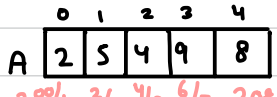


⑤ `int A[] = {2,4,6,8,10,12}`



→ Depending upon the number of elements, size of the array is automatically allocated.

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



→ Array addresses are contagious.

```
printf("%d", A[2]);
printf("%d", 2[A]);
printf("%d", *[A+2]);
```

→ OUTPUT
4

```
for(i = 0; i<5; i++)
    printf("%u", &A[i]);
```

%u → for printing address.

STATIC VS DYNAMIC ARRAY

Size of the array is static

Size of the array is dynamic

→ Once an array is created, it's size cannot be modified.

In C

→ Size of the array is decided at compile time

In C++

→ The size of the array can be decided at run time also.

```
int n;
cin>>n;
int A[n];
```

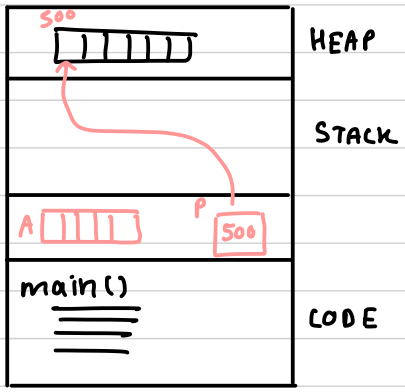
ACCESSING HEAP

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

C++ p = new int[5];

C   p = (int *)malloc(5 * sizeof(int));
                <stdlib.h>
    :
    :

C++ delete []p;
  C free(p);
}
```

p[0] = 5

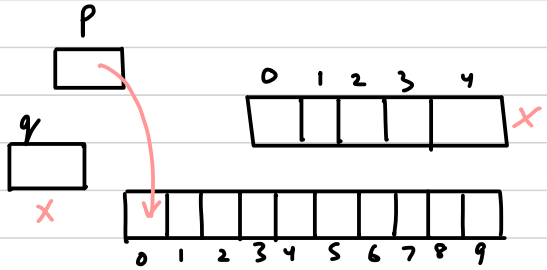
Otherwise MEMORY LEAK

Shortage of memory.



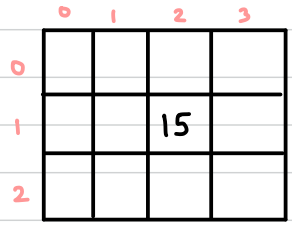
ONE WAY OF INCREASING SIZE OF ARRAY

```
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;
```



2D - ARRAY

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



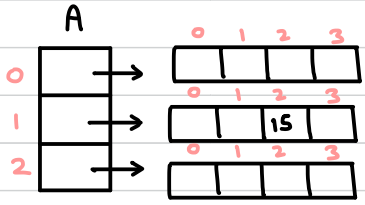
↓ Array will be stored inside stack.

`A[1][2] = 15`

② `int *A[3];` → Array of pointers

```
A[0] = new int[4];
A[1] = new int[4];
A[2] = new int[4];
```

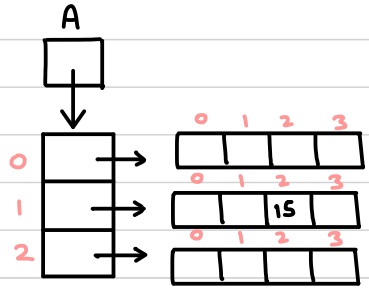
Memory is created inside heap.



`A[1][2] = 15;`

We can access array of pointers in the same way.

③



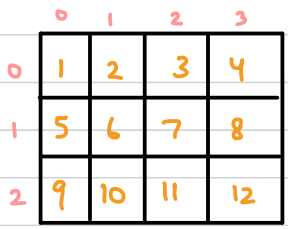

```
int **A;
A = new int*[3];
A[0] = new int[4];
A[1] = new int[4];
A[2] = new int[4];
```

All the memory is allocated in heap.

for column

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

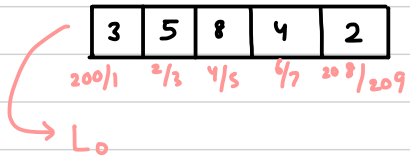
for row



- Elements accessed in order

HOW COMPILER GENERATES FORMULA FOR ADDRESS OF AN ARRAY

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



ADD (A[3]) = 200 + 3 * 2 = 206

ADD (A[3]) = L_0 + 3 * 2

ADD (A[i]) = L_0 + i * w → FORMULA

No of operations = 2

L_0 : Base Address

i : Index

w : Size of datatype.

ADD (A[i]) = L_0 + (i-1) * w → Formula, when indices are starting from one onwards

No of operations = 3

More time consuming.

That's why C and C++ do not start array index with 1. Only for one extra operation, time taken by the program will increase and this will make the program slower.

ROW MAJOR


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

Add (A[1][2]) = 200 + [4 + 2] * 2 = 212

Add (A[2][3]) = 200 + [2 * 4 + 3] * 2 = 222

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

4 operations

Add([i][j]) = L_0 + [(i-1) * n + (j-1)] * w

6 operations

COLUMN MAJOR

$$ADD(A[1][2]) = 200 + [2*3+1]*2 = 214$$

$$ADD(A[1][3]) = 200 + [3*3+1]*2 = 220$$

$$ADD(A[i][j]) = L_0 + (j*m+i)*\omega$$

FORMULAS FOR nD ARRAYS

$A[d1][d2][d3][d4];$ 4D ARRAY

Row Major

$$ADD(A[i_1][i_2][i_3][i_4]) = L_0 + [i_1*d_2*d_3*d_4 + i_2*d_3*d_4 + i_3*d_4 + i_4]*\omega$$

Column Major

$$ADD(A[i_1][i_2][i_3][i_4]) = L_0 + [i_4*d_3*d_2*d_1 + i_3*d_2*d_1 + i_2*d_1 + i_1]*\omega$$

ROW MAJOR FOR nD

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

COLUMN MAJOR for nD

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

(self tried)
(please check)

A[d1][d2][d3][d4]; 4D ARRAY HORNER'S RULE

Row Major

$$ADD(A[i_1][i_2][i_3][i_4]) = L_0 + [i_1 * d_2 * d_3 * d_4 + i_2 * d_3 * d_4 + i_3 * d_4 + i_4] * \omega$$

3 2 1

4D → 3 + 2 + 1

5D → 4 + 3 + 2 + 1

nD → n-1 + n-2 + n-3 .... + 1 = $\frac{n(n-1)}{2}$

$O(n^2)$

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

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

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

$O(n)$

FORMULA FOR 3D ARRAYS

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

Row Major

$$Add(A[i][j][k]) = L_0 + [i * m * n + j * n + k] * \omega$$

Column Major

$$Add(A[i][j][k]) = L_0 + [k * m * l + j * l + i] * \omega$$