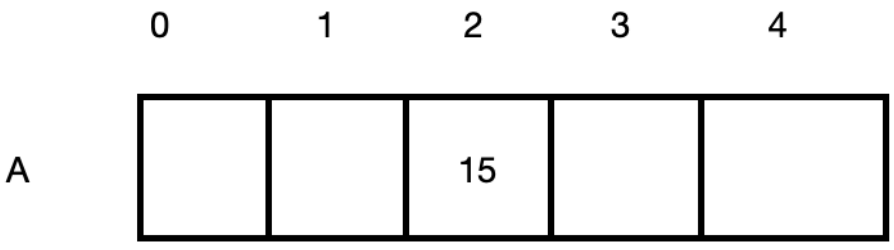
Arrays

- Array is a collection of similar data types grouped under one name
- Its also called vector value
- We can Access or differentiate all the elements in an array using index values
- This concepts is supported by many programming languages

Example :

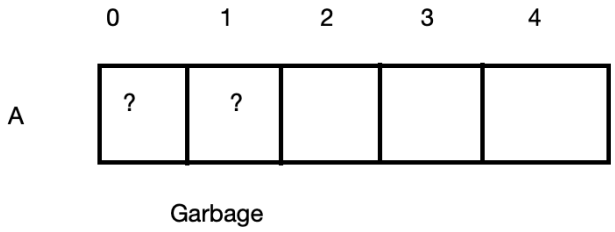
Int A [5]; // Initialise or declaration

A[2] = 15; // Access

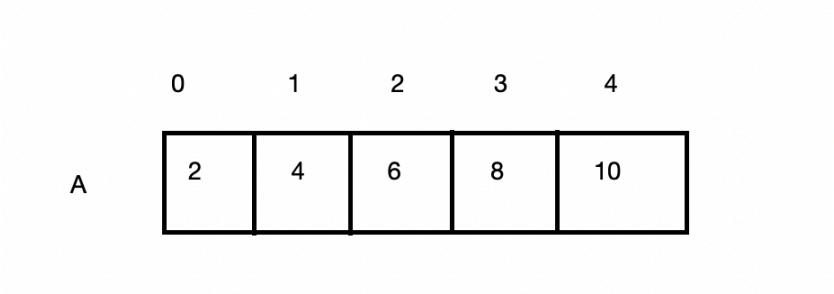


- Some ways of Declaring and initialisation of array are as follows

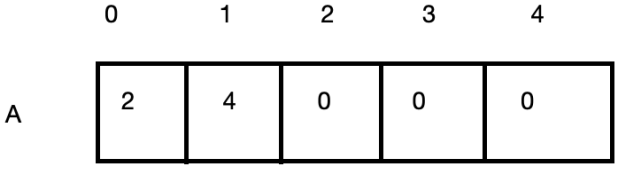
int A[5] ;



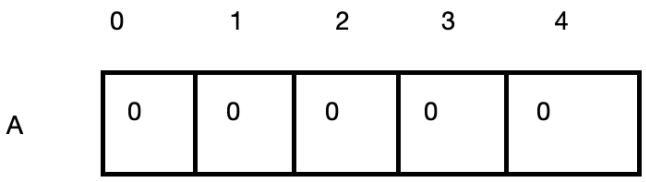
int A[5] = {2,4,6,8,10} ;



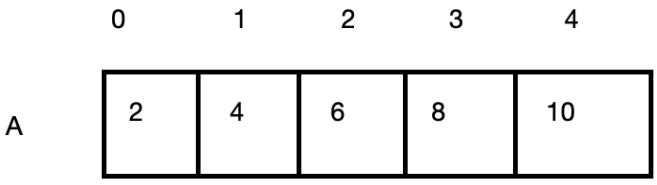
int A[5] = {2,4} ;



int A[5] = {0} ;



int A[ ] = {2,4,6,8,10} ;



- To access all elements in an array , we can traverse through it for example

```
int A[ 5 ] = {2,4,6,8};

for (i = 0; i < 5 ; i++)
{
        printf( “%d”, A[ i ] );

}
```

- The elements inside the array can be access through the subset or through the pointer

```
int A[ 5 ] = {2,4,6,8};

for (i = 0; i < 5 ; i++)
{
        printf( “%d”, A[ i ] );

        printf( “%d”, A[ 2 ] );

        printf( “%d”, 2[ A ] );

        printf( “%d”, *(A + 2 ) );

}
```

// Example of accessing elements inside an array

Static Vs Dynamic Arrays

- Static array means the size of an array is static I.e; fixed
- Dynamic array means the size of an array is Dynamic I.e; flexible
- When an array is created it is created inside Stacking the memory
- The size of the array is decided during at compile time
- When declaring an array it must be a static value only and not variable type in c language however in c++ dynamic allocation is possible during compile time

We can create array inside Heap

- When accessing any value inside a heap it must be done through a pointer

Example :

```
Void main( )
{

        int A[5];
        int *p;
c++     p = new int[5];

C lang  p =( int * ) malloc ( 5* sizeof ( int ) );

.
.
.
```

- When the work in heap is done it must be deleted or it will cause memory leak which will cause problem
- To release the heap memory we do

c++ delete[ ] p;

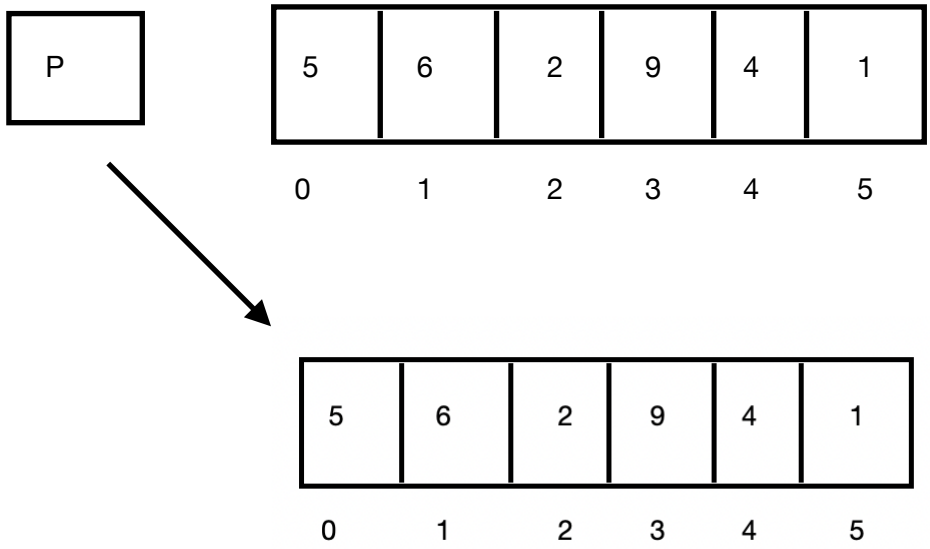
C lang free(p);

Static vs Dynamic Arrays

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int A[5]={2,4,6,8,10};
    int *p;
    int i;

    p=(int *)malloc(5*sizeof(int));
    p[0]=3;
    p[1]=5;
    p[2]=7;
    p[3]=9;
    p[4]=11;

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

    printf("\n");
    for(i=0;i<5;i++)
        printf("%d ",p[i]);

    return 0;
}
```

How to increase Array Size

- An array is created in stack, so in order to increase the array size we can use another pointer of larger size then point it to the array this will transfer all the elements to the new array
- After allotting the array to the new pointer it a must to delete the previous pointer so that there is no memory leakage.
- The command use to delete is delete[ ]



Array Size

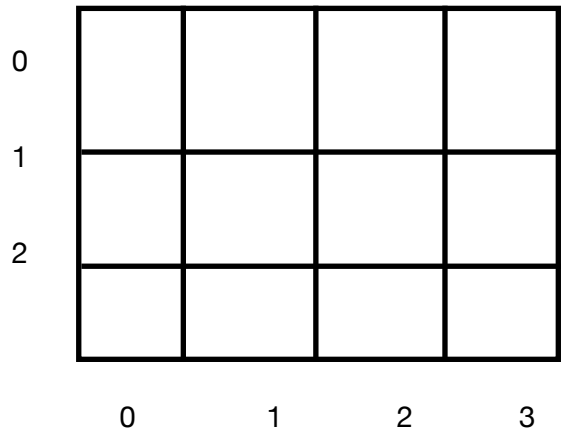
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *p,*q;
    int i;
    p=(int *)malloc(5*sizeof(int));
    p[0]=3;p[1]=5;p[2]=7;p[3]=9;p[4]=11;

    q=(int *)malloc(10*sizeof(int));

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

    free(p);
    p=q;
    q=NULL;

    for(i=0;i<5;i++)
        printf("%d \n",p[i]);

    return 0;
}
```

2D Array

- In programming language it is possible to create multi dimensions arrays
- One of the common multi dimensions array is 2D array this is used to implement matrices
- They are 3 methods of declaring 2D arrays
- We can access 2D array using 2indices (one for row and other for column)

1. Normal Declaration

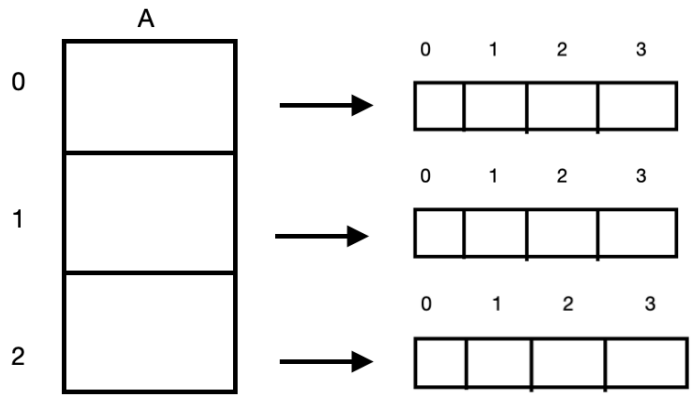
Int A[3][4] ;



- Memory will be created like a single dimension array , but compiler will allow us to access that array as a 2D arrays with rows and columns
- We can directly mention the array list and initialise it
- Ex: int A[3][4] = { {1,2,3,4} , {2,4,6,8} , {3,5,7,9} }
- It is partial is stack

2. Array Of Pointers

```
int * A[ 3 ] ;

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

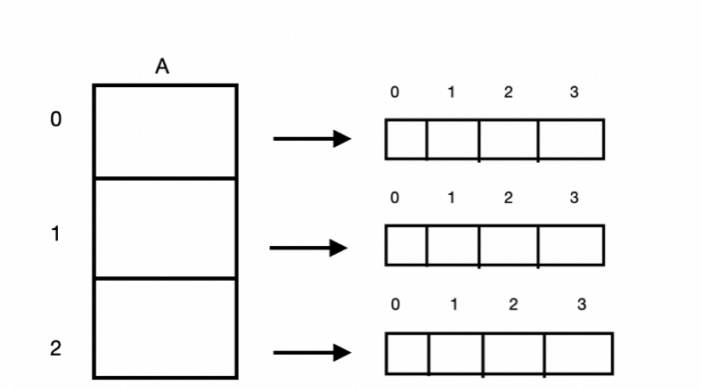


- The pointer will be created inside heap memory and through that we can access , initialise and declare all the elements inside the array
- This is array of integer pointer
- It is partial in Heap

3. Double Pointer

- Here almost everything is inside the heap pointer
- Here the pointer will be like a variable there is no new operator so it is created inside stack in the memory

```
int * *A;

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



- Here everything is inside heap.
-

2D Array

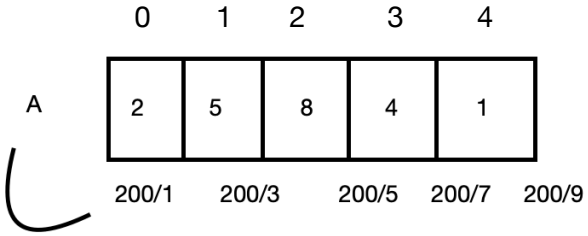
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int A[3][4]={{1,2,3,4},{2,4,6,8},{1,3,5,7}};

    int *B[3];
    int **C;
    int i,j;


    B[0]=(int *)malloc(4*sizeof(int));
    B[1]=(int *)malloc(4*sizeof(int));
    B[2]=(int *)malloc(4*sizeof(int));


    C=(int **)malloc(3*sizeof(int *));
    C[0]=(int *)malloc(4*sizeof(int));
    C[1]=(int *)malloc(4*sizeof(int));
    C[2]=(int *)malloc(4*sizeof(int));

    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
            printf("%d ",C[i][j]);
        printf("\n");
    }

    return 0;
}
```

Array in Compilers

- Any location in an array can be accessed with the help of base address

Example :

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



- The formula used by any compiler to convert it is

Addr(A[i]) = Lo + i * w

Where ;

Lo is - base address
i - Index
w - Size of Data Type

- Base address of an array will be updated when the program starts running and once the memory is allocated
- So the address of this is known during run time
- As the base address is relative the formula for it is also relative formula

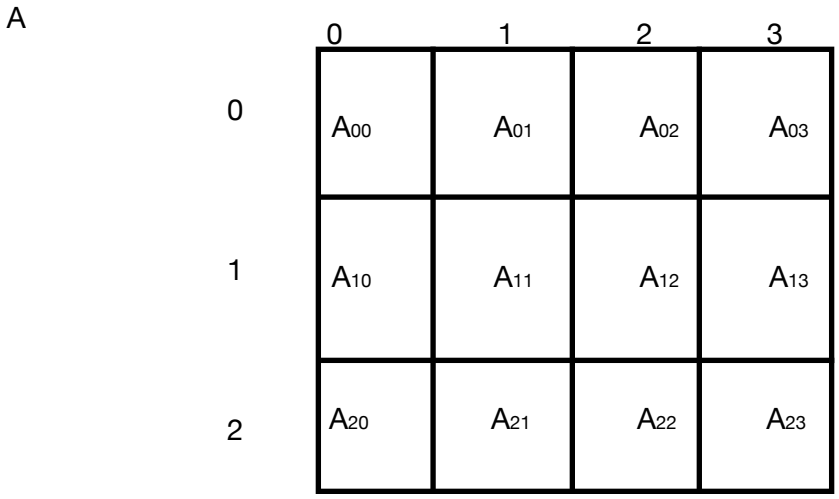
- Suppose a in a different language if the index value starts from 1 then the formula for the compiler is as follows

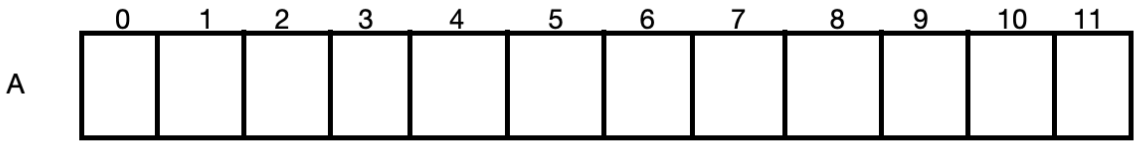
Addr(A[i]) = Lo + (i -1)* w

Row Major Formula for 2D Arrays

- In compiler a 2D array is in linear form (1D Array) in terms of down and column

Example :





The 2D array can be represented in 2 ways

1. Row major mapping
2. Column major mapping

1. Row major mapping

- Here the elements are stored row by row

Example :

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

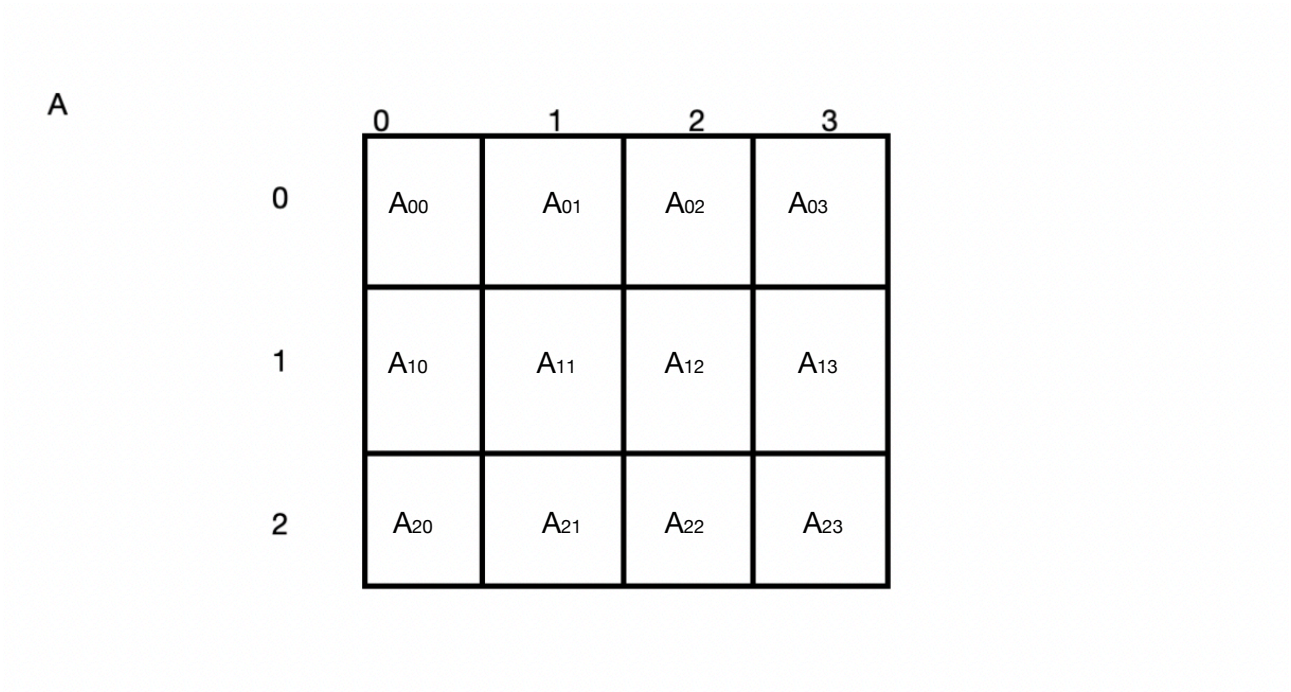
0	1	2	3	4	5	6	7	8	9	10	11
A_{00}	A_{01}	A_{02}	A_{03}	A_{10}	A_{11}	A_{12}	A_{13}	A_{20}	A_{21}	A_{22}	A_{23}

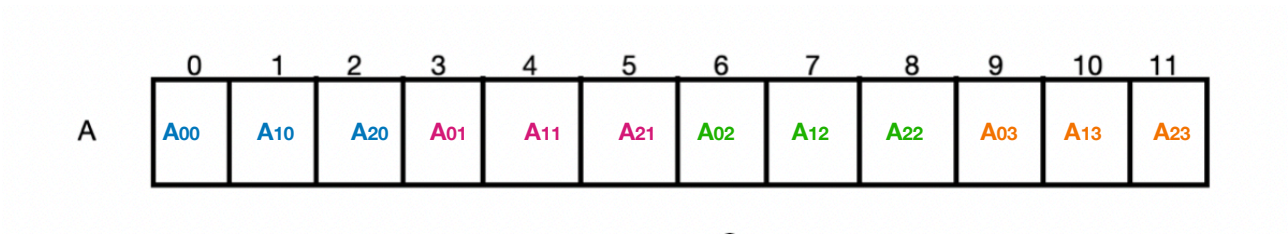
- The formula for accessing this 2D array is

$$\mathbf{Add(A[i] [j]) = L_0 + [i * n + j\ ] * w}$$

Column Major Mapping

- Here the elements of array are stored in column by column





- The formula for obtaining address for each element is

$$\text{Add}(A[i][j]) = L_0 + [i * m + j] * w$$

Ds

Formula for nD Array

Example :

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

Row Major

Add(A[i1] [i2][i3] [i4]) = L0 + [i1 * d2* d3 * d4 + i2 * d3 * d4 + i3 *d4 +i4] * w

Column Major

Add(A[i1] [i2][i3] [i4]) = L0 + [i4 * d1 * d2 * d3 + i3 * d1 * d2 + i2 *d1+i1] * w

Ds

Formula for 3D Array

Example :

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

Row Major :

Addr (A[i] [j] [k]) = L0 + [i * m * n + j * n + k] * w

Column Major :

Addr(A[i] [j] [k]) = L0 + [k * l * m + j * l + i] * w