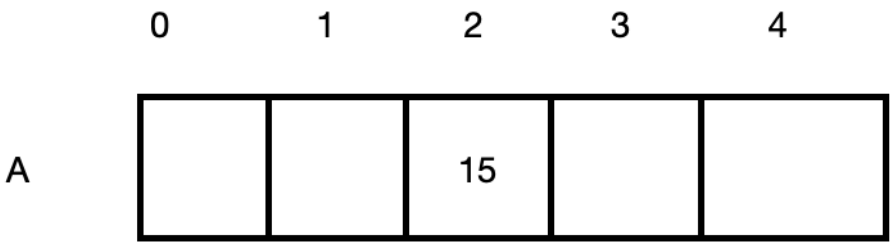
Arrays

- 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 :

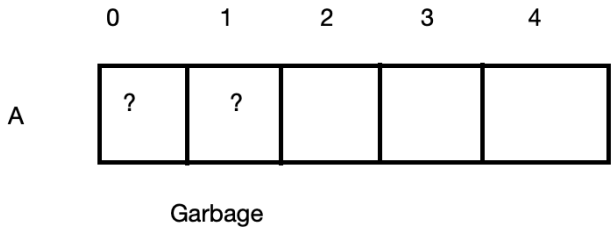
Int A [5]; // Initialise or declaration

A[2] = 15; // Access

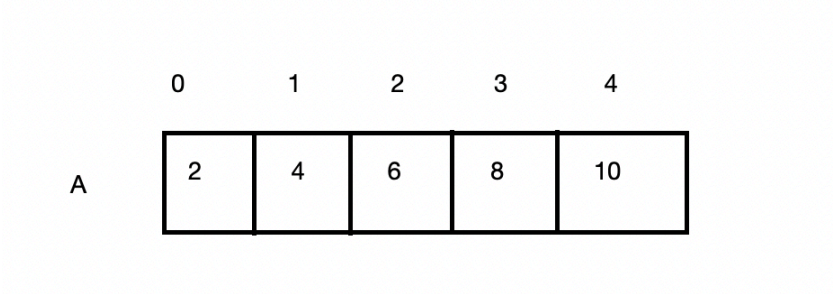


- Some ways of Declaring and initialisation of array are as follows

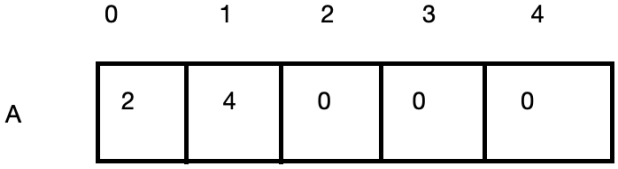
int A[5] ;



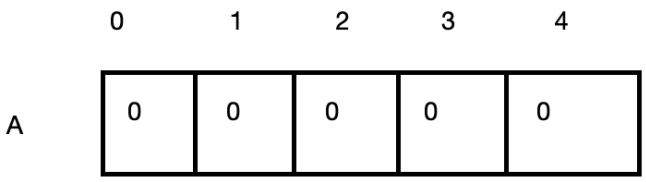
int A[5] = {2,4,6,8,10} ;



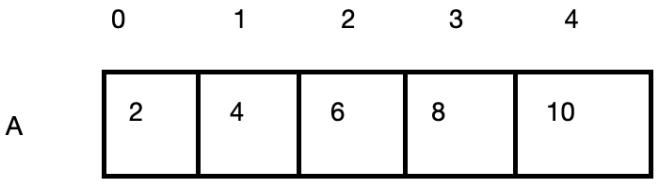
int A[5] = {2,4} ;



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;

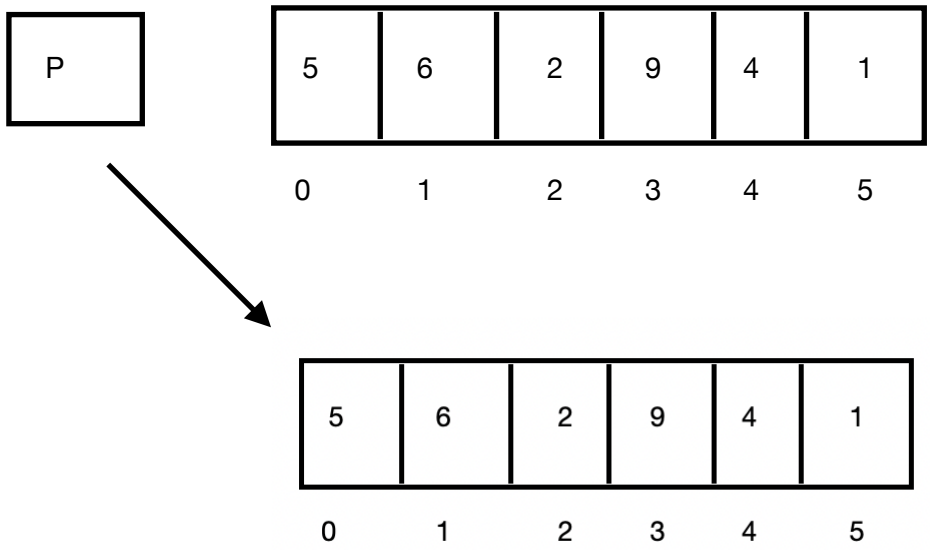
C lang free(p);

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;
}
```

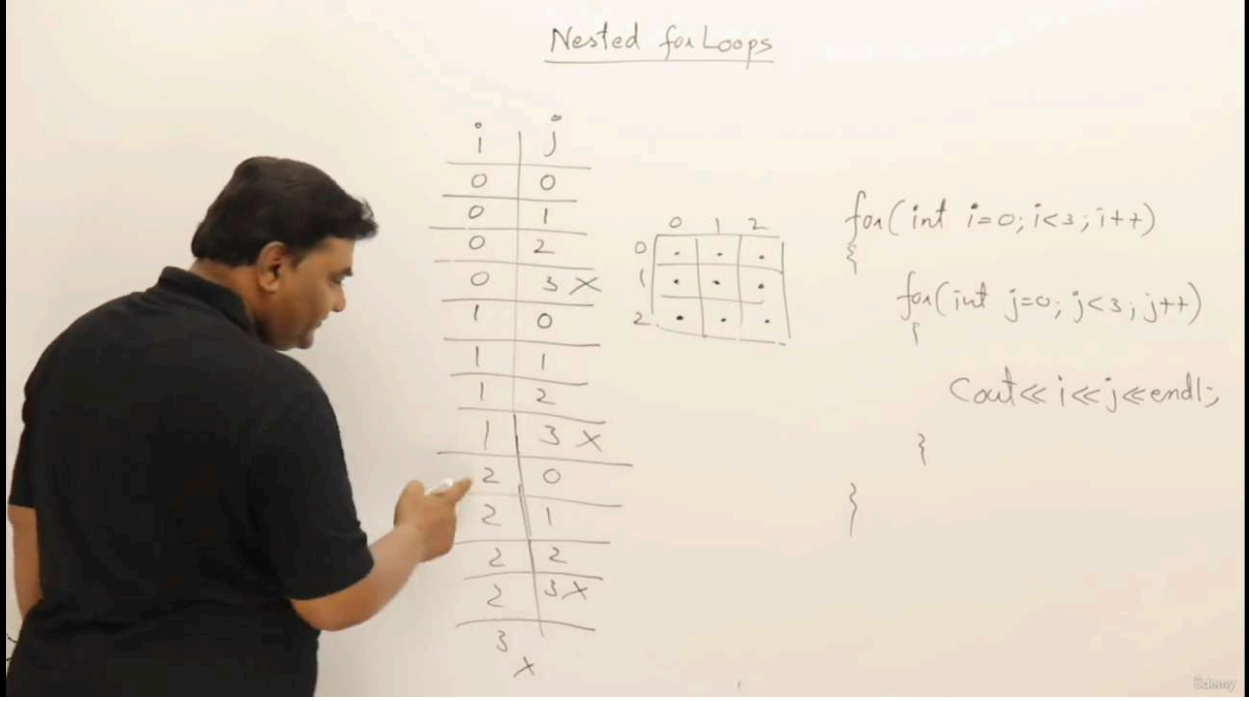
How to increase Array Size

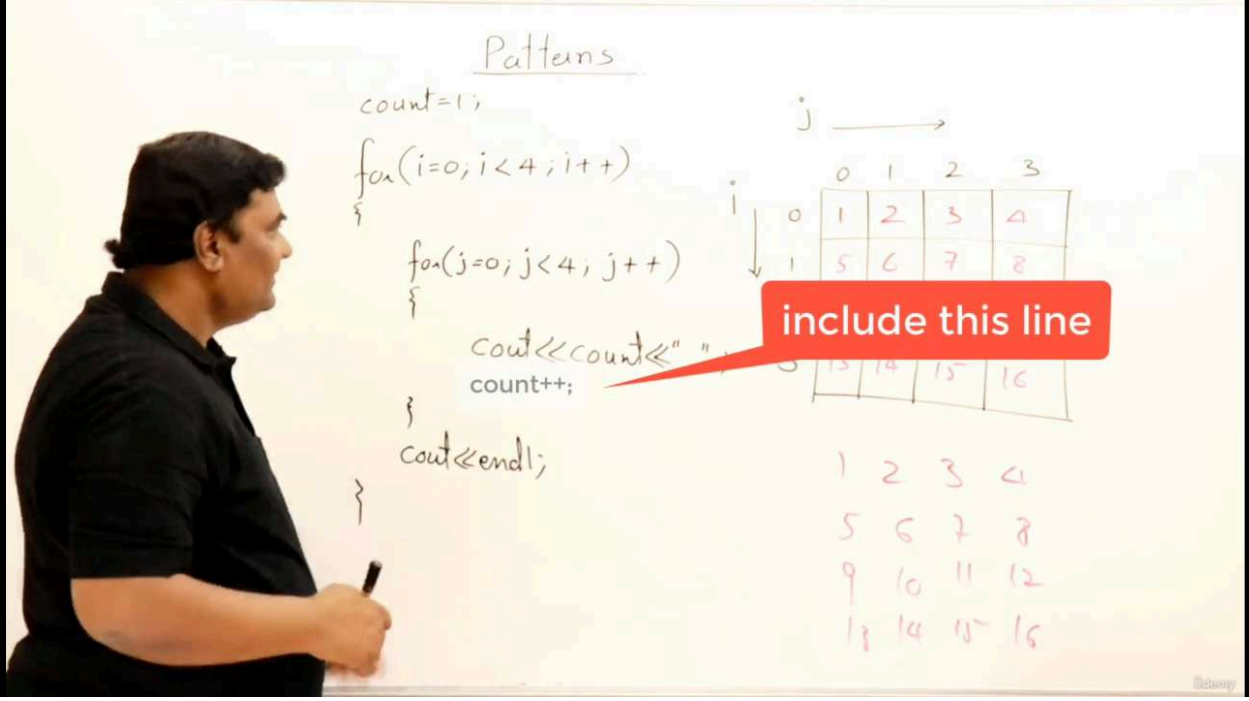
- An array is created in stack, so in order to increase the array size we can use another pointer of larger size then point it to the array this will transfer all the elements to the new array
- After allotting the array to the new pointer it a must to delete the previous pointer so that there is no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



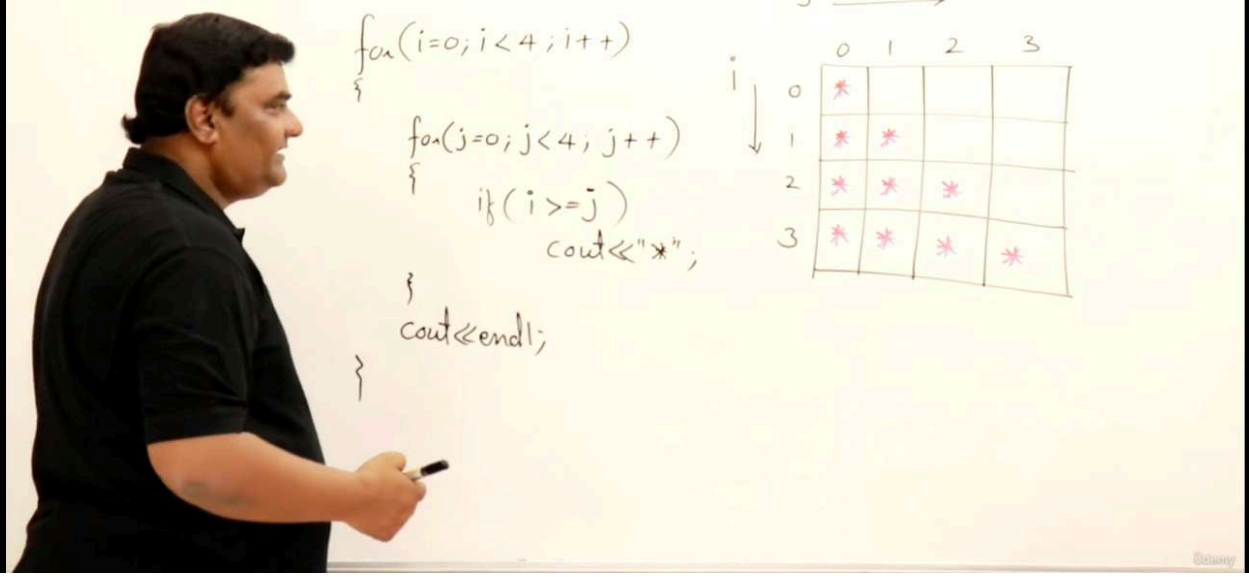
Array Size

```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;
}
```

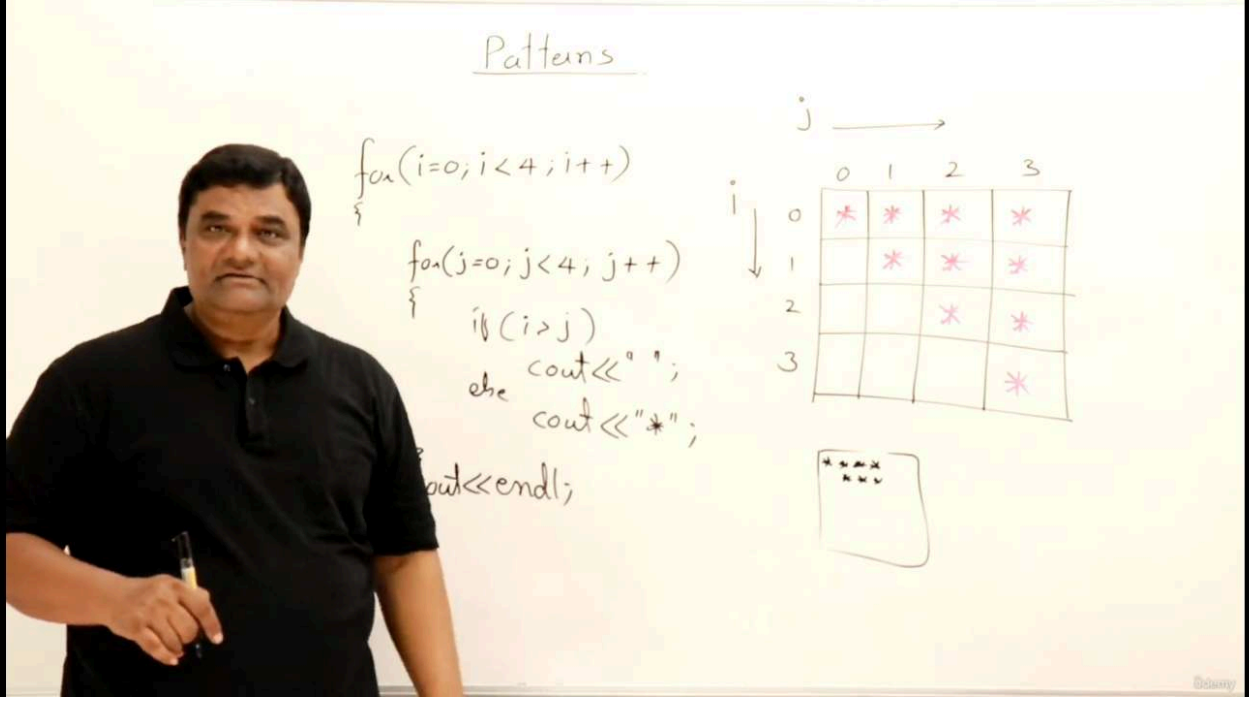

Nested for Loops
i j
0 0
0 1
0 2
0 3 X
1 0
1 1
1 2
1 3 X
2 0
2 1
2 2
2 3 X
3 X
0 1 2
0 . . .
1 . . .
2 . . .
for(int i=0; i<3; i++)
{
for(int j=0; j<3; j++)
{
cout<<i<<j<<endl;
}
}

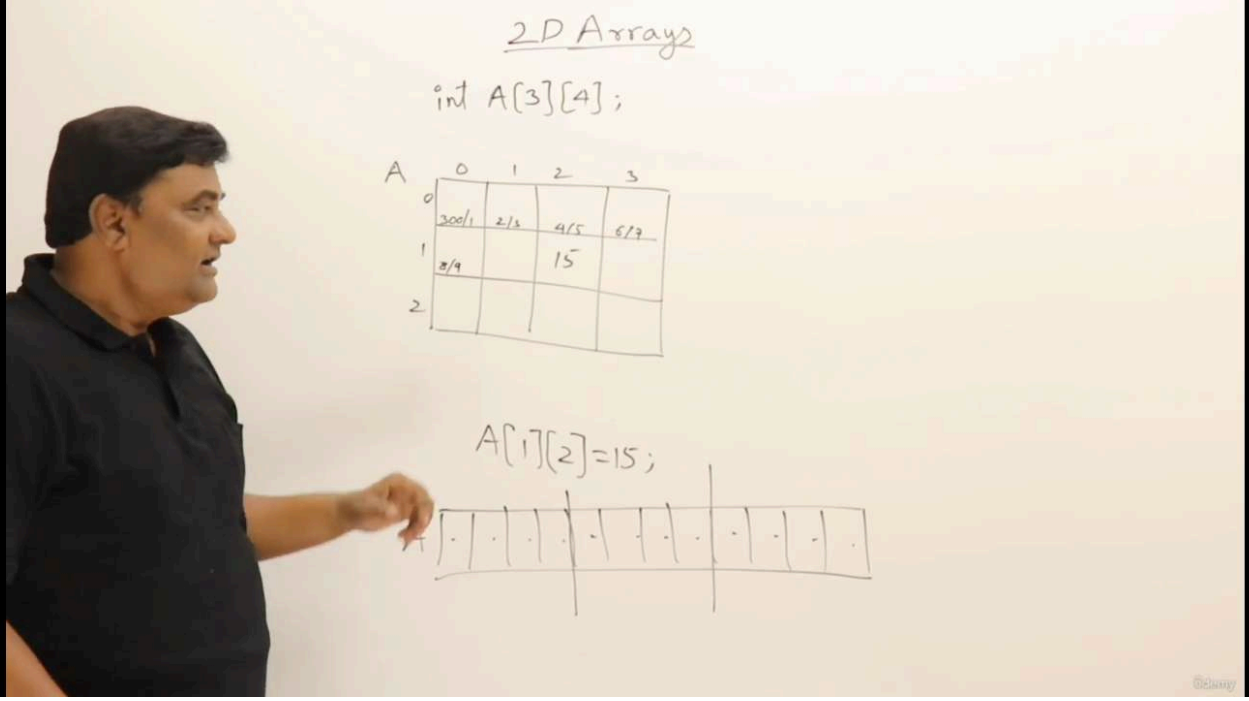

Patterns
count=1;
for(i=0;i<4;i++)
{
for(j=0;j<4;j++)
{
cout<<count<<" ";
count++;
}
cout<<endl;
}
j
i
0 1 2 3
0 1 2 3 4
1 5 6 7 8
include this line
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Patterns



```
for(i=0;i<4;i++)
{
    for(j=0;j<4;j++)
    {
        if(i>=j)
            cout<<"*";
    }
    cout<<endl;
}
```


Patterns
for(i=0;i<4;i++)
{
for(j=0;j<4;j++)
{
if(i>j)
cout<<" ";
else
cout<<"*";
out<<endl;
j
i
0 1 2 3
0
1
2
3

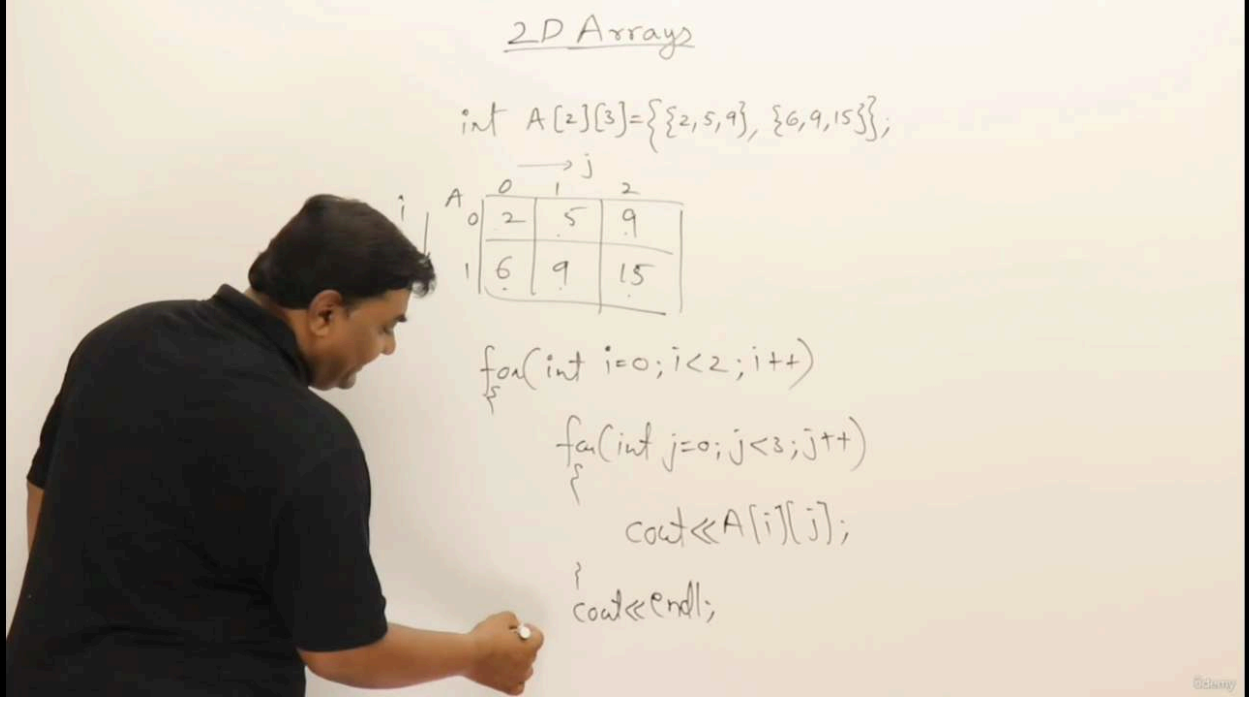

2D Arrays
int A[3][4];
A
0 1 2 3
0
300/1 2/1 4/5 6/7
1
8/9 15
2
A[1][2]=15;
A

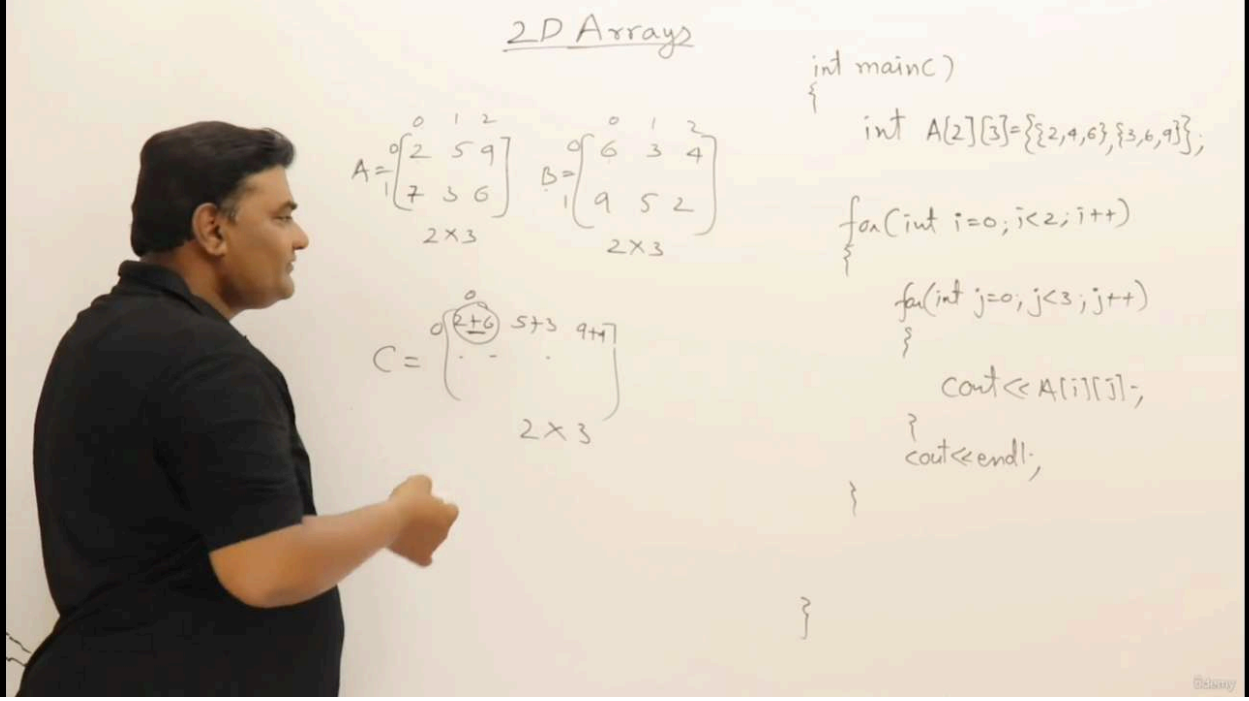
2D Arrays

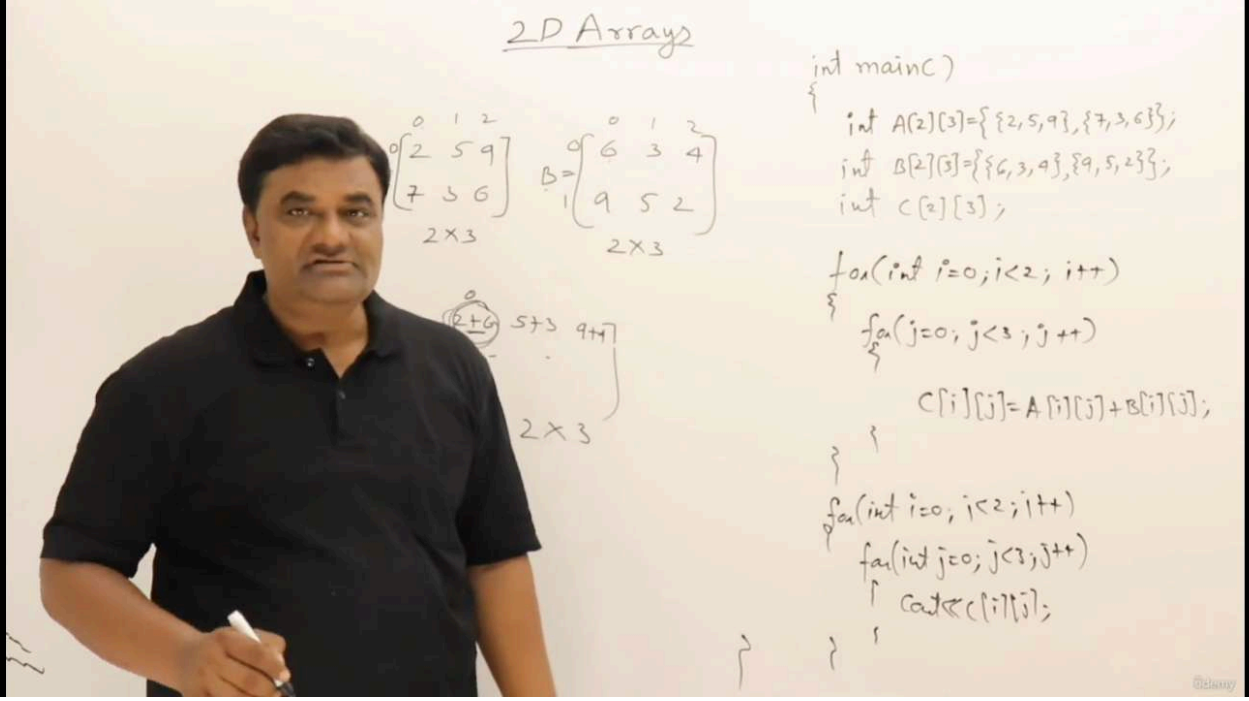
```
int A[2][3]={{2,5,9}, {6,9,15}};
```

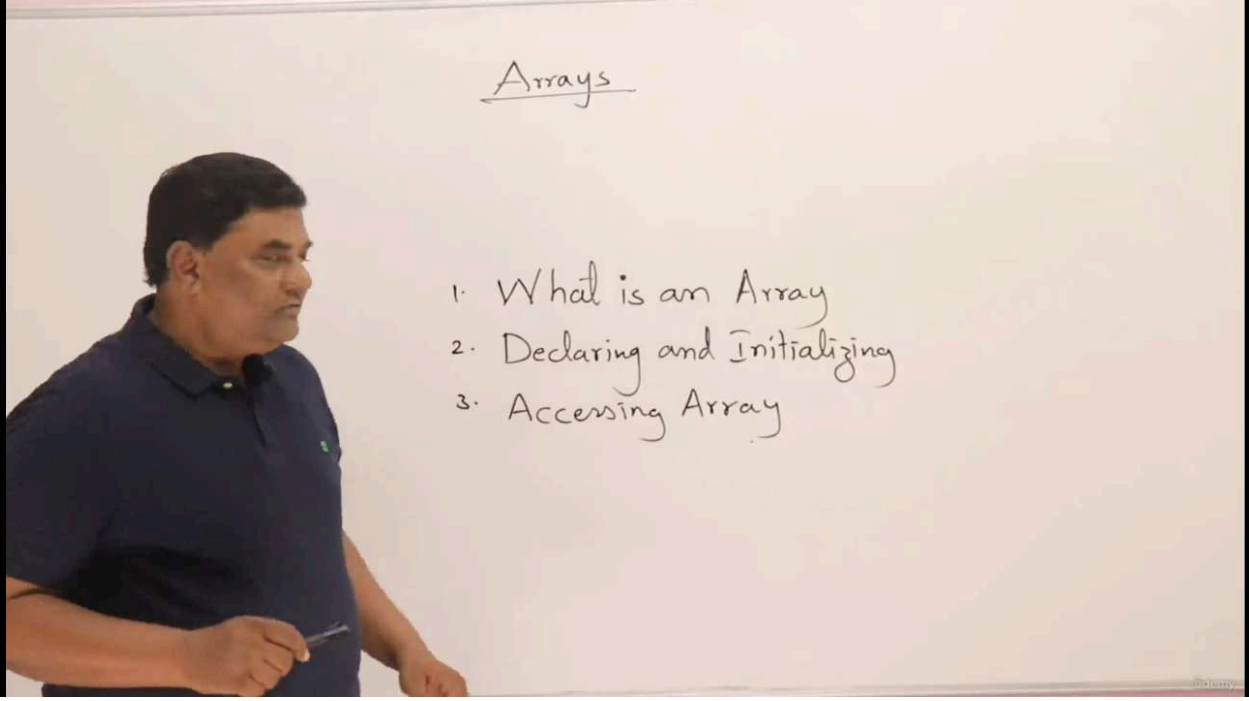
A	0	1	2
0	2	5	9
1	6	9	15

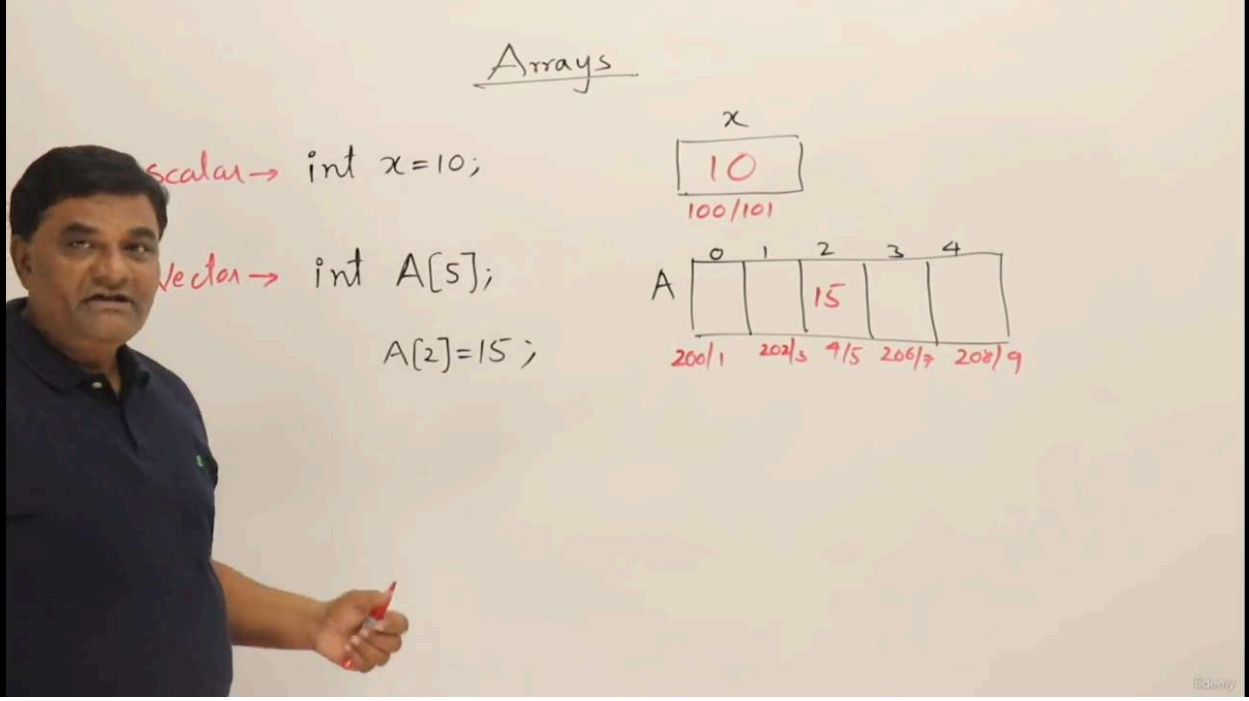
```
int A[2][3]={2,5,9,6,9,15};
int A[2][3]={2,5};
```


2D Arrays
int A[2][3]={{2,5,9},{6,9,15}};
j
i
A
0 1 2
0 2 5 9
1 6 9 15
for(int i=0;i<2;i++)
{
for(int j=0;j<3;j++)
{
cout<<A[i][j];
}
cout<<endl;

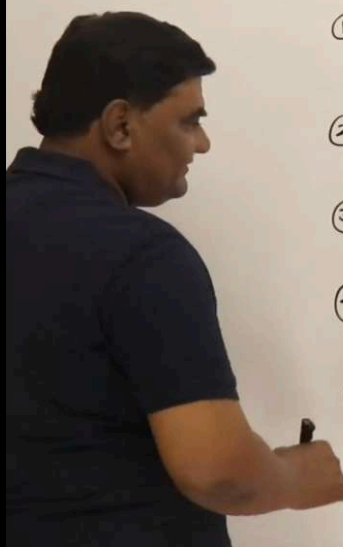

2D Arrays
A = 2 5 9 / 7 3 6 (2×3)
B = 6 3 4 / 9 5 2 (2×3)
C = 2+6 5+3 9+4 / . . . (2×3)
int main()
{
int A[2][3]={{2,4,6},{3,6,9}};
for(int i=0;i<2;i++)
{
for(int j=0;j<3;j++)
{
cout<<A[i][j];
}
cout<<endl;
}
}


2D Arrays
int main()
{
int A[2][3]={{2,5,9},{7,3,6}};
int B[2][3]={{6,3,4},{9,5,2}};
int C[2][3];
for(int i=0;i<2;i++)
{
for(j=0;j<3;j++)
C[i][j]=A[i][j]+B[i][j];
}
for(int i=0;i<2;i++)
for(int j=0;j<3;j++)
cout<<C[i][j];
}
2×3
2×3
2×3

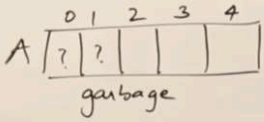

Arrays
1. What is an Array
2. Declaring and Initializing
3. Accessing Array


Arrays
scalar → int x=10;
vector → int A[5];
A[2]=15;
x
10
100/101
A
0 1 2 3 4
15
200/1 202/3 4/5 206/7 208/9

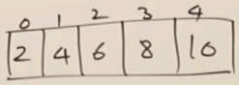
Arrays



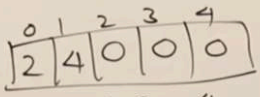
① `int A[5];`



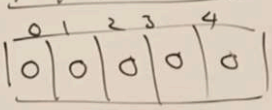
② `int A[5]={2,4,6,8,10};`



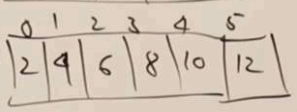
③ `int A[5]={2,4};`



④ `int A[5]={0};`



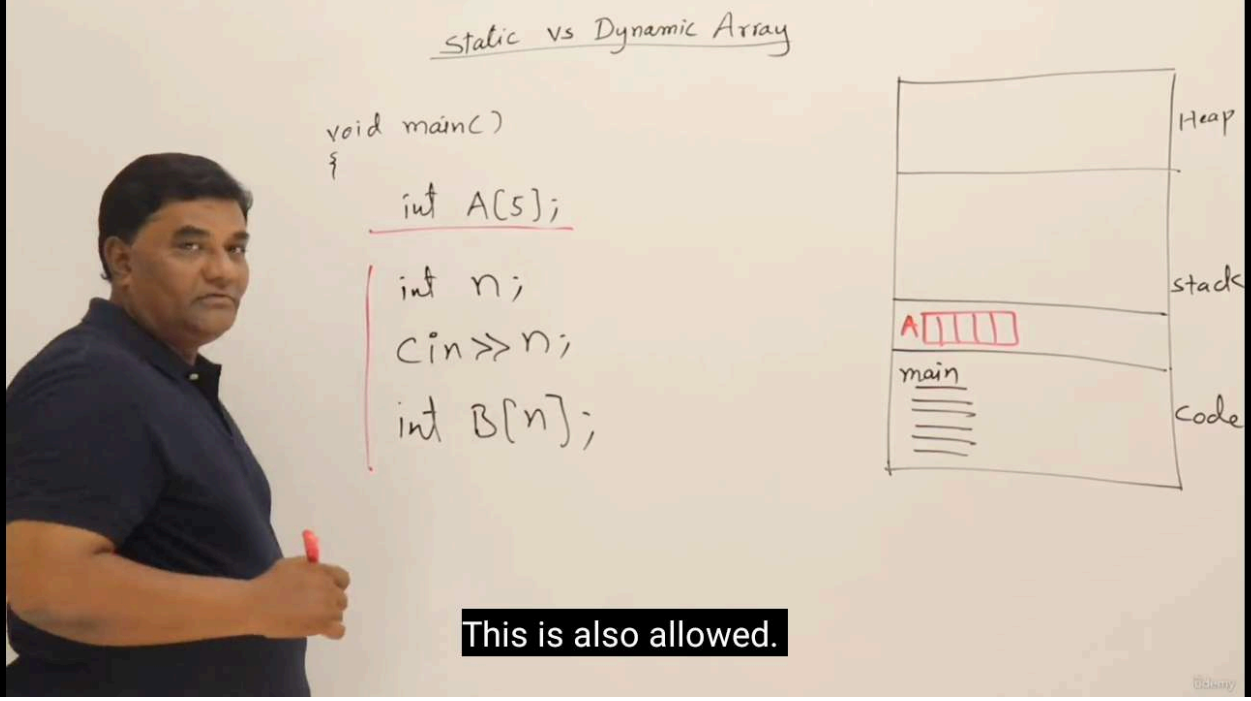
⑤ `int A[]={2,4,6,8,10,12};`



Arrays

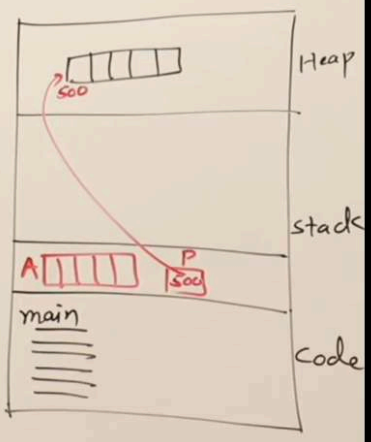
```
int A[5]={2,5,4,9,8};

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

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

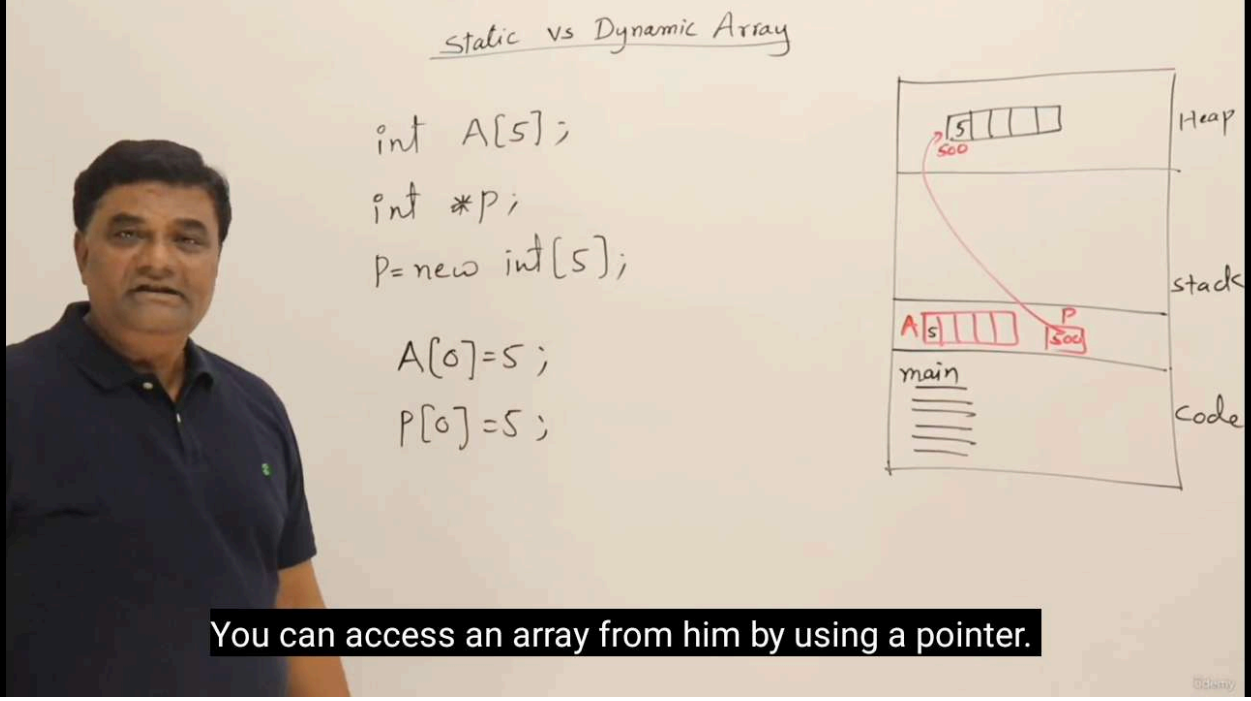
	0	1	2	3	4
A	2	5	4	9	8
	200/1	2/3	4/5	6/7	208/9


Static vs Dynamic Array
void main()
{
int A[5];
int n;
cin >> n;
int B[n];
Heap
stack
A
main
code
This is also allowed.

Static vs Dynamic Array

```
void main()
{
    int A[5];
    int *P;
C++     P=new int[5];
'C' lang P=(int *)malloc(5*sizeof(int));

C++     delete []P;
C-lang  free(p);
```





You can access an array from him by using a pointer.

Static vs Dynamic 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;
```