Pointers:

int main() {

int a=2;		//variable a with value 2
int *p;		//creating a variable which is pointer * is must to use it as pointer
p = &a;		// &a gives the address of a to p
cout << p << endl;		//will print address of a i.e. value stored in p (&a)
cout << *p << endl;		//will print value in the address stored in p *(&a)
cout << &p << endl;		//will print address of p
cout << *(&p) << endl;		//will print value stored in the address of p i.e. address of a
*p=8;		//updating the value in the address stored in p i.e. A=8

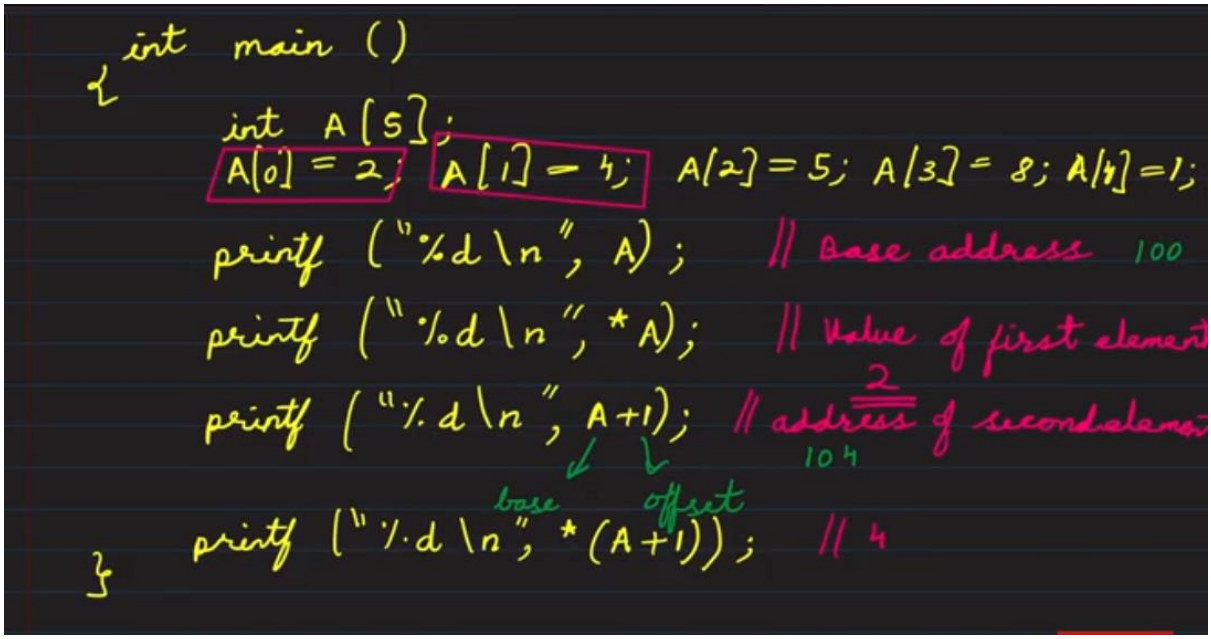
return 0;
}

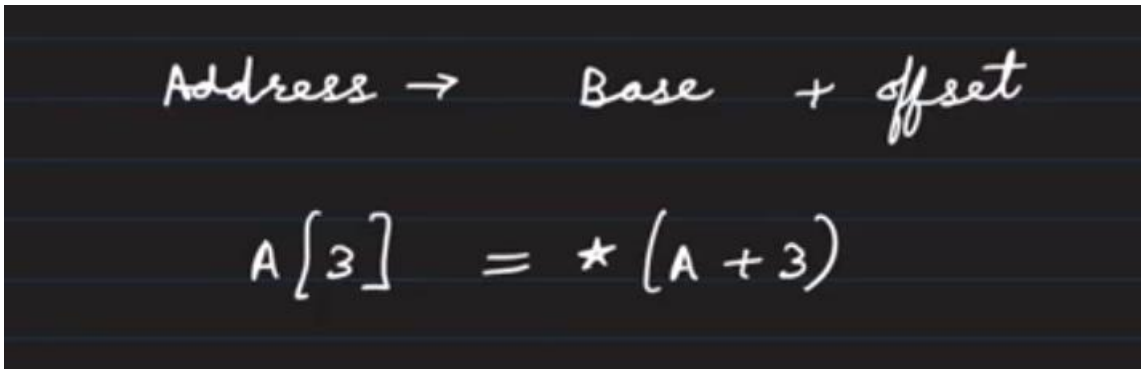
Array:

- Array elements are stored consecutively/ Continuously. Like memory storage is together.
- Declaring an array A of size 5, print A will give base address(address of 1st element)
- A+1 returns address of next element, A is base address and "+1" means offset i.e. 4 byte in case of integers.

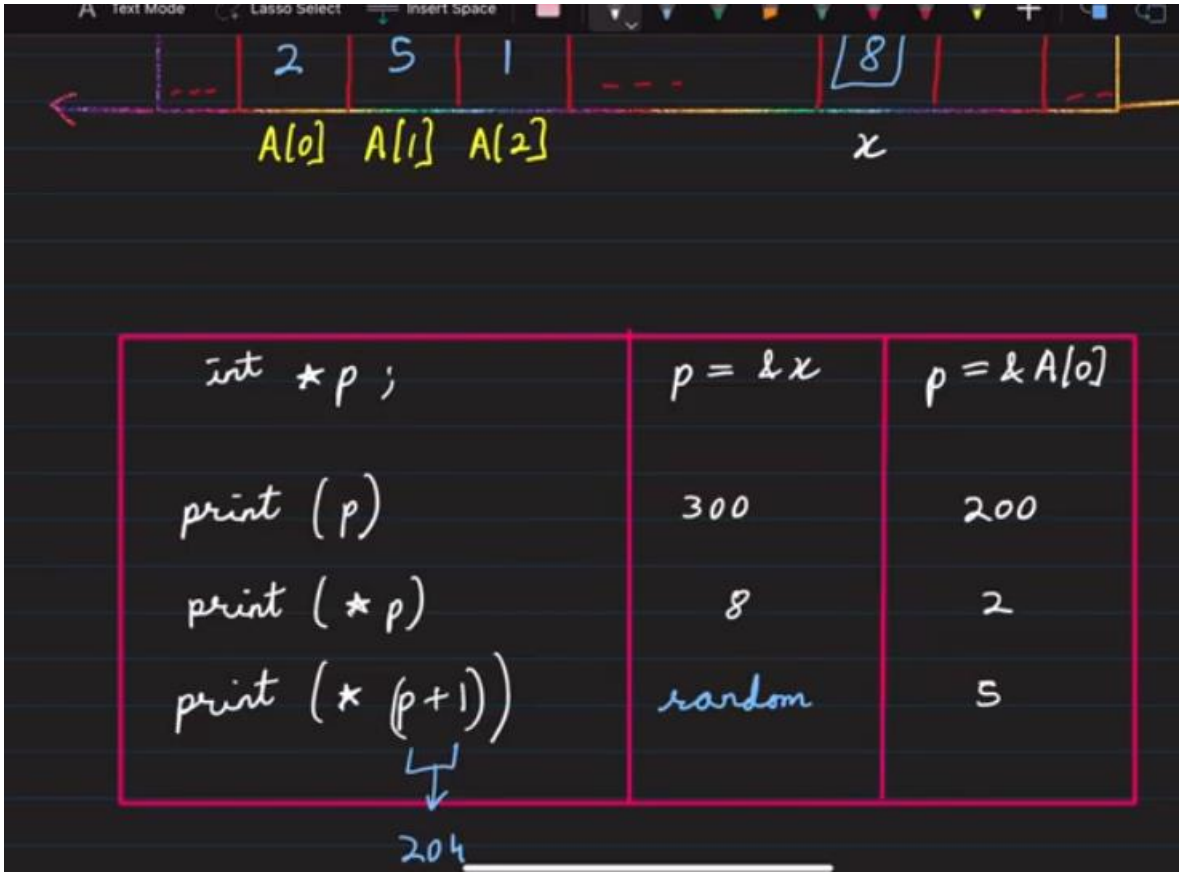
1-D array:

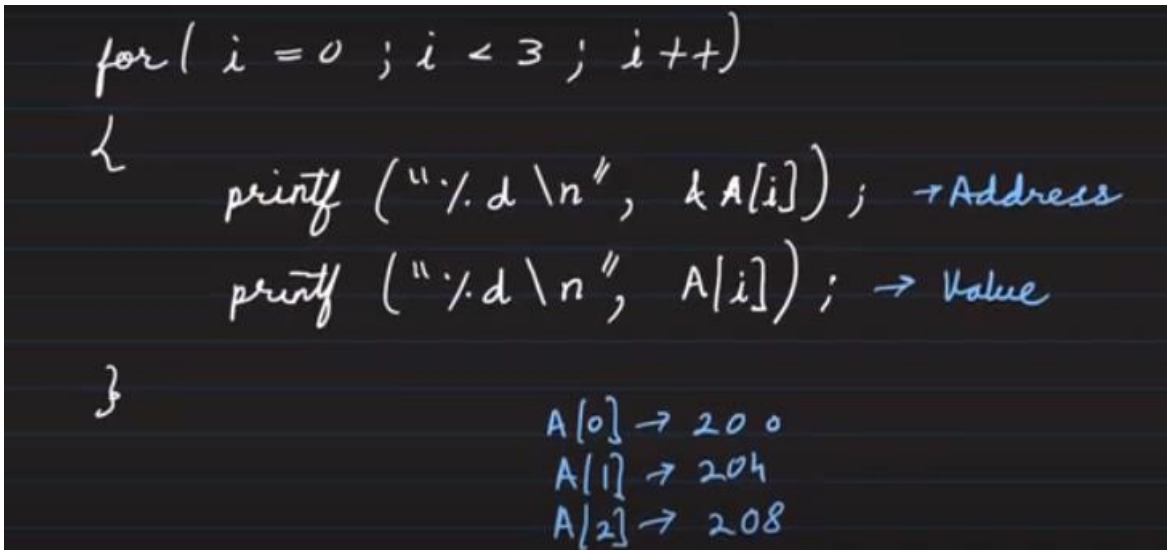
```
// Array declaration by specifying size
int arr1[10];

// declare an array of user specified size
int n = 10;
int arr2[n];

// Array declaration by initializing elements
// Compiler creates an array of size 4.
int arr[] = { 10, 20, 30, 40 }


// Array declaration by specifying size and initializing
// elements
int arr[6] = { 10, 20, 30, 40 }
```



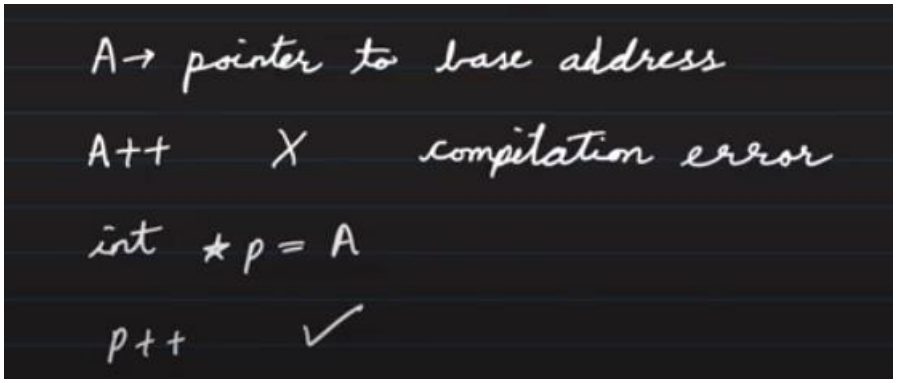


- Pointer arithmetic does not make much sense when we have a single variable. So it is helpful in arrays.

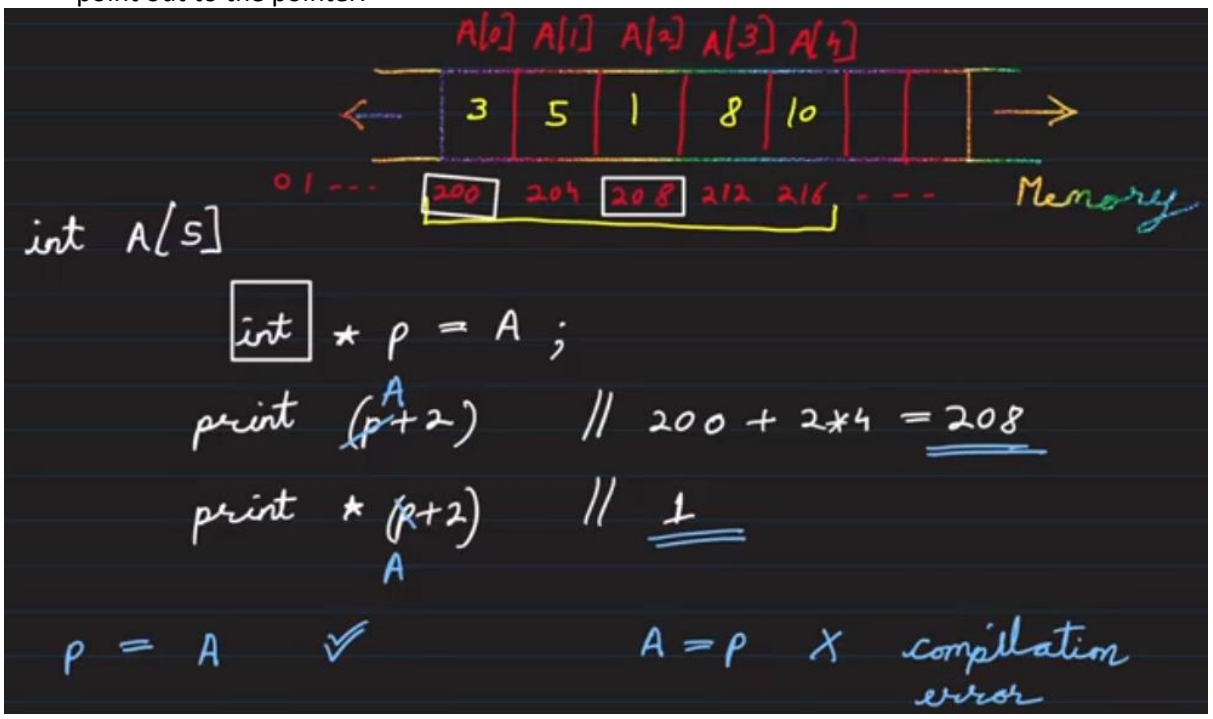




- In above loop if we write A++ instead of A[i] it will give error, but if we declare a pointer and then do p++ it will work,



- Any integer pointer can point to the base address of an array but the base address can not be point out to the pointer.



2-D array:

Creating 2-D array:

```
vector<vector<int> > B;
B.resize(A.size());
for (int i = 0; i < A.size(); i++) {
   B[i].resize(A[i].size());
```

Now we have defined size of this array, we can add element to it like
A[i][i]=n;

Short way: (vector of size n*n-2)

```
vector<vector<int>> ans(n, vector<int>(n-2));
```

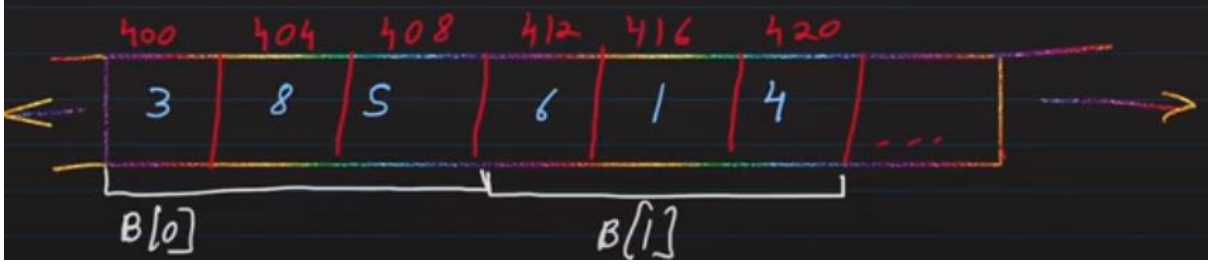
OR:

```
Int arr[n][n];
```

int B[2][3]

B[0]
B[1]
} 1D Arrays of 3 integers

B[0] = B[1] = 3 * 4 = 12 Bytes



int * p = B ✗ compilation error

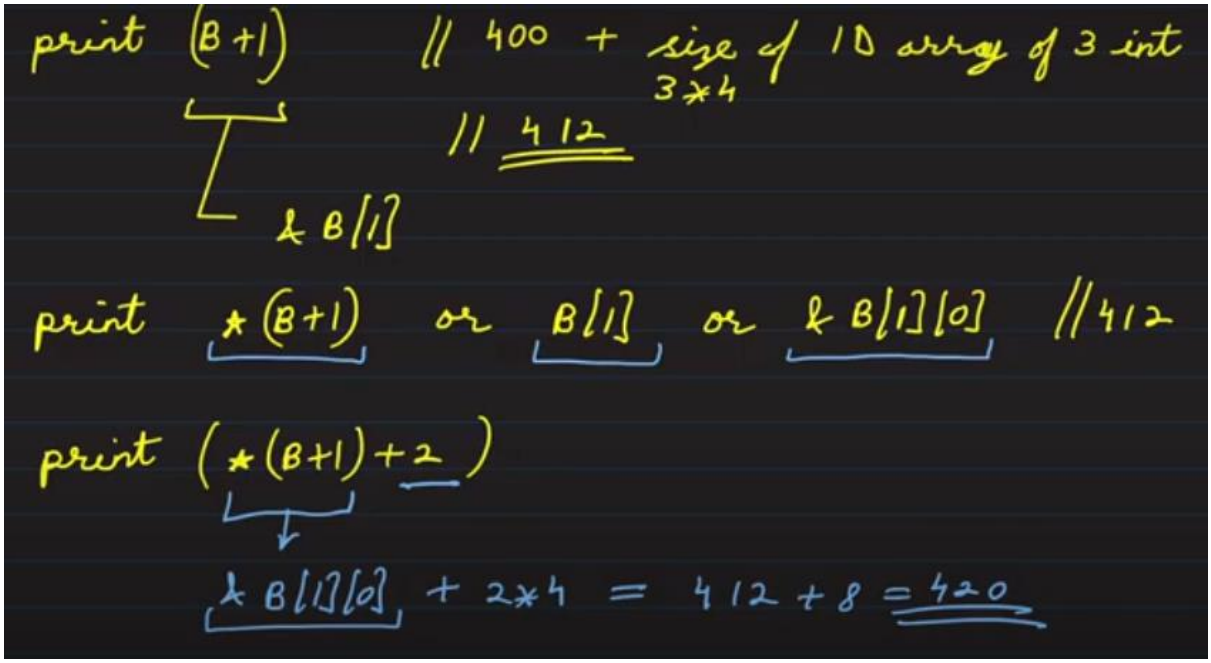
↳ pointer to 1D array

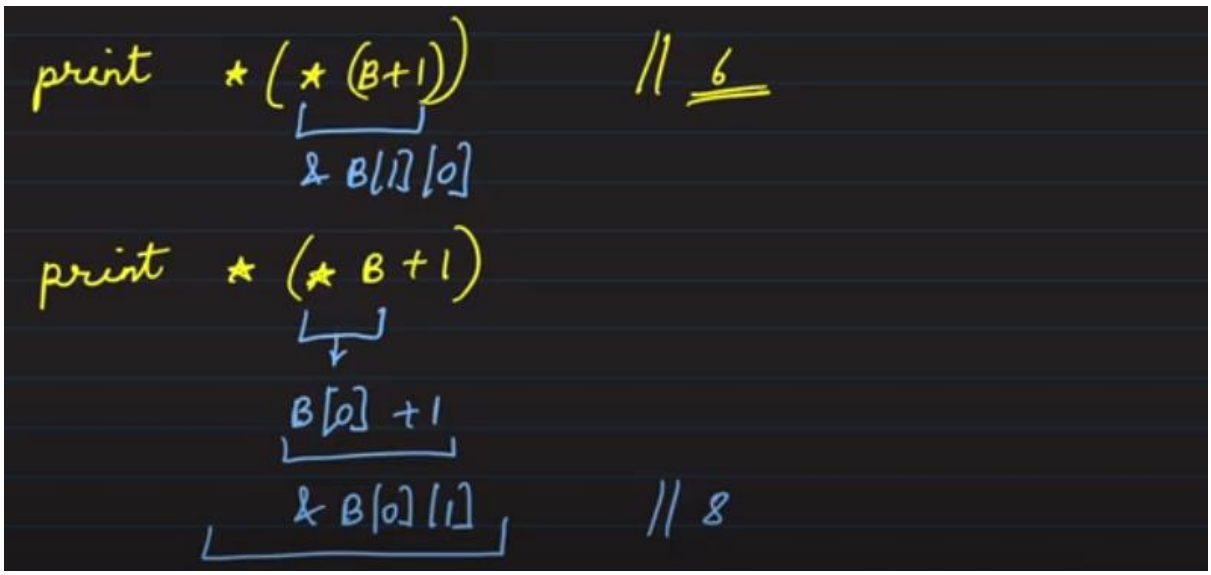
int (*p)[3] = B ; ✓

print B or &B[0] // 400

print *B or B[0] or &B[0][0] // 400

print (B+1) // 400 + size of 1D array of 3 int
3*4
// 412





To sort the array A in ascending order use:
sort(A.begin(),A.end()); -->(starting address, ending address)

For descending order:
sort(A.rbegin(), A.rend());
or
```
sort(arr, arr + n, greater<int>())
```

Fill elements in the array:
fill((v.begin(),v.end());

To swap the elements:

```
swap(A[i],A[i-1]);
```

Transpose of a matrix:

```
for(int i=0;i<n;i++){
    for(int j=i;j<n;j++){
        swap(A[i][j], A[j][i]);
    }
}
```

Reverse of a vector:

```
reverse(v.begin(),v.end());
```

IMPORTANT: (take care of size, show error after size of string > long)

stoi -> string to integer

Stof -> string to float

Stod -> string to double

//IMPORTANT(creating pair)

```
pair<int, char> name1;
pair<int, char> name2;
 if(name1.first==name2.first) return....;
else return....;
```

We can make vector of it:

```
//declaring vector of pairs
vector< pair <int,int> > vect;
```

From <https://www.geeksforgeeks.org/sorting-vector-of-pairs-in-c-set-1-sort-by-first-and-second/>

See the sorting of vector here ^

Removing last character: arr.pop_back()

Removing first character: arr.pop_front()

TO remove element from particular position:

```
it = myvector.begin();
myvector.erase(it);

b.erase(0,i);  //remove zero from index I

 vectorname.erase(position)
vectorname.erase(startingposition, endingposition)
```

vec.erase(std::remove(vec.begin(), vec.end(), 8), vec.end());

From <https://stackoverflow.com/questions/3385229/c-erase-vector-element-by-value-rather-than-by-position>

Inserting element to vector at position:

```
v.insert(it, 1);
```

From <https://www.tutorialspoint.com/cpp_standard_library/cpp_vector_insert_single_element.htm>

For binary search in use:

binary_search(starting address, ending address, element to search)

Max element in array;(similarly min)

*max_element(a.begin(), a.end());

From <https://www.geeksforgeeks.org/how-to-find-the-maximum-element-of-a-vector-using-stl-in-c/>

Index of Max element:

max_element(A.begin()+i,A.begin()+j+1)-A.begin();

Upper_bound: https://www.geeksforgeeks.org/upper_bound-in-cpp/

Making Tupple vector<pair<int, int>>

Way of printing array:

```
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
```

Subarrays of an given array:

```
#include <stdio.h>
int main(){
    int A[] = {1,2,3,4,5};
    int len=sizeof(A)/sizeof(int);
    for( int i=0; i<len; i++ ){
        for( int j=i; j<len; j++ ){   // Now A[i..j] is the subarray
            for( int k=i; k<=j; k++ )
                printf("%d ", A[k]);
            printf("\n");
        }
    }
    return 0;
}
```