ARRAYS [Level-1]

Page No.
Date

* It is a kind of container / data structure which can store similar types of elements.

Why we need Arrays?

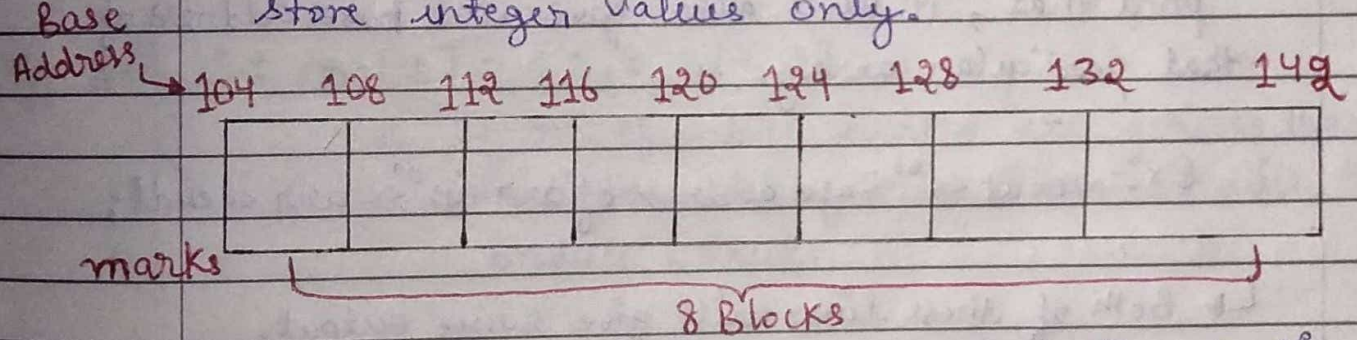
→ If we want sum of 10000 numbers and we have to store them in memory without writing 10000 lines of code to make variables and store them. Then array came into picture which can able to create continuous memory allocations with one single line of code.

Syntax:-

datatype array_name [size];

Ex:- int marks [8];

* This line of code will allocate 8 continuous memory block for the array named 'marks' and it can store integer values only.



Base Address:- It is also called as the starting address of the array

→ We can also refer the first block of array with its name so, we can also name the block with address 104 as 'marks'.

→ Int take 4 bytes to store integers in memory, so this array will take 8 × 4 = 32 bytes space in memory.

→ int arr[20];

↳ This type of array creation is called as static array. which takes fixed number of blocks of memory.

↳ Here, we fixed 20 blocks for array named 'arr'. It will take 20×4 = 80 byte storage space in 64 bit OS.

→ Address of operator :- '&'

* To get the base address of any array, we can use this operator with array name and it will return the hexa-decimal value of that array.

Ex:- int arr[10];
cout << "Base address of arr is:" << &arr << endl;

* Keep in mind - If we only write array name and print that, it will also give the base address of that particular array.

Ex:- cout << "Base address of arr is:" << arr << endl;

↳ Both of these lines will give same output.

→ Size of function :- 'sizeof(array_name)'

↳ This is an inbuilt function of C++ which gives the size of any variable or array passed to it.

Ex:- int a = 5;
int arr[5];
cout << "Size of a:" << sizeof(a) << endl;
cout << "Size of arr:" << sizeof(arr) << endl;

Output:-
Size of a: 4
Size of arr: 20

} int take 4 byte
So, 5×4 = 20 (size of the array)

Page No.
Date

Array Initialisation :-

Type I :- `int arr[] = { 1, 3, 2, 6, 8 }`

∴ Size is not mentioned here, so it will automatically detect the size of array from no. of values.

Type II :- `int brr[5] = { 1, 2, 4, 3, 7 }`

∴ Size is mentioned, so we just initialise 5 values in it.

Type III :- `int crr[5] = { 2, 5 }`

∴ If no. of values inserted are less than the mentioned size, then it will insert '0' zero in empty blocks.

2	5	0	0	0

Type IV :- `int drr[2] = {1, 3, 5, 4, 8}`

∴ If no. of values inserted are greater than the mentioned size of array, then it will produce ERROR.

* If we want to create an array of user specified size, like :-

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

∴ This is compiler dependent, it might execute seamlessly or throw error.

↳ But writing this kind of code is considered as BAD PRACTICE, if you have a choice then must specify the size of array.

↳ Because OS will allocate some memory to each program if that integer input exceeds that memory limit then program show WRONG BEHAVIOUR and array can't be created !!

It might happen that our program don't have that much continuous memory which user demanded.
To prevent this case, we use DYNAMIC ARRAYS.

Indexing in Array :-

→ Array indexing is the same as accessing an array element.

→ You can access an array element by referring to its index number.

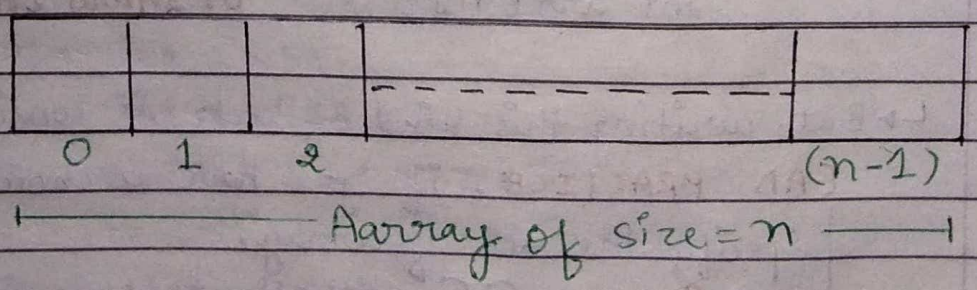
→ Indexing is same as the address of our home in continuous street where we live.

→ To reach at any place, we have to know its address. similarly in arrays, to access any array element, we can use index to get its value.

→ Every block of array have its hex address (hexa-decimal address) which is difficult to learn so, that's why we can use indexing in array.

→ Remember that, indexing starts with '0' in arrays.

So,
if we have an array of 'n' size, then first memory block have 0 index and the last block will have (n-1) index always.



Page No.
Date

Start n=5 End

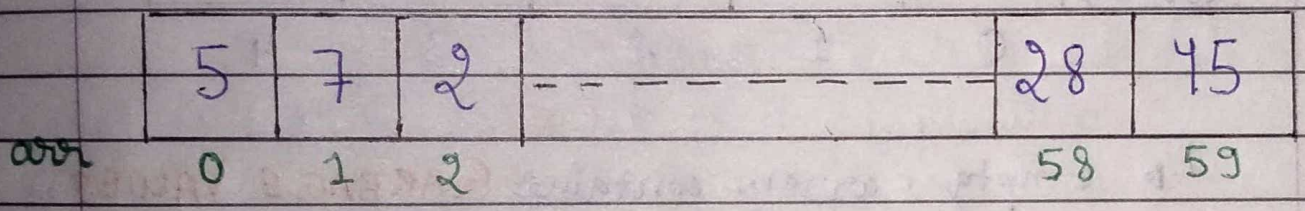
1-Based Indexing: 1 2 3 4 5 → n

0-Based Indexing: 0 1 2 3 4 → (n-1)

∴ To access any memory block of array using indexing, we can just follow this syntax and get the value inside particular block.

array_name [index_number]

Ex:- int arr[60];



To access the values of particular index we can write:-

arr[0] = 5

arr[1] = 7

arr[2] = 2

arr[58] = 28

arr[59] = 45

Page No.
Date

Ex:- `int arr[5] = {5, 8, 9, 12, 13}`
`int n = 5;`

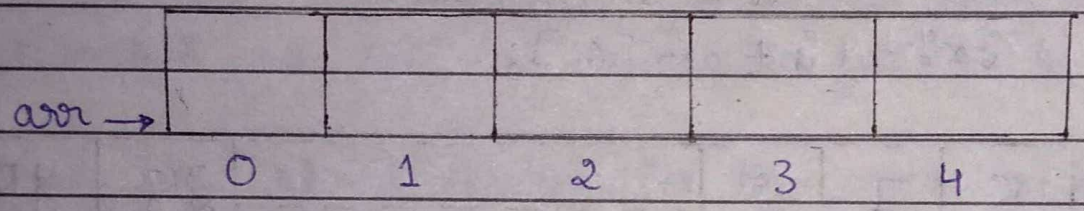
```
for (int i=0; i<n; i++) {
    cout << arr[i] << " ";
}
```

Output:-
5 8 9 12 13

→ Taking input in an Array :-

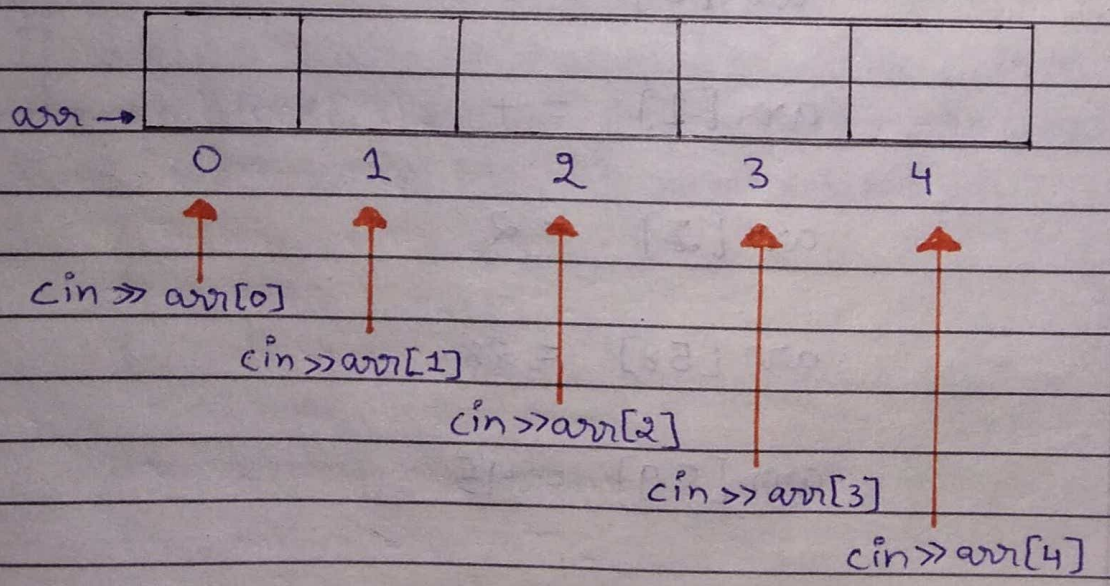
`int arr[5];`

∴ This line will create the continuous memory blocks like :-



↳ Empty array contains GARBAGE VALUES.

↳ If some blocks have values in it, then other empty blocks will initialised with '0'.



→ But, if we want to store 1000 values in an array then we can use loops to store the values without writing 1000 lines of code.

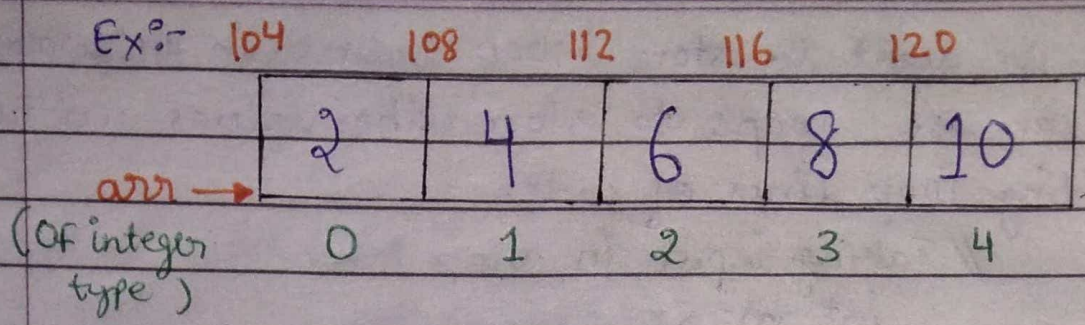
```
// Taking input in arr
    int m=5;
for (int i=0; i<m; i++){
    cin >> arr[i];
    cout << endl;
}
// Printing the arr array
cout<<"Printing the array"<<endl;

for (int i=0; i<m; i++){
    cout << arr[i] << " ";
}
```

Output

```
10
20
30
40
50
Printing the array
10 20 30 40 50
```

Formula to calculate value of arr[i] -

$$arr[i] = \text{Value at}\left(\underbrace{\text{Base Address} + (\text{Datatype size} * \text{index})}_{\text{Address Calculation corresponding to arr[i].}}\right)$$



If we want to access value of arr[2] then,

arr[2] → Value at (Base Address 104 + (Data Type 4 × Index 2))

→ Value at (104 + 8)

→ Value at (112)

↳ Address of index 2

∴ So, behind the scene memory block is accessed using this formula and then returned the value corresponding to index numer.

Q 1. Make an array of size = 10 then take input values from user and double-up each value of that array.

```
int arr[10];        // Array of size = 10
int n = 10;
// Taking values from user
For (int i=0; i<n; i++) {
    cin >> arr[i];
}
// Double-Up the array
For (int i=0; i<n; i++){
    arr[i] = arr[i] * 2;
}
```

```
// Printing the doubled-Up array
for( int i=0; i<n; i++){
    cout << arr[i] << " ";
}
```

Output :-

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

Q2. Create an array of size = 5 the take values in input and print the total sum.

```
// Array of size = 5
int arr[5];
// Take input
int n=5;
cout<<" Enter the input :" <<endl;
for (int i=0; i<n; i++) {
    cin >> arr[i];
}
// Calculate sum
int sum =0;
for( int i=0; i<n; i++){
    sum = sum + arr[i];
}
// Printing total sum
cout<<" Total sum : "<<sum;
```

Output :-

Enter the input :

2 4 6 8 10

Total sum : 30

Linear Search in Array :-

* In linear search, we traverse the array from starting to end or vice-versa to perform specific task.

Q1. Find the value of target in an array.

```
int arr[5] = { 5, 6, 7, 8, 9};
int n=5;
int target = 8;
bool flag = 0;
// 0 ⟶ Not found
// 1 ⟶ Found

for(int i=0; i<n; i++) {
  if (target == arr[i]) {
      flag = 1;
      break;
  }
}
// Checking the value of flag
if (flag == 1) {
  cout<< " Target found";
}
else {
  cout << " Target Not found";
}
```

Output:-

Target Found

Explanation:-

- We got array and target in input, to find the target in array we used a boolean variable named 'flag'.
- Flag = 0 means target not found and flag = 1 means target found.
- for loop is checking every value of array with the target value, if they got matched then flag becomes 1 and loop will breaks.
- finally we are checking the value of flag.

Arrays & functions :-

↳ While passing arrays to a function, keep in mind to pass the size of array.

↳ Size here means the total no. of values present in array.

```
int main()
{
    int arr[5];
    int size = 5;
    solve(arr, size);
}
```

function call

```
solve(int arr[], int size)
{
    //logic
}
```

∴ Here, we passed a 1-dimensional array and integer size as arguments in function named 'solve()'.

Ex:-

function which print values of array :-

```
void printArray(int arr[], int size) {
    for(int i=0; i<size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[5] = {2, 4, 6, 8, 10};
    int size = 5;
    // Function call
    printArray(arr, size);
}
```

Output :-

2 4 6 8 10

Page No.
Date

Q1. Finding the value of target present in an array using function which takes array in argument.

```
bool linearSearch (int arr[], int size, int target){

    for (int i=0; i<size; i++) {
        if (arr[i] == target) {
            // Target found
            return true;
        }
    }
    // Not found
    return false;
}

int main () {
 int arr[5] = { 5, 4, 2, 6, 1 };
 int size = 5;
 int target = 2;
 // function call
 bool ans = linearSearch (arr, size, target );

 if(ans == 1){
    cout << "Target found";
 }
 else {
    cout<<"Target Not found";
 }
}
```

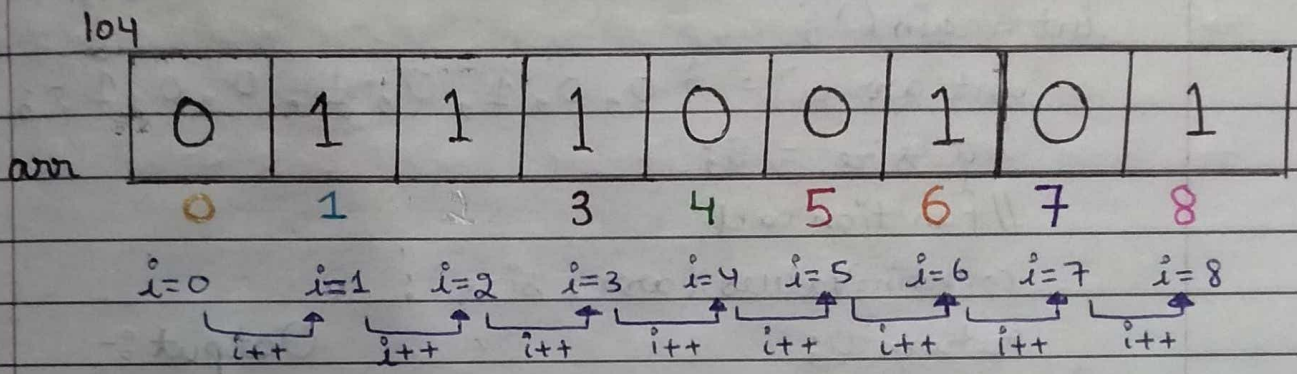
Output:-
Target Found

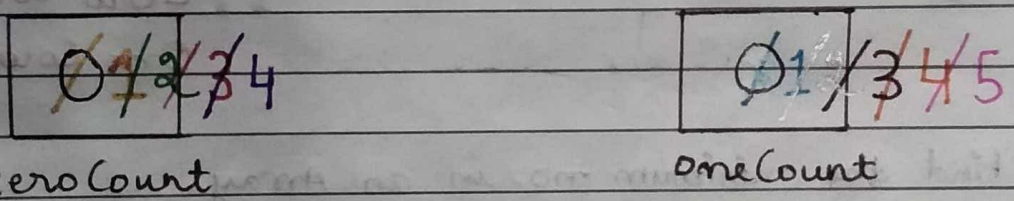
↳ Finding the target value without using flag.

Page No.
Date

Q2. Count 0's and 1's in an Array.

To solve this, we create two variables which count no. of 0's and 1's seperately. for complete array traversal we use loop which check array values and compare them with '0' and '1'. when a value matched with them, it will increment its corresponding count by 1. Both variables are initialised with 0.





∴ finally zeroCount = 4 (which means 4 zeros in array i.e True)

And oneCount = 5 (which means 5 ones in array i.e True)

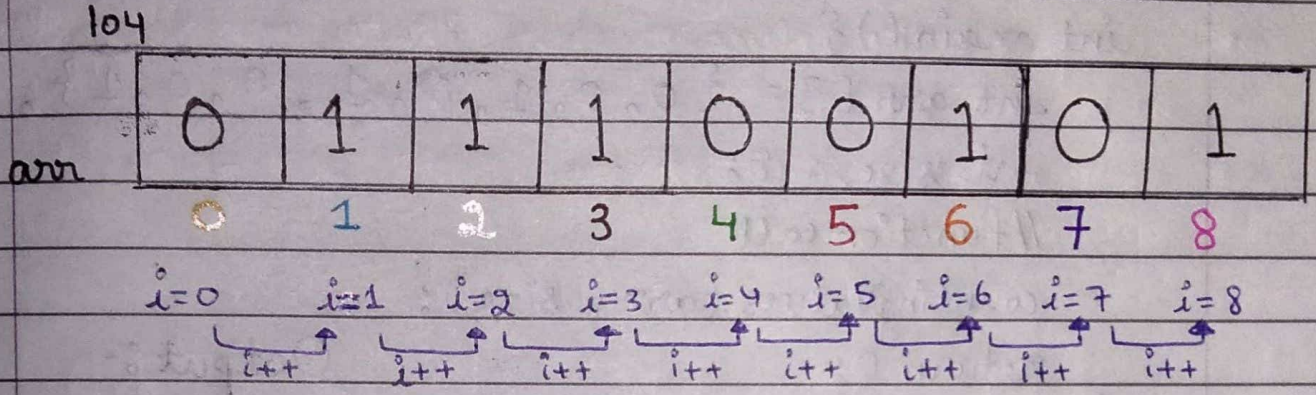
Code :-

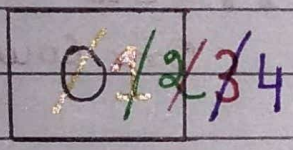
```
void countingValues(int arr[], int n){
    int zeroCount = 0;
    int oneCount = 0;
    // loop to traverse array
    for(int i=0; i<n; i++){
        if(arr[i] == 0){
            zeroCount++;
```

Page No.
Date

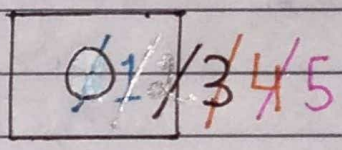
Q2. Count 0's and 1's in an Array.

To solve this, we create two variables which count no. of 0's and 1's seperately. for complete array traversal we use loop which check array values and compare them with '0' and '1'. when a value matched with them, it will increment its corresponding count by 1. Both variables are initialised with 0.





zeroCount



oneCount

∴ finally zeroCount = 4 (which means 4 zeros in array i.e True)

And oneCount = 5 (which means 5 ones in array i.e True)

Code :-

```
void countingValues(int arr[], int n){
    int zeroCount = 0;
    int oneCount = 0;
    // loop to traverse array
    for(int i=0; i<n; i++){
        if(arr[i] == 0){
            zeroCount++;
```

```
        }
        if (arr[i] == 1) {
            oneCount++;
        }
    }
    cout << "Zero Count = " << zeroCount << endl;
    cout << "One Count = " << oneCount << endl;
}

int main(){
    int arr[] = {0,0,1,0,1,0,0,1};
    int size = 8;
    //function call
    countingValues(arr, size);
    return 0;
}
```

Output :-
Zero Count = 5
One Count = 3

Q3. Find the minimum no. in an Array.

→ Range of signed integer is from -2^{31} to $2^{31}-1$

→ We have 2 utility values in "limits.h" header file named INT_MIN and INT_MAX.

→ So, INT_MIN represents :- $-2^{31} = -2147483648$
and INT_MAX represents :- $2^{31}-1 = 2147483647$

→ Best Practice :-
  └→ To find min no., make a variable to store the minimum value & initialise it with → INT_MAX.
∴ When we compare any value with this, always that value is smaller and keep comparing it till the end of array.

↳ Similarly, always initialise the variable which is going to store the maximum answer with → INT_MIN.
∴ while comparing array values with this 'int min', it will always gives the greater value and thats how we can retrieve the maximum higher value in an array.

Examples:-

```
int findMinimumInArray (int arr[], int size) {
    // Variable to store minimum answer
    int minAns = INT_MAX;

    // Loop, to traverse the array
    for (int i=0; i<size; i++) {
        if (arr[i] < minAns) {
            minAns = arr[i];
        }
    }
    return minAns;
}

int main() {
    int arr[] = {10, 8, 4, 7, 2, -3, 1};
    int size = 7;

    int minimum = findMinimumInArray (arr, size);
    cout << " Minimum number is: " << minimum << endl;
    return 0;
}
```

Output:-

Minimum number is: -3

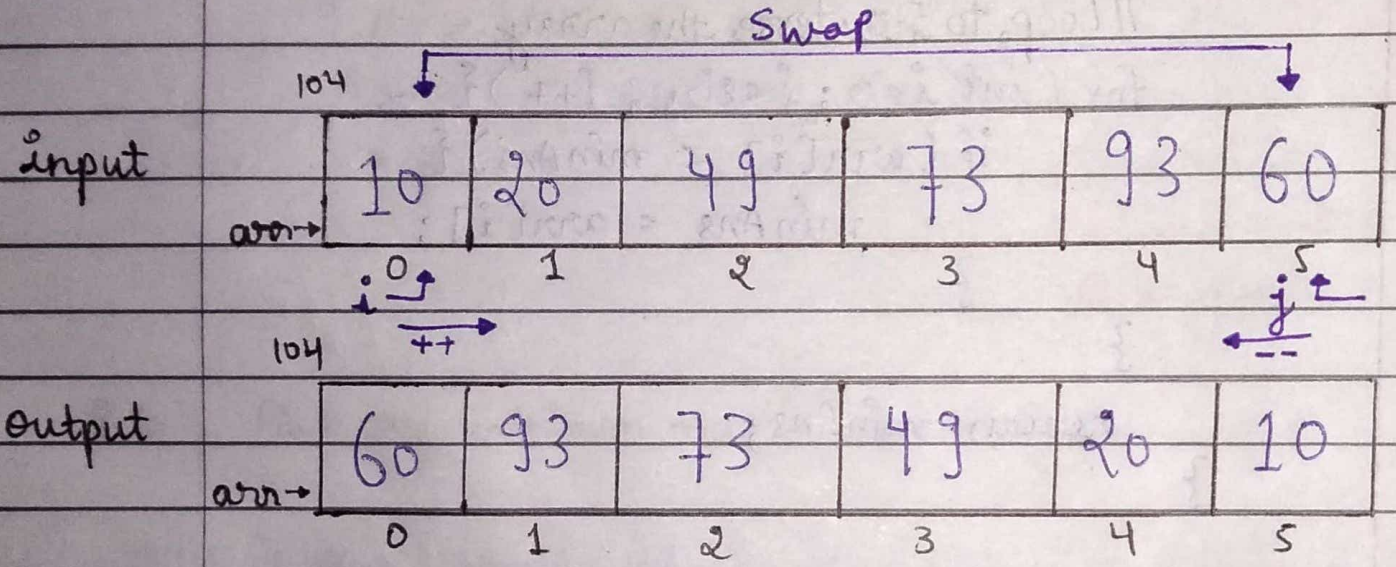
Q4. Reverse the array given in input.

→ swap() is a utility function in C++, which takes two values as input and swap them with eachother.

→ Take two iterators, one is initialised with '0' and traverse array from left to right and initialise other iterator with 'size-1' and traverse array from right to left.

→ Compare both values of each iterator and swap them with each other.

→ Both iterators will iterate until left <= right.



Example:-

```
void reverseArray (int arr[], int size){
    int left = 0;
    int right = size-1;
    while (left <= right){
        swap(arr[left], arr[right]);
        left++;
        right--;
    }
```

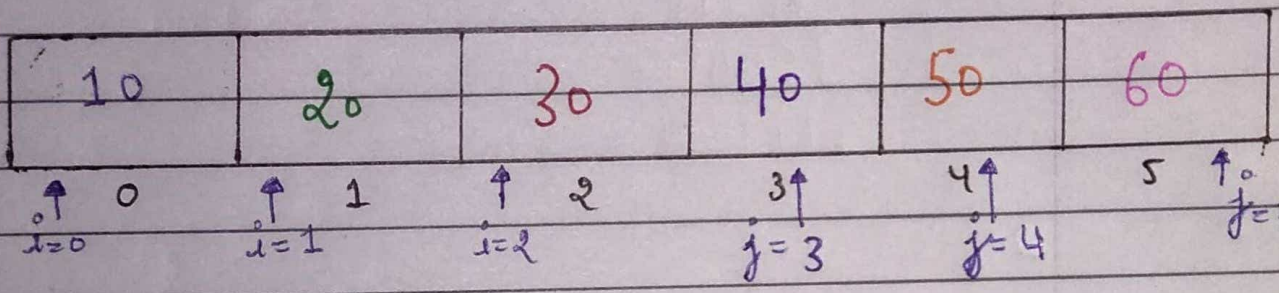
```
// Printing reversed array
for(int i=0; i<size; i++) {
    cout << arr[i] << " ";
}
}
int main(){
    int arr[] = { 10, 20, 49, 73, 93, 60 };
    int size = 6;
    reverseArray(arr, size);
    return 0;
}
```

Output:-

60 93 73 49 20 10

Qs. Extrem print in an Array.

input →



output → 10 60 20 50 30 40

Code:-

```
void extremPrint(int arr[], int size){
    int left = 0;
    int right = size-1;
    while( left <= right ){
        if(arr[left] == arr[right]){
// when both iterators are at same value, in odd no. of
// values we print that central value only at once.
            cout << arr[left];
        }
        else{
        cout << arr[left] << "_" << arr[right] << "_";
        }
```

```
            i++;
            j--;
        }
    }
int main() {
    int arr[] = {10, 20, 49, 73, 93, 60};
    int size = 6;
    // function call
    extremePrint(arr, size);
    return 0;
}
```

Output:-

10 60 20 93 49 73