Array

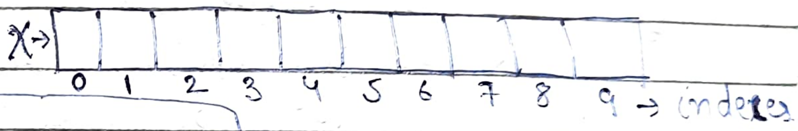
→ What is an Array? → List → Collection of similar data type.

⊛ It can be defined as a data structure that is used to store a group or collection of items or elements sequentially inside a memory

→ **Syntax and declaration**

int x[10];

data-type array-name array-size



→ x[3] = 5; → This will assign 5 value to the 3rd index of array x.

→ x[3] = 8; → This will update the 3rd index of array x to 8.

⇒ Array elements can be accessed by using indices

⊛ eg → arr[4];

→ **Printing all the elements of an array**

```
int arr[7] = {2, 4, 5, 7, 8, 10, 13};
for (int i=0; i<=6; i++){
      Cout << arr[i] << " ";
}
```

O/P =

Output : 2 4 5 7 8 10 13

→ **Taking input (Using for loop)**

```
for (int i=0; i<=6; i++){
    cin >> arr[i];
}
```

Q) Given an array of marks of students, if the marks of any student is less than 35 than print its roll no. [roll no. = index]

```
→ int main(){
      int marks [6];
      // input
      for (int i=0 ; i<=5; i++) {
          cin >> marks[i];
      }
      for (int i=0 ; i<=5; i++){
          if (marks[i] < 35) cout << i << " ";
      }
  }
```

→ In this question we can also take the no. of students manually as n and can run the for loop from i=0 to i<=n-1 to take input and print output.

→ **Types of arrays**

1) 1 dimensional array
2) 2 dimensional array → matrix

→ **Size and ~~size of~~ sizeof operator**

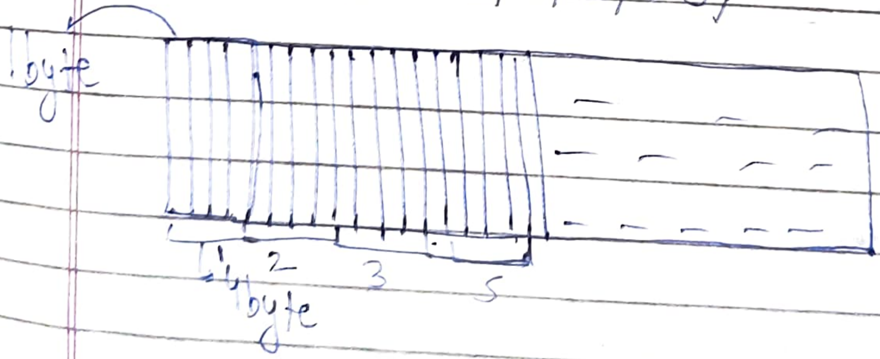
⇒ int arr[] = {2, 3, 4, 1, 2, 9, 10, 3, 14, 15, 20};

int n = size of(arr) / Sizeof(arr[0]);
↳ This will be the length of the array

⇒ **Memory Allocation in Arrays**

→ Continuous memory allocation.

→ int arr [4] = {1, 2, 3, 5};



→ Cout << &arr; Or Cout << arr;
↳ This will give us the address of first byte of first element of the ~~arr~~ array

→ Calculate the sum of all elements in the given array.

→ Approach

```
int sum = 0;

for (int i = 0; i <= Size -1; i++) {
    Sum = Sum + arr[i];
}
Cout << sum ;
```

⟹ Linear Search

→ Find the element x in the array. Take array and x as input

→ Approach

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

// Take inputs of the array elements

int x;
cin >> x;

// Loop through array elements
if (arr [i] == x) Cout << " Element is present';
```

→ Find maximum value out of all the elements in an array

Approach

arr	2	3	5	10	13	15
	0	1	2	3	4	5

```
int max = arr[0];
// loop from i=1 to i<=n-1

    if (arr[i] > max){
            max = arr[i]
    }

→ Cout << max;
```

→ Find second largest element

Approach

① Find max element
② Traverse through the array,

```
        if (Smax < arr[i] && arr[i] != max)
                Smax = arr[i];
```

⇒ Pointers in arrays

```
int arr[] = {4, 2, 3, 4}
int* ptr = arr; → store the address of
                  first element of an array.
int* ptr1 = &arr;
```

=

Date : / /

Page No.

→ Now, we can access the array elements using pointer.

Eg→ ptr[0] ; //It will give us the elemenent at 0th index.

⇒ **Note→** Whenever we pass an array to a function, we pass the address of the first element of the array.

→ **Printing the array using pointer**

```
int arr[]={2,4,6,8}
int* ptr = arr;

for (int i=0; i<=4; i++){
        cout<<*ptr<<" ";
        ptr++;
}
```

⇒ Vectors in C++

——x——x——

↳ Dynamic array

→ **Syntax**

declaration → vector <data_type> name;

→ Basic operations ~~of~~ on vector

→ vector <int> v;

① v.push_back (6); → v | 6 |

v.push_back (1); → v | 6 | 1 | (Size increased by x2)

v.push_back (9); → v | 6 | 1 | 9 | | (")

v.push_back (0); → v | 6 | 1 | 9 | 0 |

→ v.size → It will give the no. of elements in the vector

→ v.capacity - It gives us the total capacity of the vector (It increase by x2)

② v.pop_back(); - Remove the last element of the vector.

→ To print vector elements

```
for (int i=0; i<v.size(); i++){
    cout << v[i] << " ";
}
```

// v.pop_back() → updates the size but not the capacity.

→ We can also initilize vector with a size

(*) vector <int> v(5) ;
↳ Size of vector is 5

(*) vector <int> v(5, 7) ;
↳ all the places in vector will be assigned 7.

→ Taking input in a vector

```
(*) vector <int> v(5);

(*) for (int i=0; i<5; i++){      // Only possible if
        cin >> v[i];              we have declared
    }                             the size of the vector.
```

If we have not declared the size

```
(*) for (int i=0; i<n; i++){
                    ↳ Can be asked to user
        int x;
        cin >> x;
        v.push_back(x);
    }
```

(3) v.at(index) → it gives the element present at a that index.

(4) sort (v.begin() , v.end());

{ This will sort all the elements in the ascending order.

⇒ **Passing vectors to function**

→ ~~We can~~ Normally vectors are passed by ~~[illegible]~~ values in a function unlike array that gets passed by reference.

→ But we can also pass it by reference using (& → ampercant)

```
⇒ void change (Vector<int> a){
        a[0]=100;
  }
  int main () {
        Vector<int> V;
        V.push_back (10);
        change (V);  //     Pass by value
  }
```

```
⇒ Void change (Vector <int> & a) {
                    a[0]=100
  }
  int main () {
        Vector <int> V;
        V.push_back(10);
        Change (V); //Pass by reference
  }
```

→ WAP to reverse the array without using any extra array.

Approach
↳

	0	1	2	3	4	5	6	7
V	6	1	2	4	3	4	3	1
	↑							↑
	i							j

→ swap(v[i], v[j])

→ i++ , j--

	0	1	2	3	4	5	6	7
V	1	1	2	4	3	4	3	6
		↑					↑	
		i					j	

→ Again repeat the process.

⊛ This is known as 2 pointers approach

⇒ Reverse part of array

	0	1	2	3	4	5
arr →	1	6	2	3	7	4

reverse → 4 4 3 2 6 1

reverse (1,4) → 1 7 3 2 6 4

reverse (0,3) → 3 2 6 1 7 4

→ Rotate the given array 'a' by 'k' steps, where k is non-negative.

$$\begin{array}{lcccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \# \Rightarrow \text{arr} & 1 & 6 & 2 & 3 & 7 & 4 \end{array} \quad [K=2]$$

2   4 1 6 2 3 7

→ 7 4 1 6 2 3

→ Algorithm

1 6 2 3 7 4 8   [K=3]

3 → 7 4 8 1 6 2 3

→ For K = 2

$$\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

$$\underbrace{1\ 6\ 2\ 3\ 7}_{\downarrow (n-k)\ \text{reverse}}\ \ \underbrace{4\ 8}_{\downarrow (k)\ \text{reverse}} \quad [K=2]$$

$$\underbrace{7\ 3\ 2\ 6\ 1\ \ 8\ 4}_{\downarrow \text{reverse}}$$

4 8 1 6 2 3 7

① → reversePart (0, n-K-1, v)   n = v.size();
reversePart (n-K, n-1, v)
reversePart (0, n-1, v)

→ Sort the arrays of 0's and 1's .

int n;

0 1 0 0 1 1 0 1

⇒ Method-I (Two Pass Method)

int noZ = 0; → count of zeros
int noO = 0; → count of ones

//noZ = 4 //noO = 4

① → Count the number of ~~0~~ zeroes and no. of One's in One pass (i.e in One iteration)

② → In second pass fill the 0th to (noZ-1)th index with 0 and rest with 1.

→ T.C → O(2n)

⇒ Method-II (Two Pointers Method)

0 1 2 3 4 5 6 7
→ 1 1 0 0 1 1 0 1

① Make 2 pointer variables
int i=0;
int j=n-1; //n → no. of elements.

```
② while (i<j){
        if(arr[j]==1) j--;      if (i>j) break;
        if (arr[i]==0) i++;
        if (arr[i]==1 && arr[j]==0) swap(   );
        i++;
        j--;
  }
```

⇒ Sort the array of 0's, 1's and 2's

(Leetcode-75)

→ Method 1 (Two Pass Method)

```
Void sortColors (vector <int> & nums){
        int n = nums.size();
        int noZ = 0;
        int noO = 0;
        int noT = 0;
        for (int i=0; i<n; i++){
            if (nums[i]==0) noZ++;
            else if (nums[i]==1] noO++;
            else noT++;
        }

        //Fill
        for(int i=0; i<n ;i++){
            if(i<noZ) nums[i]=0;
            else if ( i<(noZ+noO)) nums[i]=1;
            else nums[i]=2;
        }
        return;
```

⇒ Method-2 (3-Pointer Algorithm) (dutch flag algorithm)

→ Approach →

```
int lo = 0;
int mid = 0;
int hi = n-1; //n → size of array
```

→

0 - lo-1 → all the 0's

hi+1 - n-1 → all the 2's

```
⇒ while (mid <= hi) {
    ① if (nums[mid] == 2) {
          swap (mid, hi);
          hi--;
      }

    ② if (nums[mid] == 0) {
          swap(lo, mid)
          mid++;
          lo++;

    ③ if (nums[mid] == 1) {
          mid++;
  }
```

⇒ Merge two sorted arrays (Leetcode 88)

arr1 → 1 4 5 8

arr2 → 2 3 6 7 10
→ (Sorted)

→ Merge these 2 sorted array by creating another array.

⇒ Algorithm

→ arr1

0	1	2	3
1	4	5	8
i			

→ Size → n

arr2

0	1	2	3	4
2	3	6	7	10
j				

Size → m

arr 3

0	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	10
K								

→ Size → n+m

```
int i= 0, j=0, K=0;
while (i< n && j< m){
        if (arr1[i] < arr2(j]){
            arr3[K] = arr1[i];
            i++;
            K++
        }
        else {
            arr3(K] = arr2[j];
            j++;
            K++ ;
        }
}
```

```
~~if (i>m)~~
if (i==n){ arr1 poora uth gya
    while (j<m)
            ~~arr[3~~
            arr3[k] = arr2[j];
            j++;
            k++;
    }
}
if (j==m){ // arr2 poora uth gya
    while (i<n){
            arr[3
            arr3[k] = arr1[i];
              k++;
              i++;
    }
}
```

⟹ Find the next Permutation of array

(Leetcode - 31)

→ Algorithm

① Find Pivot index

```
int idx = -1;
for (int i = n-2; i>=0; i--) {
    if (arr[i] < arr[i+1]){
        idx = i;
    }
```

Date : / /

Page No.

```
② if (idx == -1) {
      reverse / Sort the whole array.
  }
```

③ Find the element just greater than the (idx)th element from idx+1 to end

④ Swap that element with arr[idx];