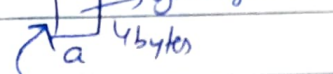
Need of Functions
Easy debugging
Neat code
Reusability
Modularization

Data Structure "Array"

We need to store 10 integers and in order to do that we need to create 10 variables & figure out the maximum element out of them which will be a tedious task.

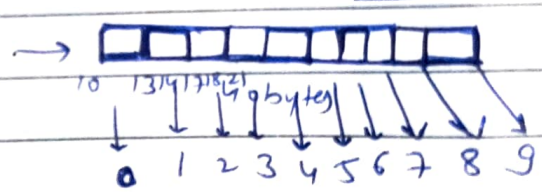
A list of similiar kind of elements stored at contiguous locations (homogenous)

→ int a;



→ a = 5;

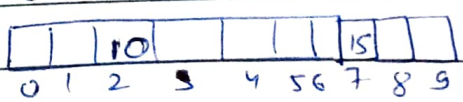
→ int a[10]
- length of array
- To save collectively
- 40 bytes contiguous block



→ indexing from 0 to (n-1)
→ Memory block → random garbage value sets.

→ a[index] = 5; ✓
↓
(0 to n-1)

int b[10]; (0-9)



b[2] = 10; ✓
b[7] = 15; ✓
~~b[10] = 3;~~ ✗

cout << b[2] << " " << b[3] << " " << b[4] << endl;

WOW!

Dt
Pg

→ Double Array

double arr[10]; → 80 bytes

→ Char array

char arr[10]; → 10 bytes

→ Take Away Input

int n;

cin>>n; → size → constant value → arr[50] arr[40] arr[10]

× int arr[n]; — × avoiding variable size array

✓ int arr[100]; → wasting of lot of space

Input an array

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

↓ index of current element

n=4 [0-3] index

arr → | 10 | 15 | 16 | 20 | | | | | | | 99

0 1 2 3 4

cin>>arr[0]; → 10
cin>>arr[1]; → 15
cin>>arr[2]; → 16
cin>>arr[3]; → 20

| 10 | 12 | 15 | 16 | | | | | |

0 1 2 3 4 5 6 7 8 -- 99

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

Largest Element in Array

input → | 10 | 5 | 15 | 17 | 9 |

0 1 2 3 4

n = 5

max = ~~10~~ 7

written in climits

INT_MIN

Approach-1

```
{
int max = arr[0];
for(int i=0; i<n; i++){
    if(arr[i] > max){
        max = arr[i]; }
}
cout<<max<<endl;
}
```

n = 5

arr → | 10 | 12 | 15 | 17 | 16 | | | |

0 1 2 3 4 99

max	i
16 ~~15~~ ~~12~~ ~~10~~	1 2 3 4

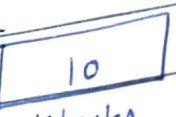
n = 0

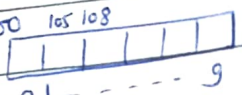
Approach-2

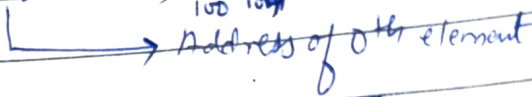
Put max = INT_MIN; → $-\infty$

How are Arrays Stored?

```
int a=10;
cout<< a <<endl;
int b=5;
cout<< b<< endl;
int c[10];
cout<< c[5]<<endl;
```








```
c[1] = 15;
```

$100 + 1 \times 4 = 104$

$b[2] = 100 + 4 \times 2 = 108$

→ $b[i] = 100 + i * 4$

↓

Address of 0th element

```
cout<< b[1]; → 5
cout<< b[2];
```

→ Address of 0th element is in hexadecimal [0x]

→ Sizeof operator → tells the size of a particular datatype

```
int a;
double b;
cout<< sizeof(a) << sizeof(b)<<endl;
```

↓ 4 bytes ↓ 8 bytes

```
cout<< sizeof(int) <<endl; → 4 bytes
cout<< sizeof(double)<<endl; → 8 bytes
```

Dt.
Pg.

Arrays and Functions

```
int main(){
    int b[5];
    fun(b);
}
```

sizeof(b) → 20 bytes

b → Address of 1st element

20 bytes

100 104

```
void fun(int b[]){
    for(int i=0; i<size; i++){
        cout << b[i] << endl;
    }
}
```

sizeof(b) → 8 bytes → pointer variable

b → 100 → int size

Linear Search - Technique or algorithm to search the position of an element in the array.

Consider an array and a key value of 9 in the array

arr → | 7 | 2 | 3 | 8 | 9 | 1 |

0 1 2 3 4 5

ans → 4

Complexity → Time → O(n) → worst case as we need to compare the element (target) with each and every element of the array in order to find the element.

Space → O(1) → As no need of an auxiliary space (extra space)

Populate the array

1 to N

1 to 4

| 1 | 3 | 4 | 2 |

int val = 1

start

end

1 to 5

| 1 | 3 | 5 | 4 | 2 |

0 1 2 3 4

start = 0 1 2

end = 4 3 2

val = 1 2 3 4 5

Passing Arrays To Functions

Initialize Array at time of declaration

variable → int a = 5;

Array → int b[100];

int n; cin>>n;

for(i=0 → n) { cin>>input[i];

Initialization of Array at time of declaration

→ int input[] = {1, 2, 3, 4};

→ int input[3] = {1, 2, 3};

→ int input[6] = {6, 5, 4}; → | 6 | 5 | 4 | 0 | 0 | 0 |
0 1 2 3 4 5

→ int input[10] = {0}; → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

→ int input[4] = {1}; → | 1 | 0 | 0 | 0 |

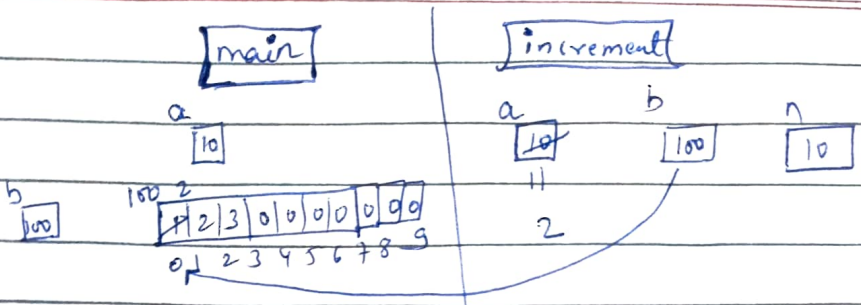
Function Increment

```
void increment (int arr[], int b, int n){
    arr[0]++;
    b++;
}

int main(){
    int b=10;
    int arr[10] = {1,2,3}
    increment(arr, b, 10);
    cout<< b <<endl;      → 10
    cout<< arr[0] <<endl; → 1
}
```

arr → | 1 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
0 1 2 3 4 5 6 7 8 9

Dt
Pg



increment (a, b, 10)

10 100

Reverse An Array

start	end
0	4
1	3
2	2

arr → | 1 | 2 | 3 | 4 | 5 |
0 1 2 3 4

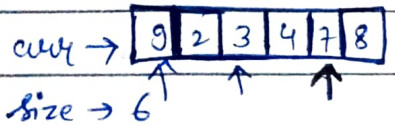
| 5 | 4 | 3 | 2 | 1 |

```
void fun(int arr, int size){
    int start = 0, end = size-1;
    while(start <= end){
        swap(arr[start], arr[end]);
        start++;
        end--;
    }
}
```

i = 0 → i++

j = n-1 → j--

Swap Alternate Elements



Dt
Pg

Assignment Questions

#1) Find Unique Element

arr[] → | 2 | 3 | 1 | 6 | 3 | 6 | 2 | , size = 7
0 1 2 3 4 5 6

i	j	count
0	0 - 6	1
1	1 - 6	1
2	2 - 6	1
3	3 - 6	1
4	4 - 6	1

```
for(int i=0; i<n; i++){
  for(int j=0; j<n; j++){ int count=0;
  if (arr[i] == arr[j]){
      count++; }
  if (count == 1){
      cout << arr[i] << endl;
  }
```

```
for(int i=0; i<size; i++){
    int count=0;
    for(int j=0; j<size; j++){
        if(arr[i] == arr[j]) count++; }
    if(count==1) cout << arr[i] << endl;
}
```

Solution
Time complexity
Worst case $O(n^2)$

#2) Sort the array and we get the consecutive elements together

| 1 | 2 | 2 | 3 | 3 | 6 | 6 | → $O(n \log n)$

#3) XOR all elements with 0 ... we get unique element.

ans = 0

ans ^= arr[i];

cout << ans << endl;

$0 \wedge 1 \wedge 2 \wedge 2 \wedge 3 \wedge 3 \wedge 6 \wedge 6$

$\Rightarrow 1$

Dt.
Pg.

#2) Find Duplicate Element -

arr → | 0 | 7 | 2 | 5 | 4 | 7 | 1 | 3 | 6 |
0 1 2 3 4 5 6 7 8

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

Time Complexity → $O(n^2)$ → n is the size of the array

(2) # Using XOR operation

$(0^0^7^2^5^4^7^1^3^6)^(0^1^2^3^4^5^6^7)$

$0^0^7^2^5^4^7^1^3^6^0^1^2^3^4^5^6^7$

$O(n)$

(2) ArrSum += arr[i]

ActualSum = (n-1)*(n-2)/2;

ArrSum - ActualSum = Duplicate Number

Example - | 1 | 5 | 2 | 8 | 9 | 1 |
0 1 2 3 4 5

$\frac{4*3}{2} = \frac{12}{2} = 6$

26 − 6 = 20

$\frac{5*4}{2} = 10$

26 − 10 = 16

n = 6 → $\frac{5*4}{2}$ → $\frac{20}{2}$ → 10

Sum = $\frac{5*4}{2}$

26

Dt
Pg

Duplicate Number

arr → | 1 | 3 | 0 | 2 | 1 |
0 1 2 3 4

Approach - 1 Using comparing Approach by picking one element and then comparing it by itself.

i	j	count
0	0	1 2

```
for(i=0 to size){
    int count=0;
    for(j=0 to size){
        if(arr[i]==arr[j]){
            count++;}
    if(count==2) return arr[i];
    return -1;
```

⇒ 1

Approach - 2 → Using XOR operation

```
int ans=0;
for(i=0 to size){ ans^=arr[i];
for(i=0 to size-2) ans^=i;
return ans;
```

(0^1^3^1^2^0^1)^(0^1^2^3)

⇒ 1

Dt. _____
Pg. _____

Approach-3: Sum of array by

sum += arr[i];

Sum from 0 to size-2,

S = ((size-2)*(size-1))/2;

return sum-S;

sum = 7

S = 4*3/2 = 12/2 = 6

return (7-6); => 1 ✓

Intersection of Two Arrays

arr1[] = {2, 6, 8, 5, 4, 3}

arr2[] = {2, 3, 1, 7}

-∞

Intersection of two arrays = {2, 3, 4}

Sorting arrays → {2, 3, 4, 5, 6, 8}

→ {2, 3, 4, 7}

i = 0 (arr1) j = 0 (arr2)

```
while (i<size1 && j<size2){
    if (arr1[i]<arr2[j]){
        i++; }
    else if (arr2[j]<arr1[i]){
        j++; }
    else {
        cout<<arr2[j];
        i++;
        j++;
    }
```

Dt
Pg

Pair Sum

```
int count=0;
for (i=0 to n) {
    for (j=i+1 to n) {
        if (arr[i] + arr[j] == k) {
            count++; }
    }
}
return count;
}
```

arr → | 1 | 3 | 6 | 2 | 5 | 4 | 3 | 2 | 4 |
indices: 0 1 2 3 4 5 6 7 8

target → 7

i	j	target	count
0	1-8	7	1
1	2-8	7	2
2	3-8	7	X
3	4-8	7	1
4	5-8	7	1
5	6-8	7	1
6	7-8	7	1
7	8	7	0
8			

Triplet Sum

```
int cnt=0;
for(i=0 → n) {
    for(j=i+1 → n) {
        for(k=j+1 → n) {
            if(arr[i]+arr[j]+arr[k]==target){
                count++; }}}}
return count; }
```

WOW!

→ arr[] → | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
0 1 2 3 4 5 6

target → 12

i	j	k	target	count
0	1	2		~~1 2 3 4~~ 5
1				
2				
3				
4				

Sort Zero One (0 & 1)

arr → | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

```
int count=0;
for(int i=0; i<size; i++){
        if (arr[i]==0){
                count++;}
int i=0;}
for([redacted]; i< count; i++){
        arr[i]=0;}
for([redacted]; i<size; i++){
        arr[i]=1;}
```

END OF TOPIC