Week-3

Array level-1

What is an Array?

→ List of similar items
→ collection of elements
→ Datastructure
→ continous memory allocation

A container which contains same datatypes elements

Why we used Array?

As numbers or variables increases it is not possible to create such large number of variables.

```
int  arr[10000];
```

- `int` → array k andar int type ka data hai
- `arr` → Array name
- `10000` → 10000 blocks

```
int a = 10;
```
← Takes 4 bytes

```
int arr[10];
```
← Takes 4 x 10 bytes

Creation of Array:

```
int money[27];
char alphabet[26];
bool flags[27];
```

Note -
Computer maintains one symbol Table where it works like

variable name	Address
a	→ 104

Address of operator
e.g →
&a

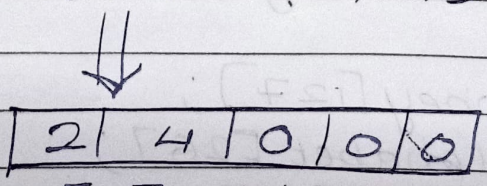
&arr , arr are same, both stores address.

Size of operator.

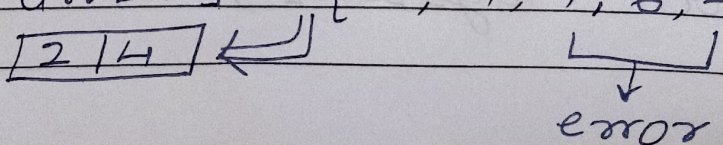
sizeof(a) gives size of a variable

Array Initialization

1. int arr[] = {1, 3, 2, 6, 8}
2. int brr[5] = {1, 2, 3, 6, 8}
3. int crr[5] = {2, 4,}

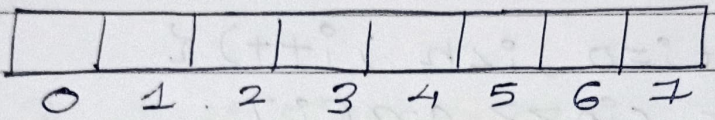


4. int drr[2] = {2, 4, 4, 6, 4}



Taking n from user
then giving size,
int arr[n];
But this is bad practice. There is a chance of memory overflow.

Indexing in Array



To access array elements, indexing is neccessary.

if n=5,
1-based indexing → start from 1
end at 5
0-based indexing → start from 0
end at 4

Code:

```
int arr[5] = { 5, 8, 9, 12, 13 };

cout<<arr[0]<<endl;
cout<<arr[1]<<endl;
cout<<arr[2]<<endl;
cout<<arr[3]<<endl;
cout<<arr[4]<<endl;
return 0;
}
```

O/P: 5
8
9
12
13

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

O/p → 5, 8 9 12 13

Taking input

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

Formula

arr[i] ⇒ value at (Base Add + (datatype size * index))

Behind the scene it is working like this.

Q. 1 Problem statement. ⇓
1. 10 size array based
2. Take i/p in that array
3. Double up each value of that array.

Code:

```
//array create kardo
int arr[10];
// input lelo
int n=10;
for(int i=0; i<n; i++){
    cin>> arr[i];
}
```

```
//double-UP
for(int i=0; i<n; i++){
    arr[i] = 2*arr[i];
}
//print
for(int i=0; i<n; i++){
    cout<<arr[i]<<" ";
}
```

Q.2 1. 5 size Array
2. Take input
3. Total sum print

Code:

```
int arr[5];
//input
int n=5;
for(int i=0; i<n; i++){
    cin >> arr[i];
}
//calculate sum
int sum=0;
for(int i=0; i<n; i++){
    sum += arr[i];
}
// Then print
```

Linear search in an Array

i/p target = 10

2	4	6	8	10	12
0	1	2	3	4	5

o/p → Found / Not Found

Code:

```
int arr[5] = {2, 4, 6, 8, 10};
int target = 10;
int n = 5;
for(int i=0; i<n; i++) {
    if(arr[i] == target) {
        // Found
        cout << "target found" << endl;
        break;                          (Not Necessary)
    }
}                                       flag = 1;

if(flag) {
    cout << "Target Found";
}
else {
    cout << "Target Not found";
}
```

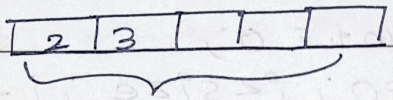
Arrays and functions

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

```
solve(          ) {
}
```

size

If we want to pass array in other function then we compulsory pass its size also.

why?
Because sizeof function will give you  but we want to the exact size filled in the array.

Code:

```
void printArray(int arr[], int size) {
    for(int i=0; i<size; i++) {
      cout << arr[i] << " ";
    }
    cout << endl;
}
```

```
bool linearSearch(int arr[], int size, int
                                    target) {
  for(int i=0; i<size; i++) {
    if(arr[i] == target) {
      return true;
    }
  }
  return false;
}
```

Count 0's & 1's in an Array.

0	1	1	1	0	0	1	1

1. Traverse the array
2. count zeros & ones
3. Print.

Code:

```
void countZeroOne(int arr[], int size){
    int zerocount = 0;
    int onecount = 0;
    for(int i=0; i<size; i++){
        if(arr[i]==0){
            zerocount++;
        }
        if(arr[i]==1){
            onecount++;
        }
    }
cout << "zerocount: " << zerocount<<endl;
cout << "onecount: "<< onecount<< endl;
```

Minimum number in an array

Integar minimum & maximum

$-2^{31} \longrightarrow 2^{31}-1$

INT_MIN INT_MAX

inside header file ⇒ limits.h

Steps ⇒

(A) int minAns = INT_MAX

(B)
```
for (i=0; i<n; i++) {
    if (arr[i] < minAns) {
        minAns = arr[i]
    }
}
```

Alternate

(B)
```
for (i=0; i<n; i++) {
    minAns = min(arr[i], minAns);
}
```

Code:
```
int findMinimumArray(int arr[], int size){
    // ans store variable
    int minAns = INT_MAX;
    for (int i=0; i<size; i++) {
        if (arr[i] < minAns) {
            minAns = arr[i];
        }
    }
    return minAns;
}
```

Reverse an Array

i/p →

10	20	30	40	50	60
0	1	2	3	4	5

o/p →

60	50	30	30	20	10
0	1	2	3	4	5

Taking / maintaining 2 pointer Left and right. swapping the elements and shrinking them. Stop when Left cross Right.

code;

```
void reverseArray (int arr[], int
                   size) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
    swap (arr[Left], arr[right]);
     Left++;
    right--;
    }
```

Can be done using for loop too.

H.W ⇒ Swap Implement using 3 Approach

Page 75

Extreme print in an Array

I/p →

10 20 30 40 50 60

O/p →

10 60 20 50 30 40

Using Left and right Pointers

Code:

```
void extremePrint (int arr[], int size)
{
 int Left=0;
  int Right=size-1;

while (Left<=right){
    if (Left==right){
     cout<< arr[Left]<<endl; }
    else {
    cout<< arr[left] <<endl;
    cout<< arr[right] <<endl;
  }
   Left++;
  Right++;
 }Right--;
```

Home Work

(1) swapping using temp variable

```
int main() {
    int a=2, b=3;
    int temp;
    temp=a;
    a = b;
    b=temp;
    cout << a << " " << b << endl;
    return 0;
}
```

o/p ⇒ 3 2

↑ a ↑ b

(2) swapping using '+' '−', without using temp variable

```
int main() {
    int a=2, b=3;
    b= a+b;
    a= b-a;
    b= b-a;
    cout << a << " " << b << endl;
    return 0;
}
```

o/p ⇒ 3 2

↑ a ↑ b

(3) swapping using XOR operator

```
int main() {

    int x=10, y=5;

    x = x^y;
    y = x^y;
    x = x^y;
    cout<< x << " " << y <<endl;
    return 0;
}
```

o/p ⇒ 5 10