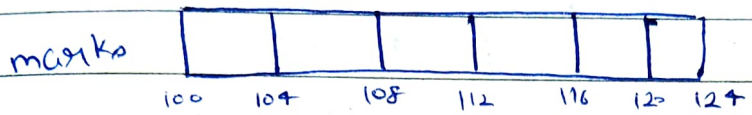
★ Arrays



- Same type
- Contiguous in memory + Linear

जैसे

```
int marks[5];
```

OR

```
int marks[] = {99, 100, 85, 90, 91};
```

Array की index 0 से Start होता है!

```
int marks[] = {99, 100, 85, 90, 91};
int size = sizeof(marks) / sizeof(marks[0])
for (int i = 0 ; i < size ; i++) {
    cout << marks[i] << endl;
}
```

★ ⇒ Arrays are linear collection of same type of elements that are stored together in contiguous memory space.

Memory is statically allocated

(at compile time)

DATE / /
PAGE No.

★ Arrays are passed by reference

Array का नाम Pointer की तरह काम करता है और वह index first के Element को Point करता है।

```
void printArr(int arr[])
void printArr(int *arr)
```
} Same

Example:

```
int arr[] = {0, 1, 2, 3, 4}
```

0	1	2	3	4

arr

★ arr first index के Address को
★ store करता है।

```
int a = 5;
int *ptr = &a;
cout << ptr << endl;      // Address of a
cout << *ptr << endl;     // value of a
```

★ Same
★

```
cout << arr << endl;      // Address of arr[0]
cout << *arr << endl;     // value of arr[0]
cout << *(arr+1);         // value of arr[1]
```

DATE / /
PAGE No.

- Function में हमेशा Array Pass by reference होता है!!

- Passing array name is equivalent to passing the pointer.

क्योंकि Array का नाम Base address Store करता है और Pointer भी Address Store करता है।

Example

```
void func(int *ptr);
int main() {
    int arr[] = {0,1,2,3,4};
    func(arr);
    return 0;
}
void funct(int* ptr) {
    cout << ptr[0];      // output is 0
    cout << ptr[1];      // output is 1
    ptr[3] = 100;
    cout << ptr[3];      // output is 100
```

क्योंकि array हमेशा Pass by reference होते है तो कोई भी change Original array में भी change लाता है।

सार

- Array Function में Pass by reference होते है!
- Array का नाम Base address store करता है!!
- (int arr[]) == (int * ptr)
- arr[0] == ptr[0]

DATE / /
PAGE No.

Important Baat

जब हम Function में Array को Pass करते हैं तो उसके size को भी Pass करना होता है।

क्योंकि Loop चलाते हुए size का होना जरूरी है! लेकिन Function में जब हम Array Pass करते हैं, तो उस Array की Memory location लेने के लिए वहां Pointer (ptr) होता है।

Example

```
int arr[] = {1, 2, 3, 4, 5}
int size = sizeof(arr) / sizeof(arr[0]);
fof(arr);
```

यहां sizeof(arr) total memory देगा 20 लेकिन !!

Function → int fof (int *ptr),

यहां size of (ptr) / size of (ptr[0]) ✗ गलत

✗

sizeof(ptr) ≠ 20 = 8 size of pointer

इसलिए Array को Pass करते हुए, उसका size भी Pass करना होता है।

```
void fun (int *ptr, int n);
void fun (int *ptr, int n);
```

DATE / /
PAGE No.

★ Reverse an array

यह दो तरह से हो सकता है।

(1) With extra space
(2) Without extra space

With extra space में एक नई Array बनाते है और Reverse करके Array element fill करते है।

Without extra space, No new array is made, just reverse the element in the main array.

```
int arr[] = {1, 2, 3, 4, 5};
int n = sizeof(arr) / sizeof(arr[0]);

for (int i = 0; i < n; i++){
        int temp = arr[i];
        arr[i] = arr[n-i-1];
        arr[n-i-1] = temp;
}
```

Now cout array.

Searching

DATE / /
PAGE No.

★ Linear Search

Both for sorted array and unsorted (sorted) array.

```
int target;
cin << target;
for(int i=0; i<n; i++) {
    if (arr[i] == target) {
        cout << "Element found";
            flag = 1;
            return;
    }
```

★ Binary Search

Just for sorted array.

This is for better time complecity

Target = 12

2	4	6	8	10	12	14	16

(i) Find out mid

$$mid = \frac{(start + end)}{2}$$

$$mid = \frac{0 + n-1}{2}$$

DATE / /
PAGE No.

Best for searching

Pseudocode

```
int str = 0 ; end = n-1
while (str <=end){
        mid = (str + end)/2
        if (arr[mid] == key)
            return mid,
        else if (arr[mid] < key)   //2nd half
            str = mid+1
        else                       // Ist half
            end = mid-1
}
```

Binary search is more acceptable and has better time complecity.

Note : But just for sorted array.

⇒ Time complexity for binary search

Ist iteration — array size
n

2nd — $n/2$
3rd — $n/4$
4th — $n/8$
⋮
x^{th} time — $n/2^{x-1} = 1$

$$\frac{n}{2^{x-1}} = 1$$

$n = 2^{x-1}$ $\qquad$ $x = \log n + 1$

$\log_2 n = x-1$ $\Rightarrow$ $x = \log n$

constants and base are being ignored

Time complexity $= O(\log n)$

$$O(\log n) << O(n)$$

So binary search is far better than linear search.

★ Array Pointer.

```
int x = 25;
int * ptr = &x;
cout << *ptr;        // 25
int y = 10;
ptr = &y;
cout << *ptr;        // 10
```

Here we can change address stored in any pointer, but this is not possible in array pointer.

```
int arr[5];
int x = 25
arr = &x;
```

→ Not possible

Because array pointer only stores address of array element.
It can't store address of others.

Array is Constant Pointer !!

DATE / /
PAGE No.

★ Pointer Arithmetic

Increment & Decrement Operator.

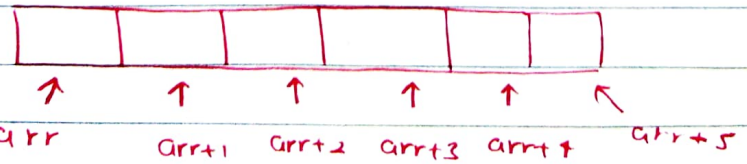
ptr ++ , ++ ptr
ptr -- , -- ptr

we can increment or decrement a pointer but how much it will be increment depends upon its type

<u>example</u> int *ptr
Let ptr = 104 (Address)
ptr ++ = 108
increment 4 because it is int type
char *ptr = 100
ptr ++
ptr = 101 Because of char

- ptr + 3
Let ptr is int type
than 3 * int = 3 x 4 = 12 space skip
Let address is 101
ptr + 3 , Address 113

सार ★



इस तरह हम Array की Diff. Location को Access करते हैं।

DATE / /
PAGE No.

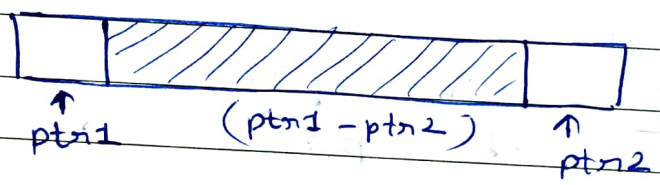
- Addition & Substraction of Pointers

ptr 1 + ptr 2 // Invalid ✗

ptr 1 + ptr 2 is incorrect
But

ptr 1 − ptr 2 // ✓
But both should be of same type

This gives space between both address



ex

```
int a = 10;
int * ptr1 = &a;
int * ptr2 = ptr1 + 3;
cout << ptr2 - ptr1; // 3 int space
```

DATE / /
PAGE No.

★ Print Subarrays

Subarray is continuous part of array

1	2	3	4	5

Subarrays :- 1 1,2 1,2,3 1,2,3,4
1,2,3,4,5 2, 2,3 etc.

Subarray → continous part
Subsequence → non - continous part

If there is n size of an array then, total no. of subarrays is:
$n + (n-1) + (n-2) + (n-3) + \dots + 1$

$$\boxed{\text{Total no. of subarrays} = \frac{n * (n+1)}{2}}$$

Code !!

```
int arr[] = {1, 2, 3, 4, 5, 6}
int n = sizeof(arr) / sizeof(arr[0]);

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

Time complexity ⇒ $O(n^3)$

★ Max Subarray Sum

Brute Force Approach

For positive array

10	20	30	40	50

Max. Subarray Sum = Sum of array

But for negative :

→ Find out all subarray
→ Find out max by max( ),
→ Print max sum.

code !!

```
int arr[]= {2, -3, 6, -5, 4, 2};
int n= size of (arr) / size of (arr[0]);
int sum = 0;
int maxi = INT_MIN;
for (int str =0; str < n; str++){
    for (int end =str; end <n; end++){
        sum = 0;
        for (int i= str; i<=end; i++){
            sum + = arr [i];
            maxi = max(sum, maxi);
        }
    }
}
cout << "Max. Subarray Sum " << maxi << "
return 0;
```