25 August 2023 20:48

- Pointers
- 2D Arrays.
- Doubts

Max. number in an array.

arr = {5, 9, 11, 10, 4, 3, 12, 7}

max = arr[0] = 5

9 > max → max = 9

11 > max → max = 11

10 > max ✗

4 > max ✗

3 > max ✗

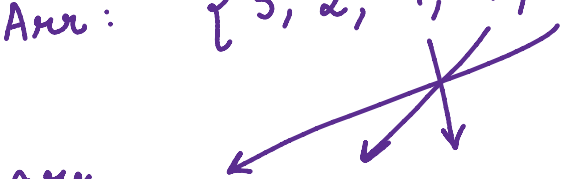
12 > max → max = 12

7 > max ✗

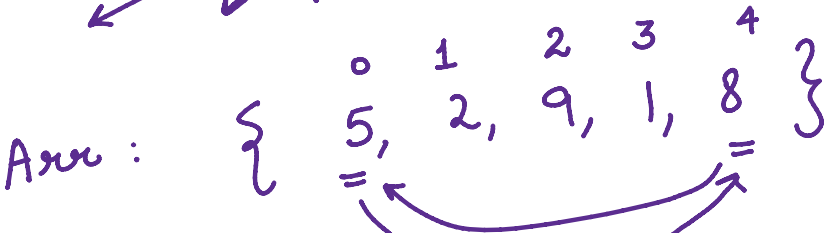
max = 12

Reverse an array.

Arr: {5, 2, 9, 1, 8} → {8, 1, 9, 2, 5}



arr



Arr : 1

n

8, 2, 9, 1, 5

8, 1, 9, 2, 5

arr[i], arr[j]

$i = 0$, $j = n-1$

```
while (i < j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++; j--;
}
```

① Arr: ~~11~~ 22 33 ~~44~~ (11)

0 1 2 3

$i = 0$, $j = 3$ — $i < j$ True

temp = arr[0] = 11

arr[0] = arr[3] = 44

arr[3] = temp = 11

Arr: 44 ~~22~~ ~~33~~ 11

$i++$, $j--$ — $i = 1$, $j = 2$ — $i < j$ True

temp = arr[1] = 22

arr[1] = arr[2] = 33

arr[2] = temp = 22

Arr: 44 33 22 11 ✓

i++ j-- i=2 j=1 i<j False

② Arr = { ~~50~~, 20, 30, 40, ~~10~~ }
0 1 2 3 4

i=0 j=4 i<j True

temp = arr[0] = 10
arr[0] = arr[4] = 50
arr[4] = temp = 10

Arr: 50, ~~20~~ 40, 30, ~~40~~ 20, 10

i++ j-- i=1 j=3 i<j True

temp = arr[1] = 20
arr[1] = arr[3] = 40
arr[3] = temp = 20

Arr: 50, 40, 30, 20, 10
↑ Reverse

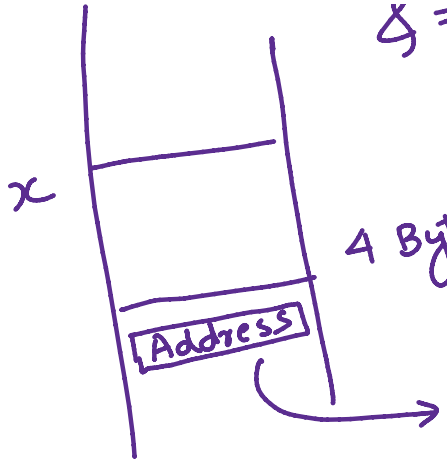
i++ j-- i=2 j=2 i<j False ✓

Pointers

| | & = address operator

Pointers

```
int x;
x = 5;
```



& = address operator

Address of the variable = & var-name

cout << x → Value stored at x

cout << & x → Location where x is stored

Pointers are the variables that stores the address of other variables

```
int x;
int * y = & x;
```

Integer pointer

→ char * ch
Store address of character variables

→ Stores address

```
float *f;
```
→ Stores address of floating point variable

```
int a;
int *b = &a;
```
→ address of a

```
cout << &a
cout << b
```
→ address of a

```
int a = 5;
int *b = &a;
```

```
cout << &a
cout << b
```
→ Address of a

```
cout << a;
cout << *b;
```
→ Value of a → '5'

& → reference operator

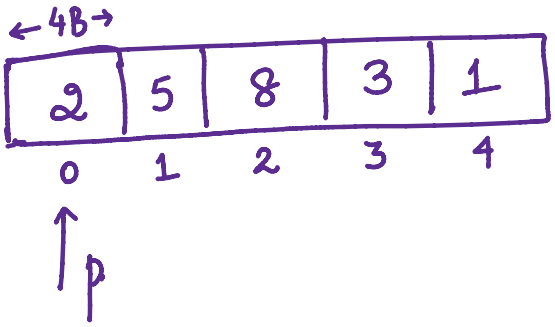
* → dereference operator

Address of a

$\&a \equiv b \equiv$ Address of a

$a \equiv *b \equiv$ Value of a

Pointer and array




```
int *p;
p = &arr[0];
p++;
```

```
cout << *p;   (2)
cout << *p;   (5)
```

```
char* ch;
ch++;
```

↳ Go to address of next character

1B forward

```
int * i;
i++;
```

↳ Go to address of next integer

4B forward

```
double * d;
d++;
```

↳ Go to address of next double

8B forwar

```
double *dl;
dl++;
```

dl = 1000

dl = 1008

p = p + 1

p = p + (1 * size of datatype)

2D Array → Array of arrays.

```
int arr[2][3] = { {5, 6, 7}, {2, 4, 3} }
                  arr[0]     arr[1]
                  0 1 2      0 1 2
```

2 Rows
3 columns.

arr[0][1] = 6
arr[1][2] = 3

arr:

5	6	7
2	4	3

"matrix"

Print all the elements of 2D array:

```
for (int i = 0; i < R; i++) {
```

R, C

```
for (int i = 0; i < R; i++) {
    for (int j = 0; j < C; j++) {
        cout << arr[i][j];
    }
    cout << endl;
}
```

2D arrays: → Row Major Order R_1, R_2, R_3

→ Column Major Order C_1, C_2, C_3

$$arr = \begin{bmatrix} 2 & 5 & 7 \\ 8 & 9 & 1 \end{bmatrix}$$

✓ Row Major Order:

2	5	7	8	9	1

Column Major Order:

2	8	5	9	7	1

$$arr: \begin{bmatrix} 2 & 1 & 5 \\ 3 & 8 & 6 \end{bmatrix}$$

arr[0][0], arr[0][1], arr[0][2], arr[1][0], arr[1][1], arr[1][2]

R, C

R, C

arr =

```
for ( int i = 0; i < R; i++) {
    for (int j= 0; j < C ; j++) {
        cout << arr[i][j];
    }
    cout << endl;
}
```

2D Vector

Vector of vectors

→ 1D vector

```
vector <int> vec1;
```

→ 2D vector

```
vector < vector<int> > vec;
```

```
vector < datatype > name;
```

datatype → vector of int

```
{ { 1, 2, 3 } , { 5, 6, 7 } }
```

{ → "vector <int>"

{1, 2, 3} , {5, 6, 7} → vector <int>

"vector <int>"

vector <int>

vector < datatype > name;

vector < vector <int> > vec;

```
vector<vector<int>> vec={{1,2},{4,7},{0,8,9},{6,1,4,3}};
   for(int i=0;i<vec.size();i++){ //vec.size() is the number of rows
      for(int j=0;j<vec[i].size();j++){ //vec[i].size() is the number of columns
in vec[i]
         cout<<vec[i][j]<<" ";
      }
      cout<<endl;
   }
```

```
int arr[10]={0};
   for(int i=0;i<10;i++){
      cout<<arr[i]<<" ";
   }
```

2D array

"R = C"

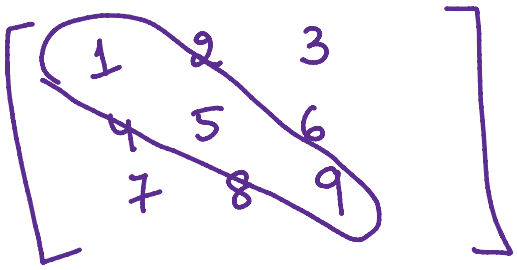
Sum of all the elements

```
int sum = 0;
for (int i = 0; i < R; i++) {
    for (int j = 0; j < C ; j++) {
        sum += arr[i][j];
    }
}
```

Sum of diagonal elements:

[1 2 3]

R=C



1+5+9

```
for (int i=0; i<R; i++) {
    for (int j=0; j<C; j++) {
        if (i==j) {
            sum+= arr[i][j];
        }
    }
}
```

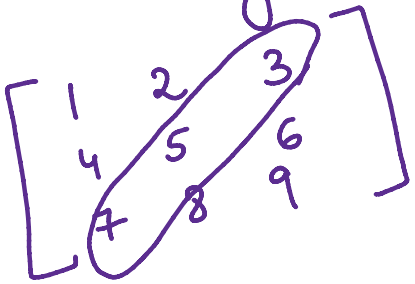
```
for (int i=0; i<R; i++){
    sum+ = arr[i][i];
}
```

i=0: sum+ = arr[0][0]

i=1: sum+ = arr[1][1]

i=2 sum+ = arr[2][2]

Sum of anti diagonal elements

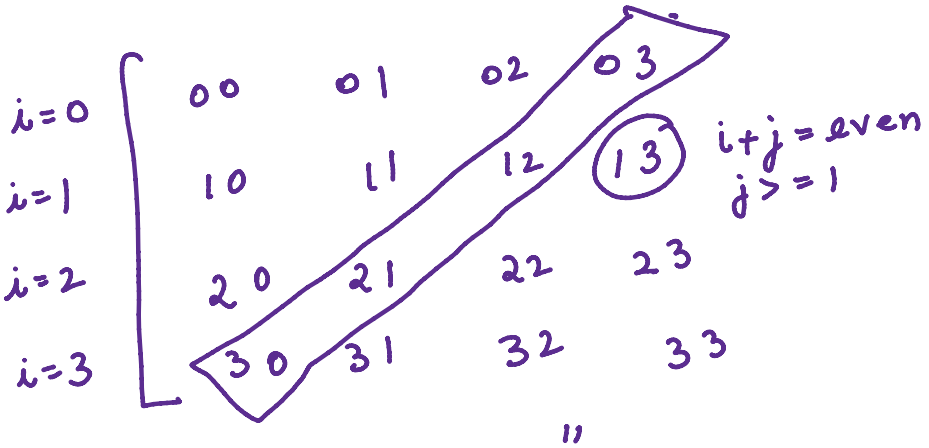


3+5+7

LC 3+5+7

4 X 4 matrix

N = 4



" $i + j = n - 1$ "

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

$i + j = n-1$

$j = n-1-i$

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

Sum of each row

. [. 5 3]

~~sum~~

Arr: $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 7 & 1 \\ 8 & 2 & 5 \end{bmatrix}$ 9, 10, 15

```
int sum;=~~0;~~
for (int i = 0; i < R; i++) {
    sum = 0;
    for (int j = 0; j < C; j++){
        sum += arr[i][j];
    } cout << sum;
}
```

1 2 3

sum=0 → 4 5 1
2 3 4

6
16 ×

Arr:
1 2 3
3 2 5
4 3 1

i=0: sum = 0 Sum = ~~1~~ 1+2 = ~~3~~ 3+3 = 6 (6)

i=1 sum=0 Sum = ~~3~~ 3+2 = ~~5~~ 5+5 = 10 (10)

i = 1 Sum = 0 Sum = ~~3~~ 3 + 2 = ~~5~~ 5 + 5 = 10 (10)

i = 2 Sum = 0 Sum = ~~4~~ 4 + 3 = ~~7~~ 7 + 1 = 8 (8)

i = 3 X