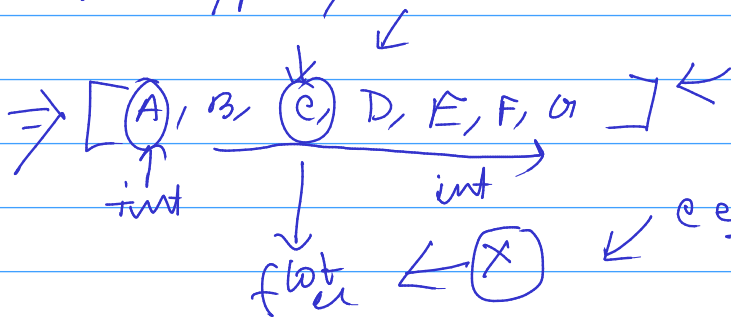
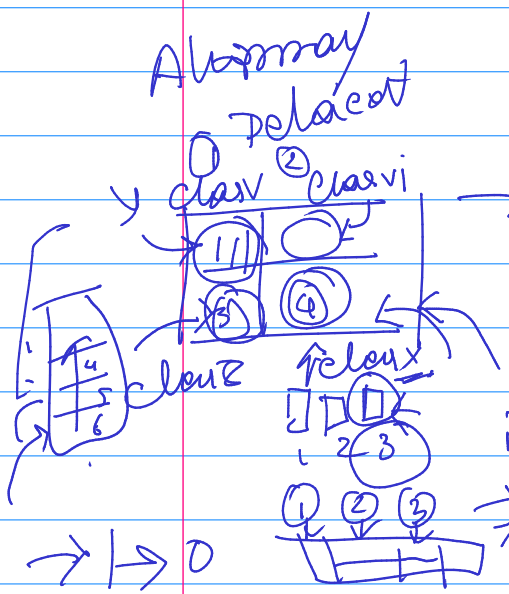
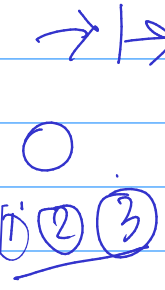
Array → collection of elements of same type

C - varible type → int, char, float, double



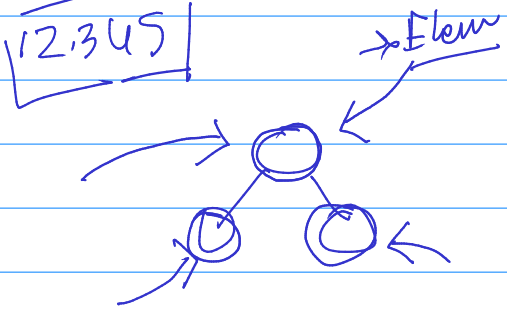


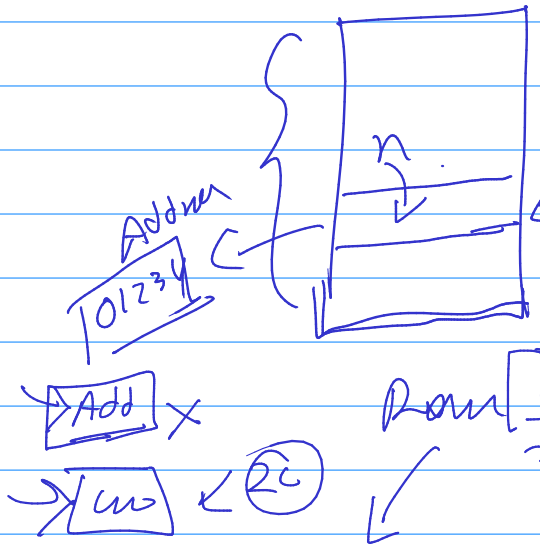


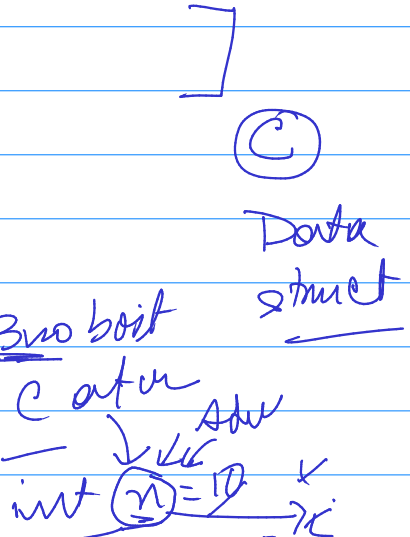
① scanf("%d", &n);

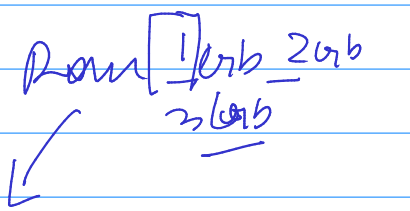
Address of n

① garbage value







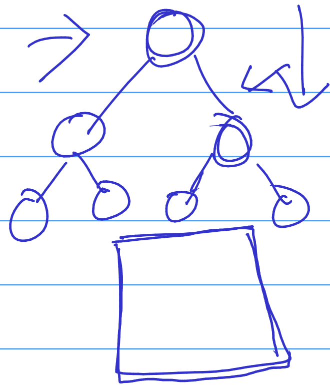


⇒ C → Data Structure ⇒ ① [Array]

② Linked List

③ Tree

→ Tree → man



⇒ ① Array ⇒ [1, 2, 3, 4, 5, 6]

↳ Continious memory Address

⇒ ①

① Same type

② Continious memory location

[1234] → [1235] [1230]

① ② ③ ① ⇏ ③

Array

→ Indexing ⇒ [1, 2, 3, 4, 5] ← Element ⇒ 5

↳ position → 0th 1st 2nd 3rd 4th position

(n−1) the point.

int [x] = 10?

⇒ Array Delecaration ⇒

① ⇒ int arr[5] = {1, 2, 3, 4, 5}; int x = 10; int n; ?

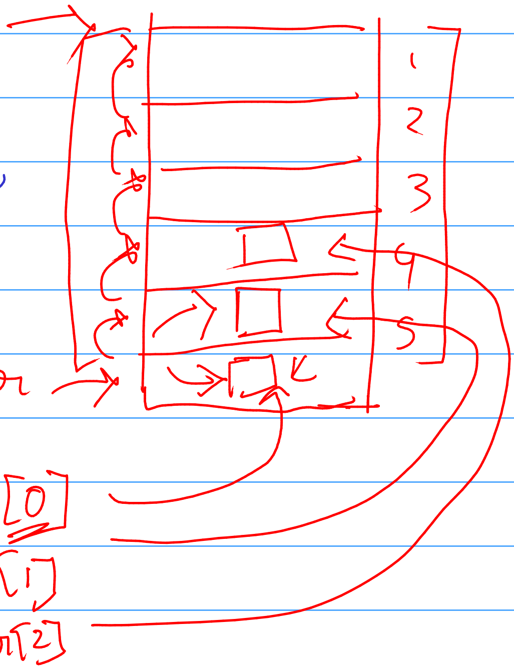
type name size

a b, l

② ⇒ int arr[ ] = {1, 2, 3, 4, 5};

size Auto calculate.

③ ⇒ int arr[5];

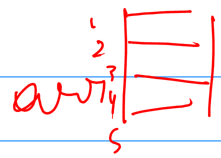


① Insert 5 elements in an array and print it?

Ans:

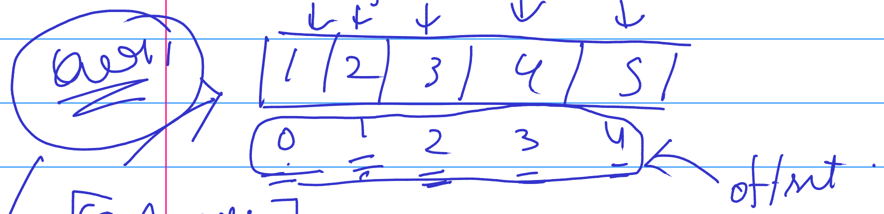
```c
#include <stdio.h>

int main(){
    int arr[5] = {1,2,3,4,5};
    printf("%d", arr);
}
```



int x = 10;

```c
int main(){
    scanf("%d", &arr);
}
```

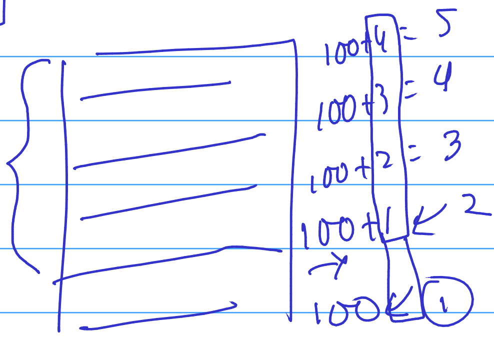


printf("%d", n) ; n = address

① Access

Base

offset



int x = 10
int y = 20
int z = 30

printf("%d", x, y, z);

arr → Base Address

Array size

```c
for(int i=0; i<n; i++){
    printf("%d", arr[i]);
}
printf("%d", arr[0]);   => 1
printf("%d", arr[4]);
```

→ array[2] = ③

0101 [C → memory] ⇒ [memory] [0xfffff] (H) → C

⇒ [Array memory | layout] Array[5]

↳ Example :⇒ int arr[5] = {(10) 20, 30, 40, 50}; (→ Addr)

→ 4byte → Unit → Size of Int :⇒ (4) space

memory layout :⇒

Address	value	
0x1000	10	← arr[0] (base Address)
0x1004	20	← arr[1]
0x1008	30	← arr[2]
0x100C	40	← arr[3]
0x1010	50	← arr[4]

4+↓

char = 1

Decimal Binary

[12 → C]

(10) 1001

8 4 2 1 → 1 0 1 0 → 1010

16 8 4 2 1 → 0 1 0 1 0

⇒ Formula for element address :⇒

⇒ arr[5] = {1, 2, 3, 4, 5}
   0 1 2 3 4

⇒ Address of arr[i] = base Address + (i * sizeof(type))
(↓ int)

Example :⇒
Addr arr[2] ⇒ 0x1000 + (2 * 4)
⇒ 0x1000 + 8
⇒ 0x1008

⇒ [Data Structure] ← ⇒ (1, 2, 3) ⇒ { 100 ← A, 200 ← B, 300 ← 3 } [400] e ↓ An

⇒ Avg ⇒
Total ⇒
A+B+C → K
So lowest mark ⇒
highest mark :?

Array | Li

Array | Linked list