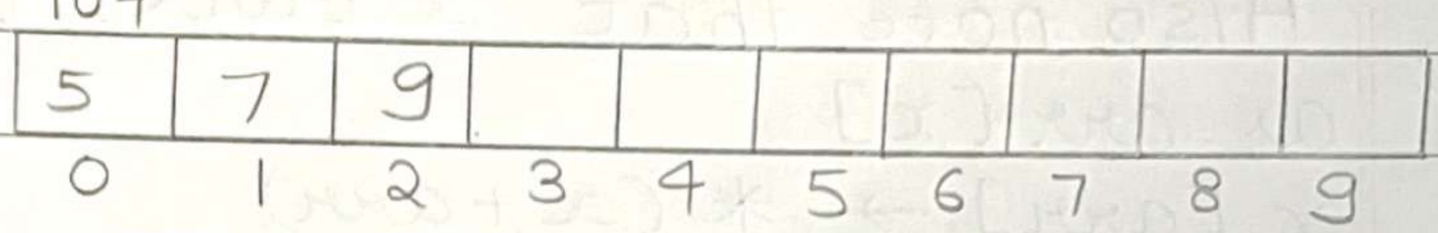
4/03/2023

int arr [10];
This will create 10 contiguous blocks & in each block will be storing integer.



cout << arr [0] << endl; → 5
cout << arr << endl; → 104 } Base
cout << & arr << endl; → 104 } address
cout << & arr [0] << endl; → 104 ↑

int * p = arr;
cout << p << endl; → Base address i.e. 104
cout << & p << endl; → Address of pointer
↳ We can get stuck here.

Note → arr and & arr will give output as base address of the array. This is because of entry maintained in the symbol table.

Symbol Table
arr ⟶ 104 } Actually this is happening in the memory

arr, & arr, & arr [0] represents the base address of array named arr.

As arr is representing some address, so we can use dereference operator on arr. The answer is yes.

* arr → Value at arr [0] = 5
* arr + 1 → 5 + 1 = 6

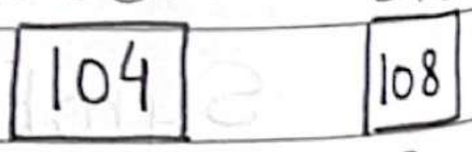
* (arr) + 1 → 6
* (arr + 1) → 7 } value at arr [1]
* (arr + 2) → 9 } value at arr [2]

Note → * (arr + x) = arr [x]
Also note that x [arr] will be same as arr [x]
x [arr] → * (x + arr)
We can get stuck over this point also

int arr [5];
Here arr is the pointer to the 1st location of the array i.e. arr [0].

Note → arr = arr + 2; → This will not work as we can't change value present in symbol table.
int * p = arr;
p = p + 2; → This will work & hence another use case of the pointer

Ex → int arr [4] = {5, 6, 7, 8};
int * p = arr;
int * q = arr + 1;



arr	104	&q	316
&arr	104	*q	6
arr [0]	5	*p + 1	6
&arr [0]	104	*p + 2	7
p	104	*q + 3	9
&p	208	* (q + 4)	Segmentation fault (gargage value)
*p	5		
q	108		

Pointer vs Array

① int brr[10];
cout << sizeof(brr); → 40 (10 × 4 = 40)
int * p = brr;
cout << sizeof(p); → 8 (Depends on system)

② arr = arr + 1; → This won't work
p = p + 1; → This will work

Can't change symbol table entry.

↳ Here entry in symbol table is not get modified.

arr is a constant pointer and hence we can't change its value.

Note → int arr[10]; arr → 104
arr + 1 → 104 + 1 × 4 = 108
arr + 3 → 104 + 3 × 4 = 116
arr + 5 → 104 + 5 × 4 = 124
↳ sizeof(int) → 4 bytes

Character array

char ch[10] = "Babbar";
char * c = ch;
cout << c; → Babbar

The reason is different implementation of cout for int* and char*.

ch → Babbar
&ch → Address of ch[0]

ch[0] → B
&c → Address of pointer
*c → B
c → Babbar

Ex→ char name[10] = "SherBano";
char *cptr = &name[0];

name	SherBano	cptr+2	erBano
&name	Address → 104	*cptr	S
*(name+3)	r	cptr+8	error
cptr	SherBano		
&cptr	Address → 216		
*(cptr+3)	r		

Understanding cptr+2 & cptr

2nd index
SherBano
↔ will be printed

0th index
SherBano
↔ will be printed

cptr+2 means start from 2nd index & print all character from 2nd index till the null character.

Special case
char ch = 'v';
char *ptr1 = &ch;
cout << ptr1; → This will print v & garbage value until we get the null character.

Important case

```
char name[10] = "Bhavya"
cout << name << endl;  → Bhavya
```

2-step process

→ First of all a temperory storage is there in which Bhavya is stored.

→ Then it is copied to name array storage.

```
char * c = "Bhavya"
cout << c << endl;  → Bhavya
```

2-step process

→ First of all a temperory storage is there in which Bhavya is stored.

→ Now c is pointer which is pointing to this this temperory storage & hence it is considered as a bad practice.

Pointers with function

```
int main () {
  int arr[10];
  sizeof (arr); → 40
  solve (arr);
}
```

```
solve (int arr[]){
  sizeof (arr); → 8
}
```

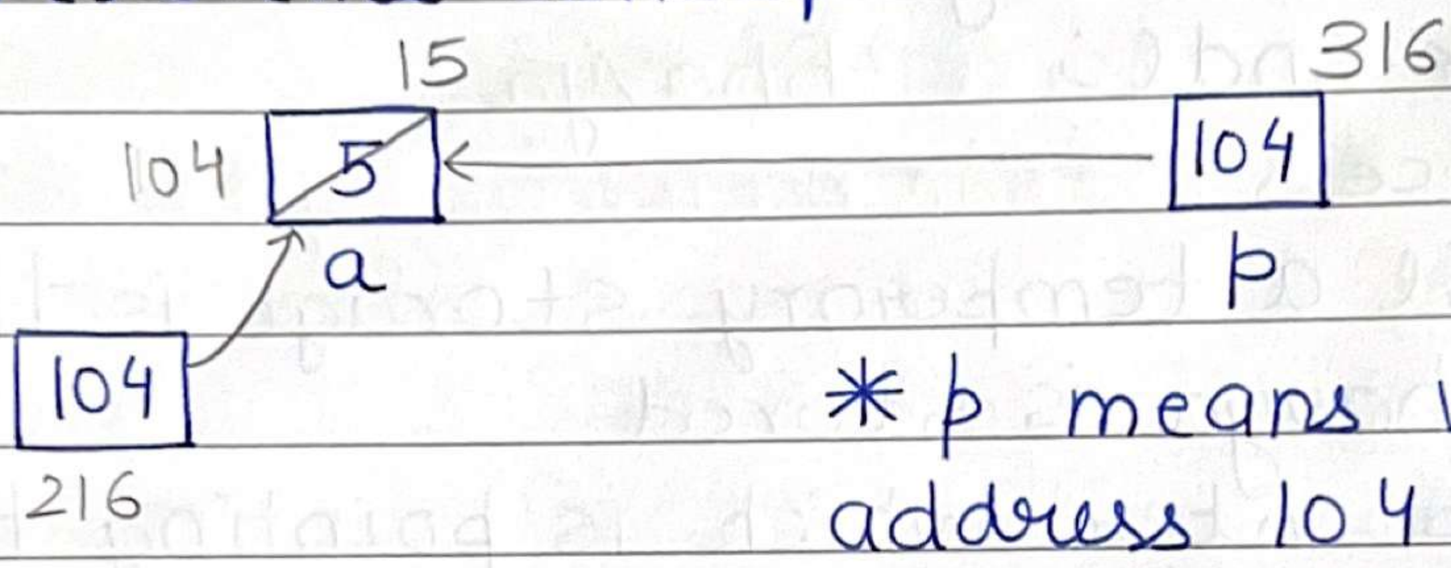
location of this will be diff

By default array is passed as the pointer & this is known as pass by reference. Hence sizeof (arr); will come out to be the size of the pointer. If we change anything in array in solve function, then those changes will be reflected in original array.

Ex→
```
main() {
    int a = 5;
    int *ptr = &a;
    solve (ptr);
}
```

```
solve (int *p){
    *p = *p + 10;
}
```

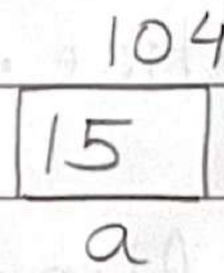
Here a will get modified to 15 as by reference concept is used.



*p means value at address 104 i.e. 5

*p = 5 + 10

*p = 15

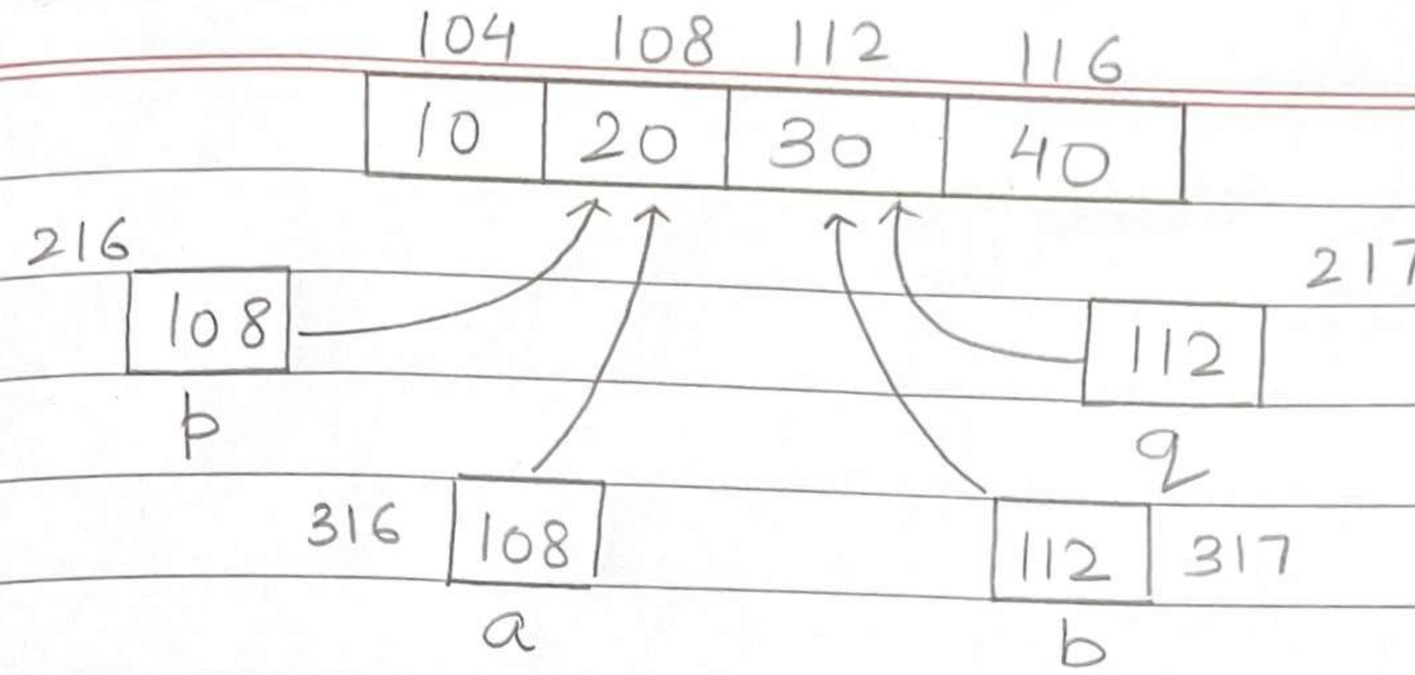


solve (ptr); // This is copying of ptr to p.

Ex→
```
main() {
    int arr[4] = {10, 20, 30, 40};
    int *p = &arr[1];
    int *q = &arr[2];
    update (p, q);
    //print entire array

}

update (int *a, int *b) {
        *a = 100;
        *b = 200;
}
```



* a ⇒ value at 108

* a = 100 ;

10	100	30	40

* b ⇒ value at 112

* b = 200 ;

10	100	200	40

o/p → 10 100 200 40