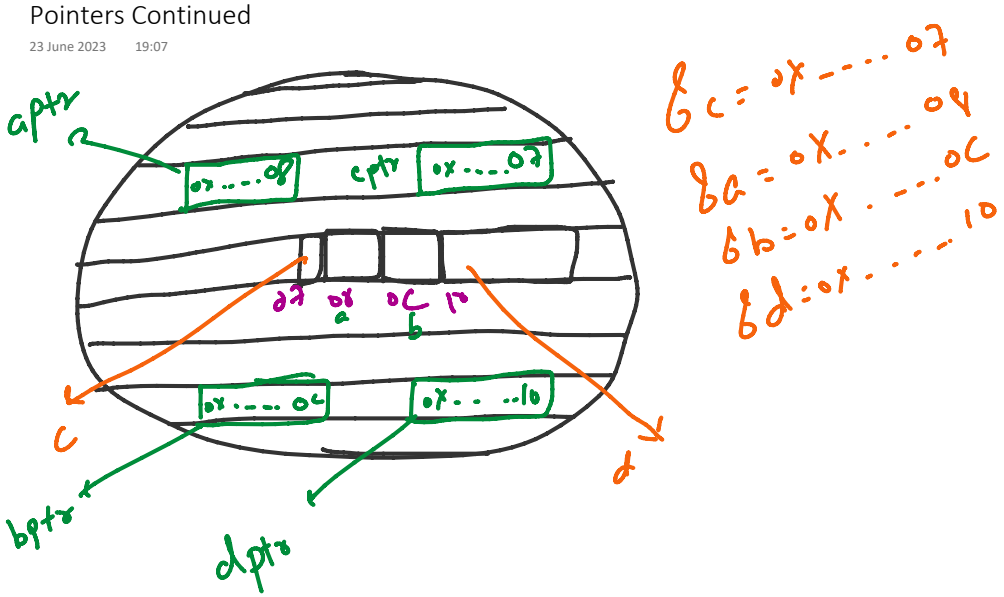
Pointers Continued

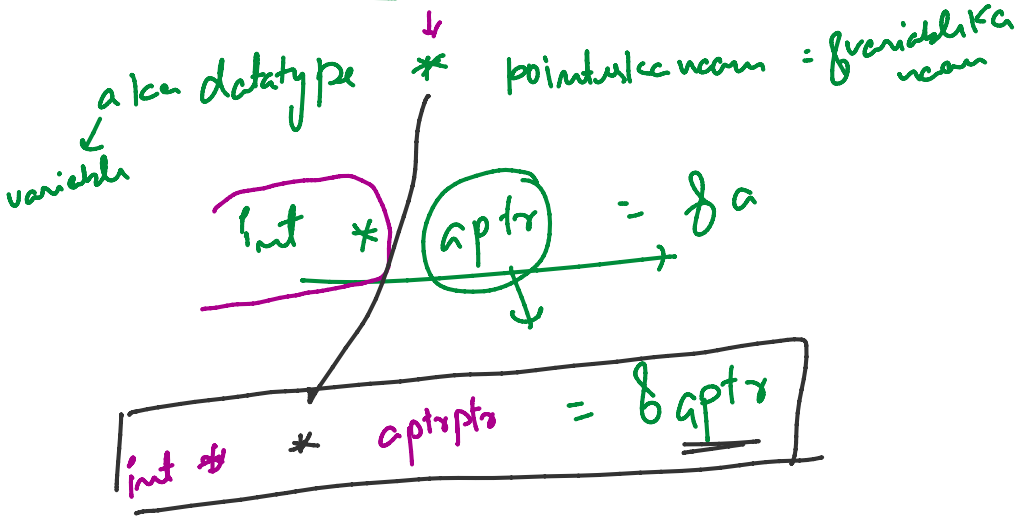
23 June 2023 19:07



---

int a = 10;

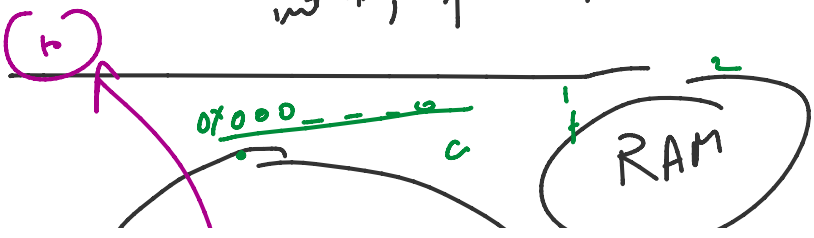
&a.

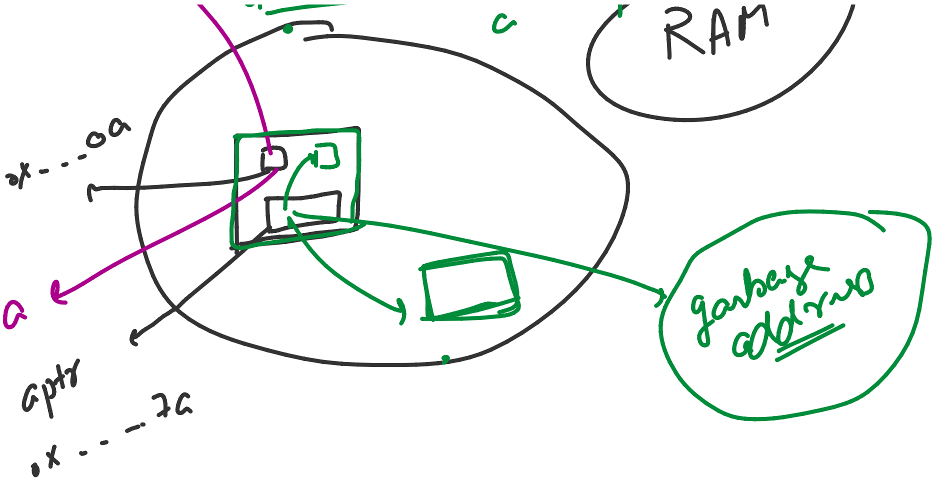


Fooling the compiler

(int*) &ch → value

int *, float *,





int a = 10;

int * aptr = &a;

dereference op also

I can access the value stored inside 'a' using 'aptr'.

cout<< aptr << endl → &a.

Dereference operator se.

*

when you are declaring/initializing a pointer variable '*' is used as pointer datatype

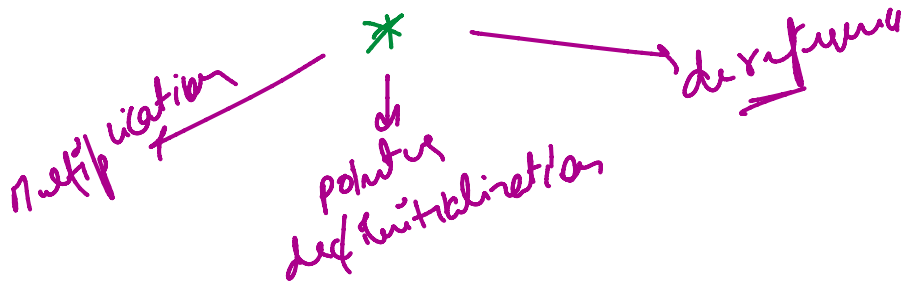
int * aptr = &a;

datatype

After that whenever you will use * with that pointer, it will have only 2 meanings.

① multiplication

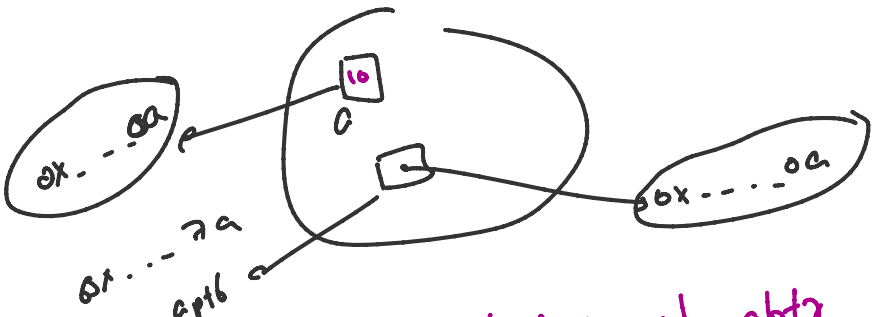
① multiplication
② Dereferencing



b

Syntax for Dereference :-

meaning of Dereference :- It means accessing the value stored inside the address it is storing / holding / pointing towards.



Dereference of aptr means accessing the value stored inside address (0x----0a) it is storing

so Dereference of aptr will give (10).

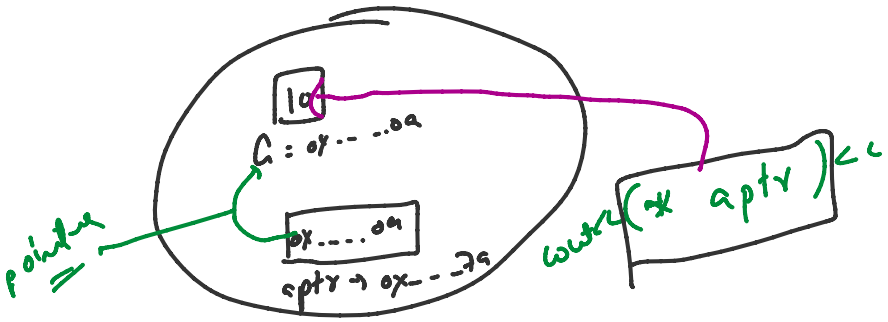
syntax

cout << aptr << endl; // b c

... endl; // a

cout << aptr << endl; // a

cout << *aptr << endl; // a



---

int a=10;
int *aptr;

-3 → -6

cout << (*aptr) * 2 << endl;

exp. o/p = 20

1st case
aptr is storing address inside of program memory
↓
o/p → our prog. will get executed.

2nd case
aptr is storing address of memory outside prog. memory
Error → Seg. fault.

If you don't have any address to give to pointers, then assign it with NULL value.

int* aptr = NULL; // ox0.....0
= 0;

NULL pointers

↳ And dereferencing a NULL pointer will always give seg. fault.

Arrays & pointers, pointer arithmetic

Pointers Arithmetic

```
int a = 20, b = 30;
cout << a+b << endl;
```

meaningful things to do:

```
int *aptr = &a, *bptr = &b;
```

[aptr + bptr] → This is no meaning in adding two hexadecimal nos. (addresses)

→ address (hexadecimal no.) + address (hexadecimal no.)

But there is significance in adding integral values to addresses.

When we add integers to a pointer

```
int arr[7] = {10, 20, 30, 40};
```

→ 4 bytes

```
cout << (arr+1)
```

+1 → go to next

address of arr[0] '10'

+1 means go to next bucket available

int double → pointervar + n

o/p will be + [n x sizeof (variable whose pointer we are updating)]

pointervar.

0x---80

+ (n x 4)
+ (n x 8)
+ (n x 1)

---

arr[3]

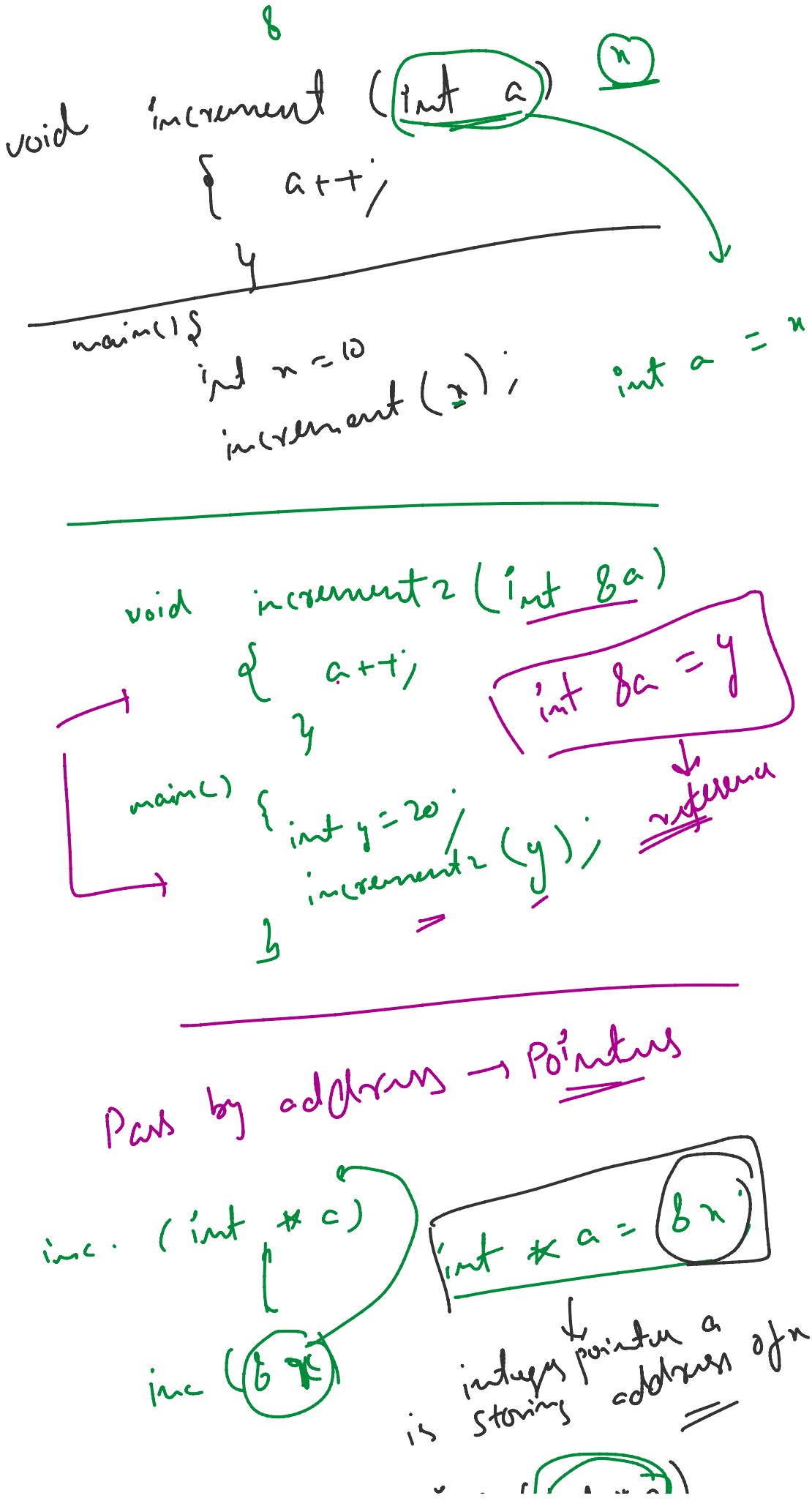
→ *(arr+3)

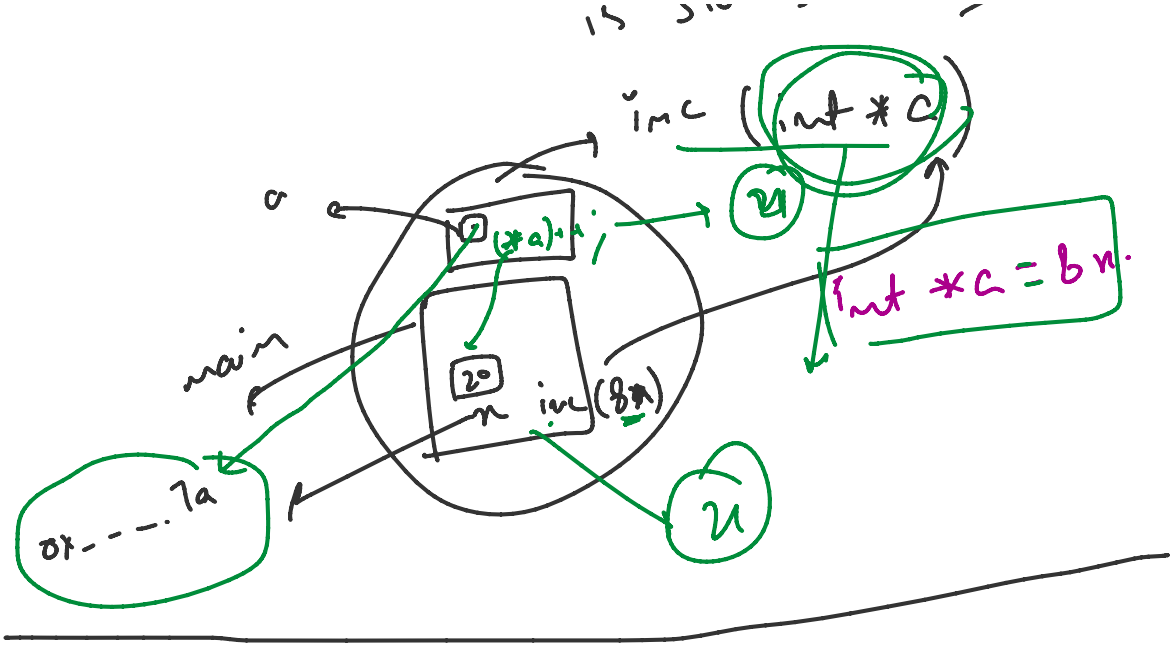
array name is nothing but an internal pointer inside array.

(ap 10 x3) → meaningless

↓ hexadecimal

Pointers And functions


void increment (int a)
{ a++;
}
main()
int x = 10
increment (x);
int a = x
void increment2 (int &a)
{ a++;
}
main()
{ int y = 20;
increment2 (y);
}
int &a = y
reference
Pass by address → Pointers
inc. (int *c)
{
inc (&x)
int *a = &x
integer pointer a is storing address of x.



int arr[] = int *arr

in function definition

*(arr+i)

arr[i]