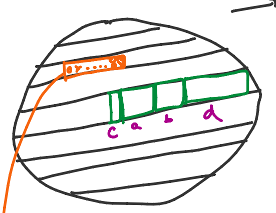
Pointers contd.

24 June 2023 20:12



→ Program memory

&a = 0x 98
&b = 0x 9c
&c = 0x 87
&d = 0x 90

aptr : 8 bytes → address of aptr - 0x .... 60.

again a pointer only

datatype * ptrname

int * * aptrptr = &aptr

---

int a;
char ch;

cout << a << → number integer

cout << ch << endl → character

---

int * aptr;

cout << aptr << endl.
→ some garbage value

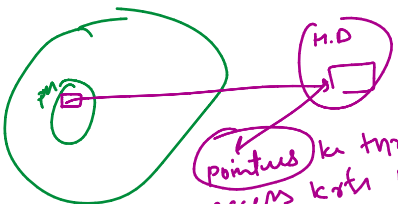
Some hexadecimal no. or some address

Dereferencing a pointer

int a = 10;
int *aptr = &a;

Through 'aptr', I can access the value of a.

→ // pass by address in functions.
similar to pass by reference

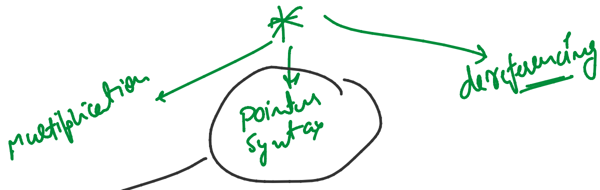


pointers ke through access karte ho.

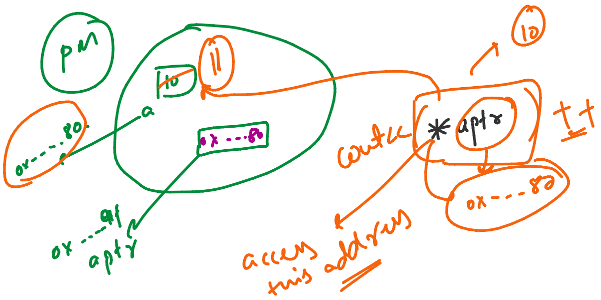
Accessing values of variables through their pointers.

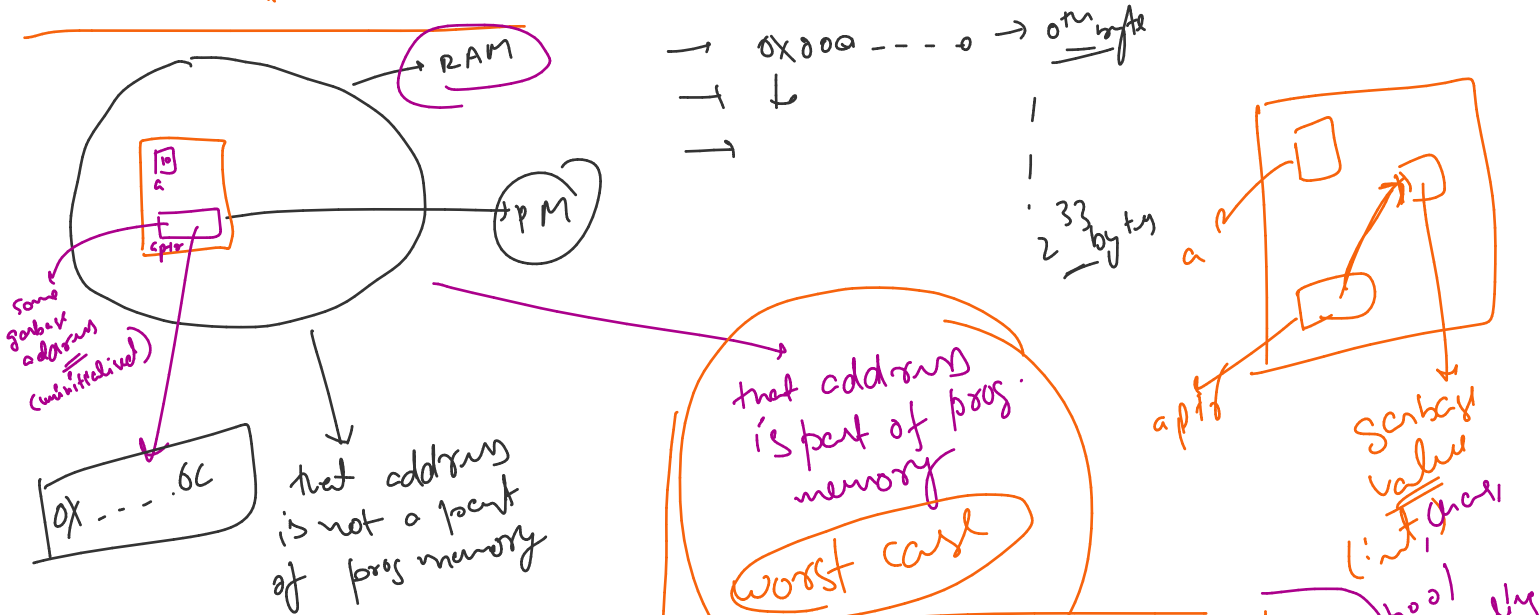
```
int a=10;
int *aptr = &a;

cout<< a <<      // 10.
cout<<(*aptr) << //10
```

Using * with pointer name, we can dereference (access the value of) variable a.



whenever you are declaring a pointer variable then only * is used for declaration (then only * means pointer variable) other than that, it has only 2 meanings → (1) multiplication (2) Dereferencing





... de-reference both these types ... depending

Now if we try to de-reference both these types (depends on type of pointers) what will happen.

segmentation fault	program will execute

```
int a = 10;
int * aptr;
cout << (*aptr) * 2 << endl;
```

(-4)

20 → expected

-40 → got

So to prevent dangerous / unwanted outcomes, we can initialize pointers with either some variable's address or with NULL;

→ 0th address i.e. → 0x000 _ _ _ .00

```
int * aptr = NULL;
              0;
```

} both will work.

whenever you will dereference a NULL pointer it will always give seg. fault.

Pointers & Arrays / Pointer Arithmetic

array's name is basically the address of 1st memory location inside that array. So it behave like a pointer. & Arrays & pointers are very similar.

So before moving on any further let's discuss pointer arithmetic.

int a = 10, b = 20;   a + b = 30 → int

int *aptr, *bptr;
aptr → address of a, bptr → &b

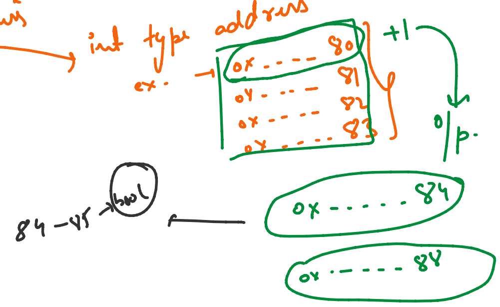
aptr + bptr = ? → Hexcode
Hexadecimal Hexadecimal

So adding of two addresses doesn't make any sense. Similarly multiplication & division also doesn't make any sense.

Arithmetic of integers with pointers

```
int * aptr = &a;        // 10
cout << *aptr + 1 << endl;   // 11      (highest)
cout << (aptr + 1) << endl;
aptr ++ ;   //  cout ++aptr ;
cout << aptr
```

address → int type address



general syntax.

pointer name ± x → integer.

general syntax.

pointername (address) ± x → integer

↓ x × (sizeof (variable/datatype whose pointers we are updating)

int *

0x 8- - - .80 ± x.

o/p 0x - - - .80 ± 4x.

bool *

0x - - - 80 ± x.

arr[2] = *(arr + 2)

dereference

arr + 2x4 = (arr + 8) bytes → 8

---

pointers & functions

1. call by value → changes will not reflect
2. call by reference → will reflect
3. call by address → will reflect

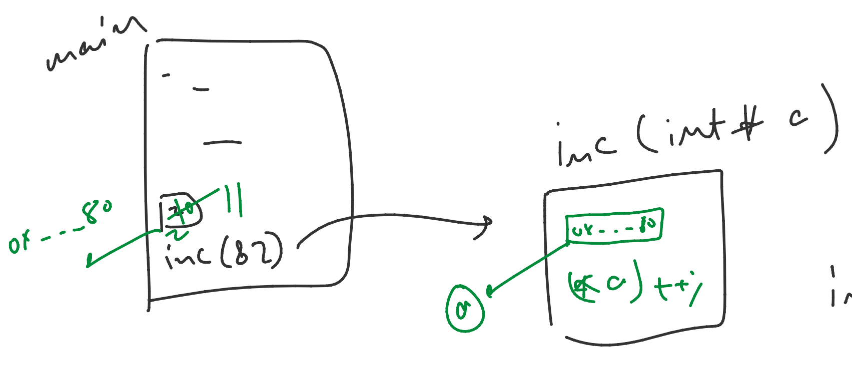
Syntax

```
void inc (int *b) {    // variable
  // inc by 1, the value we will provide.
  (*b)++;
}
int a = 10;
inc (&a)    // value
cout << a → 11
```

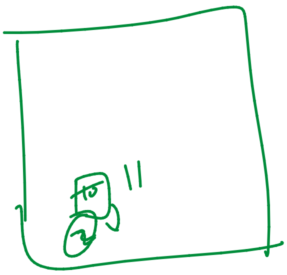
variable = value

_____ = &a;

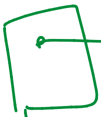
main



int * a = &2;



inc2(int &a)



{ a++; }

there is no bucket formed in call by reference.