Pointers - 1

`int a = 5;` In memory, 

a Integer block stored 5 in it, and it's name is a.

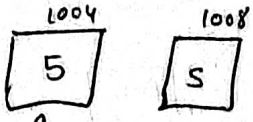
But we can't give name to a mem loc. The only way of accessing is to use address.

Symbol Table
a → 104

In symbol table variable's are mapped with their addresses.

⊙ Does `a = 5` , `b = 5` points to same memory block?

`int a = 5;`
`int b = 5;`



Symbol Table
a → 1004
b → 1008

We don't decide the addresses of these variables. This memory mgmt is done by OS.

⊙ Can we find out the address of a variable?

⇒ Yes, by address of (&) operator.

```
int a = 5;
cout << a;          //o/p → 5
cout << &a;         o/p → 0xffabcdef1234   ← This is the address of a.
int b = 5;                  ← some hexa-decimal value.
cout << &b;         o/p → 0x45321---   ← This address of b will be different from address of a.
```

pointer → A type of ~~datatype~~ variable that store address.

is a pointer to integer data.

`int * p = &a;` → p is a pointer to integer data.

- `int` : datatype
- `*` : dereference operator / syntax
- `p` : variable name.
- `&a` : address of operator.

is a pointer to char data.

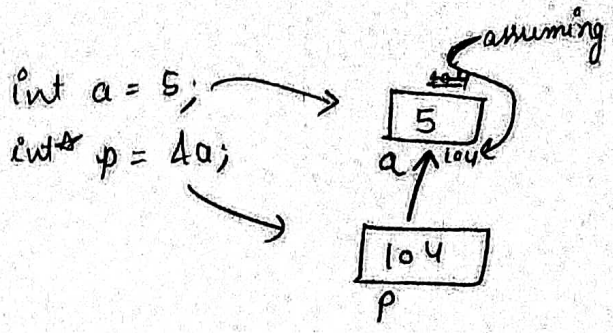
`char * ptr = &ch;` → ptr is a pointer to char data.

```
int a = 5; //creation
int* ptr = &a;
// access the value ptr is pointing to
cout << *ptr;
```

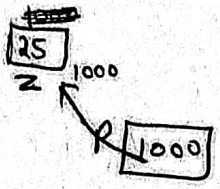
ptr is a pointer of a which contains integer data.

*ptr → dereference operator.

O/P → 5




```
int z = 25;
int* k = &25;
```

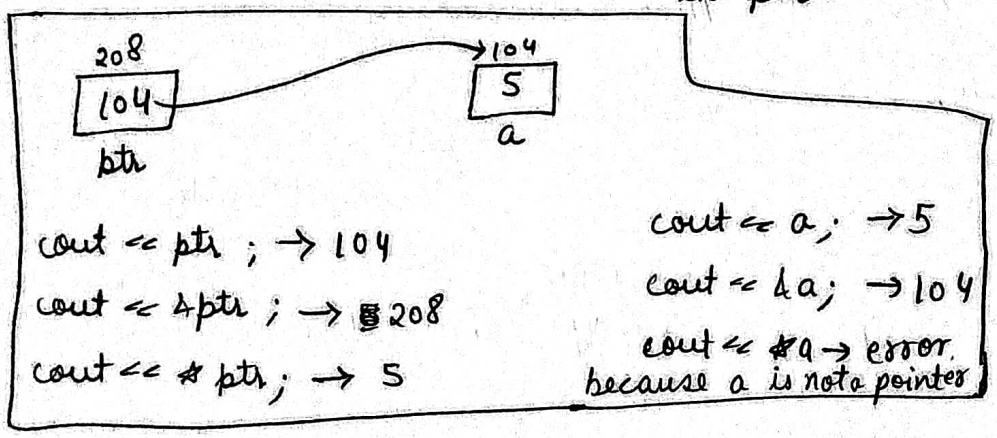


pointer → In C++, pointer is a variable that stores address of another variable.

ptr → address
ptr → value

```
cout << ptr;     //O/P → 104   (value stored at ptr)
cout << *ptr;    //O/P → 5
```

↳ value at location stored in ptr.



Find the size of these pointers :→

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

char c = 'b';
char* ch = &c;

double d = 1.03;
double* dtr = &d;
```

All three pointer's size will be same, because pointer stores address of variables ~~all all~~ and address size of any type of data will be same. (no matter their datatype is same or not).

```
cout << sizeof(p) << sizeof(ch) << sizeof(dtr);   → o/p → 8 8 8
cout << sizeof(a) << sizeof(c) << sizeof(d);      o/p → 4 1 8
```

Here the size of pointer is 8.
~~8~~ Architecture dependent.

Find out?

- What do you mean by 64 system?
- Why pointer's size is 8?

- **Why do we need pointers?**
  - ↳ Dynamic memory allocation
  - ↳ Memory mgmt
  - ↳ To access hardware.
  - ↳ Pointers arithmetic, handling NULL pointers
  - ↳ To pass a function as an argument inside another function.

- **Declaring a pointer →**

```
int * ptr;   // declaration.
```

In this case a garbage value is assigned to the ptr.

```
cout << *ptr;   ← And here we are trying to access that memory.
```

We get **Segmentation fault**. (means we trying to access that memory which either don't exist or out of the alloted memory space ~~of~~ program.

This is a **BAD PRACTISE.**

Now how to correct this?

↳ Null pointer.

```
int * ptr = 0;   ← This is how null pointer is created.
```

means that pointer is pointing to nothing.

Error - segmentation fault.

But here we can check this before running.

```
if (ptr == 0)
    cout << " ptr is a NULL pointer ";
```

Other way to create NULL pointer →

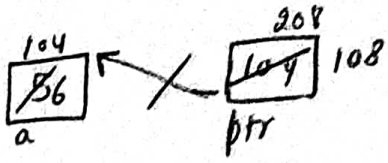
```
int* ptr = NULL;
```

New/modern way to create NULL pointer →

```
int* ptr = nullptr;
```

①

→
```
int a = 5;
int* ptr = &a;
a = a+1;
ptr = ptr+1;
```

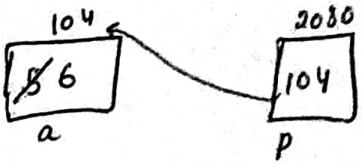


104
ptr + 1 = 108

→
```
int a = 5;
int* p = &a;
```

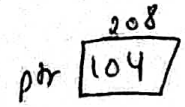
*p = *p + 1

↓
value at address stored in p

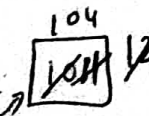


*p = 5
*p = *p + 1
5 + 1 = 6

→



→ a → 10
→ &a → 104
→ ptr → 104
→ *ptr → 10
→ &ptr → 208
→ *ptr * 2 → 20
→ (*ptr)++ → 10



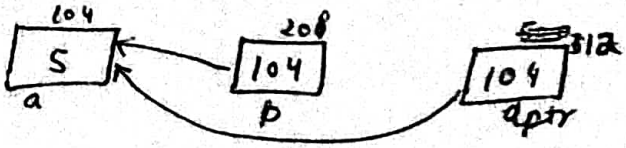
→ ++(*ptr) → 12
→ a = a+1 → 11+1=13
→ *ptr = *ptr + 2 → 15
→ *ptr = *ptr * 2 → 30
→ *ptr = (*ptr/2) → 15

⊙ Can a pointer copy another pointer?

Yes,

```
int * q = p;
```
← p is also a pointer.

- ```int a = 5;
  int* p = a;``` ⟶ gives error. so can't assign pointer a with a value, we have to give address.
- ```int a = 5;
  int* p = &a;
  int* dptr = ptr;``` ⟶ copy a pointer into another.



*p → 5

*dptr → 5

a → 5

&a 104

p → 104

&p → 208

*p → 5

q → 104

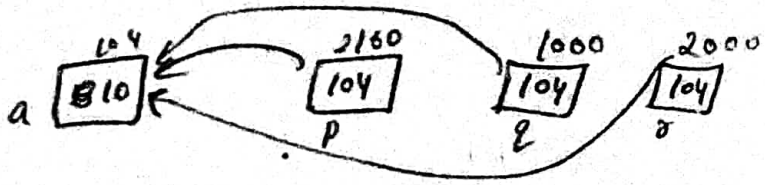
&q → 312

*q → 5

(*p/2) → 2 ⟶ 5/2 = 2

*q/2 → 2 ⟶ 5/2 → 2

⊙
```
int a = 10;
int*p = &a;
int* q = p;
int* r = q;
cout << a << &a << p << &p << *p << q << &q << *q <<
    r << &r << *r << (*p + *q + *r) << (*p)*2 + (*r)*3
<< (*p/2) - (*q/2) << endl;
```



a → 10

&a → 104

p → 104

&p → 2160

*p → 10

q → 104

&q → 1000

*q → 10

r → 104

&r → 2000

*r → 10

(*p + *q + *r) ⟶ 10+10+10 = 30

(*p)*2 + (*r)*3 → 10×2 + 10×3 = 50

(*p/2) - (*q/2) → 10/2 - 10/2 = 0