Pointers

(11) *(&y) ⟶ 10, ~~10~~, 101, 100

Q. How to swap??

- using a function

Reason: function is call-by-value. : value is copied, original variable is not used

① Get the address of the variables (to swap)

- address-of operator & (prefix)

② How to store an address.

- pointers (a different type of variable)

- its type is determined by the pointee.
  (the value it is pointing to).

③ Getting the value (bucket) pointed by a pointer.

- dereference operator, i.e. (*) prefix of an address

* → multiplication
* → pointer declaration
* → dereference (pointer variable / address) — legal

```
int x = 4;
int *xptr = &x;
```

pointer storing address of int x;

```
cout << *xptr;
```

⇒ ④

Pointer is used to store the location of other variables.

- it helps in passing values by reference to functions.

---

char c;

cout << &ch; → unexpected output.

Pointer type-casting.

Function is called — by — value: values are copied, original variables are not passed.
⇒ changes remain within function

reference
we are refering to the original variable and not its copy. ⇒ changes are reflected in call-scope.

copies (value)

```
f(int a, int b) {
}
```

any changes here remain here

```
main {
  f(a, b);
}
```

call scope

call by reference → ① Using pointers → 3 step process

```
f(int *a, int *b) {

}
main {
  f(&a, &b);
}
```

) have passed the address / reference to original variables

any changes in function (f) will be reflected in main (call scope).

② Using reference variables

→ Variables which simply point to the bucket of another variable.

→ this feature doesn't exist C. :(

---

functions

call by

- value
- reference (original variable is used)
  - using pointers
  - using reference variables

→ {using addresses (pointers) itself are passed called by value.}

Q. Can we return pointers from function??

Yes ✓

No

return_type: int*
void*
char*
??

Q. Should we return pointer from functions?

- [ ] Yes
- [ ] No
- [x] Depends

int

return_type of a function can be : { int *
void *
char *
double *
float
- - -

use case

Linked List

stack } LL
queue

Q. Passing arrays in functions.

```
func (int arr[], ....){

}
```

[] → diff between int and int array

```
main (){
    int a[n];
    func(a, ---- );
}
```

Observation: Arrays are always passed by reference.

Pointer - Array duality:

there is an intricate link b/w pointers & arrays

An array is actually a pointer that points to the first element of the array (arr[0])

- as the array is a pointer variable, we may dereference it, and perform pointer arithmetic.

32 Th

Array name is just the alias of the address of first element.

there is no seperate storage for it.

- it behaves as a pointer, but isn't a pointer.

```
int arr[10];

cout << arr;
cout << &arr[0];
```
} ⇒ same

Pointer Arithmetic

- addition, multiplication, division of two addresses is senseless

- Addition of a constant integer value is equivalent as

int* ptr + x ⇒ the cell/byte/block which is 5 * sizeof(int) ahead of ptr.

int * ptr = 3760;

int * next_ptr = ptr + 2; 3768

ptr + 2 * (sizeof(int))

3760 + 2x(4)

3760 + 8 = 3768

long int * ptr = 501; 501 + 1 * (sizeof(long int))

501 + 1 x (8) = 509

cout << (ptr + 1); 509

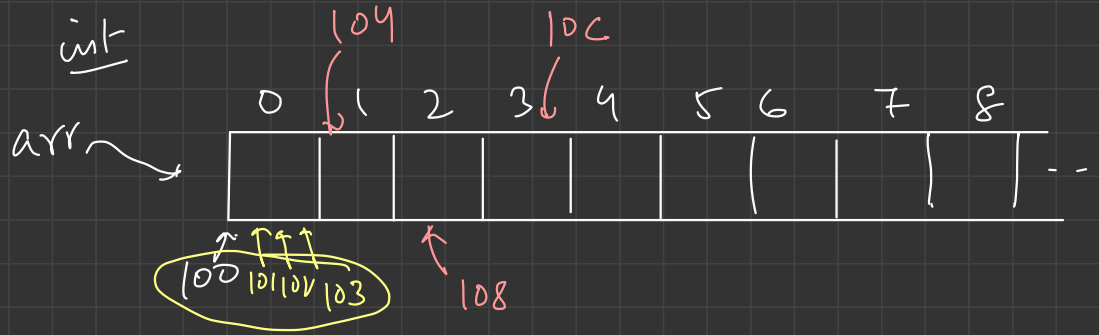
— Subtraction is similar

⇒ Subtracting two addresses ⇒ no. of blocks/elements between them

7608

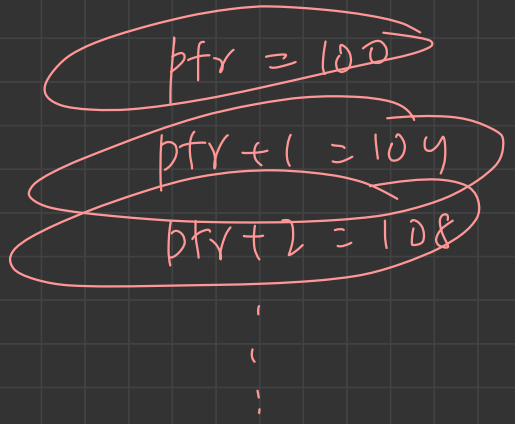
7600

$$\frac{7608 - 7600}{\text{sizeof(int)}} = \frac{8}{4} = 2$$



arr[0] → 100) size 4 bytes

arr[1] → 104



Can we iterate the array using pointer arithmetic ?!

Arrays / Pointers

sizeof : array → memory used by all the elements

pointer → memory used by pointer variable (8 byte)

& op → &arr → alias for &arr[0]

&pointer → address of the variable

int a[10];

int ptr;

ptr = a; → just a pointer pointing to the address of first element

a = ptr; ?! → a is an array, and this is illegal.

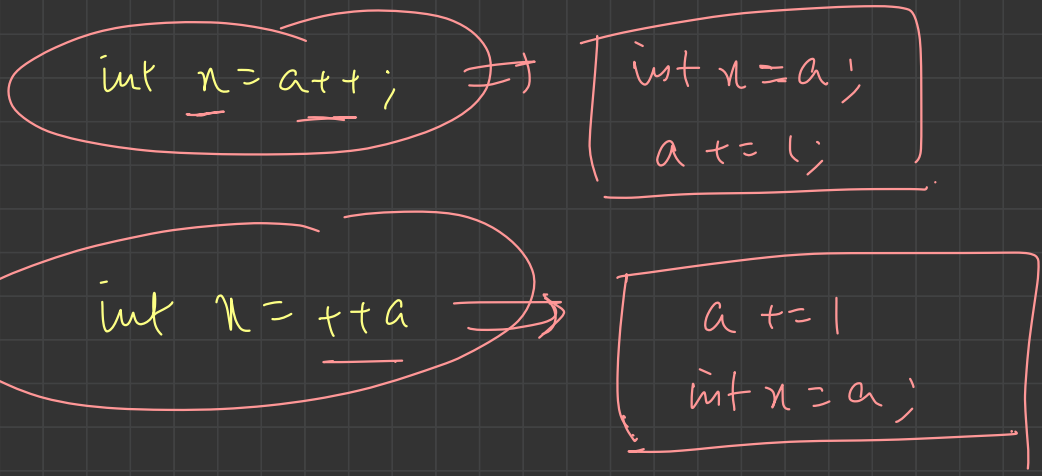
ptr++; ✓ → ptr = 80 : ptr++ ⇒ 84

a++; XX increment / decrement not allowed!

a → 70 [a++ X]

Post increment Pre increment

a++; ++a; increment



→ Operator Precedence / Associativity → reading assignment

→ Binary search
character arrays / Strings
2D / multi-dimensional arrays
} next class

Revise all (pointer, array, function).
sort