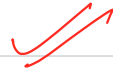



Pointers & References ✓

Q You have a sorted array (arranged in asc order)
You are also given a number S . You need to
find if there exists a pair that sums up to S .

1 2 3 4 5 6
↳ 9



$N \leq 10^5$
length of the array

$S = 8$

$N C_2$

Brute force

We can try to find all possible pairs & check if any one of them sums upto S .

$$nC_2 \approx \underline{\underline{n^2}} \rightarrow 10^{10} \times$$



$a[i]$



sorted

for all pairs
with same i
& $j = \neq i$

$i=0$
 $j = n-1$

$$\underline{\underline{a[i] + a[j] \neq S}}$$
$$> S$$
$$< S$$

while ($i \leq j$) {

→ 2 pointers
[]

if ($a[i] + a[j] > S$) {

// discard everything to right of j

} else if ($a[i] + a[j] < S$) {

// discard everything to the left of i

} else {

// we found a pair

}

}

Pointers

↳ Datalyfes → store memory address → pointers

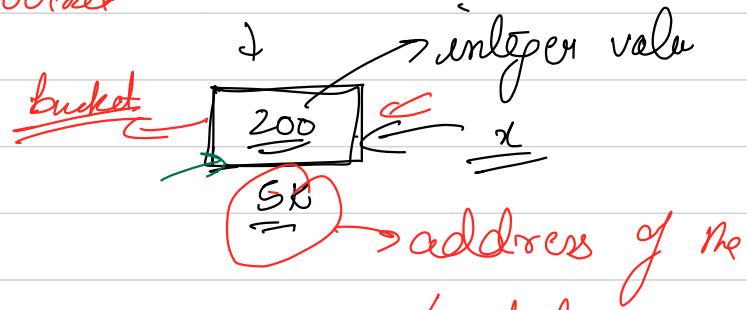
So declaration of a pointer depends on what you are going to point.

<data type> *name

name doesnot include * in
pointer

x = 10 → up doo

↪ int x = 200;
 ↗ name of the bucket



cout << x; → 200

↪ int * ptr = 5K;
 ↗ bucket → 5K → ptr

→ operator (&) → unary operator
↳ returns address of a variable.

und- ** double ptr = &ptr;

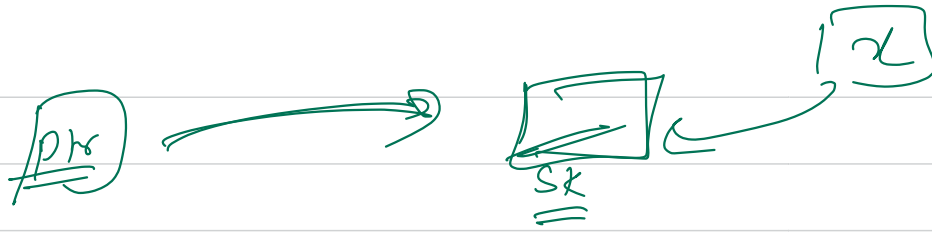
**ptr = 20

When we stored address of int value we use
 int^* .

When we store address of a pointer pointer
we use int^{**} .

$*ptr = *(ptr)$

$**ptr = **(*ptr)$



*ptr = *(ptr)

*ptr = 10 → update the variable

**double - ptr = 12348

Pointer Supports arithmetic

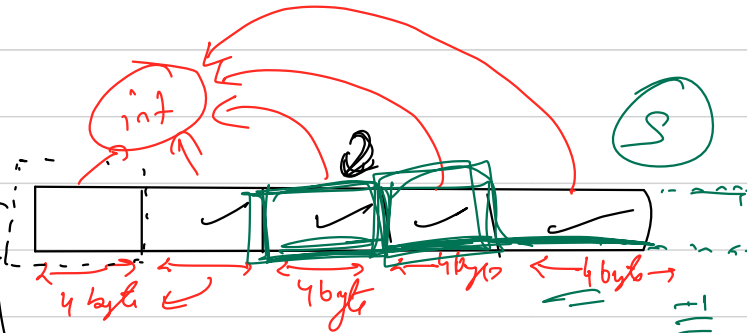
A pointer can be incremented or decremented

Any value can be added or subtracted from a

pointer

`int arr[100];`

first
under
address
(base address
of array)



base address of an
array can be
accessed by name
of the array

(arr + 2)

(arr + 3)

*** ptr \rightarrow 7/c

10K
 \rightarrow 100

* (* (* (ptr)))

* (* (* 7K))

* (* 10K)

(* 11K)

100

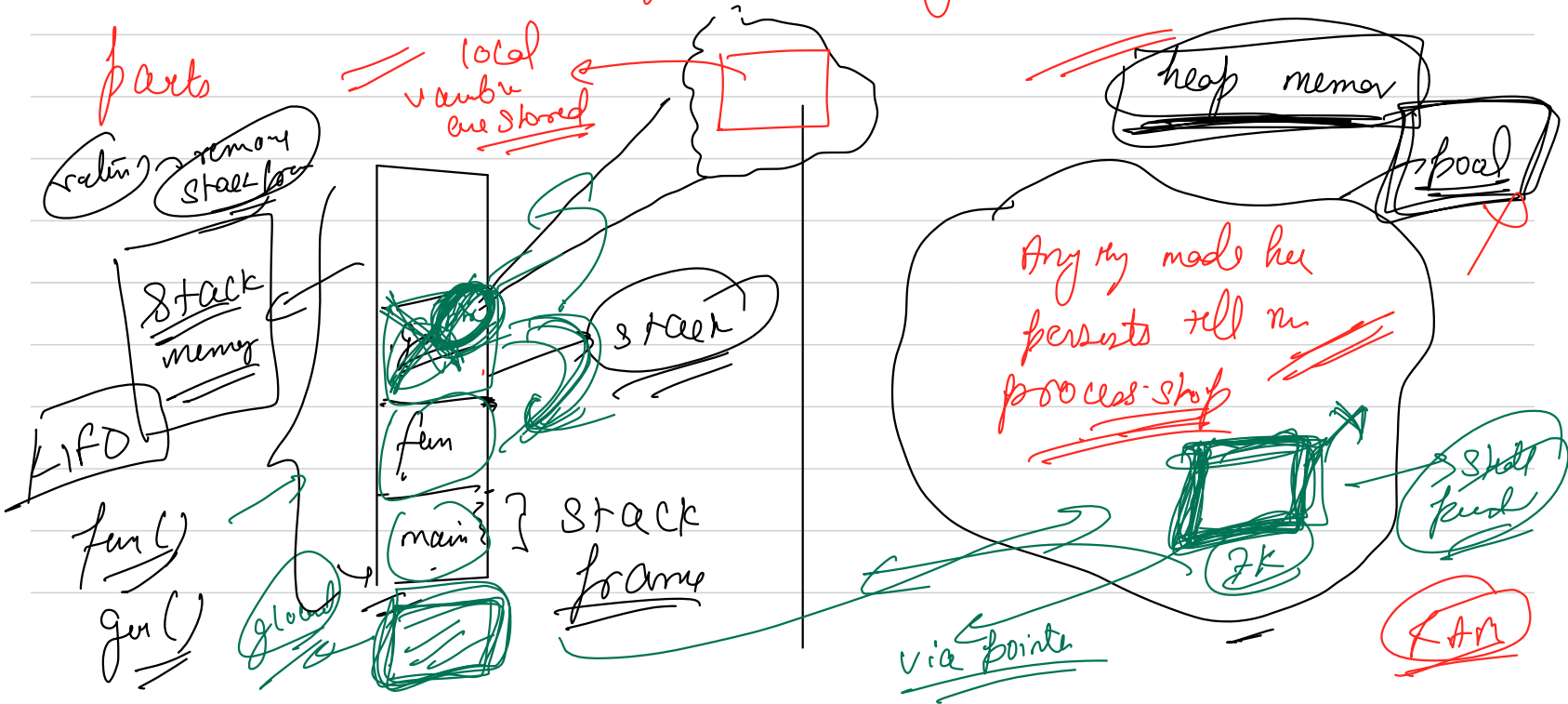
→

Memory

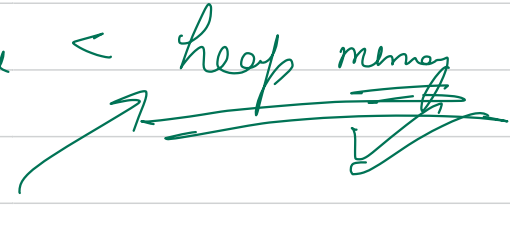
→ $\text{unit of } 10$

When a process is running the memory allocated has 2

parts

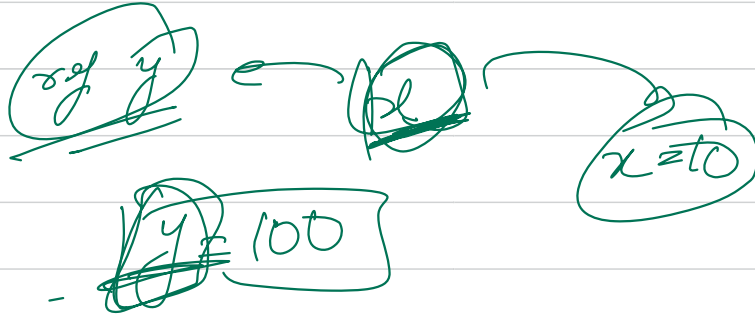


Memory given by stack frame < heap memory
more storage area



References

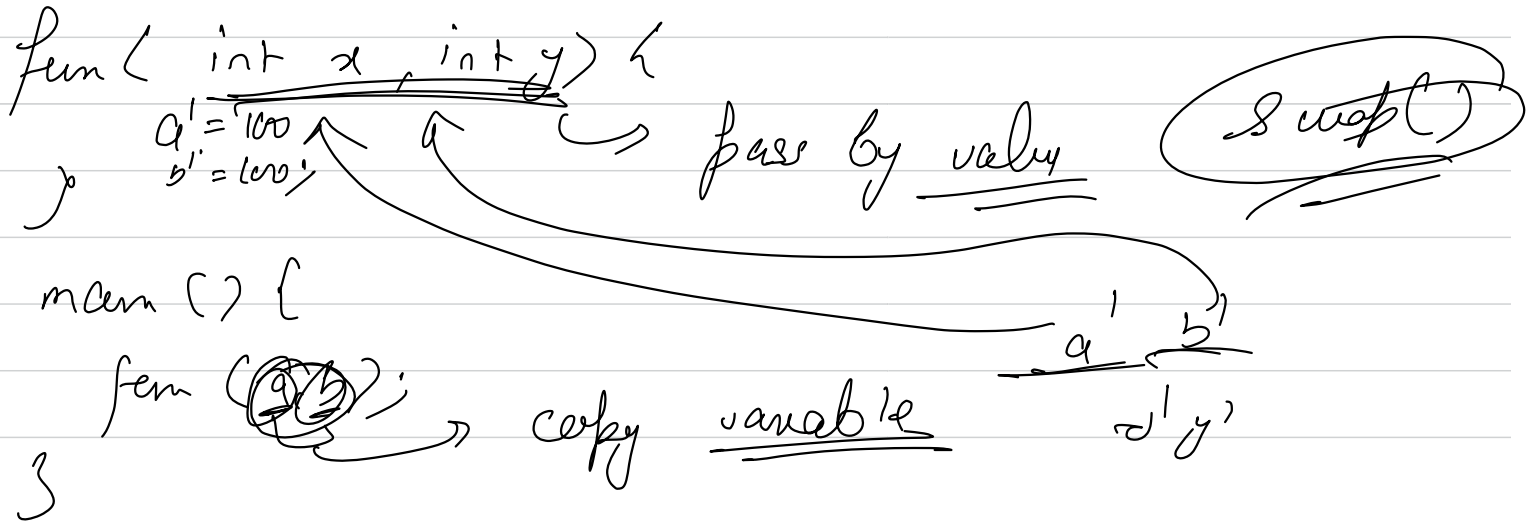
When a variable is declared reference w.r.t another variable then it becomes an alternative name to the memory bucket of the argument variable.



Passing arguments to function

① Pass by value / Pass by copy

② Pass by reference / Pass by original address



fun (int & x, int & y) {

x = 1000

y = 2000;

}

main() {

fun(1, 2)

}

Pass by reference

x & y can update the

original a, b

no copy here, original

bucket is passed

advantages → No copy is done

copy can be expensive

disadvantages → can hampers original value

↳ new

↳ delete

int *ptr = new int;

go in the heap and create
a bucket for `ptr`

whenever a function is called a new stack frame

is added in stack

Sort (arr, arr + n);

↙
base address
of the
array

↘ (last index + 1) address
of array

If we have address of a bucket how can we get the value of bucket??

Dereferencing operator \rightarrow It is a unary operator applied on a pointer variable which gives the value stored at that memory loc pointed by the pointer $(*)$

