We will start at 8:33

#Agenda → Basics of memory management

Bases of pointers & references

Differences btw other data types & pointer & references

Types of pointers

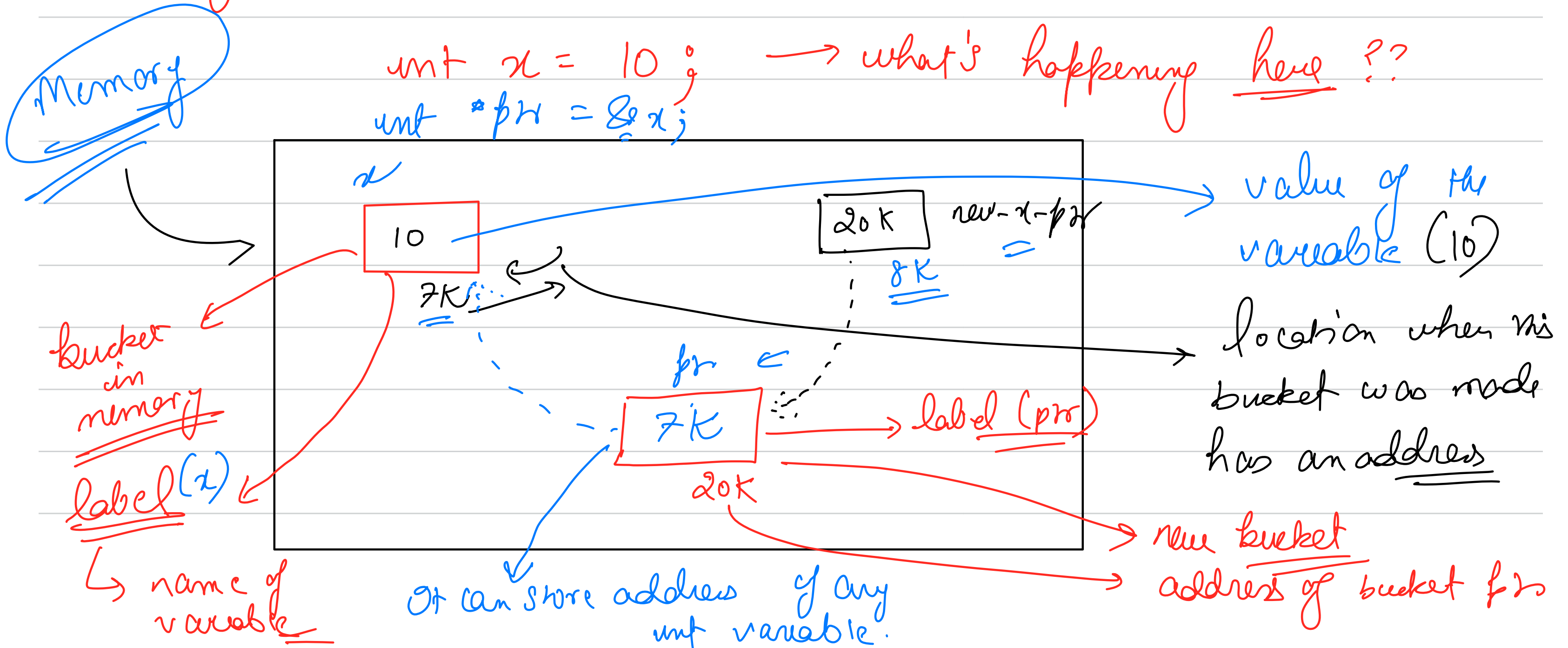
Operators & arithmetics of pointers

function pointer, smart pointer, new/delete

What are pointers ??

Pointer is a variable whose value is any address from memory

int x = 10; → what's happening here ??
int *ptr = &x;



Memory
x
10
7K
20K
new-x-ptr
8K
ptr
7K
20K
value of the variable (10)
location where this bucket was made has an address
bucket in memory
label(x)
↳ name of variable
label (ptr)
new bucket
address of bucket ptr
It can store address of any int variable.

In normal variable we write

<datatype> <name> = <value>;

So when we initialise a pointer, the declaration depends on the type of variable whose address is stored in pointer.

```
int x = 10; // normal integer
```

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

<datatype> * <name> = address

* → during declaration is defines that variable is a pointer.

```
char *ptr;
```

Q ⇒ How to access address of a variable ??

→ using "&" operator (ampersand)

→ unary operator.

→ It returns the address of any variable

```
int x = 10;

int *ptr = &x;
```

Q- How to access anything stored at an address using the address ??

→ 1) Store the address in a pointer.

2) Use dereferencing operator (*) to get the value.

```
cout << *ptr;
```

↓ dereferencing operation

What is dereferencing?

→ It is the process of getting the value stored at an address using that address.

```
int a =10;
cout<<a;     <10
cout << (a+1);   <11
```

```
int x = 10;
float y = 1.2;
char z = 'b';

int * xptr = &x;
float * yptr = &y;

char * zptr = &z;
```

Q-> What to do if we want to store address of xptr?

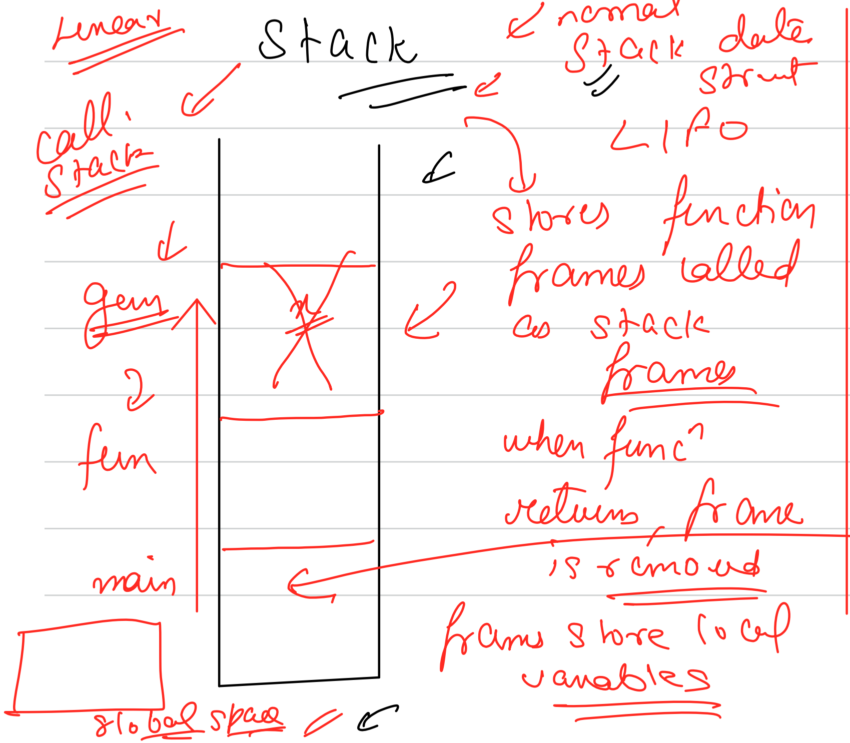
```
<datatype> * <name> = &<variable_name>
int ** new_x_ptr = &xptr;
```

Qn what is the size of int in C++?

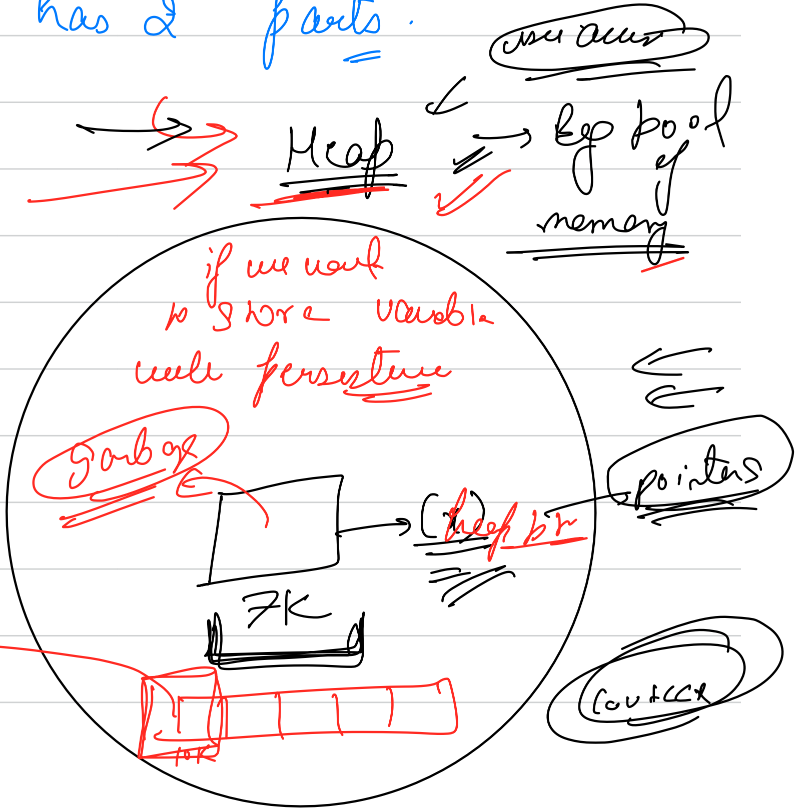
a) 1 byte
b) 2 bytes
c) 4 bytes
d) None ✓ → depends on environment.

sizeof(x);

when a C++ program is run, a memory from RAM is allocated.
Now that memory majorly has 2 parts.



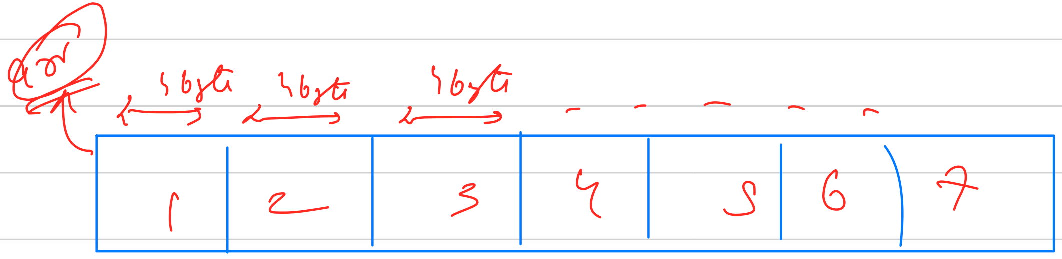
Linear
Stack
normal stack data struct
LIFO
call stack
gen
fun
main
stores function frames called as stack frames
when func returns, frame is removed
frames store local variables
global space



$$\begin{tikzcd}
\text{int } x = 10; \arrow[r] & \text{Stack}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{operator} \arrow[dl] \arrow[dr] & & \\
\underline{\text{new}} \arrow[dr] & & \underline{\text{delete}} \arrow[r] & \text{free}() \\
 & \underline{\underline{\text{malloc}()}} & &
\end{tikzcd}$$

new operator → makes a new memory space in heap & returns the address.



name of array stores the base address of array

cout << arr[0]; // 1

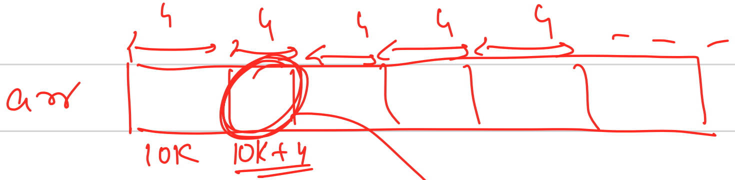
cout << arr; // 25k address

Points

int arr[6];

Pointer support arithmetics

1) increment (++)
2) decrement (--)
3) addition (+=) (+)
4) subtract (-=) (-)



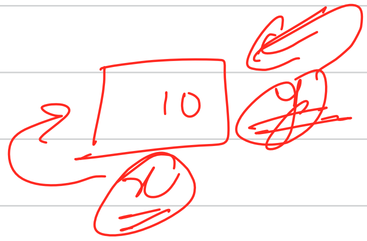
cout<<arr → addr → 10K

cout<<arr+1 // print address of index 1

References → A reference variable, acts as an alternate label or alias, for an original variable.

Q→ How to make a reference?

we use & operator



```
int& y = x;

cout << y << x;
```

Q:- Can references refer to invalid location in C++??

Yes

```
int *ptr;   // garbage

int& ref = *ptr;
```

$\underbrace{\text{int\& ref}}\ = \underbrace{\text{*ptr}}_{\text{dereference}};$ ✓

```
int& gun() {
    int temp = 1000;
    return temp;
}
```

```
int x = 10

int *ptr = &x;

int*& cptr = ptr;
```