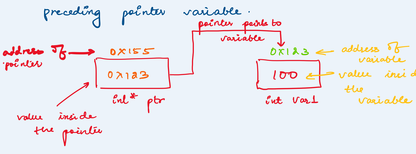
Pointers

→ A pointer refers to a variable that holds the address to another variable.

→ have a data type

→ If you want to access that particular memory location, then we use * operator preceding pointer variable.



Examples: (Initialisation of pointer)

Case 1:

```
int a = 100;
int *b = &a;
```

Case 2:

```
int *b;
b = &a;
```

→ ampersand means that address is being accessed.

Example: (Reassign existing pointer)

```
int a = 35;
int b = 30;

int *x;
x = &a;
x = &b;
```

Syntax:

datatype *variablename;

```
int *val;
float *val;
float *val;
```

& → Reference operator
returns the variable address

* → Deference operator
returns the value that has been stored in memory address

References:

Reference variable can be considered as an alias for an existing variable.

Reference is like a pointer whose address remains constant and the compiler will apply the * operator.

```
int a = 30;
int &x = a;
```

Pointers & Arrays

The array name itself denotes the base address of the array.

This means that to assign the address of an array to a pointer, you should not use an ampersand (&).

```
p = arr;
p = &arr; xx
```

```
int arr[20];
int *ip;
ip = arr;
```

Null pointer

If there is no exact address that is to be assigned, then the pointer variable can be assigned a NULL. It should be done during the declaration.

Such a pointer is known as NULL pointer.

Pointers & variables

The memory space can be assigned & reassigned as one wishes.

This is made possible by pointer variables.

Pointer variables point to a specific address in the computer's memory pointed to by another variable.

```
int *p;
int* p;
```

Application of pointers

→ Functions in C++ can only return 1 value.

→ Arguements to the functions are passed by value, & any modification done on the variables doesn't change the values of the actual variables that are passed.

Dynamic memory allocation

new operator

delete operator