📌 Pointers in C — Notes

1. What is a Pointer?

A **pointer** is a variable that stores the **memory address** of another variable.

- Normal variables hold data values (e.g., `int x = 10;`)
- Pointers hold addresses (e.g., address of `x`).

Example:

```
int x = 10;
int *ptr = &x; // ptr stores the address of x
```

---

2. Why Use Pointers?

- **Pass by reference** (modify variables inside functions)
- **Dynamic memory allocation** (`malloc`, `free`)
- **Efficient array and string handling**
- **Access hardware resources / memory directly**
- **Work with data structures** like linked lists, trees, etc.

---

3. Pointer Declaration and Initialization

Syntax:

```
dataType *pointerName;
```

Example:

```
int *p;      // pointer to int
float *q;    // pointer to float
char *r;     // pointer to char
```

Initialization:

```
int num = 25;
int *p = &num;  // store address of num in p
```

---

4. Accessing Value through Pointer (Dereferencing)

The `*` (asterisk) operator is used to **dereference** a pointer — access or modify the value stored at that address.

```
printf("%d", *p);     // prints value of num
*p = 50;                    // changes num to 50
```

---

5. Pointer Operators

Operator	Meaning
`&`	Address-of (gives address of a variable)
`*`	Dereference (access value stored at the address)

Example:

```
int x = 5;
int *p = &x;
printf("%p\n", &x);        // address of x
printf("%p\n", p);         // same address stored in p
printf("%d\n", *p);        // value at address (5)
```

---

6. Null Pointer

A pointer that stores **no valid address** is called a null pointer.

```
int *p = NULL;
```

- Useful for checking if a pointer is initialized before use.

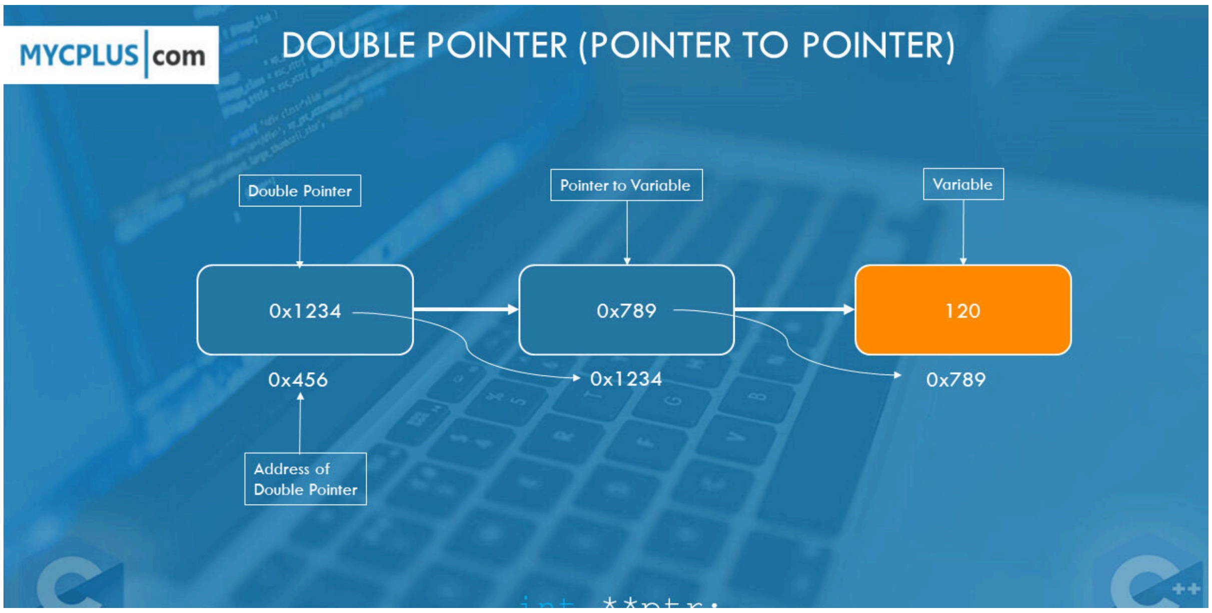
---

7. Pointer to Pointer

A pointer can store the address of another pointer.

```
int x = 10;
int *p = &x;
int **pp = &p;

printf("%d", **pp); // prints 10
```



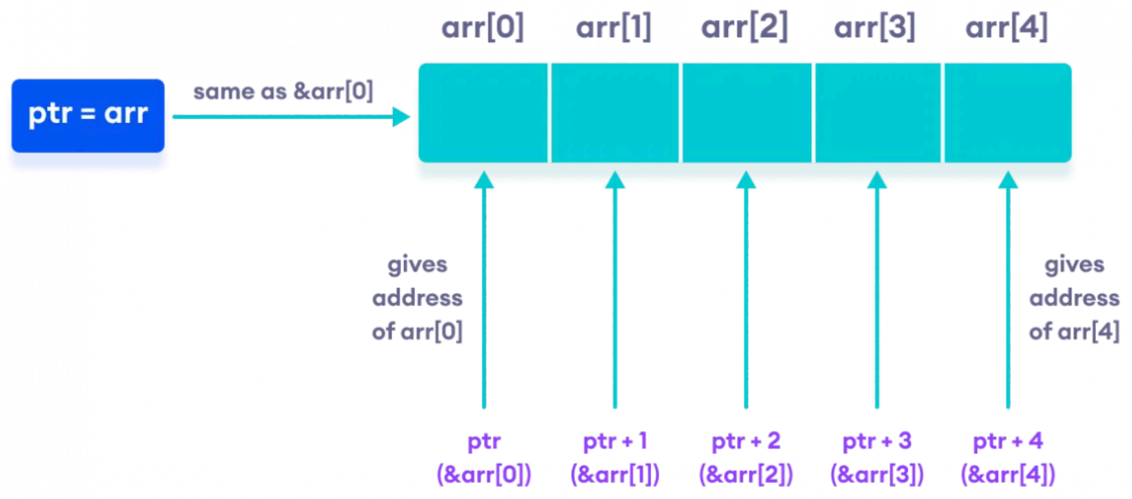
---

8. Pointers and Arrays

- Array name acts like a pointer to the first element.

```
int arr[3] = {1, 2, 3};
int *p = &arr[0];          // can also be written as arr

printf("%d", *(p+1));  // 2
```



9. Pointers and Functions (Pass by Reference)

```
void changeValue(int *n) {

    *n = 100;        // changes original variable

}

int main() {

    int x = 5;
    changeValue(&x);
    printf("%d", x); // 100

}
```

10. Pointer Arithmetic

You can increment/decrement pointers to move through memory.

```
int arr[] = {10, 20, 30};
int *p = arr;

p++; // now points to arr[1]
```

- Moves by the size of the data type (`int` → +4 bytes on most systems).

11. Common Mistakes

❌ Using an **uninitialized pointer** (can cause crashes)
❌ Dereferencing a **NULL** or invalid pointer
❌ Forgetting to `free()` dynamically allocated memory

12. Diagram Example

```
int x = 10;
int *p = &x;

x    = 10    →  [10]
&x   = 1000   (address)
p    = 1000  → points to x
*p   = 10
```

✅ Summary Table

Term	Example	Meaning
Address-of	`&x`	Address of variable `x`
Pointer	`int *p`	Stores address of an int
Dereference	`*p`	Value at stored address
Null Pointer	`p = NULL`	Pointer to nothing
Pointer to Pointer	`int **pp`	Stores address of a pointer