📘 Pointers in C++

🧠 What is a Pointer?

A **pointer** is a variable that stores the **memory address**. Pointers are a powerful feature in C++ that allow for **dynamic memory management**, **efficient array handling**, and **function argument manipulation**.

🧱 Pointer Declaration and Initialization

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

Person bruce("Bruce Wayne");
Person* pBats = &bruce; //pBats stores the memory address bruce
```

- `int* ptr`: Declares a pointer to an integer.
- `&x`: The **address-of** operator, returns the memory address of `x`.

🔍 Accessing Values with Pointers

```cpp
cout << *ptr << endl;  // Dereferencing: prints the value at the address stored in ptr
cout << pBats->Name(); // use the arrow notation to access the class members of bruce
```

- `*ptr`: The **dereference** operator, accesses the value at the memory address.
- `->` arrow notation (aka member access operator) is used to dereference the pointer and access the members of an object when you have a pointer to an object

NOTE: `ptr->member` is equivalent to `(*ptr).member`

🧪 Example: Basic Pointer Usage

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 42;
    int* p = &a;

    cout << "Value of a: " << a << endl;
    cout << "Address of a: " << &a << endl;
    cout << "Pointer p: " << p << endl;
    cout << "Value pointed to by p: " << *p << endl;

    return 0;
}
```

🧮 Pointer Arithmetic

Pointers can be incremented or decremented to traverse arrays:

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

cout << *p << endl;     // 10
cout << *(p + 1) << endl; // 20
```

🧰 Pointers and Functions

Pass-by-Reference Using Pointers

```cpp
void increment(int* num) {
    (*num)++;
}

int main() {
    int x = 5;
    increment(&x);
    cout << x << endl;  // Output: 6
}
```

🧠 Dynamic Memory Allocation

C++ allows dynamic memory allocation using `new` and deallocation using `delete`.

```cpp
int* p = new int;  // dynamically allocate memory
*p = 100;
cout << *p << endl;
delete p;          // free the memory
```

For arrays:

```cpp
int* arr = new int[5];  // allocate array
delete[] arr;           // deallocate array
```

🧱 Creating Class Instances on the Heap

In C++, you can create objects either on the **stack** or on the **heap**. When you create an object on the heap, you use the `new` keyword, and you manage its lifetime manually using `delete`.

🔹 Syntax

```cpp
ClassName* obj = new ClassName();  // Allocate on heap
...
delete obj;  // Deallocate memory
```

🧪 Example: Class Instance on the Heap

```cpp
#include <iostream>
using namespace std;

class Student {
public:
    string name;
    int age;

    Student(string n, int a) : name(n), age(a) {}

    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    // Create object on the heap
    Student* s = new Student("Alice", 20);

    // Access members using pointer
    s->display();

    // Clean up memory
    delete s;

    return 0;
}
```

🧠 Why Use Heap Allocation?

- **Dynamic lifetime**: Object exists until you explicitly delete it.
- **Useful for large objects** or when object size is not known at compile time.
- **Shared across scopes**: Can be passed around without worrying about scope-based destruction.

⚠️ Best Practices

- Always `delete` what you `new` to avoid **memory leaks**.
- Prefer using **smart pointers** (`std::unique_ptr`, `std::shared_ptr`) in modern C++ to manage memory automatically.

⚠️ Common Pitfalls

- **Dangling pointer**: Pointer pointing to deallocated memory.
- **Memory leak**: Forgetting to `delete` dynamically allocated memory.
- **Uninitialized pointer**: Using a pointer before assigning it a valid address.

📊 Summary Table

Concept	Syntax Example	Description
Declare pointer	`int* p;`	Pointer to an integer
Assign address	`p = &x;`	Store address of `x` in `p`
Dereference	`*p`	Access value at address
Dynamic allocation	`p = new int;`	Allocate memory
Deallocation	`delete p;`	Free memory

🎯 Quiz!

Here's a short quiz on the topic: quiz

🔭 Markdown Viewer

How to view the markdown files in a browser...

- Markdown Viewer

🧠 Lecture Practices

Here are the lecture Practices...

- Day 7
- Day 8
- Day 9

🔍 Lecture Quizzes

Here are the lecture quizzes...

- Day 7
- Day 8
- Day 9

Weekly Topics

Here are the topics for the week...

- Classes
- Structs
- Fields
- Getters and Setters
- Constructors
- Instances
- Inheritance
- Polymorphism
- Pointers
- Upcasting
- Misc. Concepts
- 4 Pillars of OOP