C++ Static Members

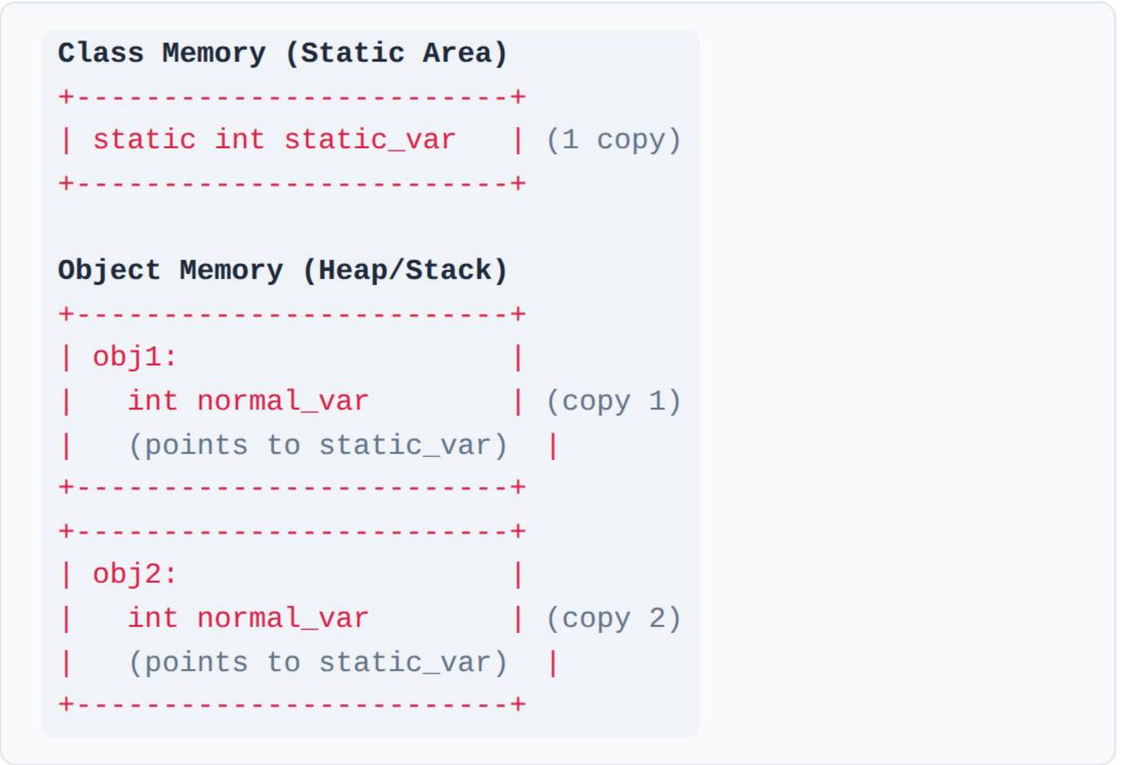
Exploring `static` data and functions to create class-level shared resources.

Static Data Members

Shared Memory: The Static Data Member

Definition & Purpose

- Declared **inside** the class using the `static` keyword, but defined **outside**.
- The member is **common to all objects** of the class.
- Memory is allocated **only once** for the entire class, regardless of how many objects are created.
- Its value persists throughout the life of the program.



Static Data Member Rules

Declaration vs. Definition

A static member must be defined outside the class scope using the scope resolution operator (`::`).

```
 class MyClass {
public:
    static int count; // Declaration inside
};

// Definition outside (MUST be done once)
int MyClass::count = 0;
```

Key Properties

- Must be explicitly defined outside the class definition.
- Can be accessed even if no object of the class exists.
- Cannot be initialized inside the class definition (except for `const static int`).
- Can be used to maintain a global count or a shared property for all objects.

Code Example: Using a Static Counter

```
 #include
using namespace std;

class DataDemo {
public:
    int normal_var;
    static int static_counter; // Static declaration

    DataDemo() {
        normal_var = 10;
        static_counter++; // Increments the SAME counter for every object
    }
};

// Static member definition and initialization (MUST be outside)
int DataDemo::static_counter = 100;

int main() {
    DataDemo obj1;
    DataDemo obj2;

    cout << "obj1.normal_var: " << obj1.normal_var << endl;
```

Static Member Functions

Access Restrictions: Static Member Functions

Definition & Calling

- 📦 Declared with the `static` keyword inside the class.
- ✅ Can be called **without** creating an object (using `ClassName::FunctionName()`).

CRITICAL Restriction

- ✋ A static member function **CANNOT** access non-static data members or non-static member functions.
- 🗄️ It can **ONLY** access other **static** data members or **global** variables/functions.

Code Example: Using a Static Function

```
 #include
using namespace std;

class MemberDemo {
private:
    int non_static_var = 5;
    static int static_var;
public:
    static void showStaticVar() {
        cout << "Static var: " << static_var << endl;
        // cout << non_static_var; // ERROR! Cannot access non-static member
    }
};

int MemberDemo::static_var = 25;

int main() {
    // Call without creating an object
    MemberDemo::showStaticVar();

    // You can still call it via an object, but it's less common
    MemberDemo obj;
```

Summary: Non-Static vs. Static

Feature	Non-Static Member	Static Member
Memory	Separate copy for every object.	One single copy, shared by all objects.
Access	Must be accessed via an object (instance).	Accessed via Class Name (`ClassName::member`).
Functions	Can access ANY class member (static or non-static).	Can ONLY access static members or global variables.
Lifetime	Created when object is created, destroyed when object is destroyed.	Created before `main()`, exists until program ends.