Introduction to Smart Pointers in C++

What are Smart Pointers?

Smart pointers are class templates in `<memory>` that automatically manage the lifetime of dynamically allocated objects. Unlike raw pointers, they prevent memory leaks by automatically deallocating memory when it's no longer needed.

Types of Smart Pointers

1. `std::unique_ptr`

A `unique_ptr` owns an object exclusively. It cannot be copied, only moved.

Example:

```
#include <memory>
#include <iostream>

struct Foo {
    Foo() { std::cout << "Foo created\n"; }
    ~Foo() { std::cout << "Foo destroyed\n"; }
};

int main() {
    std::unique_ptr<Foo> p = std::make_unique<Foo>();
} // Foo is destroyed automatically here
```

2. `std::shared_ptr`

A `shared_ptr` allows multiple pointers to share ownership of the same object. The object is deleted when the last `shared_ptr` is destroyed.

Example:

```
#include <memory>
#include <iostream>

int main() {
    auto p1 = std::make_shared<int>(42);
    auto p2 = p1; // reference count increases
    std::cout << *p2 << std::endl; // prints 42
} // object deleted when last shared_ptr goes out of scope
```

3. std::weak_ptr

A `weak_ptr` is a non-owning reference to a `shared_ptr` managed object. It is used to avoid cyclic dependencies.

Example:

```
#include <memory>
#include <iostream>

int main() {
    std::shared_ptr<int> sp = std::make_shared<int>(100);
    std::weak_ptr<int> wp = sp;

    if (auto locked = wp.lock()) {
        std::cout << *locked << std::endl;
    } else {
        std::cout << "Object no longer exists.\n";
    }
}
```

Comparison Table

Pointer Type	Ownership	Reference Counted	Auto Deletes
`unique_ptr`	Exclusive	No	Yes
`shared_ptr`	Shared	Yes	Yes (last one)
`weak_ptr`	None	No	No

Summary

- Use `unique_ptr` when only one owner exists.
- Use `shared_ptr` when ownership is shared.
- Use `weak_ptr` to observe an object managed by `shared_ptr` without extending its lifetime.