Introduction

This lesson will introduce smart pointers in Modern C++.

WE'LL COVER THE FOLLOWING

Introduction

Smart pointers are one of the most important additions to C++ because they enable us to implement explicit memory management in C++. Beyond the *deprecated* `std::auto_ptr`, C++ offers three different smart pointers. They are defined in the header `<memory>`.

First, there is `std::unique_ptr`, which models the concept of exclusive ownership. Second, there is `std::shared_ptr`, which models the concept of shared ownership. Lastly, there is the `std::weak_ptr`. `std::weak_ptr` is not smart, as it has only a thin interface, and it serves only to break cycles of `std::shared_ptr`. It models the concept of temporary ownership.

Smart pointers manage their resources according to the RAII idiom. Therefore, the resource is automatically released if the smart pointer goes out of scope.

> **i Resource Acquisition Is Initialization** **R**esource **A**cquisition **I**s **I**nitialization, also known as RAII, is a popular technique in C++. Resource acquisition and release are bound to the lifetime of an object. This means that the memory for the smart pointer is allocated in the constructor and deallocated in the destructor. We can use this technique in C++ because the destructor is called when the object goes out of scope.

Various Types of Smart Pointers

Name	Standard	Description
`std::auto_ptr` (*deprecated*)	C++98	Owns exclusively the resource. Moves the resource while copying.
`std::unique_ptr`	C++11	Owns exclusively the resource. Can't be copied.
`std::shared_ptr`	C++11	Has a reference counter for the shared variable. Manages the reference counter automatically. Deletes the resource, if the reference counter is 0.
`std::weak_ptr`	C++11	Helps to break cycles of `std::shared_ptr`. Doesn't modify the reference counter.

Now, let's move on to the various types of smart pointers.