Introduction

This section will deal which a very powerful new category called smart pointers. Let's begin.

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

Firstly there is the `std::unique_ptr`, which models the concept of exclusive ownership. Secondly, there is the `std::shared_ptr`, who models the concept of shared ownership. Lastly, there is the `std::weak_ptr`. `std::weak_ptr` is not so smart, because it has only a thin interface. Its job is it to break cycles of `std::shared_ptr`. It models the concept of temporary ownership.

The smart pointers manage their resource according to the RAII idiom. So 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, short RAII, stands for a popular technique in C++, in which the resource acquisition and release are bound to the lifetime of an object. This means for the smart pointer that the memory is allocated in the constructor and deallocated in the destructor. You can use this technique in C++ because the destructor is called when the object goes out of scope.

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.

Overview smart pointers

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