Phantom and indulgent shared pointers

devblogs.microsoft.com/oldnewthing/20230818-00

August 18, 2023

Raymond Chen

Last time, we looked at various ways to convert among different `shared_ptr`s. We finished with this diagram:

	Null control block	**Non-null control block**
Null stored pointer	Empty	Phantom
Non-null stored pointer	Indulgent	Full

You are familiar with an empty shared pointer, which manages no object and has no stored pointer. You are also familiar with a full shared pointer, which manages an object and has a non-null stored pointer (to the managed object, or something whose lifetime is controlled by the managed object). But what about those other two guys?

In the upper right corner, you have the case of a shared pointer that manages an object but whose stored pointer is null, which I've called a *phantom* shared pointer. If you convert the shared pointer to a `bool`, it produces `false`, because you can't use it to access anything. The phantom shared pointer looks empty at a casual glance, but it secretly manages an object behind the scenes. That secretly-managed object remains alive for no visible reason. There is no way to access that secretly-managed object, but it's still there. It's a phantom which follows you around.

```
struct Sample
{
    int value;
};
std::shared_ptr<Sample> p = std::make_shared<Sample>();
std::shared_ptr<char> q = std::shared_ptr<char>(p, nullptr);
p = nullptr;
```

In the above example, we start by creating a shared pointer to a freshly-constructed `Sample` object. We then use this shared pointer to provide the control block to a new aliasing shared pointer whose stored pointer is `nullptr`. Finally, we clear the original shared pointer, so that the only reference to the `Sample` is in the phantom shared pointer `q`. The `Sample` is now being kept alive by what looks like a null pointer!

In the opposite corner, you have a shared pointer with no control block, but with a non-null stored pointer, which I've called an *indulgent* shared pointer because it points to something, yet owns nothing. "Go ahead, have fun with this pointer all you like, I don't care!"

```
int globalVariable;

std::shared_ptr<int> p(std::shared_ptr<int>(), &globalVariable);
```

We use the aliasing constructor for `shared_ptr<int>` which takes another `shared_ptr<int>` (which provides the control block) and a raw pointer (which serves as the stored pointer). The donor `shared_ptr` is empty and has no control block. We give that nonexistent control block to the aliasing constructor, and the result is a shared pointer that owns nothing, yet which points to something, namely, `globalVariable`.

This style of shared pointer is useful if you need a shared pointer that points to memory with static storage duration. The pointer is valid for the duration of the process, so you don't need a control block to manage the object's lifetime: The object's lifetime is "forever".

How can you detect whether you have one of these unusual shared pointers?

If you treat a `shared_ptr` as a `bool`, the resulting value tells you whether the stored pointer is non-null. On the other hand, if you ask `use_count()`, it will return a nonzero value if there is a managed object. Now we can finish our table:

	Null control block (use_count() == 0)	**Non-null control block (use_count() != 0)**
Null stored pointer (falsy)	Empty	Phantom
Non-null stored pointer (truthy)	Indulgent	Full

Note that phantom shared pointers are falsy, and indulgent shared pointers are truthy. I couldn't think of a synonym for indulgent that begins with a "t", sorry.

In code, we can detect the four cases by using `use_count()` and `get()` (or equivalently, converting to `bool`).

```cpp
void detect(std::shared_ptr<T> const& p)
{
    bool has_control_block = p.use_count();
    bool has_pointer = p.get();
    // equivalently, has_pointer = static_cast<bool>(p);

    if (has_control_block && has_pointer) {
        // full
    } else if (has_control_block && !has_pointer) {
        // phantom
    } else if (!has_control_block && has_pointer) {
        // indulgent
    } else { // !has_control_block && !has_pointer)
        // empty
    }
}
```