# 【C++】Day53(2)

| | |
|---|---|
| Class | C++ |
| Date | @February 7, 2022 |
| Material | |
| Series Number | |
| Summary | |

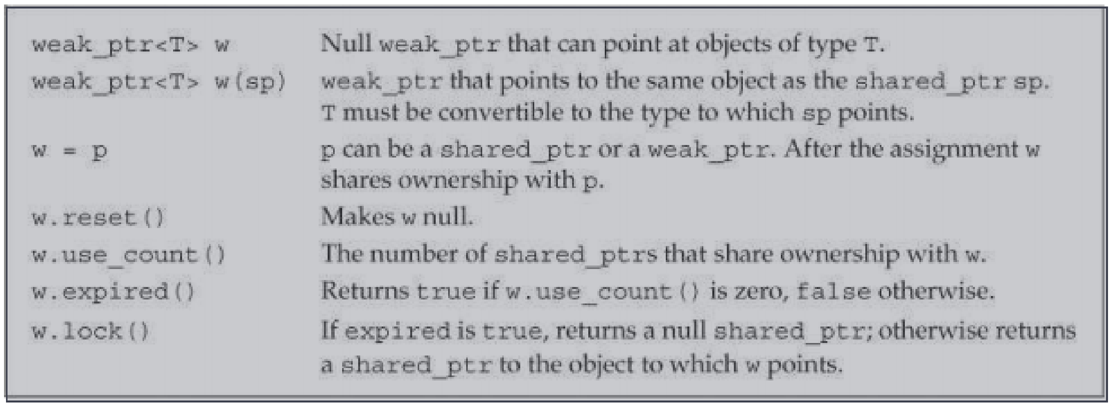
## 【Ch12】Dynamic Memory

### 12.1.6 weak_ptr

A `weak_ptr` is a smart pointer that does not control the lifetime of the object to which it points. Instead, a `weak_ptr` points to an object that is managed by a `shared_ptr`. Binding a `weak_ptr` to a `shared_ptr` does not change the reference count of that `shared_ptr.`

Once the last `shared_ptr` pointing to the object goes away, the object itself will be deleted. That object will be deleted even if there are `weak_ptr`s pointing to it-hence the name `weak_ptr`, which captures the idea that a `weak_ptr` shares its object "weakly."

**Table 12.5.** `weak_ptrs`



| | |
|---|---|
| `weak_ptr<T> w` | Null `weak_ptr` that can point at objects of type `T`. |
| `weak_ptr<T> w(sp)` | `weak_ptr` that points to the same object as the `shared_ptr sp`. `T` must be convertible to the type to which `sp` points. |
| `w = p` | `p` can be a `shared_ptr` or a `weak_ptr`. After the assignment `w` shares ownership with `p`. |
| `w.reset()` | Makes `w` null. |
| `w.use_count()` | The number of `shared_ptr`s that share ownership with `w`. |
| `w.expired()` | Returns `true` if `w.use_count()` is zero, `false` otherwise. |
| `w.lock()` | If `expired` is `true`, returns a null `shared_ptr`; otherwise returns a `shared_ptr` to the object to which `w` points. |

When we create a `weak_ptr`, we initialize it from a `shared_ptr`:

```
auto p = make_shared<int>(42);
weak_ptr<int> wp(p); //wp eakly shares with p; use count in p is changed
```

Here `wp` and `p` point to the same object. It is possible that the object to which `wp` points might be deleted.

Because the object might no longer exist, we cannot use a `weak_ptr` to access its object directly. To access that object, we must call `lock`. The `lock` function checks whether the object to which the `weak_ptr` points still exists.

If so, lock returns a `shared_ptr` to the shared object. As with any other `shared_ptr`, we are guaranteed that the underlying object to which that `shared_ptr` points continues to exist at least as long as that `shared_ptr` exists. For example:

```
if(shared_ptr<int> np = wp.lock()) {
  //true if np is not null
  //inside the if, np shares its object with p
}
```