 𝐬𝐡𝐚𝐫𝐞𝐝\_𝐩𝐭𝐫 𝐢𝐧 𝐂++:  
  
Now, let's step into the world of smart pointers with `shared\_ptr`. Think of it as a shared ownership pointer. Multiple `shared\_ptr` instances can point to the same resource, and it keeps track of how many are using it.  
  
[#include](https://www.linkedin.com/feed/hashtag/?keywords=include&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7106160291301765120) <memory>  
  
std::shared\_ptr<int> shared = std::make\_shared<int>(42);  
  
🔗 Multiple `shared\_ptr` instances can point to this integer, and the resource (in this case, an integer with a value of 42) will be automatically deleted when the last `shared\_ptr` that owns it goes out of scope.  
  
🔒𝐮𝐧𝐢𝐪𝐮𝐞\_𝐩𝐭𝐫 𝐢𝐧 𝐂++:  
  
On the other hand, we have `unique\_ptr`, which is like a sole proprietorship pointer. It ensures exclusive ownership of the resource.  
  
[#include](https://www.linkedin.com/feed/hashtag/?keywords=include&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7106160291301765120) <memory>  
  
std::unique\_ptr<int> unique = std::make\_unique<int>(42);  
  
🔐 With unique\_ptr, only one pointer can own the resource. If you try to create another `unique\_ptr` pointing to the same resource, it won't work. When the `unique\_ptr` goes out of scope or is explicitly reset, it will automatically delete the resource.