# Improving `std::shared\_ptr` Usability with Classes Having Protected or Private Destructors

Author: Davit Kalantaryan  
GitHub: https://github.com/davitkalantaryan

## Abstract

This paper proposes a revision to the internal deletion mechanism of `std::shared\_ptr` to make it compatible with classes whose destructors are declared as protected or private, as long as `std::shared\_ptr<T>` is explicitly declared a friend. The goal is to improve safety, encapsulation, and prevent accidental misuse, particularly in large codebases. Additionally, it suggests a secondary extension: optional notification hooks on reference count changes.

# Appendix B: Clarification After Initial Feedback

After reading the initial feedback from the community, I decided to include the following clarification in the proposal:  
  
Many of you are absolutely right — if a smart pointer manages lifetime, manual deletion should be avoided. But in large, evolving codebases, it’s not uncommon for someone to forget or misunderstand ownership. That’s exactly why C++ provides access control like private and protected: not because developers are careless, but to let the compiler help enforce intended usage.  
  
Saying “just don’t delete manually” is similar to saying “we don’t need private or protected; developers should just remember not to call internal functions.” But we do use those specifiers — to make misuse harder, not just discouraged.  
  
This proposal aims to bring similar safety to destructors. If shared\_ptr<T> is a friend, it should be allowed to delete T, just like any other friend.