**C++ Memory Management**

**and How the Standard Library Could Have Done Better**

I'm an old school C++ guy. I started using the language in 1993-ish and it was my main development language until 2010. I then did other things for a while, and got back into C++ with a new job last year (early 2022). During that “off” time, I kept a cursory eye on the developments in the language. As part of the new job, I dug in to using the STL smart pointers.

Now rewind to 1995, where I had a job where I probably learned the most about C++ and object oriented development. That team had written their own smart pointers. Over the years, I rewrote them time and again for the various organizations for which I worked.

One of the first things I noticed in the new smart pointers was a lack of a non-owning pointer type. I'm not a fan of using raw pointers for this purpose. They are a little too non-explicit for my tastes. Also, some things about the non-interaction between unique\_ptr and shared\_ptr bothered me. I should be able to transfer ownership of an object from a unique\_ptr to a shared\_ptr, right? And finally, why are these data type names so long, for something you would expect to use so often? So I rewrote my own, again, using modern C++. UP is essentially a unique\_ptr. SP is essentially a shared\_ptr. WP is a weak\_ptr. And I added TP, a non-owning pointer, providing the missing pointer with the semantics of “this pointer does NOT own the thing it's pointing to.” I also set up UP to work interchangeably with unique\_ptr. SP could not be made to work interchangeably with shared\_ptr, just because they would each have their own different internal structures (well, they *could* be made to work, I guess, but that's make for some really ugly code). UP and SP both replace the get() method with an UnsafeAccess() method, providing a clear indication that the caller is playing with fire. The better move is to assign a TP from the UP or SP.

So, anyway, as I dug further into this, one of the “old school” things we did was to override new and delete rather regularly. For that initial job, we were creating DLLs in Windows NT, and memory management in Windows had issues if you allocate memory in one DLL and try to free it in another. So overriding new and delete (using inheritance) forced those calls to come back to the code in the class's own DLL.

But for my very next job, we also found that overriding new and delete gave us an easier way to track memory leaks. We could assign each class (or set of classes) to a specific memory pool, and leaks could easily be tied back to the responsible class code. Further, classes that were often new'd and delete'd over and over could have the new and delete methods manage a memory cache for the class, significantly improving the runtime of memory allocation. I used these techniques through to 2010.

As I started looking at some code using modern C++ on my own, I realized I would want to use these techniques again. However, I had noticed that shared\_ptr had a deleter and an allocator as optional template parameters. As I dug more into this, I found that the std::shared\_ptr might not even call a new or delete overridden in a class. Ostensibly, this is because it combines the shared object's memory with the common object pointer and internal counters, which changes the size of the allocated memory between a shared\_ptr and unique\_ptr to the same type. All of this struck me as decidedly non-object-oriented. In order to get the performance of a memory cache on a class, EVERY shared pointer allocation would be required to reference the same allocator and deleter. Where we used to manage this at the class level, this kind of control had migrated to every bit of object allocating code. This, I feel, is a design mistake in the standard library smart pointers, and a significant one at that.

Lastly, the idea that the new and delete operators are somehow inherently evil does not resonate with me. Sure, with smart pointers, the use of delete becomes pretty much obsolete. But allocating memory with new and directly putting that pointer into an OP or SP seems rather innocuous to me. I, for one, will resume using new in this manner.

Having gone through this effort, I decided that maybe others could benefit from these classes, and perhaps, if you find them useful, you can sponsor me as I make updates here and add new repositories of useful classes to GitHub.