1) Class Description

- Goals
    - Override “new” to allow for a debug mode, changing raw pointers into AuditedPointers.
        - Setting the debug flag to false means it’s just a normal raw pointer, setting it to true means entering a debug mode with extra checks like memory management.
    - Override “delete” to ensure AuditedPointers that are deleted cannot be dereferenced or deleted a second time.
    - Have a list of active AuditedPointers to check whether all AuditedPointers have been deleted by program termination.
    - Create a MakeAudited function that works like std::make_shared to turn a standard raw pointer into an audited raw pointer (putting it in debug mode).
- High-Level Functionality
    - Template class that builds on top of a raw pointer by adding a DEBUG mode. This mode ensures a deleted pointer cannot be dereferenced or deleted a second time. When a program terminates, the class ensures all AuditedPointers have been deleted.

2) Similar Classes

- All smart pointer classes (*unique_ptr, shared_ptr, weak_ptr*) that help automatically manage memory.
- *Std::allocator* which also helps with memory management and can be associated with pointers.

3) Possible Key Functions

- Overrides of “new” and “delete” functions with functionality as stated before.
- MakeAudited - Similar to std::make_shared by turning a standard raw pointer into an AuditedPointer.
- CheckAuditedPointers - Checks to see if the list of audited pointer is empty, if it isn’t then throw an exception (as all pointers haven’t been deleted)

4) Error Conditions

- Programmer Errors
    - Trying to dereference a nullptr. (assert(AuditedPtr != nullptr)
    - Trying to delete an already deleted AuditedPtr. (assert(AuditedPtr.isDeleted() == false))
    - Using an already deleted AuditedPtr. (assert(AuditedPtr.isDeleted() == false))
- Recoverable Errors
    - MakeAudited has an allocation failure (throw std::bad_alloc())
- User Errors

    - Accessing a deleted AuditedPtr (return nullptr).
    - Deleting a nullptr (return early).

5) Expected Challenges

- Ensuring cyclic AuditedPointers do not lead to memory leaks
- Ensuring the debugging steps do not introduce too much overhead to where the usage of the debugging is inconvenient.
- Ensuring the global set of active AuditedPointers is properly checked and does not introduce race conditions.

6) Extra Topics Needed

- The design of AuditedPointer being a generic template that works seamlessly with any type while also overriding the standard raw pointer seems a bit tricky.
- Providing overrides to “new” and “delete” without introducing bugs also seems fairly difficult.
- Creating a global set of active AuditedPointers that is checked at program termination alo seems difficult.

6) Useful Class Projects

- Audited Vector (Group 1), Audited String (Group 4), Audited Array (Group 7): Not necessarily useful within implementation but would be nice to see how another group would handle a similar project that introduces a debug mode for a standard C++ class.
- Output Log (Group 7) - Could be useful to check to ensure debug checks are being met and when in the code they are being met.