

## Part D: Reflection

1. Difference between:
  - a. A pointer to stack memory points to a temporary variable that gets automatically deleted when the function ends. A pointer to heap memory points to the memory that stays until you delete it. And finally a smart pointer or a *unique\_ptr* also points to heap memory but instead it deletes it automatically when it gets out of scope.
2. Where and why *delete* was used.
  - a. *delete* was used in part B of the pointer project to free allocated arrays. I did not use *delete* in part C because *unique\_ptr* does that cleanup automatically without the use of *delete*.
3. Explanation of ownership in each design.
  - a. In part A, the *task* objects are created in the stack so the pointers do not own any of the memory. The objects are automatically destroyed when they get out of scope, meaning there does not have to be a memory management.
  - b. In part B, the tasks are being distributed on the heap using *new* and the raw pointer owns the memory. It is important to use *delete* to free memory which makes ownership manual and less prone to errors.
  - c. Finally in part C, ownership is managed by *std::unique\_ptr*. The *taskManager* has ownership of the task arrays and the memory is automatically released when the project gets out of scope which makes sure there is an automatic cleanup.
4. Which pointer method is safest and why.
  - a. *unique\_ptr* is probably the safest method since it prevents any memory leaks, dangling pointers, and also enforces ownership.