1) Pointer to stack memory

A pointer to stack memory essentially is a pointer that points or references back to another variable that has a certain memory address located on the stack. One good analogy that I found goes like this. Imagine you have a box of pizza; instead of carrying the pizza around with you and giving it to your friends, you can just tell your friends the address of where the pizza is located and they can come pick it up.

```
// From mainA.cpp
Task t1(1, "Finish project");  // Task created on the stack
Task *ptr1 = &t1;              // ptr1 holds the address of t1
// Demonstrating pointer usage:
ptr1->markCompleted();         // Access through pointer
std::cout << "Address of t1: " << &t1 << std::endl;
```

2) Pointer to heap memory

This type of pointer holds the memory address of other data located in the heap of memory(allocated by the programmer). This pointer requires manual allocation and deletion by the programmer, and any errors in this step may result in memory leaks.

```
// From mainB.cpp
int capacity = 5;
Task *tasks = new Task[capacity];  // Manual allocation on heap
// Manual cleanup required
delete[] tasks;
```

3) Smart pointer (unique_ptr)

Smart pointers are most commonly used in modern C++ as unlike pointers to heap memory, smart pointers automatically delete memory, making it a much safer and more practical tool. Through RAII, the lifecycle of the variables are directly correlated to the lifetime of the object, this in turn provides assurance that no variables are running out of scope.

4) Where and why delete was used.

Delete was used in Part B of the assignment where we used “pointers to heap memory”. We had to use delete because unlike smart pointers, this specific type of pointer does not allocate and delete memory on its own. So by adding a delete function (delete[] tasks;) at the end of my mainB.cpp, I ensured that there were no memory leaks and everything was deleted after the program ended.

5) Explanation of ownership in each design.

a) Allocation and deallocation of stack memory are handled automatically by the compiler.
b) The programmer owns and allocates memory. (Myself)
c) The unique.ptr inside TaskManager owns the memory and automatically deletes as it sees fit.

6) Which pointer method is safest and why?

Smart pointers are by far the safest as they automatically deallocate memory so you don't have to worry about memory leaks.