Double-checked Locking Solutions

Double-checked locking

- What is meant by double-checked locking? Explain how it is intended to work
    - Double-checked locking attempts to perform thread-safe lazy initialization while avoiding unnecessary locking
    - The variable is checked twice: once before locking, and once afterwards
    - The first check ensures that the mutex is only locked if the variable is uninitialized
    - This leaves potential for a data race, in case the thread is interrupted between performing the first check and locking the mutex
    - The variable is checked again, under the lock, to make sure that another thread has not initialized the variable while this thread was interrupted

Double-checked locking issues

- What issues can arise with double-checked locking in C++?
  - A typical case is where the variable is a pointer
  - Initialization involves calling the new operator and the check is whether the pointer is null
  - In C++, the compiler is allowed to reorder operations when optimizing the code
  - This can result in the pointer value being assigned before the object has been created
  - If the thread is interrupted at this point, other threads will see the variable as initialized
  - This will cause them to skip over the lock and second check and use the uninitialized object

Double-checked locking issues

- Describe some ways to avoid these issues
  - Use std::call_once to execute the initialization code. This will ensure that the code is only executed once, by one thread, which will not be interrupted
  - Use a static local variable
  - Compile under C++17 or later, where the object is guaranteed to be constructed before new saves its address

Thread-safe double-checked locking

- Write a function that performs thread-safe double-checked locking
- Use one of the techniques you mentioned in your previous answer