Deadlock Avoidance Solutions

Deadlock Avoidance

- Suggest two ways to avoid deadlock
  - Always acquire locks in the same order
  - Use language features which can acquire multiple locks in a single operation
- Write a program which causes two threads to deadlock
- Implement your solutions. Verify that the program is no longer affected by deadlock

Adopting Locks

- In the following code, why are the unique_lock objects needed when the mutexes are already locked?

```
std::lock(mut1, mut2);
std::unique_lock< std::mut> lk1(mut1, std::adopt_lock);
std::unique_lock< std::mut> lk2(mut2, std::adopt_lock);
```

  - To ensure that the mutexes are always unlocked when leaving the enclosing scope

- Rewrite this code to use std::unique_lock's defer_lock option

```
std::unique_lock< std::mutex> lk1(mut1, std::adopt_lock);
std::unique_lock< std::mutex> lk2(mut2, std::adopt_lock);
std::lock(mut1, mut2);
```