Unique Lock Solutions

std::unique_lock

- What are the main differences between std::lock_guard and std::unique_lock?
  - std::lock_guard has only a constructor and destructor
  - std::unique_lock has other member functions, including lock() and unlock()
  - std::lock_guard's constructor can only perform a blocking lock
  - std::unique_lock's constructor has a number of options for locking

Output Example using std::unique_lock

- Rewrite the program from the previous lecture to use std::unique_lock

std::unique_lock vs std::lock_guard

- We could also have avoided the problem with std::lock_guard by writing the code like this:

```
for (int i = 0; i < 5; ++i) {
   // Put the std::lock_guard object in an inner scope
   {
      std::lock_guard<std::mutex> lck_guard(task_mutex);

      // Critical section
      std::cout << str[0] << str[1] << str[2] << std::endl;
      // End of critical section
   } // Calls ~std::lock_guard

   std::this_thread::sleep_for(50ms);
}
```

std::unique_lock vs std::lock_guard

- In this program, is there any advantage to using unique_lock compared to using lock_guard in an inner scope?
  - Putting in an explicit call to unlock() instead of implicitly unlocking the mutex at the end of the scope makes the code clearer
  - There is some extra overhead to using unique_lock, but in this program it will not be noticeable
  - There are some situations where we need to explicitly unlock the mutex
  - In those cases, we have to use std::unique_lock
  - We will see some examples later in the course