Timeouts and Mutexes Solutions

std::recursive_mutex

- Briefly describe std::recursive_mutex
  - std::recursive_mutex is a mutex that can be locked more than once in the same thread, without being unlocked
  - (For std::mutex, this would cause undefined behaviour)
  - For each call to lock(), there must be a corresponding call to unlock()
- Is this a feature that programmers should use regularly?
  - No, it should only be used when necessary
  - If there is a possibility that a mutex may be locked more than once, this usually indicates a problem with the design of the code
  - However, it can occasionally be useful, e.g. for recursive functions that need to lock a mutex before calling themselves

std::timed_mutex

- Briefly describe std::timed_mutex
  - std::timed_mutex is similar to std::mutex
  - It has additional member functions that wait with a timeout to acquire a lock on the mutex
  - try_lock_for() will wait until a certain time interval elapses to acquire a lock
  - try_lock_until() will wait until a certain time point to acquire a lock
- Is there another way to obtain this functionality?
  - std::unique_lock has the same member functions
  - They can be called using any type of mutex as the lock's parameter

try_lock_for() and try_lock_until()

- Rewrite the try_lock program from "Mutex Introduction" to use
  - try_lock_for() with std::timed_mutex
  - try_lock_until() with std::timed_mutex
  - try_lock_for() with std::unique_lock
  - try_lock_until() with std::unique_lock

try_lock_for() and try_lock_until()

- What do you observe?
  - The results are similar
  - Task1 gets the lock first (because of the sleep in Task2)
  - Task2 repeatedly calls try_lock unsuccessfully, because it is locked by Task1
  - Finally, Task1 releases the lock and Task2's try_lock() call succeeds

```
Task1 trying to get lock
Task1 has lock
Task2 trying to get lock
Task2 could not get lock
Task2 could not get lock
Task2 could not get lock
Task2 could not get lock
Task1 releasing lock
Task2 has lock
```