

# Mutex Types and Locking Methods

This lesson discusses different types of mutexes and their locking methods.

## WE'LL COVER THE FOLLOWING ^

- `std::shared_timed_mutex`
- Mutex `try_lock` methods

C++ has five different mutexes that can lock recursively (i.e., multiple layers of locking), tentative with and without time constraints.

| Method                         | mutex | recursive_mutex | timed_mutex | recursive_timed_mutex | shared_timed_mutex |
|--------------------------------|-------|-----------------|-------------|-----------------------|--------------------|
| <code>m.lock</code>            | yes   | yes             | yes         | yes                   | yes                |
| <code>m.unlock</code>          | yes   | yes             | yes         | yes                   | yes                |
| <code>m.try_lock</code>        | yes   | yes             | yes         | yes                   | yes                |
| <code>m.try_lock_for</code>    | no    | no              | yes         | yes                   | yes                |
| <code>m.try_lock_until</code>  | no    | no              | yes         | yes                   | yes                |
| <code>m.try_lock_shared</code> | yes   | no              | no          | no                    | yes                |

|                                      |    |    |    |    |     |
|--------------------------------------|----|----|----|----|-----|
| <code>m.try_lock_for</code>          | no | no | no | no | yes |
| <code>m.try_lock_shared_for</code>   |    |    |    |    |     |
| <code>m.try_lock_until</code>        | no | no | no | no | yes |
| <code>m.try_lock_shared_until</code> |    |    |    |    |     |

## `std::shared_timed_mutex` #

With C++14, we have an `std::shared_timed_mutex` that is the base for reader-writer locks. It solves the infamous [reader-writers problem](#).

The `std::shared_timed_mutex` enables us to implement reader-writer locks which means that we can use it for exclusive or shared locking. We will get an exclusive lock if we put the `std::shared_timed_mutex` into an `std::lock_guard`; you will get a shared lock if we put it into an `std::unique_lock`.

### **i** `std::shared_mutex` with C++17

With C++17, we get a new mutex: `std::shared_mutex`. `std::shared_mutex` is similar to `std::shared_timed_mutex`. Like the `std::shared_timed_mutex`, we can use it for exclusive or shared locking, but we can not specify a time point or a time duration.

## Mutex `try_lock` methods #

The `m.try_lock_for(relTime)` (`m.try_lock_shared_for(relTime)`) method needs a relative time duration; the `m.try_lock_until(absTime)` (`m.try_lock_shared_until(absTime)`) method needs an absolute time point.

`m.try_lock` (`m.try_lock_shared`) tries to lock the mutex and returns immediately. Upon success, it returns true; otherwise, it's false. In contrast, the methods `try_lock_for` (`try_lock_shared_for`) and `try_lock_until` (`try_lock_shared_until`) try to lock until the specified timeout occurs or the lock is acquired, whichever comes first. We should use a steady clock for our time constraint. A steady clock cannot be adjusted.

**Tip:** We should not use mutexes directly; we should put mutexes into locks.

---

In the next lesson, we'll discuss deadlocks caused by improper mutex locking in C++.