C++
Information
Tutorials
Reference
Articles
Forum

Reference
C library:
Containers:
Input/Output:
Multi-threading:
<atomic>
<condition_variable>
<future>
<mutex>
<thread>
Other:

<mutex>
classes:
adopt_lock_t
defer_lock_t
lock_guard
mutex
once_flag
recursive_mutex
recursive_timed_mutex
timed_mutex
try_to_lock_t
unique_lock
functions:
call_once
lock
try_lock
constants:
adopt_lock
defer_lock
try_to_lock

mutex
mutex::mutex
mutex::~mutex
member functions:
mutex::lock
mutex::native_handle
mutex::try_lock
mutex::unlock

class

std::mutex

<mutex>

```
class mutex;
```

Mutex class

A *mutex* is a *lockable object* that is designed to signal when critical sections of code need exclusive access, preventing other threads with the same protection from executing concurrently and access the same memory locations.

`mutex` objects provide *exclusive ownership* and do not support recursivity (i.e., a thread shall not lock a `mutex` it already owns) -- see `recursive_mutex` for an alternative class that does.

It is guaranteed to be a standard-layout class.

Member types

member type	description
`native_handle_type`	Type returned by `native_handle` (only defined if library implementation supports it)

Member functions

(constructor)	Construct mutex (public member function)
lock	Lock mutex (public member function)
try_lock	Lock mutex if not locked (public member function)
unlock	Unlock mutex (public member function)
native_handle	Get native handle (public member function)

Example

```
1  // mutex example
2  #include <iostream>       // std::cout
3  #include <thread>         // std::thread
4  #include <mutex>          // std::mutex
5
6  std::mutex mtx;           // mutex for critical section
7
8  void print_block (int n, char c) {
9    // critical section (exclusive access to std::cout signaled by locking mtx):
10   mtx.lock();
11   for (int i=0; i<n; ++i) { std::cout << c; }
12   std::cout << '\n';
13   mtx.unlock();
14 }
15
16 int main ()
17 {
18   std::thread th1 (print_block,50,'*');
19   std::thread th2 (print_block,50,'$');
20
21   th1.join();
22   th2.join();
23
24   return 0;
25 }
```

Possible output (order of lines may vary, but characters are never mixed):

```
**************************************************
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```