# - Example

An example to demonstrate the usage of mutexes.

# Example #

```cpp
// mutex.cpp

#include <chrono>
#include <iostream>
#include <mutex>
#include <thread>

std::mutex coutMutex;

class Worker{
public:
  Worker(std::string n):name(n){};

    void operator() (){
      for (int i = 1; i <= 3; ++i){
        // begin work
        std::this_thread::sleep_for(std::chrono::milliseconds(200));
        // end work
        coutMutex.lock();
        std::cout << name << ": " << "Work " << i << " done !!!" << std::endl;
        coutMutex.unlock();
      }
    }
private:
  std::string name;
};


int main(){

  std::cout << std::endl;

  std::cout << "Boss: Let's start working." << "\n\n";

  std::thread herb= std::thread(Worker("Herb"));
```

```cpp
    std::thread andrei= std::thread(Worker("  Andrei"));
    std::thread scott= std::thread(Worker("    Scott"));
    std::thread bjarne= std::thread(Worker("      Bjarne"));
    std::thread bart= std::thread(Worker("        Bart"));
    std::thread jenne= std::thread(Worker("          Jenne"));

    herb.join();
    andrei.join();
    scott.join();
    bjarne.join();
    bart.join();
    jenne.join();

    std::cout << "\n" << "Boss: Let's go home." << std::endl;

    std::cout << std::endl;

}
```

▷         💾  ↩  ⌞⌝

# Explanation #

- `std::cout` is protected by the `coutMutex` in line 8. A simple `lock()` in line 19 and the corresponding `unlock()` call in line 21 ensures that the workers won't scream all at once.

> 🔑 `std::cout` **is thread-safe**
>
> The C++11 standard guarantees that you must not protect `std::cout`. Each character will be written atomically. It is possible that more output statements like those in the example will interleave. This is only a visual issue; the program is well-defined. This remark is valid for all global stream objects. Insertion to and extraction from global stream objects ( `std::cout`, `std::cin`, `std::cerr`, and `std::clog` ) is thread-safe.
>
> To put it more formally: writing to `std::cout` is not a ***data race*** but a ***race condition***. This means that the result depends on the interleaving of threads.

Test your knowledge about the lesson with an exercise in the next lesson.