

- Solutions

In this lesson, we'll discuss solutions to the tasks of the previous lesson.

WE'LL COVER THE FOLLOWING ^

- Solution 1
 - Explanation
 - Solution 2

Solution 1

```
//lock.cpp

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

std::mutex coutMutex;

class Worker{
public:
    explicit Worker(const 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
            std::lock_guard<std::mutex> myCoutLock(coutMutex);
            std::cout << name << ": " << "Work " << i << " done !!!" << std::endl;
        }
    }
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"));
std::thread andrei= std::thread(Worker(" Andrei"));
std::thread scott= std::thread(Worker(" Scott"));
std::thread bjarne= std::thread(Worker(" Bjarne"));
std::thread andrew= std::thread(Worker(" Andrew"));
std::thread david= std::thread(Worker(" David"));

herb.join();
andrei.join();
scott.join();
bjarne.join();
andrew.join();
david.join();

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

std::cout << std::endl;
}

```



Explanation

- The `std::lock_guard myCoutLock` in line 20 locks the mutex `coutMutex` in its constructor and releases the `coutMutex` in its destructor.
- `myCoutLock` goes out of scope in line 22 and releases its underlying mutex `coutMutex`.

Solution 2



The code will take some time to execute.

```

//countDown.cpp

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

int main() {

    std::cout << std::endl;

    for (long i=10; i>0; --i) {
        std::cout << i << std::endl;
        std::this_thread::sleep_for (std::chrono::seconds(1));
    }
}

```



```
std::cout << std::endl;  
}
```



For further information:

- [std::lock_guard](#)
- [std::unique_lock](#)
- [std::shared_lock](#)

Let's move on to thread-safe initialization of data in the next lesson.