- Examples

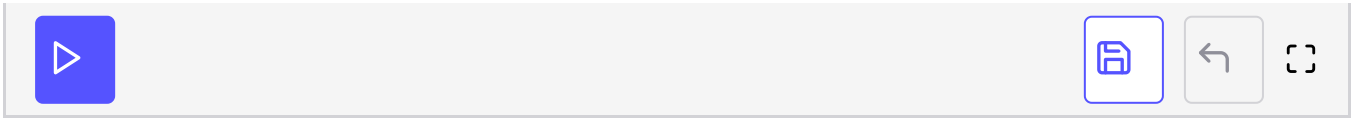
Examples for using locks in the scope of concurrency in C++.

WE'LL COVER THE FOLLOWING

Example 1

```
//lockGuard.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> myLock(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;

}
```



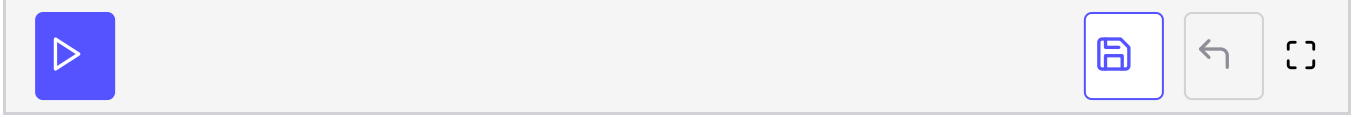
Explanation

- The program has six worker-threads (lines 36 - 41). Each worker-thread executes the function object `Worker`.
- `Worker` has three work packages. Before each work package, the worker sleeps 1/5 second (line 18) before he starts his work.
- When the `Worker` is done with the `i`-th work packages he notifies his boss (line 21).
- All threads exclusively notify their boss (line 21).

Example 2

```
//uniqueLock

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

struct CriticalData{
  std::mutex mut;
};
```

```
};

void deadLock(CriticalData& a, CriticalData& b){

  std::unique_lock<std::mutex>guard1(a.mut, std::defer_lock);
  std::cout << "Thread: " << std::this_thread::get_id() << " defer the locking of the first m

  std::this_thread::sleep_for(std::chrono::milliseconds(1));

  std::unique_lock<std::mutex>guard2(b.mut, std::defer_lock);
  std::cout << "Thread: " << std::this_thread::get_id() << " defer the locking of the second

  std::cout << "Thread: " << std::this_thread::get_id() << " locking them both atomically" <<
  std::lock(guard1, guard2);
  // do something with a and b
}

int main(){

  std::cout << std::endl;

  CriticalData c1;
  CriticalData c2;

  std::thread t1([&]{deadLock(c1, c2);});
  std::thread t2([&]{deadLock(c2, c1);});

  t1.join();
  t2.join();

  std::cout << std::endl;

}
```



Explanation

- In case, you call the constructor of `std::unique_lock` with the argument `std::defer_lock`, the lock will not be locked automatically in line 14 and 19.
- The lock operation is performed atomically in line 23 by using the variadic template `std::lock`. A variadic template is a template that can accept an arbitrary number of arguments. Here, the arguments are locks.
- `std::lock` tries to get all the locks in an atomic step. So, either it fails or gets all of them.
- In this example, `std::unique_lock` handles the lifetime of the resources, and `std::lock` locks the associated mutex. This can also be done in reverse. In the first step, you lock the mutexes, and in the second step,

`std::unique_lock` handles the lifetime of resources.

- When you remove `std::defer_lock` in line 14 and 19 and the `std::lock` call in line 23, you will likely get a deadlock.

Level up your understanding of this topic with a few exercises in the next lesson.