CppMem: Locks

This lesson gives an overview of locks used in the context of CppMem.

Both threads - `thread1` and `thread2` - use the same mutex, and they're wrapped in a `std::lock_guard`.

```cpp
// ongoingOptimisationLock.cpp

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

int x = 0;
int y = 0;

std::mutex mut;

void writing(){
  std::lock_guard<std::mutex> guard(mut);
  x = 2000;
  y = 11;
}

void reading(){
  std::lock_guard<std::mutex> guard(mut);
  std::cout << "y: " << y << " ";
  std::cout << "x: " << x << std::endl;
}

int main(){
  std::thread thread1(writing);
  std::thread thread2(reading);
  thread1.join();
  thread2.join();
};
```

The program is well-defined. Depending on the execution order (thread1 vs thread2), either both values are first read and then overwritten, or they're first overwritten and then read. The following values for `x` and `y` are possible.

y	x	Values possible?
0	0	Yes
11	0	
0	2000	
11	2000	Yes

Using std::lock_guard in CppMem

I could not find a way to use `std::lock_guard` in CppMem. If you know how to achieve this, please let me know.

Locks are easy to use but the synchronization is often too heavyweight. I will now switch to a more lightweight strategy and will use atomics.