Introduction to CppMem

This lesson gives an introduction to the case study of ongoing optimization with CppMem.

I will start with a small program and successively improve it, then verify each step of my process with CppMem. CppMem is an interactive tool for exploring the behavior of small code snippets using the C++ memory model.

First, here is the small program.

```cpp
// ongoingOptimisation.cpp

#include <iostream>
#include <thread>

int x = 0;
int y = 0;

void writing(){
  x = 2000;
  y = 11;
}

void reading(){
  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 quite simple. It consists of the two threads `thread1` and `thread2`. `thread1` writes the values `x` and `y`. `thread2` reads the values `x` and `y` in the **opposite sequence**. This sounds straightforward, but even in this simple program we can get different results if we run it several times.

I have two questions in mind for my process of ongoing optimization:

1. Is the program well-defined? In particular: is there a data race?
2. Which values for `x` and `y` are possible?

The first question is often very challenging to answer. In the first step, I will think about the answer to the first question and in the second step, I will verify my reasoning with CppMem. Once I have answered the first question, the second answer can easily be determined from the first. I will also present the possible values for `x` and `y` in a table.

But still, I haven't explained what I mean by ongoing optimization. It's pretty simple; I will successively optimize the program by weakening the C++ memory model. These are my optimization steps:

- Non-atomic variables
- Locks
- Atomics with sequential consistency
- Atomics with acquire-release semantic
- Atomics with relaxed semantic
- Volatile variables

Before I start my process of ongoing optimization, there is a short detour I have to make. I have to introduce CppMem in the next lesson.