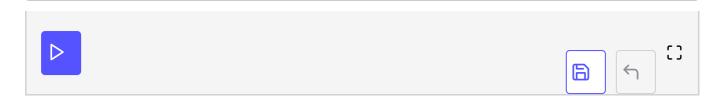
Data Races

This lesson gives an overview of data race problems which might occur during the implementation of concurrency in C++.

A data race is a situation in which at least two threads access a shared variable at the same time. Within that, at least one thread tries to modify the variable.

If your program has a data race, it will have undefined behavior. This means all outcomes are possible and, therefore, reasoning about the program makes no sense anymore. Let me show you a program with a data race.

```
// addMoney.cpp
#include <functional>
#include <iostream>
#include <thread>
#include <vector>
struct Account{
 int balance{100};
};
void addMoney(Account& to, int amount){
  to.balance += amount;
int main(){
  std::cout << std::endl;</pre>
 Account account;
  std::vector<std::thread> vecThreads(100);
  for (auto& thr: vecThreads) thr = std::thread(addMoney, std::ref(account), 50);
  for (auto& thr: vecThreads) thr.join();
  std::cout << "account.balance: " << account.balance << std::endl;</pre>
  std::cout << std::endl;</pre>
```



100 threads are adding 50 euros (line 25) to the same account (line 20). They use the function addMoney. The key observation is that the writing to the account is done without synchronization, therefore we have a data race and the result is not valid. This is undefined behavior and the final balance (line 30) differs between 5000 and 5100 euro.