- Exercises

In this exercise, you will spot a data race and synchronize the code.

Task 1

In the code below, vary the sleep time of `Sleeper` to prevent undefined behavior.

> Variation in the runtime with no synchronization results in ➡ undefined behavior.

The program has one serious issue: it has a data race and, therefore, undefined behavior. In simple words, a data race is a situation in which you have concurrent, non-synchronized read and write access to data. Do you spot the data race?

```
// threadArguments.cpp
#include <chrono>
#include <iostream>
#include <thread>

class Sleeper{
  public:
    Sleeper(int& i_):i{i_}{};
    void operator() (int k){
      for (unsigned int j= 0; j <= 5; ++j){
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        i += k;
      }
      std::cout << std::this_thread::get_id();
    }
  private:
    int& i;
};


int main(){

  std::cout << std::endl;

  int valSleeper= 1000;
  std::thread t(Sleeper(valSleeper), 5);
  t.detach();
  std::cout << "valSleeper = " << valSleeper << std::endl;

  std::cout << std::endl;

}
```

Task 2

Get the `std::thread::hardware_concurrency()` onto your system.

> 💡 Use the necessary headers.

```
#include <iostream>

int main() {
  // your code goes here
  std::cout << "Hello World";
  return 0;
}
```

The solutions to these tasks can be found in the next lesson.