- Solution

In this lesson, we'll discuss the solution to the exercise from the previous lesson.

WE'LL COVER THE FOLLOWING

Solution

```
#include <chrono>
#include <iostream>
#include <thread>

class Sleeper{

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


int main(){

  std::cout << std::endl;

  for (unsigned int i = 0; i <= 20; ++i){

    int valSleeper = 1000;
    std::thread t(Sleeper(valSleeper, (i*50)), 5);
    t.detach();
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    std::cout << "valSleeper = " << valSleeper << std::endl;

  }

  std::cout << std::endl;
```

```
}
```

Explanation

The question is what value does the variable valSleeper have in line 31?

`valSleeper` is a shared variable. Also, thread `t` gets a function object with the variable `valSleeper` and the number 5 (line 28) as its work package. The crucial observation to make here is that the thread gets valSleeper by reference (line 9) and will be detached from the main thread (line 29). Next, it will execute the call operator of the function object (lines 9 - 14). In this method, it counts from 0 to 5, sleeps in each iteration (lines 25 - 33) for an increasing amount of time, and increments `i` by `k`. In the end, it displays its id on the screen. Nach Adam Riese (a German proverb), the result should be 1000 + 6 * 5 = 1030 in each iteration.

But what happened? Something is going very wrong. Due to the variation in time spent sleeping, most of the values between 1000 and 1030 appear. The program has a few serious issues. Because of the race conditions, the result of the program depends on the interleaving of the operations. The race condition is the cause of the data race. The reading of `valSleeper` in line 31 and the updating of `valSleeper` in line 12 happen without synchronization.

Fixing the data race is quite easy. We have to join the thread instead of detaching it. This means that either the created thread is writing `valSleeper` or the creator thread is reading `valSleeper`.

In the next lesson, we'll discuss the methods commonly used by threads in C++.