

Arguments of Threads: Undefined Behavior

This lesson gives an example of the undefined behavior caused by passing arguments improperly to threads in C++.

As discussed in the previous lesson, here is another example of undefined behavior caused by improper handling of thread arguments.

```
// 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() << std::endl;
    }
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;

}
```

The question is, what value does `valSleeper` have in line 29? `valSleeper` is a global variable. Also, thread `t` gets a function object with the variable

global variable. Also, thread `t` gets a function object with the variable `valSleeper` and the number 5 (line 27) 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 28). Next, it will execute the call operator of the function object (lines 10 - 16). In this method it counts from 0 to 5, sleeps in each iteration for 1/10 of a second, 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 $1000 + 6 * 5 = 1030$.

But what happened? Something is going very wrong. In the next lesson, we'll learn how to fix this issue.