Thread arguments

A thread is a [variadic template](https://en.wikipedia.org/wiki/Variadic_template). So it can get an arbitrary number of arguments.

But now to the difference between getting the argument by copy or by reference.

std::string s{"C++11"}

std::thread t([=]{ std::cout << s << std::endl;});

t.join();

std::thread t2([&]{ std::cout << s << std::endl;});

t2.detach()

What dangers are hidden in these lines? Thread t2 gets its string s by  reference and is afterwards detached from the lifetime of its creator. On the one hand the lifetime of the string is bound to the lifetime of the invocation context, and on the other hand the lifetime of the global object std::cout is bound the lifetime of the main thread. So it may happen that the lifetime of the string s or the lifetime of std::cout is shorter than the lifetime of the thread t2.

// 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;

}

There are two issues. On the one hand, valSleeper is 1000, on the other hand, the ID is missing on the console. So, that is undefined behaviour. The reason is that the lifetime of the main thread ends, before the child thread has performed its calculation or written its ID to std::cout.

In case, the main thread waits via t.join() until this child thread is done with its work, we get the expected result.