Passing Arguments to Threads

This lesson gives an overview of how to pass arguments to threads in C++, by copy, and by reference.

WE'LL COVER THE FOLLOWING

A thread, like any arbitrary function, can get its arguments by copy, by move, or by reference. `std::thread` is a variadic template which means that it takes an arbitrary number of arguments.

In the case where our thread gets its data by reference, we have to be extremely careful about the lifetime of the arguments; not doing so may result in undefined behavior.

Copy or Reference

Let's have a look at a small code snippet:

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

int main(){
  std::string s{"C++11"};

  std::thread t1([=]{ std::cout << s << std::endl;});
  t1.join();

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

The thread `t1` gets its argument by copy whereas thread `t2` gets it by reference.

> **Thread arguments by reference**
>
> To be honest, we cheated a little. Note that the thread `t2` gets its argument by reference, and the lambda function captures its argument by reference as well. If we need to pass the argument to a thread by reference, it must be wrapped in a reference wrapper. This is quite straightforward with the helper function std::ref.
>
> ```
> void transferMoney(int amount, Account& from, Account& to){
>    ...
> }
> ...
> std::thread thr1(transferMoney, 50, std::ref(account1), std::ref(ac
> count2));
> ```

Thread `thr1` executes the function `transferMoney` and `transferMoney` gets its arguments by reference; therefore, thread `thr1` gets its `account1` and `account2` by reference.

Further information

- variadic template
- reference wrapper
- std::ref

Not convinced? Let's take a closer look at what undefined behavior may look like in the next lesson.