Introduction to Promises and Futures

This lesson gives an introduction to std::promise and std::future which are used in C++ for multithreading.

```
we'll cover the following ↑

std::promise

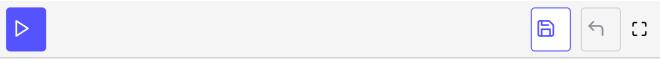
std::future
```

Promise and future are a mighty pair. A promise can put a value, an exception, or simply a notification into the shared data channel. One promise can serve many std::shared_future futures. With C++20 we may get extended futures that are compose-able.

Here is an introductory example of the usage of std::promise and std::future. Both communication endpoints can be moved to separate threads, so the communication takes place between threads.

```
// promiseFuture.cpp
#include <future>
#include <iostream>
#include <thread>
#include <utility>
void product(std::promise<int>&& intPromise, int a, int b){
  intPromise.set_value(a*b);
}
struct Div{
  void operator() (std::promise<int>&& intPromise, int a, int b) const {
    intPromise.set_value(a/b);
  }
};
int main(){
  int a = 20;
  int b = 10;
```

```
std::cout << std::endl;</pre>
  // define the promises
  std::promise<int> prodPromise;
  std::promise<int> divPromise;
  // get the futures
  std::future<int> prodResult = prodPromise.get_future();
  std::future<int> divResult = divPromise.get_future();
  // calculate the result in a separate thread
  std::thread prodThread(product, std::move(prodPromise), a, b);
  std::thread divThread(div, std::move(divPromise), a, b);
  // get the result
  std::cout << "20*10 = " << prodResult.get() << std::endl;</pre>
  std::cout << "20/10 = " << divResult.get() << std::endl;</pre>
  prodThread.join();
  divThread.join();
  std::cout << std::endl;</pre>
}
```



Thread prodThread (line 36) gets the function product (lines 8-10), the prodPromise (line 32) and the numbers a and b. To understand the arguments of prodThread, you have to look at the signature of the function. prodThread needs as its first argument a callable; this is the previously mentioned function product. The function product requires a promise of the kind rvalue reference (std::promise<int>&& intPromise) and two numbers. These are the last three arguments of prodThread. std::move in line 36 creates an rvalue reference - and the rest is a piece of cake. divThread (line 38) divides the two numbers a and b. For its job, it uses the instance div of the class Div (lines 12 - 18). div is an instance of a function object.

The future picks up the results by calling prodResult.get() and divResult.get().

std::promise

std::promise enables you to set a value, a notification, or an exception. In

addition, the promise can provide its result in a delayed fashion.

Method	Description
prom.swap(prom2) and	Swaps the promises.
<pre>std::swap(prom, prom2)</pre>	
<pre>prom.get_future()</pre>	Returns the future.
<pre>prom.set_value(val)</pre>	Sets the value.
<pre>prom.set_exception(ex)</pre>	Sets the exception.
<pre>prom.set_value_at_thread_exit(val)</pre>	Stores the value and makes it ready if the promise exits.
<pre>prom.set_exception_at_thread_exit</pre>	Stores the exception and makes it ready if the promise exits.

If the value or the exception is set by the promise more then once, a std::future_error exception is thrown.

std::future

A std::future enables you to

- pick up the value from the promise.
- ask the promise if the value is available.
- wait for the notification of the promise. This waiting can be done with a relative time duration or an absolute time point.
- create a shared future (std::shared_future).

Method	Description
<pre>fut.share()</pre>	Returns a std::shared_future. Afterwards, the result is not available anymore.
<pre>fut.get()</pre>	Returns the result which can be a value or an exception.
<pre>fut.valid()</pre>	Checks if the result is available. After calling fut.get() it returns false.
<pre>fut.wait()</pre>	Waits for the result.
<pre>fut.wait_for(relTime)</pre>	Waits for the result, but not longer than for a relTime.
<pre>fut.wait_until(absTime)</pre>	Waits for the result, but not longer than until abstime.

If a future fut asks for the result more than once, a std::future_error exception is thrown.

There is a one-to-one relationship between the promise and the future. In contrast, std::shared_future supports one-to-many relations between a promise and many futures which we will discuss in the next lesson.