Thread Creation

A thread gets its callable (*thread of execution*) and starts immediately. It needs the header `<thread>`.

- A callable unit can be a
  - Function

    ```
    std::thread t(function);
    ```
  - Function object

    ```
    std::thread t(FunctionObject());
    ```
  - Lambda expression

    ```
    std::thread t([]{ std::cout << "I'm running\n"; });
    ```

`threadCreate.cpp`

Threads Lifetime

The creator must take care of the lifetime of its child. The lifetime of the thread ends with the end of the callable unit.

- The creator
  - Waits for his child `t`: `t.join();`
  - Detaches itself from its child `t`: `t.detach();` ➡ daemon thread
- A thread `t` is joinable if a call `t.join()` or `t.detach()` was not performed.

⚠ A joinable thread `t` calls in its destructor the exception `std::terminate()`.
➡ Program termination

Arguments of Threads

The thread should get its arguments by copy. Therefore, the validity of the data is ensured. A thread can get an arbitrary number of arguments.

- Transfer of arguments

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

  - By copy

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

  - By reference

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

The lambda expression gets in this example the data.

threadArguments.cpp

Operations of Threads

Function	Description
`t.joinable()`	Checks if the thread t supports join or detach.
`t.get_id(), std::this_thread::get_id()`	Returns the ID of the thread.
`std::thread::hardware_concurrency()`	Hint for the number of threads that can run in parallel.
`std::this_thread::sleep_until(abs_time)`	Puts the thread to sleep until the time point.
`std::this_thread::sleep_for(rel_time)`	Puts the thread to sleep for a time period.
`std::this_thread::yield()`	Offers the system to execute another thread.
`t.swap(t2), std::swap(t1, t2)`	Swaps the threads.

The arguments of the sleep methods are time objects.