OpenCL exercise 1: Basics

This first exercise will cover the basics of programming with OpenCL.

1 Syntax

On the host:

```
cl::Buffer::Buffer(cl::Context context, cl_mem_flags flags, std::size_t size);

cl::CommandQueue::enqueueWriteBuffer(cl::Buffer buffer, bool blocking,
    std::size_t offset, std::size_t size, const void* ptr,
    eventsToWaitFor = NULL, cl::Event* resultEvent = NULL) const;
cl::CommandQueue::enqueueReadBuffer(cl::Buffer buffer, bool blocking,
    std::size_t offset, std::size_t size, void* ptr,
    eventsToWaitFor = NULL, cl::Event* resultEvent = NULL) const;

cl::Kernel::setArg<T>(cl_uint index, T value);

cl::CommandQueue::enqueueNDRangeKernel(cl::Kernel kernel,
    cl::NDRange offset, cl::NDRange global, cl::NDRange local,
    eventsToWaitFor = NULL, cl::Event* event=NULL) const;

Core::TimeSpan Core::getCurrentTime();
Core::TimeSpan OpenCL::getElapsedTime(cl::Event event);
```

On the device:

```
__global int* foo; // Declare foo as a pointer to global memory
__local int* foo; // Declare foo as a pointer to local memory
__private int* foo; // Declare foo as a pointer to private memory
__constant int* foo; // Declare foo as a pointer to constant memory

size_t i = get_global_id(0); // Get global index of the current work item in the x-direction
```

2 Task 1: GPU implementation

The code in the template calculates a number of cosine values on the CPU. Add code for calculating the same values on the GPU, using one work item per value.
You have to write code for

- Allocating memory on the device (`cl::Buffer` constructor)
- Initializing the memory on the device (`cl::CommandQueue::enqueueWriteBuffer()`)
- Copying the input data to the device (`cl::CommandQueue::enqueueWriteBuffer()`)
- Launching the kernel (`cl::Kernel::setArg()`, `cl::CommandQueue::enqueueNDRangeKernel()`)
- The kernel itself (in `OpenCLExercise1_Basics.cl`, use `get_global_id(0)` to get ID of current work item)

- Copying the output data to the host (`cl::CommandQueue::enqueueReadBuffer()`)

Run the code and check whether the GPU results are correct.

3 Task 2: Profiling

Add code for measuring

- The time needed for performing the calculation on the host (use `Core::getCurrentTime()`)
- The time needed for performing the calculation on the device (use an event and `OpenCL::getElapsedTime()`)
- The time needed for memory transactions (use an event and `OpenCL::getElapsedTime()`)

Display the times and the speedup you achieve as

- Time on the CPU / Time on the GPU
- Time on the CPU / (Time on the GPU + Time for Memory transactions)

Compare the times using a Debug build and a Release build.

4 Task 3: `native_cos`

Use `native_cos()` instead of `cos()` in the kernel and compare the performance.