CUDA exercise 1: Basics

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

1 Memory

1.1 CUDA Syntax

```
cudaError_t cudaMalloc(T** pointer, size_t nbytes);
cudaError_t cudaMemset(void* pointer, int value, size_t count);
cudaError_t cudaFree(void* pointer);

cudaError_t cudaMemcpy(void* dst, void* src, size_t nbytes,
   enum cudaMemcpyKind direction);
enum cudaMemcpyKind {
   cudaMemcpyHostToDevice,
   cudaMemcpyDeviceToHost,
   cudaMemcpyDeviceToDevice
};
```

You can use the macro `CUDA_CHECK` from `helper_error.h` to check whether a cuda method failed:

```
CUDA_CHECK(cudaMalloc(..., ...));
```

1.2 Task

Three arrays of the same size are allocated (two on the host, one on the device). The first array on the host is initialized and copied to the array on the device. The device array is then copied to the second array on the host. Both arrays on the host are compared to check for errors. The memory is freed.
Complete `CudaExercise1_Basics.cu` using `cudaMalloc`, `cudaMemcpy` and `cudaFree`.

2 Kernel execution

2.1 CUDA Syntax

```
dim3 dimGrid(numBlocks); // 1D grid
dim3 dimGrid(numBlocksX, numBlocksY); // 2D grid
dim3 dimBlock(numThreadsPerBlock); // 1D block
dim3 dimBlock(numThreadsPerBlockX, numThreadsPerBlockY); // 2D block

__global__ myKernel(.....);
myKernel<<<dimGrid, dimBlock>>>(....);
```

You can use the macro `CUDA_CHECK_KERNEL` to check whether a kernel call failed:

```
CUDA_CHECK_KERNEL(kernel<<<dimGrid, dimBlock>>>(....));
```

2.2 Task

In the second part of the exercise all values of the array of integers are multiplied by 2 on the host and on the device. Add an invocation to the kernel `multiplyKernel` and use `CUDA_CHECK(cudaThreadSynchronize())` to wait until the computation on the GPU has finished.