```
// includes, system
#include <stdio.h>
#include <assert.h>

// Simple utility function to check for CUDA runtime errors
void checkCUDAError(const char *msg);

// Part 3 of 5: implement the kernel
__global__ void myFirstKernel(int *d_a)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    d_a[idx] = 1000*blockIdx.x + threadIdx.x;
}

////////////////////////////////////////////////////////////////////////////////
//
// Program main
////////////////////////////////////////////////////////////////////////////////
int main( int argc, char** argv)
{
    // pointer for host memory
    int *h_a;

    // pointer for device memory
    int *d_a;

    // define grid and block size
    int numBlocks = 8;
    int numThreadsPerBlock = 8;

    // Part 1 of 5: allocate host and device memory
    size_t memSize = numBlocks * numThreadsPerBlock * sizeof(int);
    h_a = (int *) malloc(memSize);
    cudaMalloc( (void **) &d_a, memSize );

    // Part 2 of 5: launch kernel
    dim3 dimGrid(numBlocks);
    dim3 dimBlock(numThreadsPerBlock);
    myFirstKernel<<< dimGrid, dimBlock >>>( d_a );

    // block until the device has completed
    cudaThreadSynchronize();

    // check if kernel execution generated an error
    checkCUDAError("kernel execution");

    // Part 4 of 5: device to host copy
    cudaMemcpy( h_a, d_a, memSize, cudaMemcpyDeviceToHost );

    // Check for any CUDA errors
    checkCUDAError("cudaMemcpy");

    // Part 5 of 5: verify the data returned to the host is correct
    for (int i = 0; i < numBlocks; i++)
    {
        for (int j = 0; j < numThreadsPerBlock; j++)
        {
            assert(h_a[i * numThreadsPerBlock + j] == 1000 * i + j);
        }
```

```
    }

    // free device memory
    cudaFree(d_a);

    // free host memory
    free(h_a);

    // If the program makes it this far, then the results are correct and
    // there are no run-time errors.  Good work!
    printf("Correct!\n");

    return 0;
}

void checkCUDAError(const char *msg)
{
    cudaError_t err = cudaGetLastError();
    if( cudaSuccess != err)
    {
        fprintf(stderr, "Cuda error: %s: %s.\n", msg, cudaGetErrorString(
err) );
        exit(-1);
    }
}
```