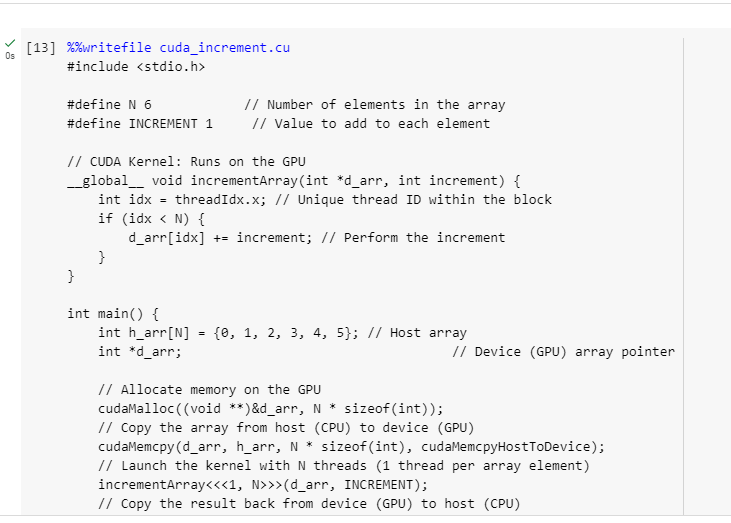
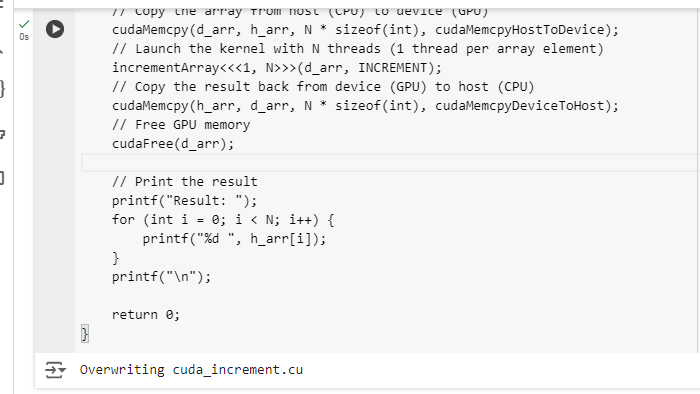
**Name: Qurat Ul Ain**

**Section: BSCS-VII-C**

**Roll no: 210958**

**Code:**

****

****

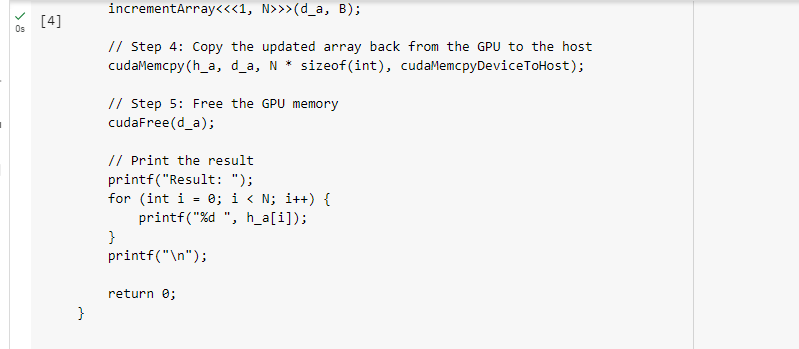
A screenshot of a computer

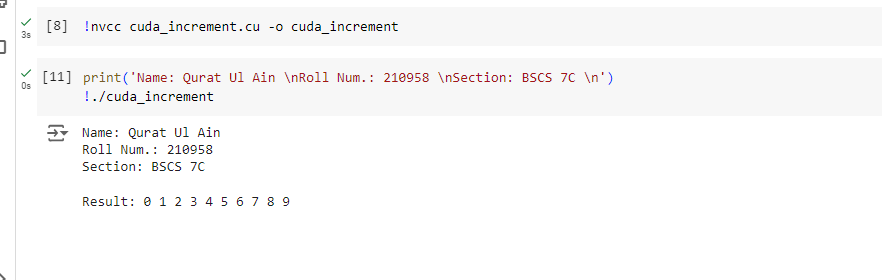
Description automatically generated

The code is entirely written in .cu and implements a simple function that performs an increment operation. GPU memory is allocated using malloc, ensuring that the necessary space is reserved on the GPU for processing. The Memcpy function is then used to transfer data and execute the increment operation on the GPU. Finally, the Free function is called to release the allocated GPU memory and clean up resources after the computation is complete.

**A screenshot of a computer program

Description automatically generated**

****

****

This code shows how CUDA can be used for parallel processing on the GPU. It has a kernel function called incrementArray that works on a device array (d\_arr) and an increment value. Each thread on the GPU handles one element of the array, and the thread ID (threadIdx.x) helps determine which element to process. The if statement makes sure the thread doesn’t go out of bounds when accessing the array.

In the main function, a simple array is created on the CPU (host), and GPU memory is allocated for it using cudaMalloc. The array is copied from the CPU to the GPU with cudaMemcpy. Then, the kernel is launched with N threads, where each thread increments one element of the array. After the computation finishes, the updated array is sent back to the CPU using cudaMemcpy. The GPU memory is freed with cudaFree, and the final result is printed on the screen. This example is a basic demonstration of how to use CUDA for parallel tasks.