University of Wolverhampton

School of Mathematics and Computer Science

Student Number: 2407710

Name: Dhadkan K.C.

6CS005 High Performance Computing Week 10 Workshop

Revision on CUDA

Tasks – CUDA Block, Grid and Thread

From what you have learnt in the Lecture, using only multiple blocks and threads, produce the following programs:

Revision on CUDA

Tasks – CUDA Block, Grid and Thread

From what you have learnt in the Lecture, using only multiple blocks and threads, produce the following programs:

1. The below cuda program attempts to launch a GPU kernel that calls a function to print "Hello World".

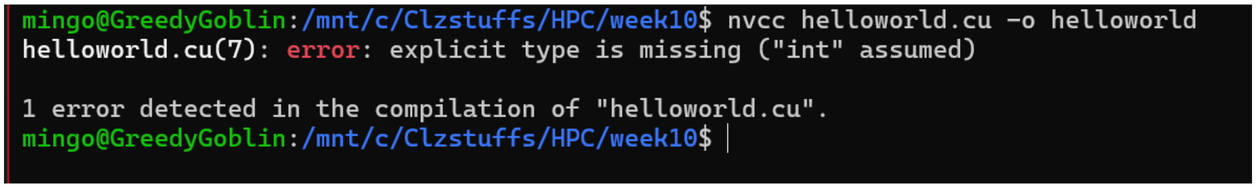
```
#include <stdio.h>
void displayHelloWorld(){
  printf("Hello World\n");
}
__global__  display(){
      displayHelloWorld();
}
int main(){
  display<<<1,1>>>();
  cudaDeviceSynchronize();
```

    return 0;

}

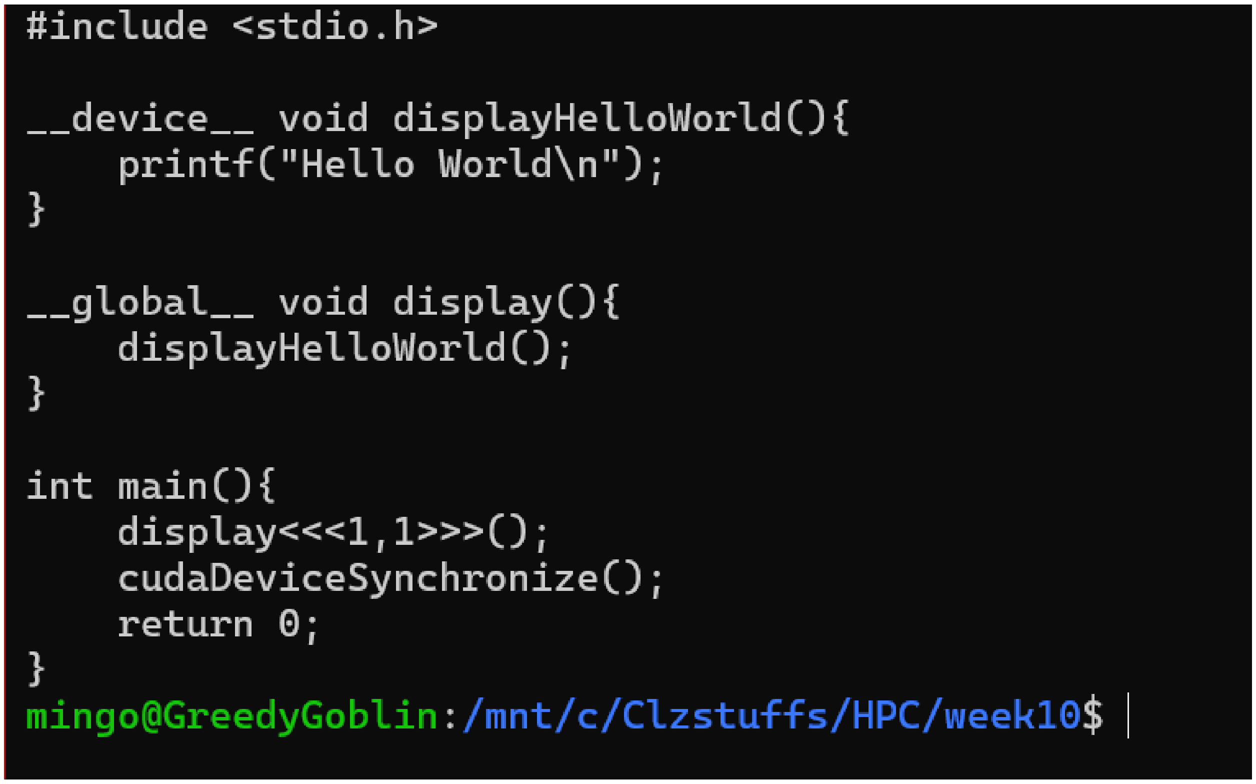
a.   Does the code work ?

Ans: No this code did not work. It did not worked because the function type of kernel is not given.



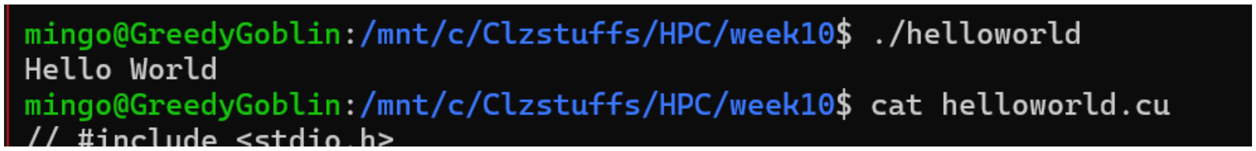
```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc helloworld.cu -o helloworld
helloworld.cu(7): error: explicit type is missing ("int" assumed)

1 error detected in the compilation of "helloworld.cu".
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

b. Modify the code such that the function displayHelloWorld is callable inside the kernel.



```
#include <stdio.h>

__device__ void displayHelloWorld(){
    printf("Hello World\n");
}

__global__ void display(){
    displayHelloWorld();
}

int main(){
    display<<<1,1>>>();
    cudaDeviceSynchronize();
    return 0;
}
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

kernel function display was missing the __global__ return type (void), it must be declared as __global__ void display().



```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./helloworld
Hello World
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ cat helloworld.cu
// #include <stdio.h>
```

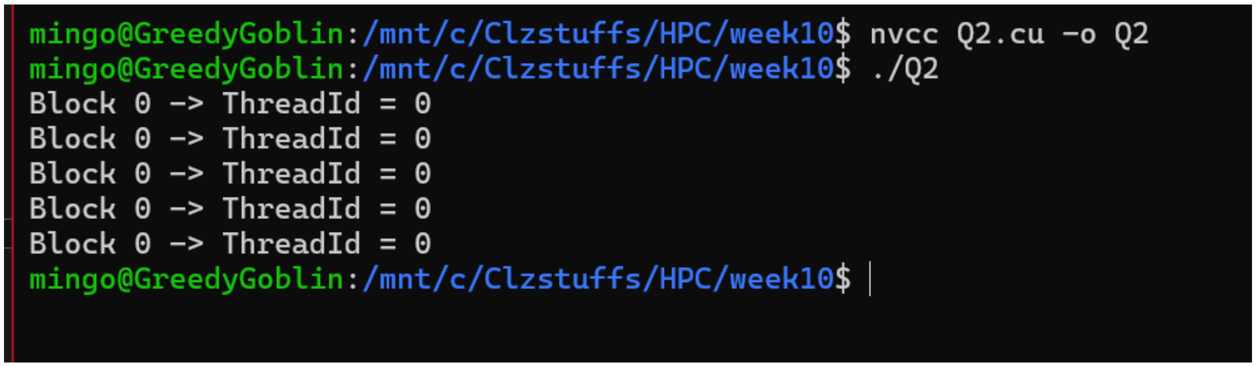
2. Run and observe the output of the following code:

```
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void display() {
   int threadID =threadIdx.y;
   int blockID = blockIdx.x;
   printf("Block %d -> ThreadId = %d\n", blockID, threadID);
}
int main() {
   dim3 gridSize(1, 1, 1);
   dim3 blockSize(5, 1, 1);
   display<<<gridSize, blockSize>>>();
   cudaDeviceSynchronize();
   return 0;
}
```

Expected Output:

```
Block 0 -> ThreadId = 0
Block 0 -> ThreadId = 1
Block 0 -> ThreadId = 2
Block 0 -> ThreadId = 3
Block 0 -> ThreadId = 4
```

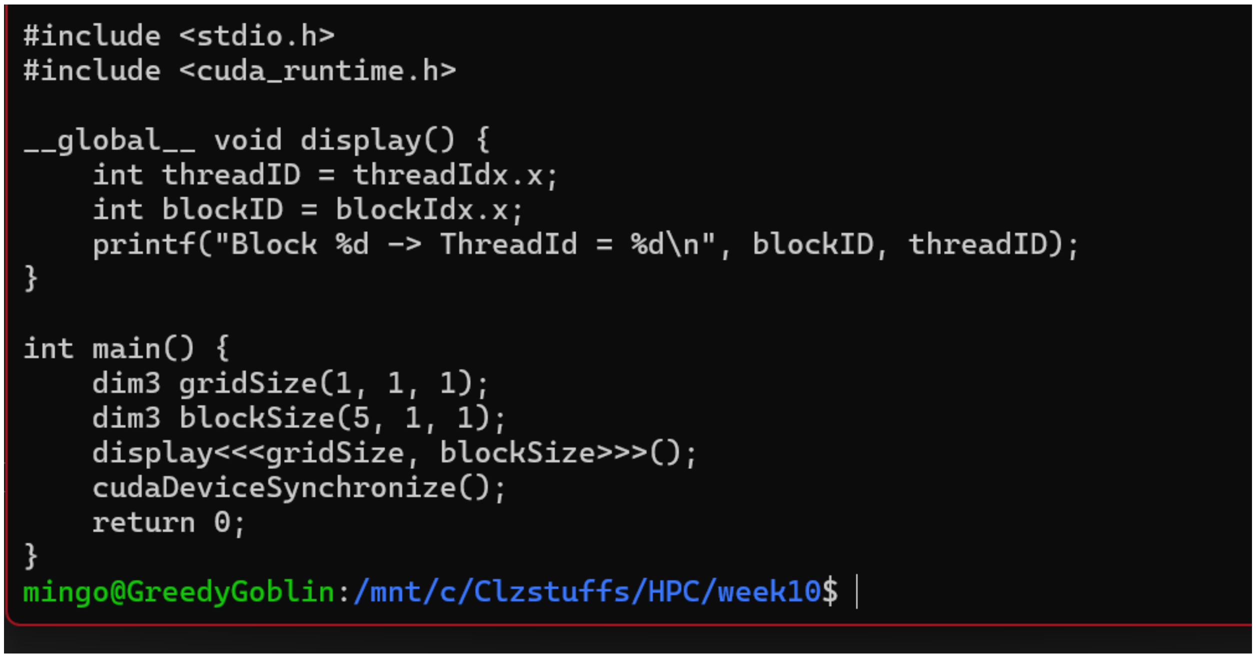
a. Does the actual output match the expected output? Identify the problem.



```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc Q2.cu -o Q2
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./Q2
Block 0 -> ThreadId = 0
Block 0 -> ThreadId = 0
Block 0 -> ThreadId = 0
Block 0 -> ThreadId = 0
Block 0 -> ThreadId = 0
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

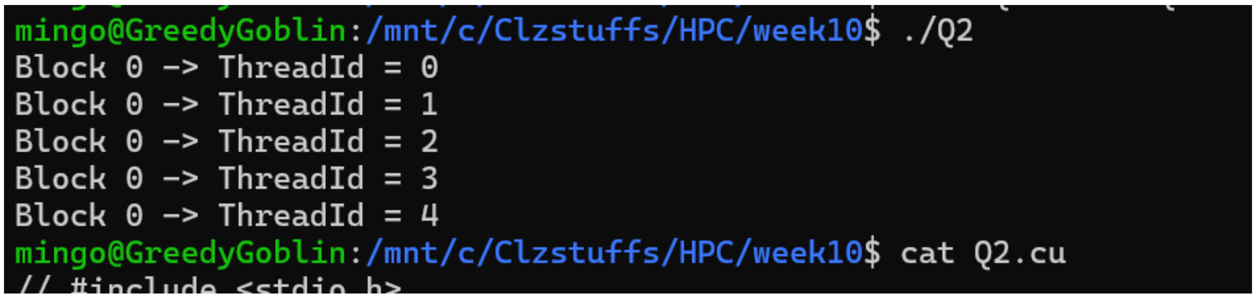
Ans : No, The actual output does not match the expected output because the block size (5,1,1) creates 5 threads in the x-direction and only 1 thread in the y-direction, but the code uses threadIdx.y to get the thread ID. Since there is only one thread in the y-direction, threadIdx.y is always 0, so every thread prints 0.

b. Modify the code so that it produces the correct output and explain your changes.



```
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void display() {
    int threadID = threadIdx.x;
    int blockID = blockIdx.x;
    printf("Block %d -> ThreadId = %d\n", blockID, threadID);
}

int main() {
    dim3 gridSize(1, 1, 1);
    dim3 blockSize(5, 1, 1);
    display<<<gridSize, blockSize>>>();
    cudaDeviceSynchronize();
    return 0;
}
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

Ans: The block is launched with 5 threads in the x-direction and only 1 thread in the y-direction, so threadIdx.y is always 0. Changing threadID to threadIdx.x correctly accesses the x-direction thread index, producing thread IDs from 0 to 4 as expected.



```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./Q2
Block 0 -> ThreadId = 0
Block 0 -> ThreadId = 1
Block 0 -> ThreadId = 2
Block 0 -> ThreadId = 3
Block 0 -> ThreadId = 4
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ cat Q2.cu
// #include <stdio.h>
```

3. In the code above, the block dimension is specified as **dim3 blockSize(5, 1, 1)**. Which means that the block has:

- 5 threads in the x-dimension
- 1 thread in the y-dimension
- 1 thread in the z-dimension

This typically means that the block we created is a **1D block** with 5 threads in the x-direction.

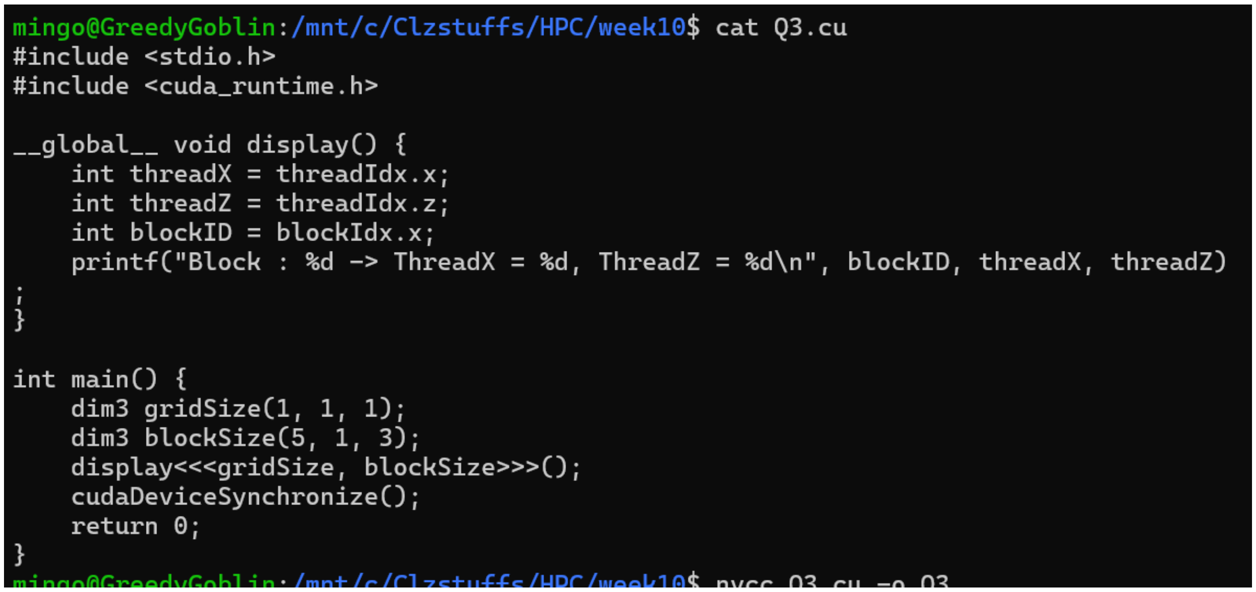
a. Modify the code so that it creates a 2D block instead of a 1D block. Also print thread ids along its corresponding directions/axis. For example if you are using a 2D block specified as **dim3 blockSize(5, 1, 3)** print thread ids along **x** and **z** direction/axis similar to the one given below:

```
Block : 0 -> ThreadX = 0, ThreadZ = 0

Block : 0 -> ThreadX = 1, ThreadZ = 0

Block : 0 -> ThreadX = 2, ThreadZ = 0

Block : 0 -> ThreadX = 3, ThreadZ = 0

Block : 0 -> ThreadX = 4, ThreadZ = 0

Block : 0 -> ThreadX = 0, ThreadZ = 1

Block : 0 -> ThreadX = 1, ThreadZ = 1

Block : 0 -> ThreadX = 2, ThreadZ = 1

Block : 0 -> ThreadX = 3, ThreadZ = 1

Block : 0 -> ThreadX = 4, ThreadZ = 1

Block : 0 -> ThreadX = 0, ThreadZ = 2

Block : 0 -> ThreadX = 1, ThreadZ = 2
```

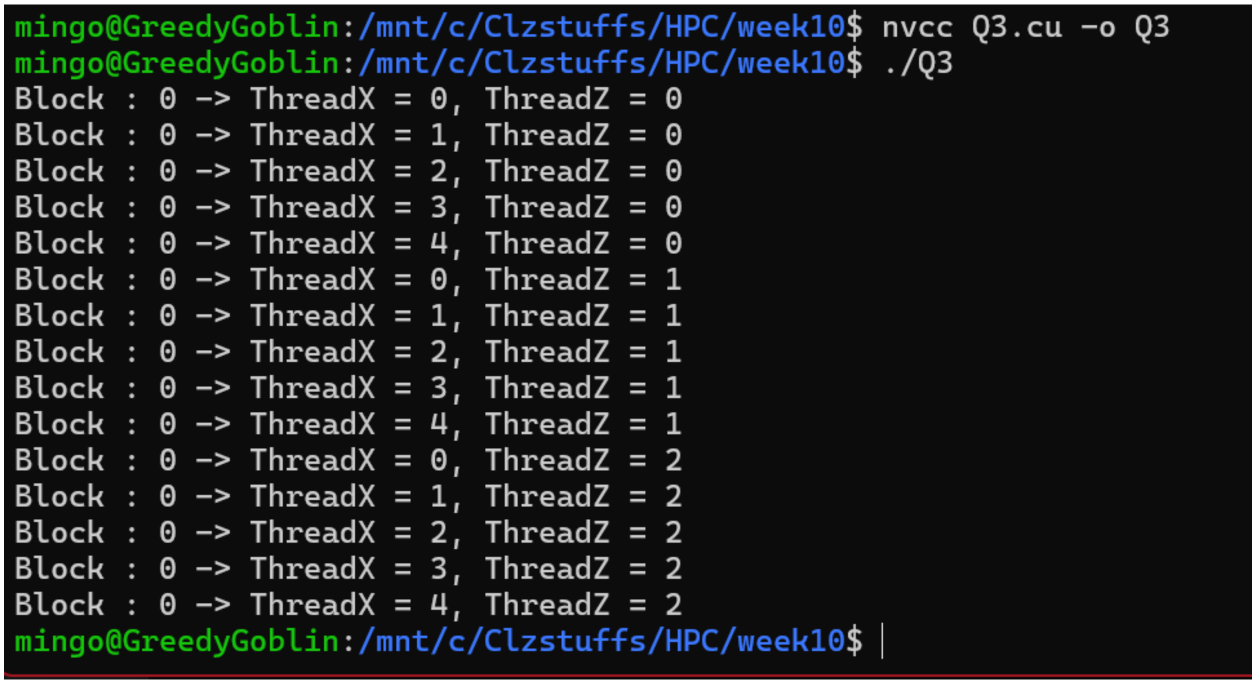
Block : 0 -> ThreadX = 2, ThreadZ = 2

Block : 0 -> ThreadX = 3, ThreadZ = 2

Block : 0 -> ThreadX = 4, ThreadZ = 2

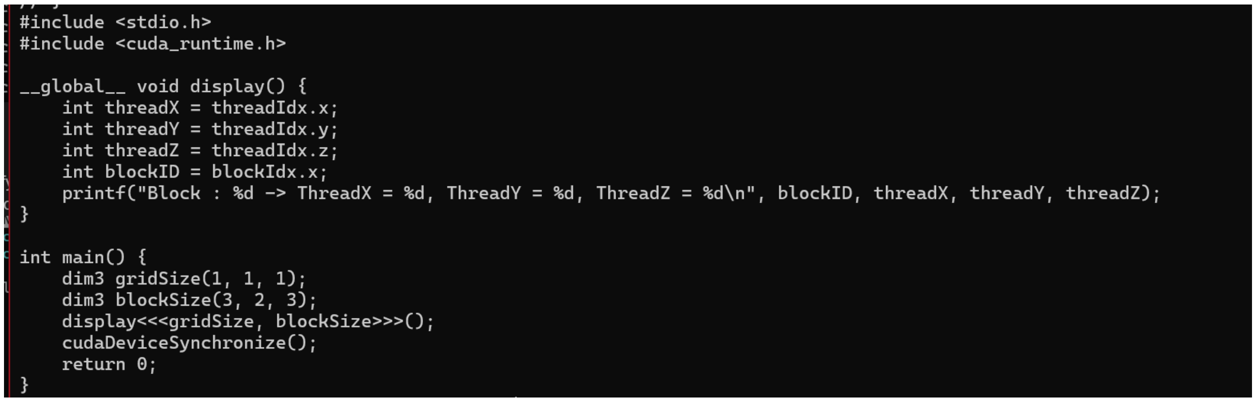


```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ cat Q3.cu
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void display() {
    int threadX = threadIdx.x;
    int threadZ = threadIdx.z;
    int blockID = blockIdx.x;
    printf("Block : %d -> ThreadX = %d, ThreadZ = %d\n", blockID, threadX, threadZ)
;
}

int main() {
    dim3 gridSize(1, 1, 1);
    dim3 blockSize(5, 1, 3);
    display<<<gridSize, blockSize>>>();
    cudaDeviceSynchronize();
    return 0;
}
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc Q3.cu -o Q3
```

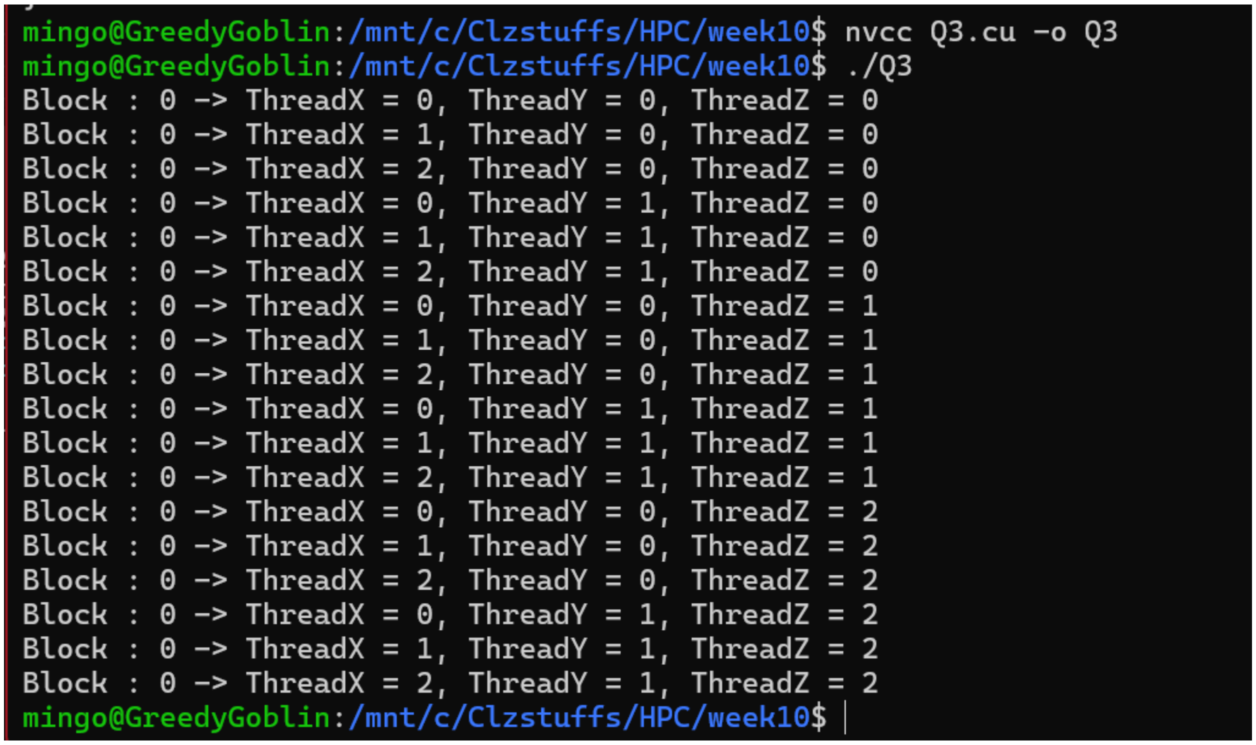


```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc Q3.cu -o Q3
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./Q3
Block : 0 -> ThreadX = 0, ThreadZ = 0
Block : 0 -> ThreadX = 1, ThreadZ = 0
Block : 0 -> ThreadX = 2, ThreadZ = 0
Block : 0 -> ThreadX = 3, ThreadZ = 0
Block : 0 -> ThreadX = 4, ThreadZ = 0
Block : 0 -> ThreadX = 0, ThreadZ = 1
Block : 0 -> ThreadX = 1, ThreadZ = 1
Block : 0 -> ThreadX = 2, ThreadZ = 1
Block : 0 -> ThreadX = 3, ThreadZ = 1
Block : 0 -> ThreadX = 4, ThreadZ = 1
Block : 0 -> ThreadX = 0, ThreadZ = 2
Block : 0 -> ThreadX = 1, ThreadZ = 2
Block : 0 -> ThreadX = 2, ThreadZ = 2
Block : 0 -> ThreadX = 3, ThreadZ = 2
Block : 0 -> ThreadX = 4, ThreadZ = 2
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

b. Modify the code again so that it creates a 3D block instead of 2D block. Also print thread ids along its corresponding directions/axis.



```
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void display() {
    int threadX = threadIdx.x;
    int threadY = threadIdx.y;
    int threadZ = threadIdx.z;
    int blockID = blockIdx.x;
    printf("Block : %d -> ThreadX = %d, ThreadY = %d, ThreadZ = %d\n", blockID, threadX, threadY, threadZ);
}

int main() {
    dim3 gridSize(1, 1, 1);
    dim3 blockSize(3, 2, 3);
    display<<<gridSize, blockSize>>>();
    cudaDeviceSynchronize();
    return 0;
}
```



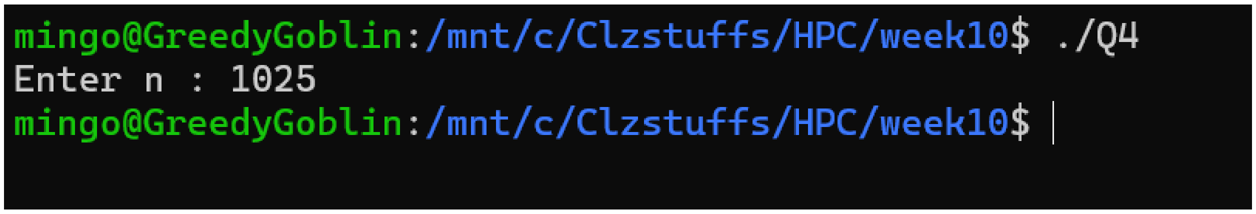
```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc Q3.cu -o Q3
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./Q3
Block : 0 -> ThreadX = 0, ThreadY = 0, ThreadZ = 0
Block : 0 -> ThreadX = 1, ThreadY = 0, ThreadZ = 0
Block : 0 -> ThreadX = 2, ThreadY = 0, ThreadZ = 0
Block : 0 -> ThreadX = 0, ThreadY = 1, ThreadZ = 0
Block : 0 -> ThreadX = 1, ThreadY = 1, ThreadZ = 0
Block : 0 -> ThreadX = 2, ThreadY = 1, ThreadZ = 0
Block : 0 -> ThreadX = 0, ThreadY = 0, ThreadZ = 1
Block : 0 -> ThreadX = 1, ThreadY = 0, ThreadZ = 1
Block : 0 -> ThreadX = 2, ThreadY = 0, ThreadZ = 1
Block : 0 -> ThreadX = 0, ThreadY = 1, ThreadZ = 1
Block : 0 -> ThreadX = 1, ThreadY = 1, ThreadZ = 1
Block : 0 -> ThreadX = 2, ThreadY = 1, ThreadZ = 1
Block : 0 -> ThreadX = 0, ThreadY = 0, ThreadZ = 2
Block : 0 -> ThreadX = 1, ThreadY = 0, ThreadZ = 2
Block : 0 -> ThreadX = 2, ThreadY = 0, ThreadZ = 2
Block : 0 -> ThreadX = 0, ThreadY = 1, ThreadZ = 2
Block : 0 -> ThreadX = 1, ThreadY = 1, ThreadZ = 2
Block : 0 -> ThreadX = 2, ThreadY = 1, ThreadZ = 2
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

4. The below code takes an integer **n** from the user and prints all numbers between **0 to n - 1** using the kernel function.

```
#include <stdio.h>

#include <cuda_runtime.h>

__global__ void displayIdx(){

   int tid = threadIdx.x;
```

```
    printf("%d\n", tid);
}
int main(){
    int n;
    printf("Enter n : ");
    scanf("%d", &n);
    dim3 gridSize(1, 1, 1);
    dim3 blockSize(n, 1, 1);
    displayIdx<<<gridSize, blockSize>>>();
    cudaDeviceSynchronize();
    return 0;
}
```

a. Does the code work when the user enters n as 1025 ? Explain.



```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./Q4
Enter n : 1025
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

Ans: No the code does not work. This is the max number of threads to use in a block is 1024. blockSize.x=1025 exceeds 1024.

b. Does the code still work when changing the dim3 **blockSize(n, 1, 1)** line of code to **dim3 blockSize(n, n, 1)** ? Modify the code so that the values are correctly printed out.

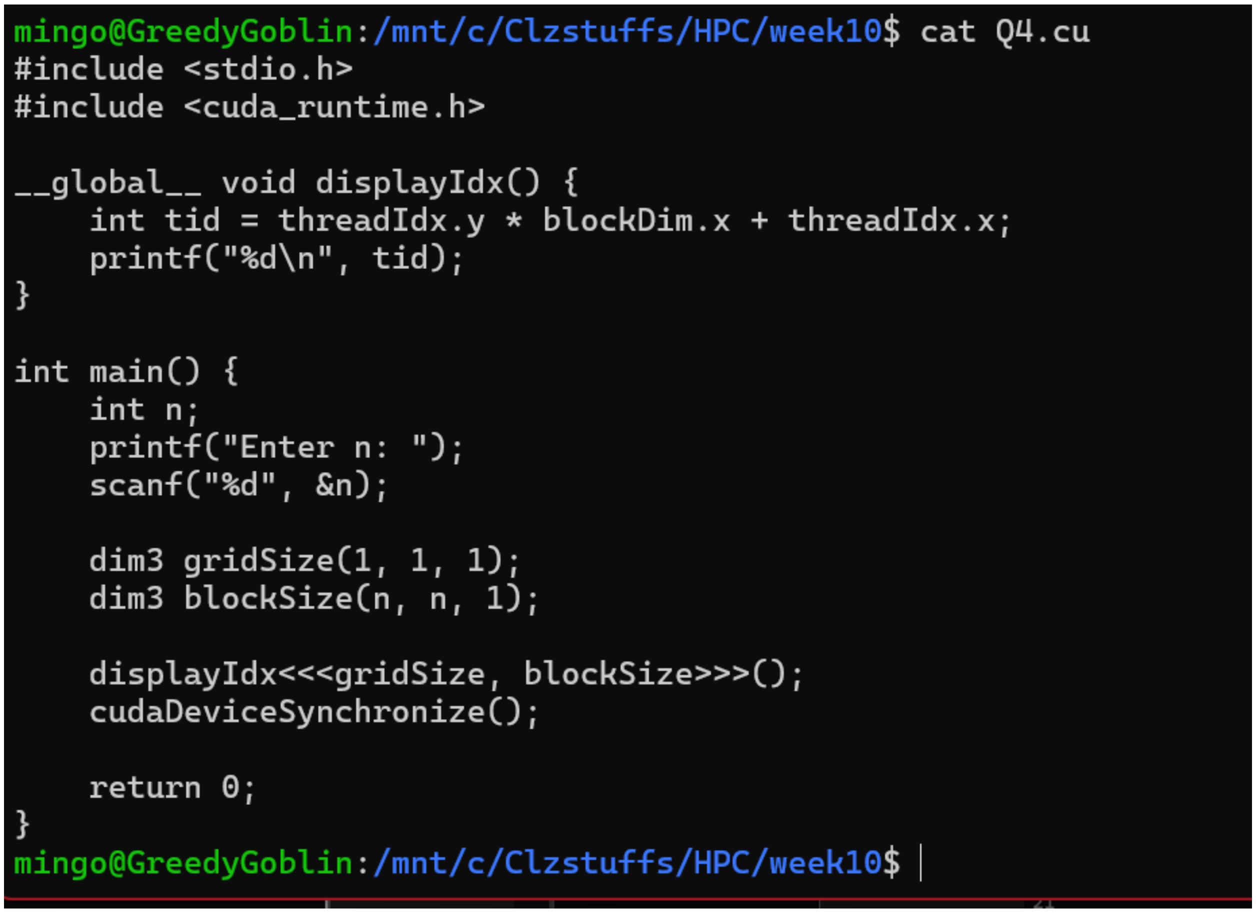
Hint:

```
__global__ void displayIdx(){
    int tid = threadIdx.x; // Try changing this line of code.
    printf("%d\n", tid);
```

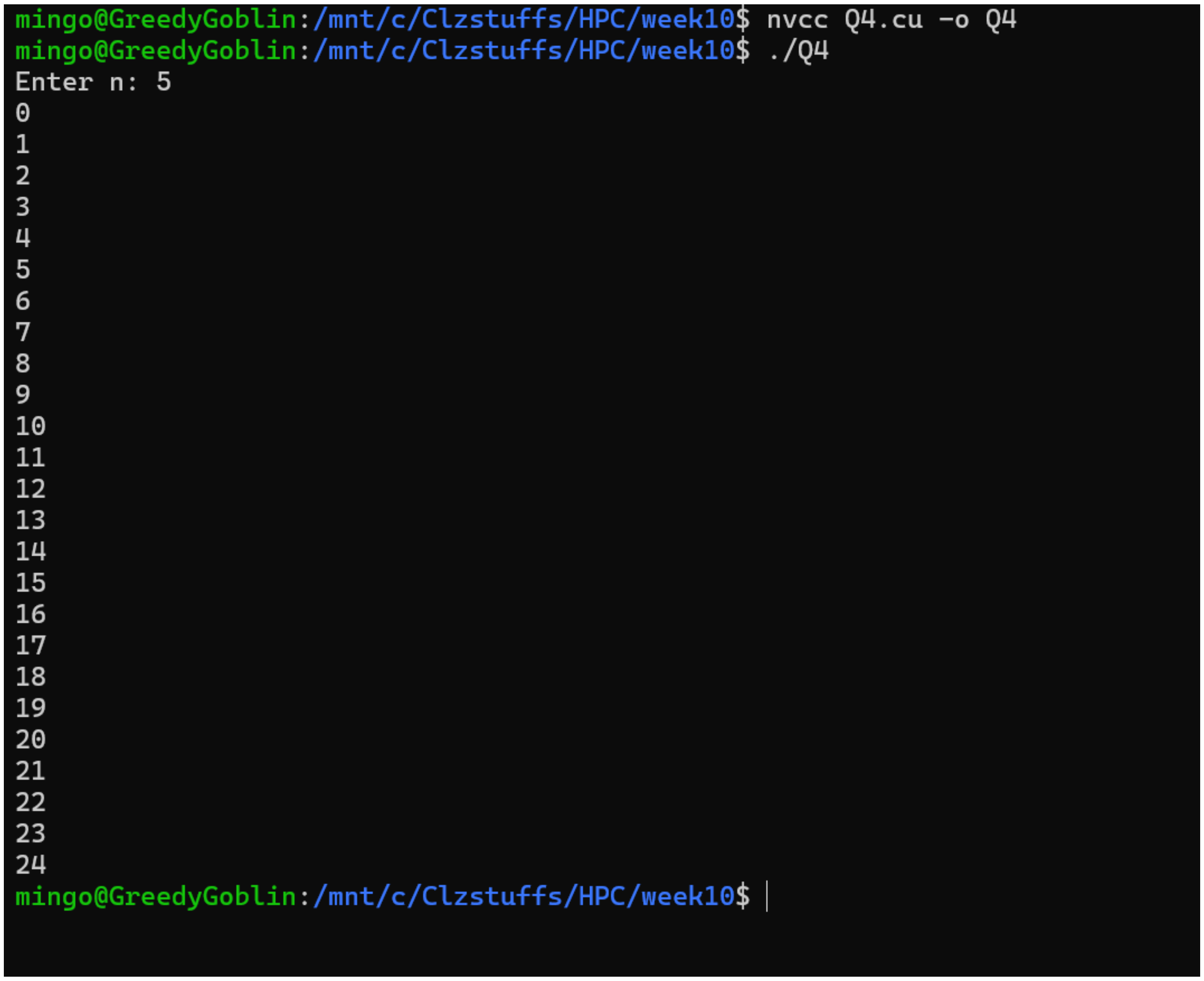
}

Note:

If n = 5, the code should print the value 0 to 24.

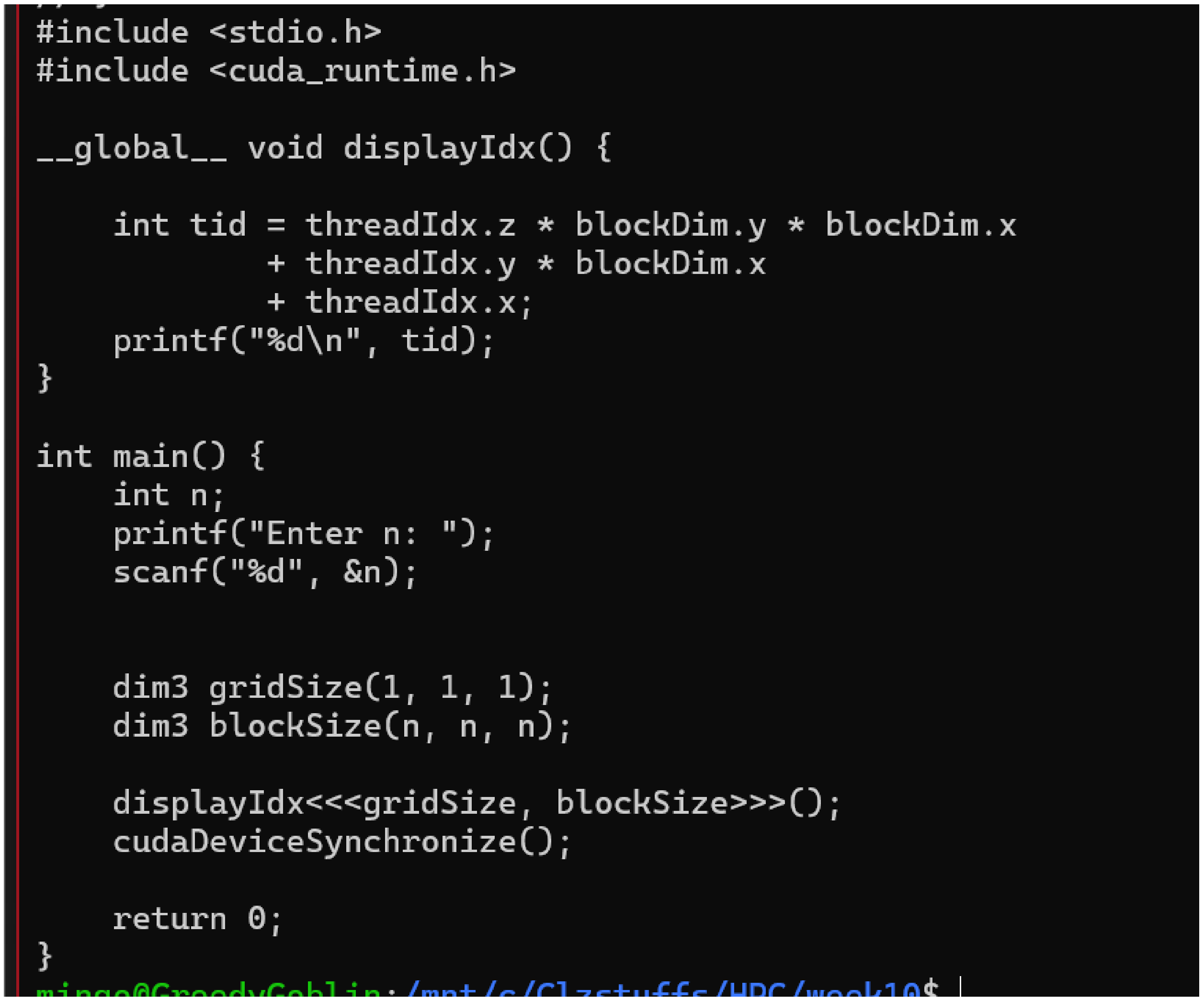


```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ cat Q4.cu
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void displayIdx() {
    int tid = threadIdx.y * blockDim.x + threadIdx.x;
    printf("%d\n", tid);
}

int main() {
    int n;
    printf("Enter n: ");
    scanf("%d", &n);

    dim3 gridSize(1, 1, 1);
    dim3 blockSize(n, n, 1);

    displayIdx<<<gridSize, blockSize>>>();
    cudaDeviceSynchronize();

    return 0;
}
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```

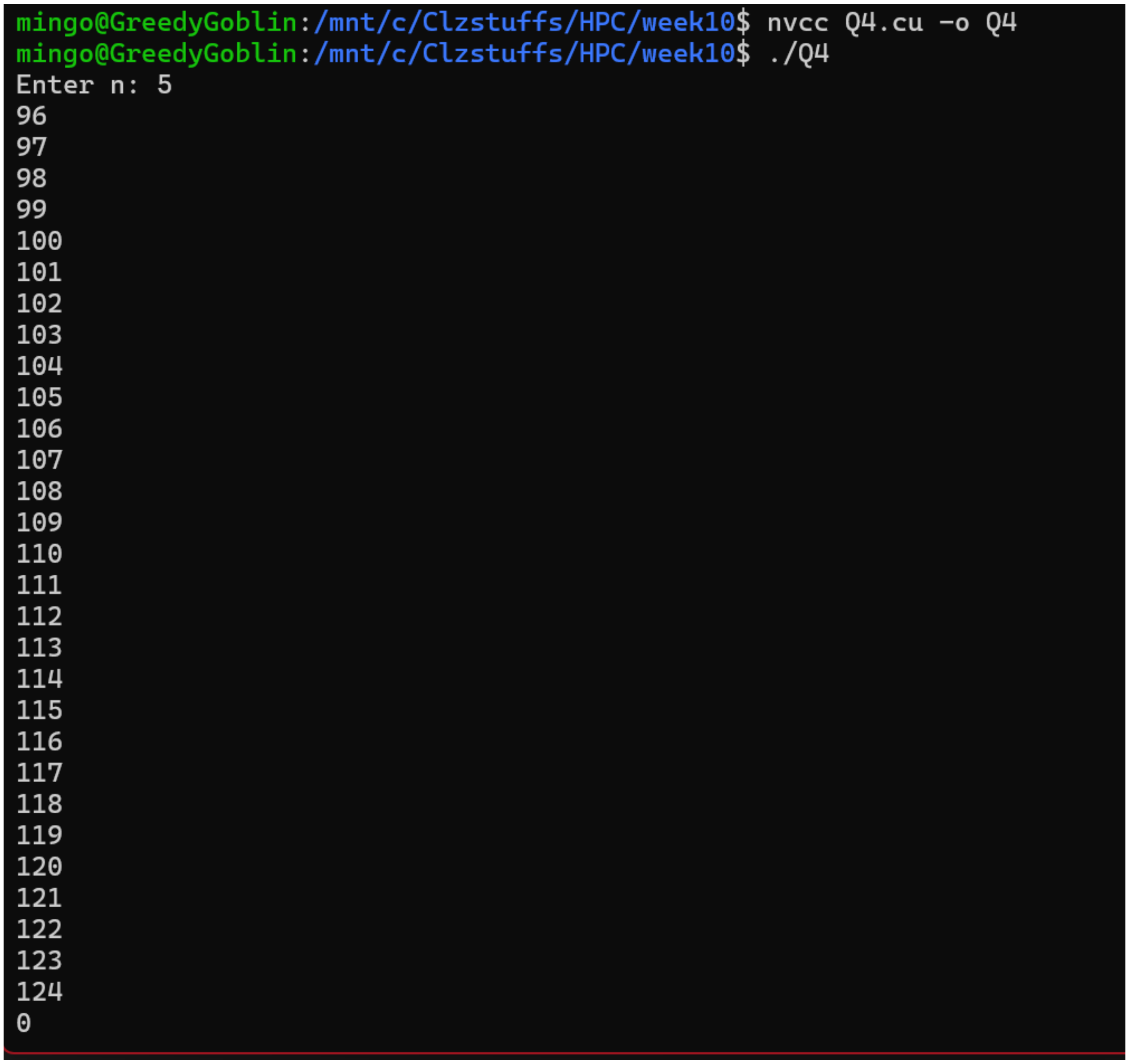


```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc Q4.cu -o Q4
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./Q4
Enter n: 5
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ 
```

c. Now change the **dim3 blockSize(n, n, 1)** to **dim3 blockSize(n, n, n)** and also modify the code so that it correctly prints out the value. Example if n = 5, the code should print the value 0 to 124.



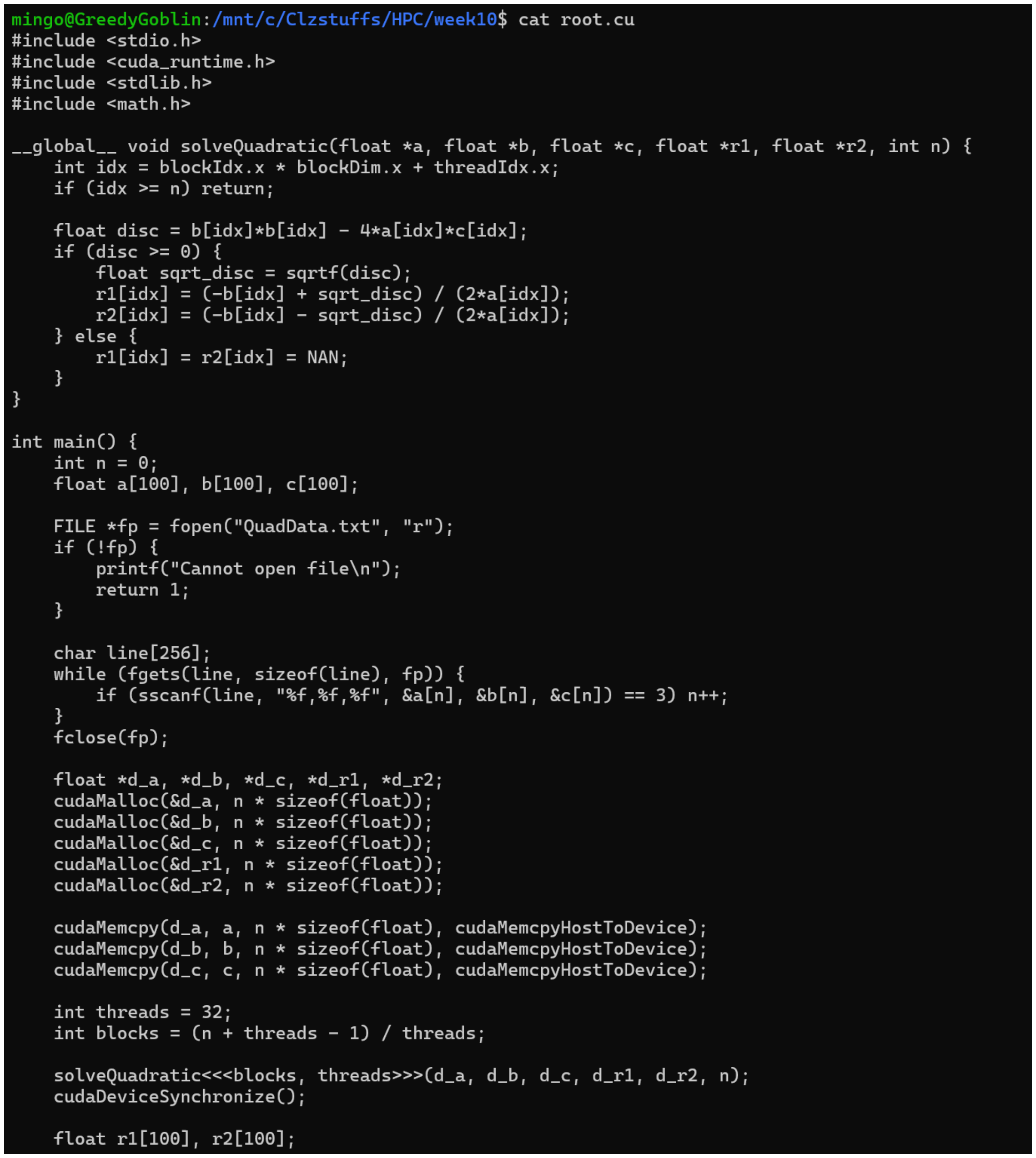
```
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void displayIdx() {

    int tid = threadIdx.z * blockDim.y * blockDim.x
            + threadIdx.y * blockDim.x
            + threadIdx.x;
    printf("%d\n", tid);
}

int main() {
    int n;
    printf("Enter n: ");
    scanf("%d", &n);


    dim3 gridSize(1, 1, 1);
    dim3 blockSize(n, n, n);

    displayIdx<<<gridSize, blockSize>>>();
    cudaDeviceSynchronize();

    return 0;
}
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$
```



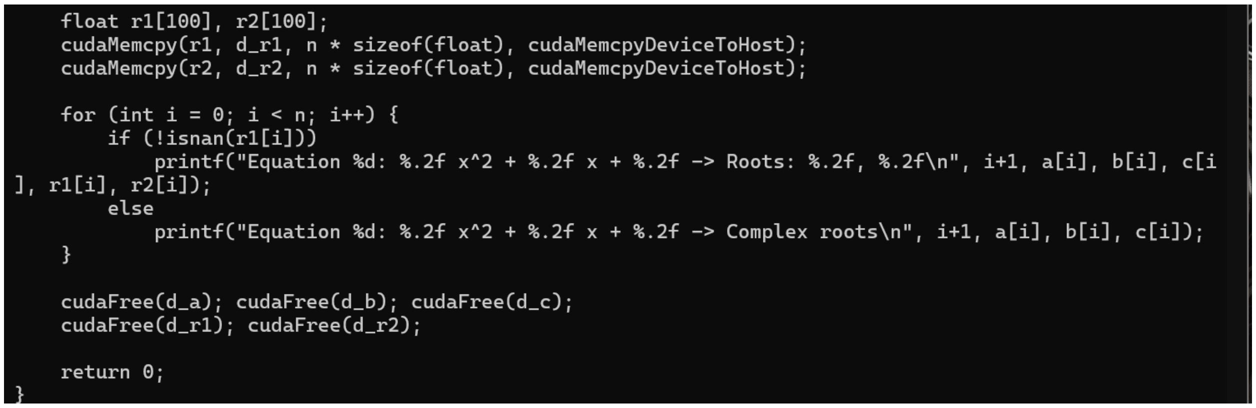
```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc Q4.cu -o Q4
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./Q4
Enter n: 5
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
0
```

6. Create a CUDA program to solve quadratics (find the roots). You will be given a text file containing 3 integers on each row which represent a, b and c for the following formula:

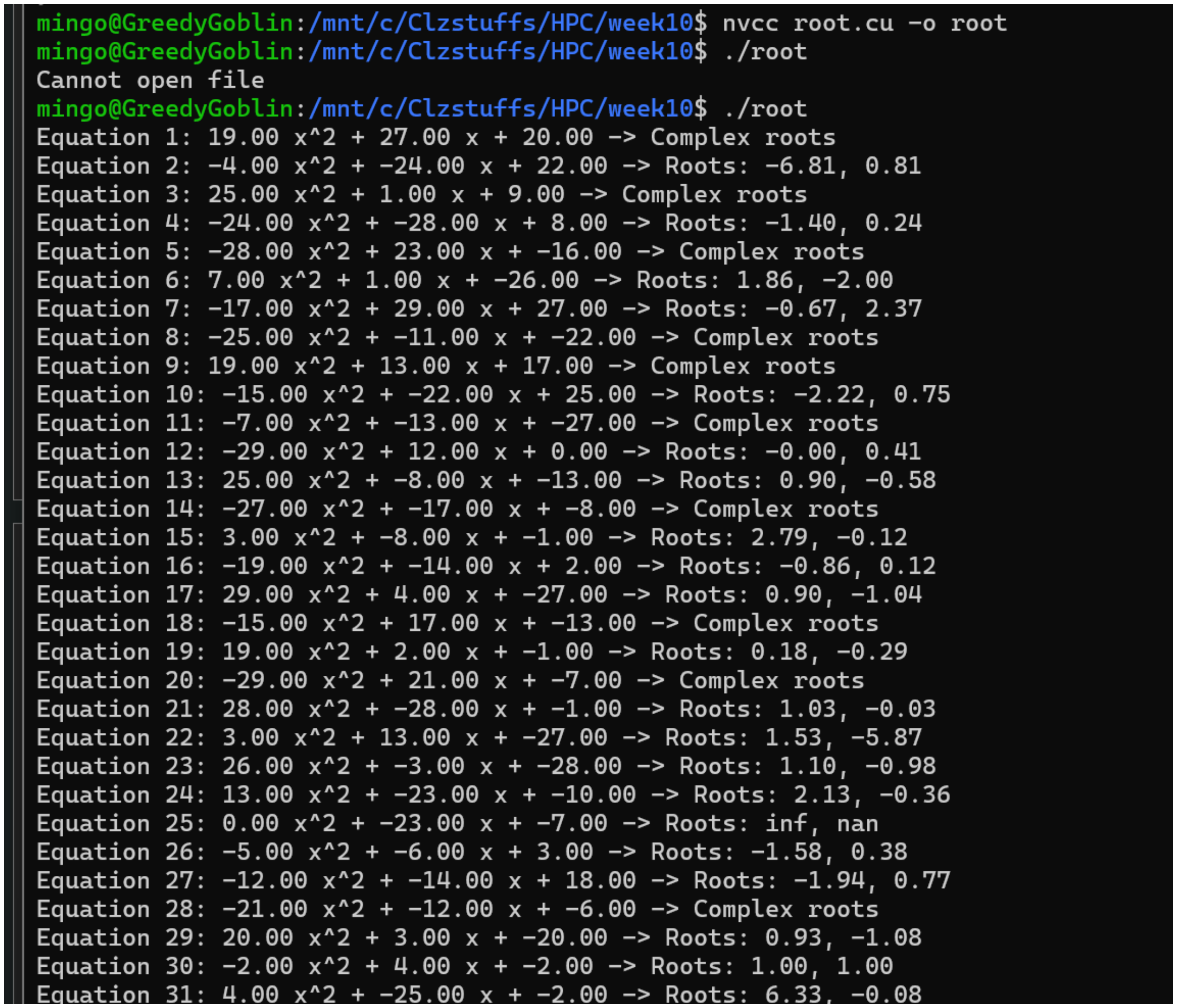
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ cat root.cu
#include <stdio.h>
#include <cuda_runtime.h>
#include <stdlib.h>
#include <math.h>

__global__ void solveQuadratic(float *a, float *b, float *c, float *r1, float *r2, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx >= n) return;

    float disc = b[idx]*b[idx] - 4*a[idx]*c[idx];
    if (disc >= 0) {
        float sqrt_disc = sqrtf(disc);
        r1[idx] = (-b[idx] + sqrt_disc) / (2*a[idx]);
        r2[idx] = (-b[idx] - sqrt_disc) / (2*a[idx]);
    } else {
        r1[idx] = r2[idx] = NAN;
    }
}

int main() {
    int n = 0;
    float a[100], b[100], c[100];

    FILE *fp = fopen("QuadData.txt", "r");
    if (!fp) {
        printf("Cannot open file\n");
        return 1;
    }

    char line[256];
    while (fgets(line, sizeof(line), fp)) {
        if (sscanf(line, "%f,%f,%f", &a[n], &b[n], &c[n]) == 3) n++;
    }
    fclose(fp);

    float *d_a, *d_b, *d_c, *d_r1, *d_r2;
    cudaMalloc(&d_a, n * sizeof(float));
    cudaMalloc(&d_b, n * sizeof(float));
    cudaMalloc(&d_c, n * sizeof(float));
    cudaMalloc(&d_r1, n * sizeof(float));
    cudaMalloc(&d_r2, n * sizeof(float));

    cudaMemcpy(d_a, a, n * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, n * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_c, c, n * sizeof(float), cudaMemcpyHostToDevice);

    int threads = 32;
    int blocks = (n + threads - 1) / threads;

    solveQuadratic<<<blocks, threads>>>(d_a, d_b, d_c, d_r1, d_r2, n);
    cudaDeviceSynchronize();

    float r1[100], r2[100];
```



```
    float r1[100], r2[100];
    cudaMemcpy(r1, d_r1, n * sizeof(float), cudaMemcpyDeviceToHost);
    cudaMemcpy(r2, d_r2, n * sizeof(float), cudaMemcpyDeviceToHost);

    for (int i = 0; i < n; i++) {
        if (!isnan(r1[i]))
            printf("Equation %d: %.2f x^2 + %.2f x + %.2f -> Roots: %.2f, %.2f\n", i+1, a[i], b[i], c[i
], r1[i], r2[i]);
        else
            printf("Equation %d: %.2f x^2 + %.2f x + %.2f -> Complex roots\n", i+1, a[i], b[i], c[i]);
    }

    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    cudaFree(d_r1); cudaFree(d_r2);

    return 0;
}
```


```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ nvcc root.cu -o root
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./root
Cannot open file
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/week10$ ./root
Equation 1: 19.00 x^2 + 27.00 x + 20.00 -> Complex roots
Equation 2: -4.00 x^2 + -24.00 x + 22.00 -> Roots: -6.81, 0.81
Equation 3: 25.00 x^2 + 1.00 x + 9.00 -> Complex roots
Equation 4: -24.00 x^2 + -28.00 x + 8.00 -> Roots: -1.40, 0.24
Equation 5: -28.00 x^2 + 23.00 x + -16.00 -> Complex roots
Equation 6: 7.00 x^2 + 1.00 x + -26.00 -> Roots: 1.86, -2.00
Equation 7: -17.00 x^2 + 29.00 x + 27.00 -> Roots: -0.67, 2.37
Equation 8: -25.00 x^2 + -11.00 x + -22.00 -> Complex roots
Equation 9: 19.00 x^2 + 13.00 x + 17.00 -> Complex roots
Equation 10: -15.00 x^2 + -22.00 x + 25.00 -> Roots: -2.22, 0.75
Equation 11: -7.00 x^2 + -13.00 x + -27.00 -> Complex roots
Equation 12: -29.00 x^2 + 12.00 x + 0.00 -> Roots: -0.00, 0.41
Equation 13: 25.00 x^2 + -8.00 x + -13.00 -> Roots: 0.90, -0.58
Equation 14: -27.00 x^2 + -17.00 x + -8.00 -> Complex roots
Equation 15: 3.00 x^2 + -8.00 x + -1.00 -> Roots: 2.79, -0.12
Equation 16: -19.00 x^2 + -14.00 x + 2.00 -> Roots: -0.86, 0.12
Equation 17: 29.00 x^2 + 4.00 x + -27.00 -> Roots: 0.90, -1.04
Equation 18: -15.00 x^2 + 17.00 x + -13.00 -> Complex roots
Equation 19: 19.00 x^2 + 2.00 x + -1.00 -> Roots: 0.18, -0.29
Equation 20: -29.00 x^2 + 21.00 x + -7.00 -> Complex roots
Equation 21: 28.00 x^2 + -28.00 x + -1.00 -> Roots: 1.03, -0.03
Equation 22: 3.00 x^2 + 13.00 x + -27.00 -> Roots: 1.53, -5.87
Equation 23: 26.00 x^2 + -3.00 x + -28.00 -> Roots: 1.10, -0.98
Equation 24: 13.00 x^2 + -23.00 x + -10.00 -> Roots: 2.13, -0.36
Equation 25: 0.00 x^2 + -23.00 x + -7.00 -> Roots: inf, nan
Equation 26: -5.00 x^2 + -6.00 x + 3.00 -> Roots: -1.58, 0.38
Equation 27: -12.00 x^2 + -14.00 x + 18.00 -> Roots: -1.94, 0.77
Equation 28: -21.00 x^2 + -12.00 x + -6.00 -> Complex roots
Equation 29: 20.00 x^2 + 3.00 x + -20.00 -> Roots: 0.93, -1.08
Equation 30: -2.00 x^2 + 4.00 x + -2.00 -> Roots: 1.00, 1.00
Equation 31: 4.00 x^2 + -25.00 x + -2.00 -> Roots: 6.33, -0.08
```