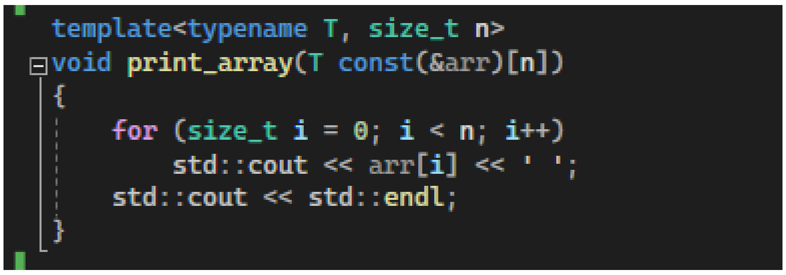
Task 1

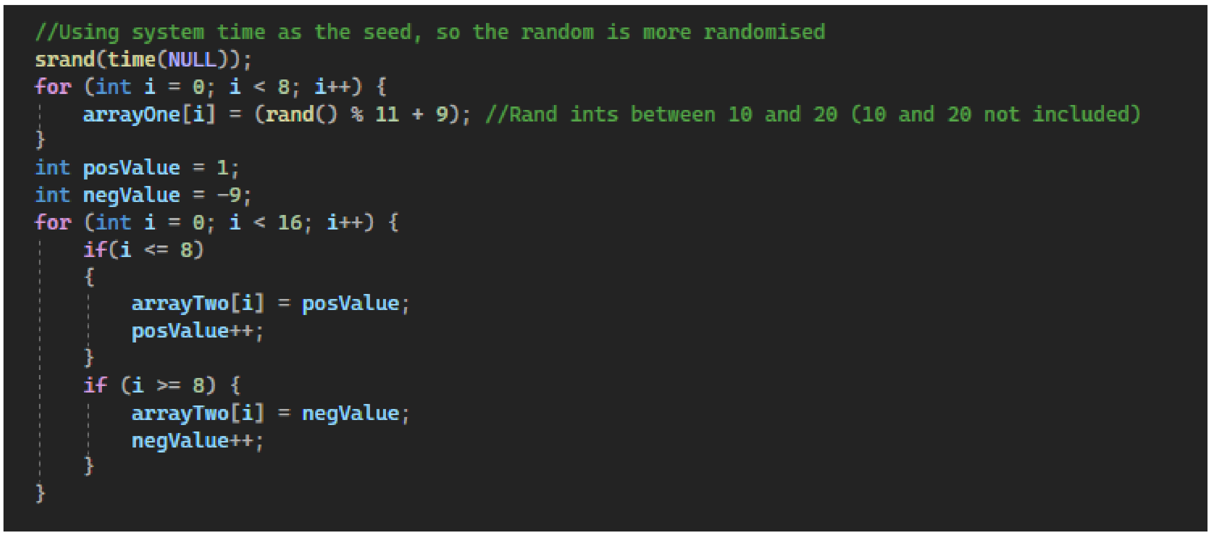
As initializing variables, Allow user to select OpenCL device/platform, creating context, building program, kernel, command queue, buffers, setting kernel arguments, enqueue kernel are all same as Assignment 1, I will not be reexplaining everything again.

Creating a template meant for printing out arrayOne and arrayTwo



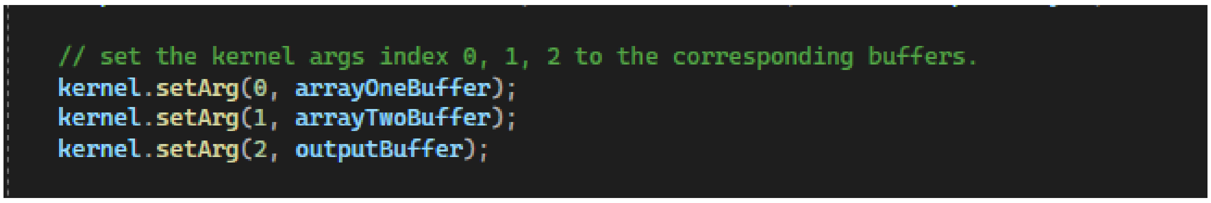
```
template<typename T, size_t n>
void print_array(T const(&arr)[n])
{
    for (size_t i = 0; i < n; i++)
        std::cout << arr[i] << ' ';
    std::cout << std::endl;
}
```

Initialising arrayOne and arrayTwo using a very for loop with a nested if loop to fill in 1 to 8 in arrayOne and -9 to (-2) in arrayTwo



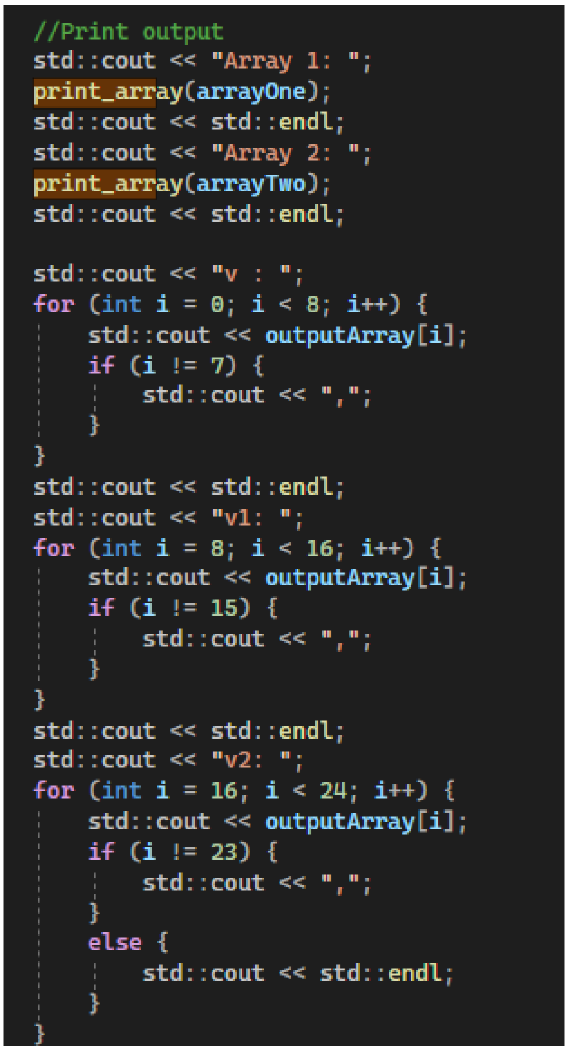
```
//Using system time as the seed, so the random is more randomised
srand(time(NULL));
for (int i = 0; i < 8; i++) {
    arrayOne[i] = (rand() % 11 + 9); //Rand ints between 10 and 20 (10 and 20 not included)
}
int posValue = 1;
int negValue = -9;
for (int i = 0; i < 16; i++) {
    if(i <= 8)
    {
        arrayTwo[i] = posValue;
        posValue++;
    }
    if (i >= 8) {
        arrayTwo[i] = negValue;
        negValue++;
    }
}
```

Set argument index 0, 1 ,2 to kernel their bufffers

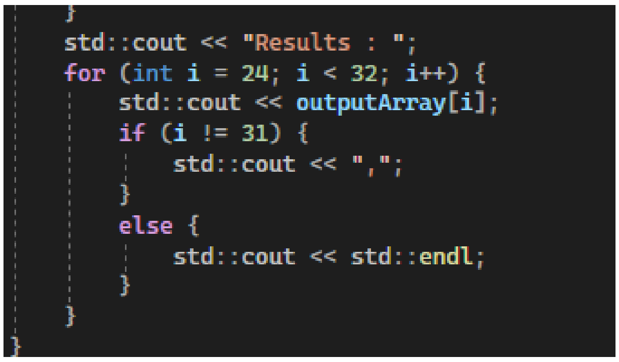


```
// set the kernel args index 0, 1, 2 to the corresponding buffers.
kernel.setArg(0, arrayOneBuffer);
kernel.setArg(1, arrayTwoBuffer);
kernel.setArg(2, outputBuffer);
```

Display output of v, v1, v2, results (stored in order in outputArray)



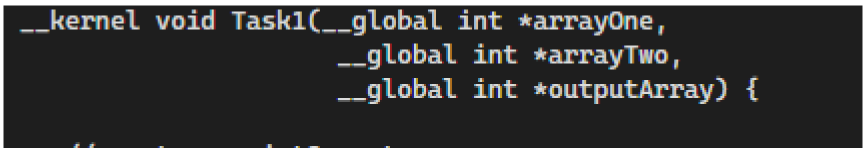
```
//Print output
std::cout << "Array 1: ";
print_array(arrayOne);
std::cout << std::endl;
std::cout << "Array 2: ";
print_array(arrayTwo);
std::cout << std::endl;

std::cout << "v : ";
for (int i = 0; i < 8; i++) {
    std::cout << outputArray[i];
    if (i != 7) {
        std::cout << ",";
    }
}
std::cout << std::endl;
std::cout << "v1: ";
for (int i = 8; i < 16; i++) {
    std::cout << outputArray[i];
    if (i != 15) {
        std::cout << ",";
    }
}
std::cout << std::endl;
std::cout << "v2: ";
for (int i = 16; i < 24; i++) {
    std::cout << outputArray[i];
    if (i != 23) {
        std::cout << ",";
    }
    else {
        std::cout << std::endl;
    }
}
```



```
    }
    std::cout << "Results : ";
    for (int i = 24; i < 32; i++) {
        std::cout << outputArray[i];
        if (i != 31) {
            std::cout << ",";
        }
        else {
            std::cout << std::endl;
        }
    }
}
```

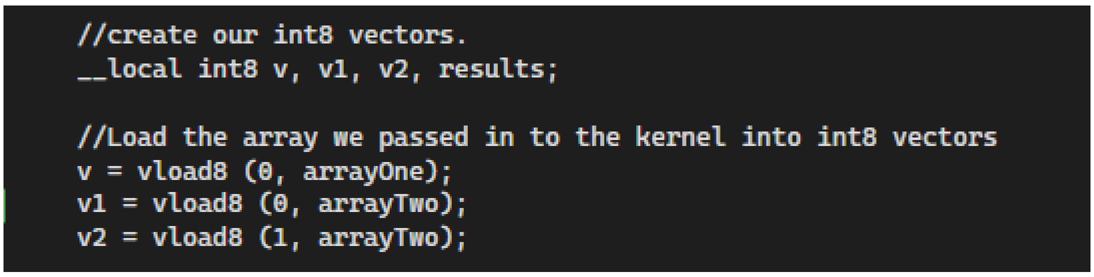
Task 1 Kernel

Initializing the 3 parameters passed in via the buffers



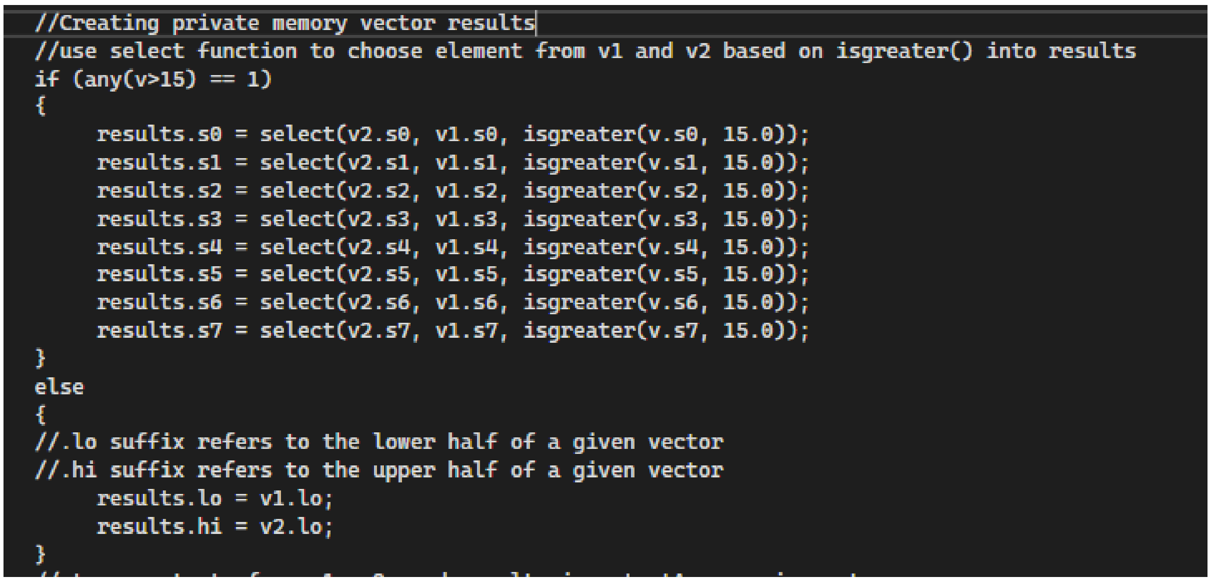
```
__kernel void Task1(__global int *arrayOne,
                    __global int *arrayTwo,
                    __global int *outputArray) {
```

Create int8 vector v as requested, use vload to copy the contents into v1 and v2.



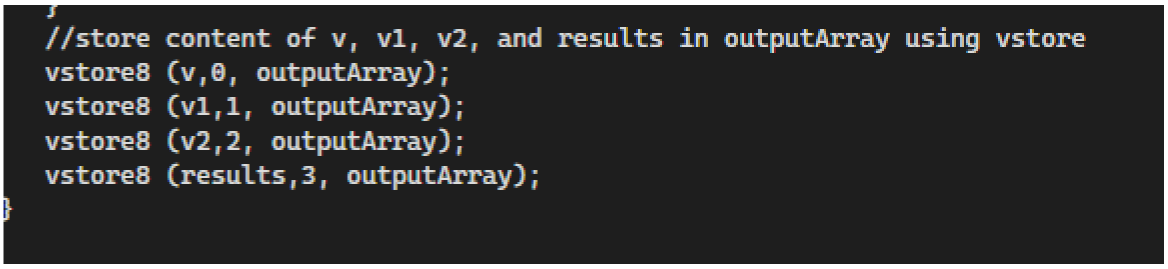
```
    //create our int8 vectors.
    __local int8 v, v1, v2, results;

    //Load the array we passed in to the kernel into int8 vectors
    v = vload8 (0, arrayOne);
    v1 = vload8 (0, arrayTwo);
    v2 = vload8 (1, arrayTwo);
```

Input the correct content into results vector while fulfilling the requirements



```
  //Creating private memory vector results
  //use select function to choose element from v1 and v2 based on isgreater() into results
  if (any(v>15) == 1)
  {
      results.s0 = select(v2.s0, v1.s0, isgreater(v.s0, 15.0));
      results.s1 = select(v2.s1, v1.s1, isgreater(v.s1, 15.0));
      results.s2 = select(v2.s2, v1.s2, isgreater(v.s2, 15.0));
      results.s3 = select(v2.s3, v1.s3, isgreater(v.s3, 15.0));
      results.s4 = select(v2.s4, v1.s4, isgreater(v.s4, 15.0));
      results.s5 = select(v2.s5, v1.s5, isgreater(v.s5, 15.0));
      results.s6 = select(v2.s6, v1.s6, isgreater(v.s6, 15.0));
      results.s7 = select(v2.s7, v1.s7, isgreater(v.s7, 15.0));
  }
  else
  {
  //.lo suffix refers to the lower half of a given vector
  //.hi suffix refers to the upper half of a given vector
      results.lo = v1.lo;
      results.hi = v2.lo;
  }
```

Store content using vstoren



```
    }
    //store content of v, v1, v2, and results in outputArray using vstore
    vstore8 (v,0, outputArray);
    vstore8 (v1,1, outputArray);
    vstore8 (v2,2, outputArray);
    vstore8 (results,3, outputArray);
}
```

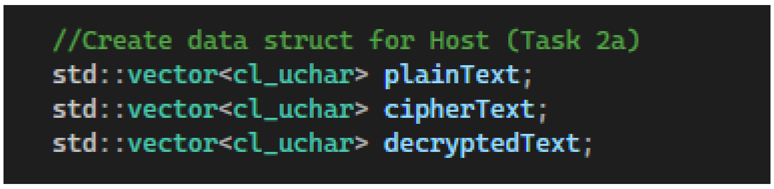
Display Output

Task2

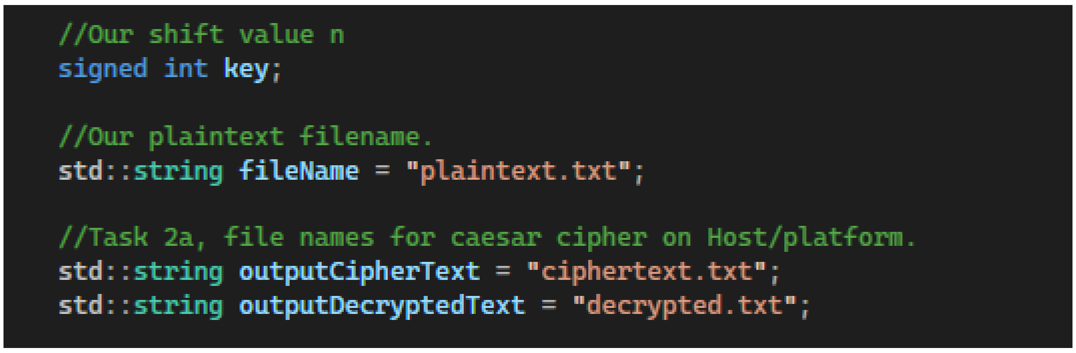
Task 2a

Write a normal C/C++ program (not using OpenCL) that reads the content from a text file called “plaintext.txt”

Initializing Variables

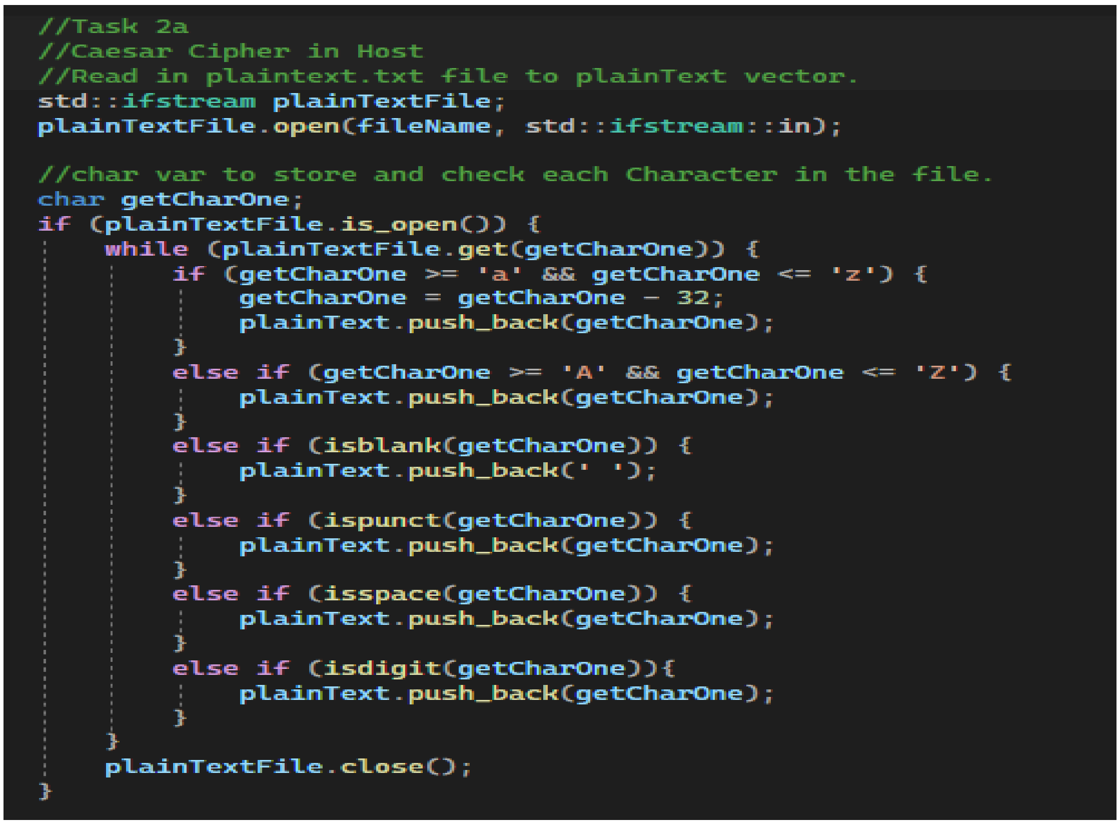


```
//Create data struct for Host (Task 2a)
std::vector<cl_uchar> plainText;
std::vector<cl_uchar> cipherText;
std::vector<cl_uchar> decryptedText;
```



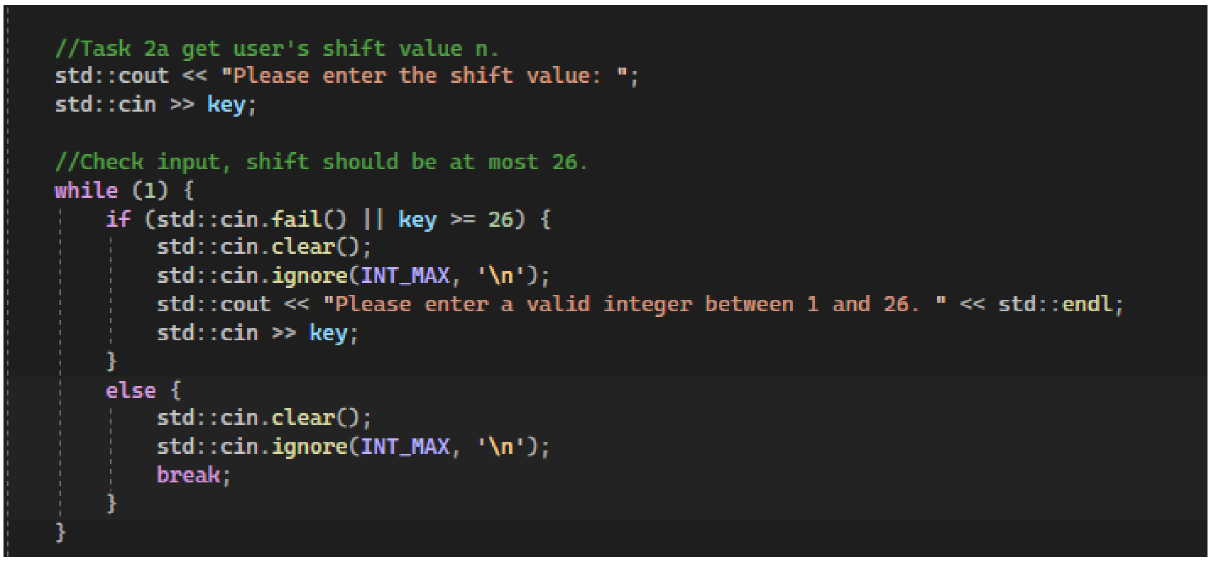
```
//Our shift value n
signed int key;

//Our plaintext filename.
std::string fileName = "plaintext.txt";

//Task 2a, file names for caesar cipher on Host/platform.
std::string outputCipherText = "ciphertext.txt";
std::string outputDecryptedText = "decrypted.txt";
```

Read file plaintext.txt and copy all content into plaintext vector.



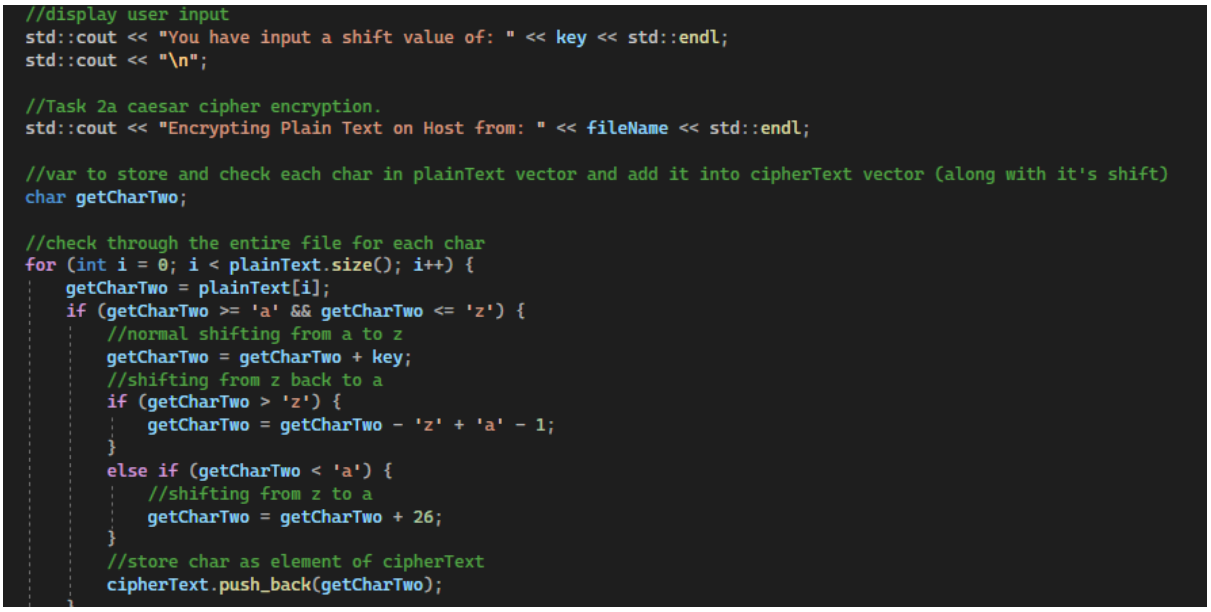
```
//Task 2a
//Caesar Cipher in Host
//Read in plaintext.txt file to plainText vector.
std::ifstream plainTextFile;
plainTextFile.open(fileName, std::ifstream::in);

//char var to store and check each Character in the file.
char getCharOne;
if (plainTextFile.is_open()) {
    while (plainTextFile.get(getCharOne)) {
        if (getCharOne >= 'a' && getCharOne <= 'z') {
            getCharOne = getCharOne - 32;
            plainText.push_back(getCharOne);
        }
        else if (getCharOne >= 'A' && getCharOne <= 'Z') {
            plainText.push_back(getCharOne);
        }
        else if (isblank(getCharOne)) {
            plainText.push_back(' ');
        }
        else if (ispunct(getCharOne)) {
            plainText.push_back(getCharOne);
        }
        else if (isspace(getCharOne)) {
            plainText.push_back(getCharOne);
        }
        else if (isdigit(getCharOne)){
            plainText.push_back(getCharOne);
        }
    }
    plainTextFile.close();
}
```

User input for shift value n, to determine left shifting or right shifting for ciphering

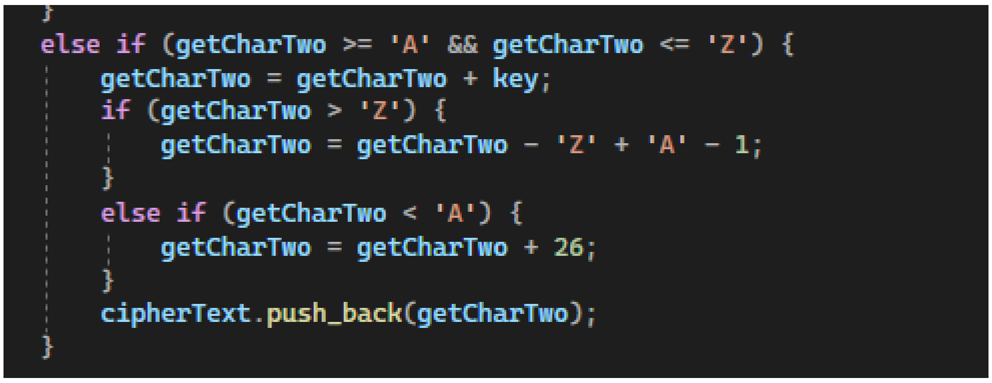


```cpp
//Task 2a get user's shift value n.
std::cout << "Please enter the shift value: ";
std::cin >> key;

//Check input, shift should be at most 26.
while (1) {
    if (std::cin.fail() || key >= 26) {
        std::cin.clear();
        std::cin.ignore(INT_MAX, '\n');
        std::cout << "Please enter a valid integer between 1 and 26. " << std::endl;
        std::cin >> key;
    }
    else {
        std::cin.clear();
        std::cin.ignore(INT_MAX, '\n');
        break;
    }
}
```

Read from plaintext vector, start shifting for small and upper case chars by user input n

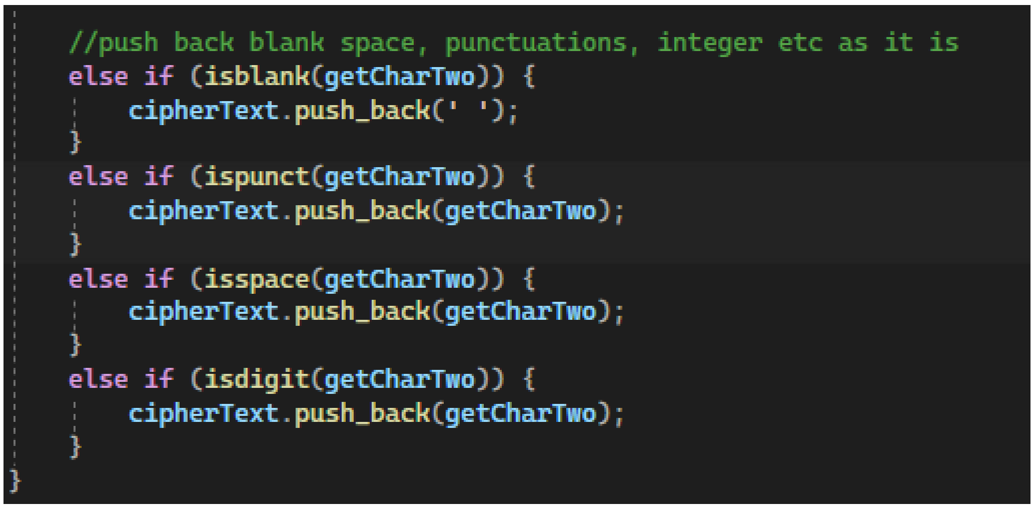


```cpp
//display user input
std::cout << "You have input a shift value of: " << key << std::endl;
std::cout << "\n";

//Task 2a caesar cipher encryption.
std::cout << "Encrypting Plain Text on Host from: " << fileName << std::endl;

//var to store and check each char in plainText vector and add it into cipherText vector (along with it's shift)
char getCharTwo;

//check through the entire file for each char
for (int i = 0; i < plainText.size(); i++) {
    getCharTwo = plainText[i];
    if (getCharTwo >= 'a' && getCharTwo <= 'z') {
        //normal shifting from a to z
        getCharTwo = getCharTwo + key;
        //shifting from z back to a
        if (getCharTwo > 'z') {
            getCharTwo = getCharTwo - 'z' + 'a' - 1;
        }
        else if (getCharTwo < 'a') {
            //shifting from z to a
            getCharTwo = getCharTwo + 26;
        }
        //store char as element of cipherText
        cipherText.push_back(getCharTwo);
```



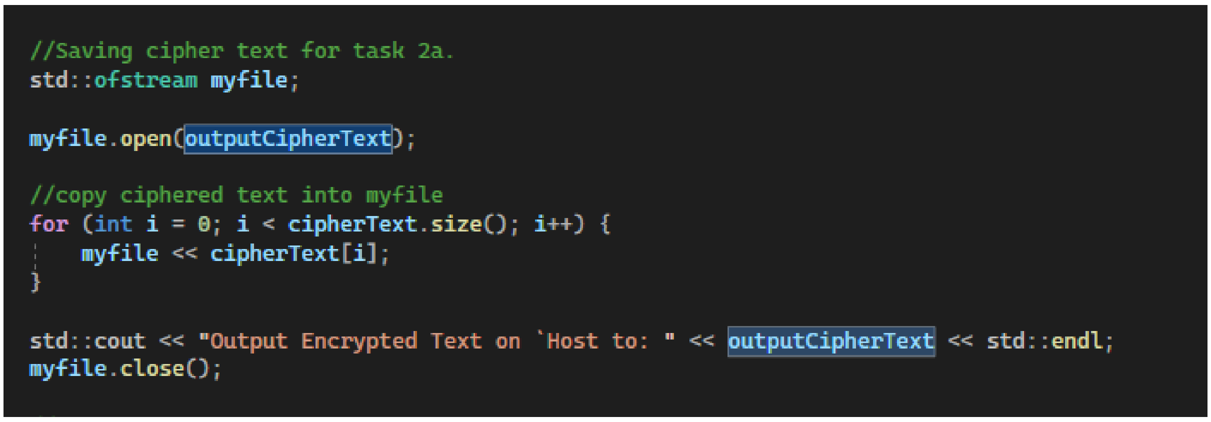
```
}
else if (getCharTwo >= 'A' && getCharTwo <= 'Z') {
    getCharTwo = getCharTwo + key;
    if (getCharTwo > 'Z') {
        getCharTwo = getCharTwo - 'Z' + 'A' - 1;
    }
    else if (getCharTwo < 'A') {
        getCharTwo = getCharTwo + 26;
    }
    cipherText.push_back(getCharTwo);
}
```

Keep blank space, punctuations, integer etc as it is



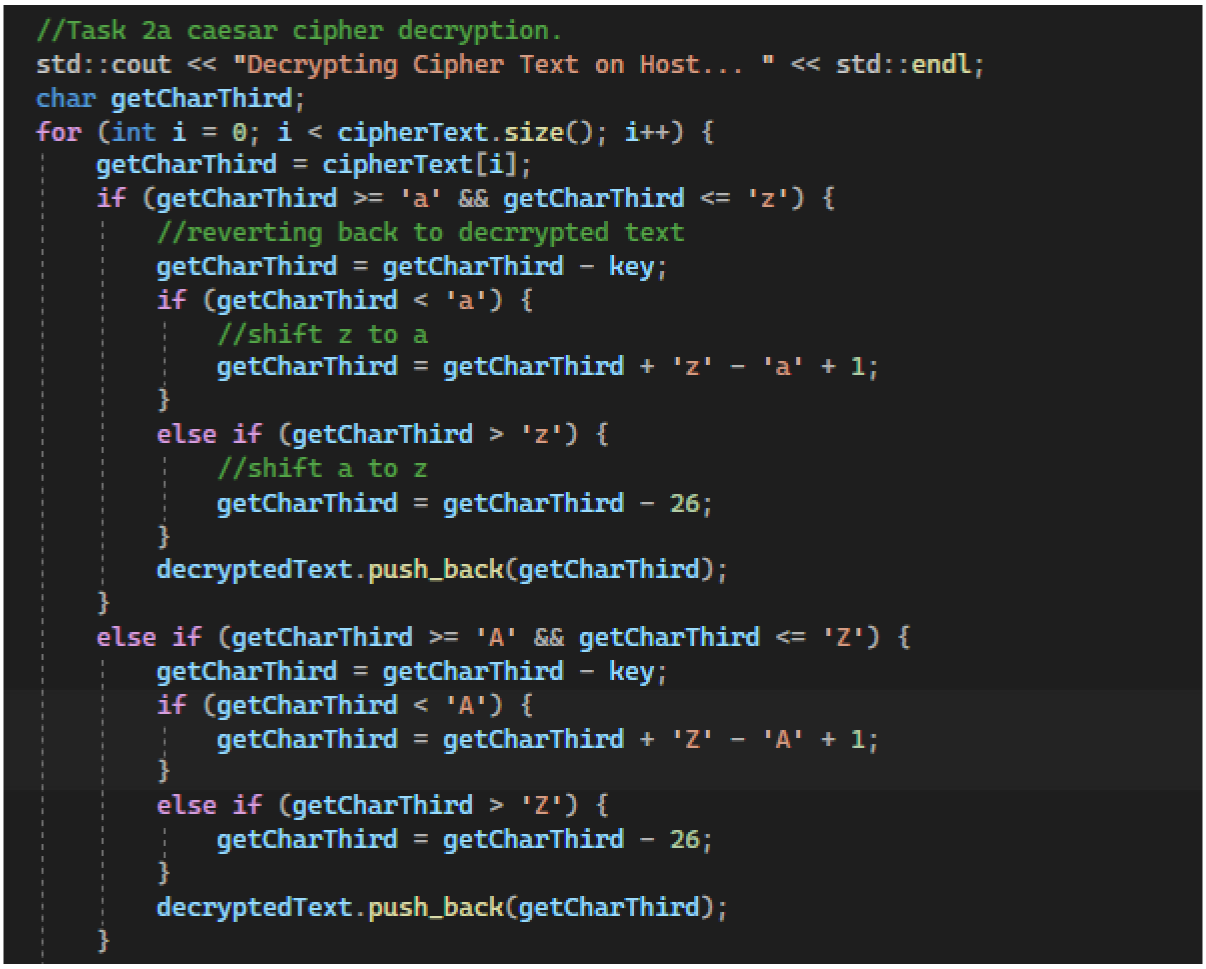
```
    //push back blank space, punctuations, integer etc as it is
    else if (isblank(getCharTwo)) {
        cipherText.push_back(' ');
    }
    else if (ispunct(getCharTwo)) {
        cipherText.push_back(getCharTwo);
    }
    else if (isspace(getCharTwo)) {
        cipherText.push_back(getCharTwo);
    }
    else if (isdigit(getCharTwo)) {
        cipherText.push_back(getCharTwo);
    }
}
```

Save cipher text into ciphertext.txt and copy it into myfile



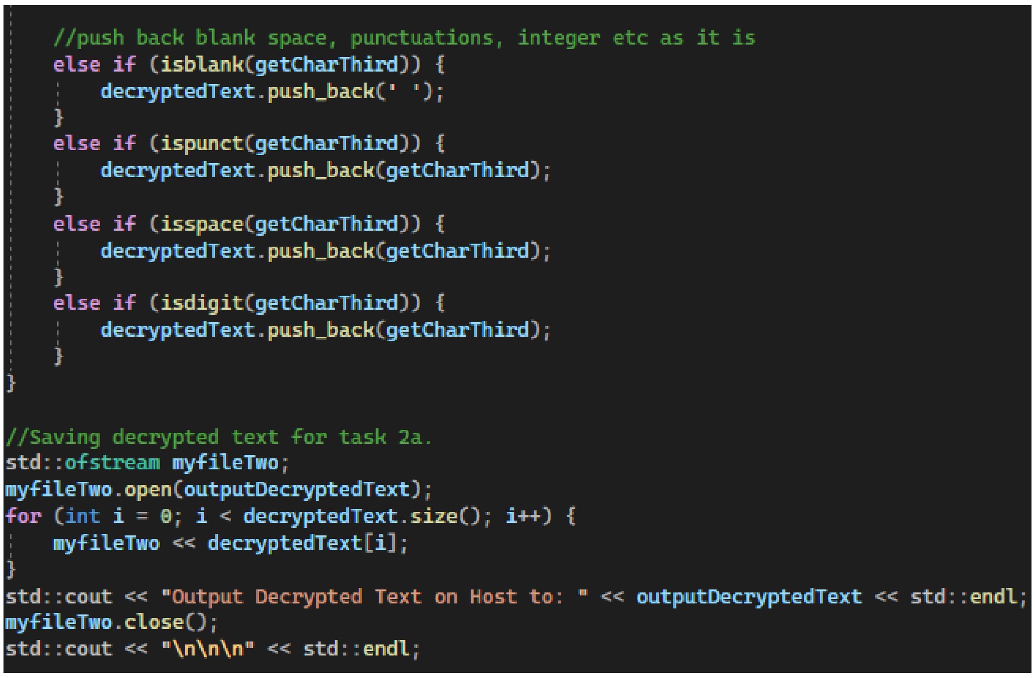
```
//Saving cipher text for task 2a.
std::ofstream myfile;

myfile.open(outputCipherText);

//copy ciphered text into myfile
for (int i = 0; i < cipherText.size(); i++) {
    myfile << cipherText[i];
}

std::cout << "Output Encrypted Text on `Host to: " << outputCipherText << std::endl;
myfile.close();
```

Decrypting the ciphered text, basically reverting the shift by n



```
//Task 2a caesar cipher decryption.
std::cout << "Decrypting Cipher Text on Host... " << std::endl;
char getCharThird;
for (int i = 0; i < cipherText.size(); i++) {
    getCharThird = cipherText[i];
    if (getCharThird >= 'a' && getCharThird <= 'z') {
        //reverting back to decrrypted text
        getCharThird = getCharThird - key;
        if (getCharThird < 'a') {
            //shift z to a
            getCharThird = getCharThird + 'z' - 'a' + 1;
        }
        else if (getCharThird > 'z') {
            //shift a to z
            getCharThird = getCharThird - 26;
        }
        decryptedText.push_back(getCharThird);
    }
    else if (getCharThird >= 'A' && getCharThird <= 'Z') {
        getCharThird = getCharThird - key;
        if (getCharThird < 'A') {
            getCharThird = getCharThird + 'Z' - 'A' + 1;
        }
        else if (getCharThird > 'Z') {
            getCharThird = getCharThird - 26;
        }
        decryptedText.push_back(getCharThird);
    }
```

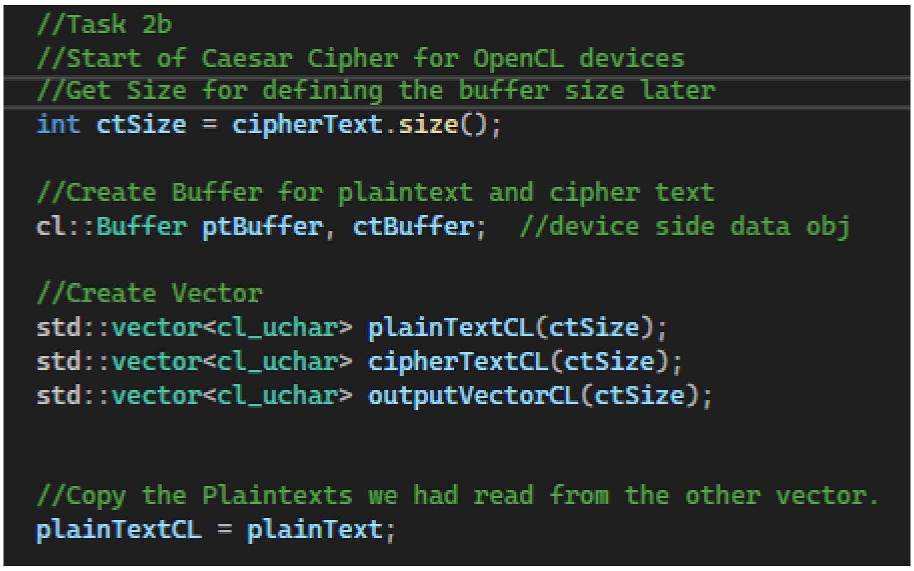
Same as ciphering text



```
    //push back blank space, punctuations, integer etc as it is
    else if (isblank(getCharThird)) {
        decryptedText.push_back(' ');
    }
    else if (ispunct(getCharThird)) {
        decryptedText.push_back(getCharThird);
    }
    else if (isspace(getCharThird)) {
        decryptedText.push_back(getCharThird);
    }
    else if (isdigit(getCharThird)) {
        decryptedText.push_back(getCharThird);
    }
}

//Saving decrypted text for task 2a.
std::ofstream myfileTwo;
myfileTwo.open(outputDecryptedText);
for (int i = 0; i < decryptedText.size(); i++) {
    myfileTwo << decryptedText[i];
}
std::cout << "Output Decrypted Text on Host to: " << outputDecryptedText << std::endl;
myfileTwo.close();
std::cout << "\n\n\n" << std::endl;
```

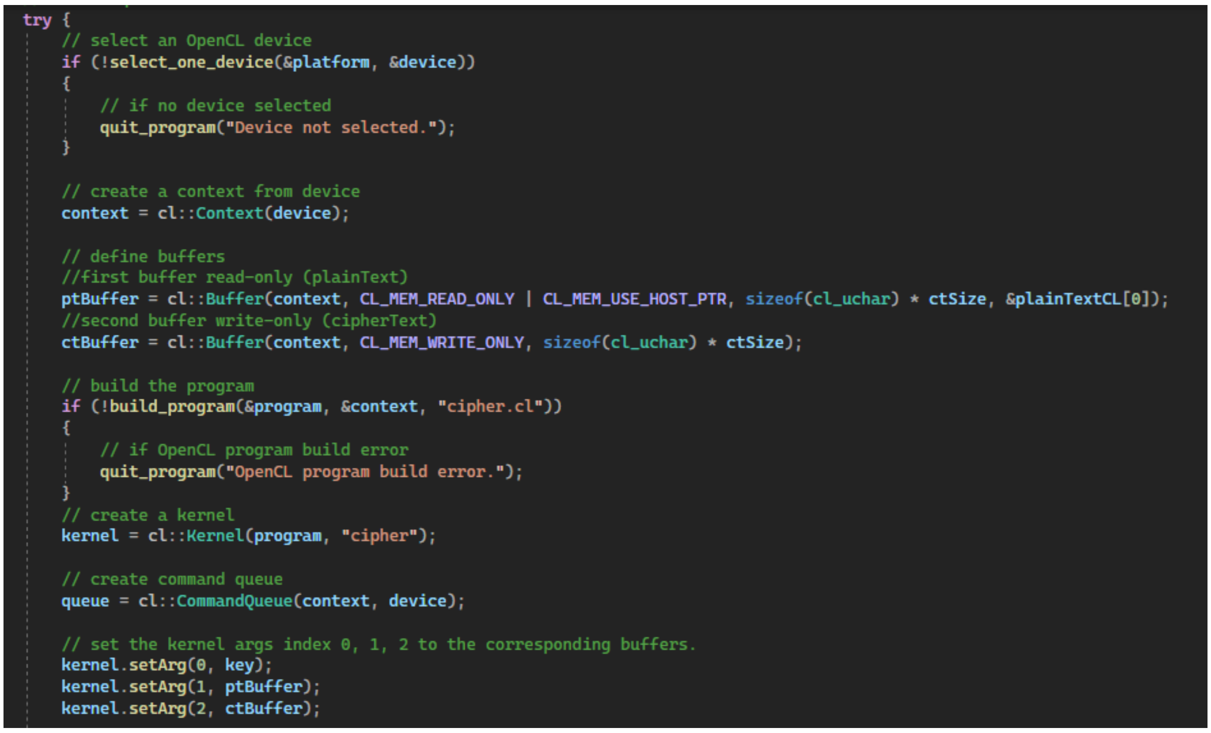
Task 2b

Initialising variable



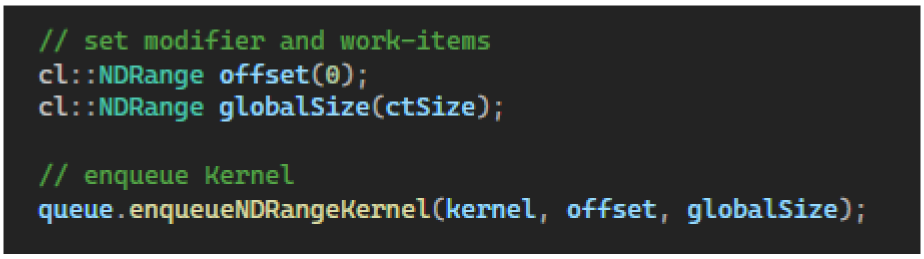
```
//Task 2b
//Start of Caesar Cipher for OpenCL devices
//Get Size for defining the buffer size later
int ctSize = cipherText.size();

//Create Buffer for plaintext and cipher text
cl::Buffer ptBuffer, ctBuffer;  //device side data obj

//Create Vector
std::vector<cl_uchar> plainTextCL(ctSize);
std::vector<cl_uchar> cipherTextCL(ctSize);
std::vector<cl_uchar> outputVectorCL(ctSize);


//Copy the Plaintexts we had read from the other vector.
plainTextCL = plainText;
```

Proceed with standard OpenCL procedures:
Select device, create context, build program, define buffer, create kernels, set arguments, create command queue etc.



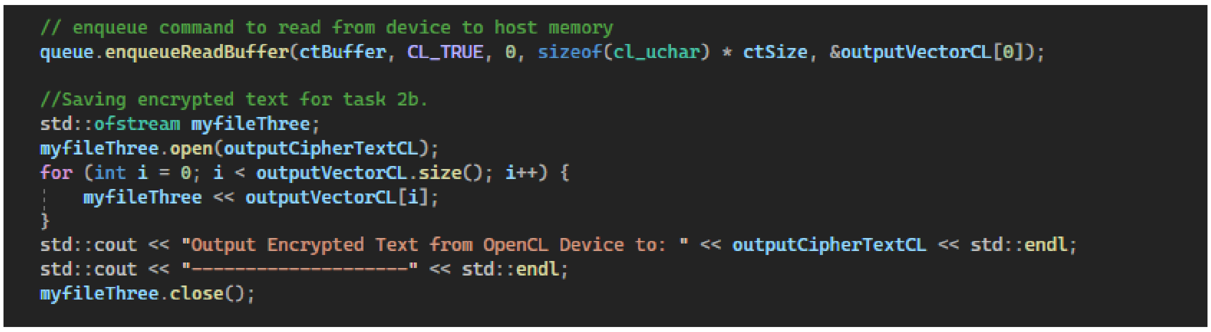
```
try {
    // select an OpenCL device
    if (!select_one_device(&platform, &device))
    {
        // if no device selected
        quit_program("Device not selected.");
    }

    // create a context from device
    context = cl::Context(device);

    // define buffers
    //first buffer read-only (plainText)
    ptBuffer = cl::Buffer(context, CL_MEM_READ_ONLY | CL_MEM_USE_HOST_PTR, sizeof(cl_uchar) * ctSize, &plainTextCL[0]);
    //second buffer write-only (cipherText)
    ctBuffer = cl::Buffer(context, CL_MEM_WRITE_ONLY, sizeof(cl_uchar) * ctSize);

    // build the program
    if (!build_program(&program, &context, "cipher.cl"))
    {
        // if OpenCL program build error
        quit_program("OpenCL program build error.");
    }
    // create a kernel
    kernel = cl::Kernel(program, "cipher");

    // create command queue
    queue = cl::CommandQueue(context, device);

    // set the kernel args index 0, 1, 2 to the corresponding buffers.
    kernel.setArg(0, key);
    kernel.setArg(1, ptBuffer);
    kernel.setArg(2, ctBuffer);
```

Set modifier, work items, and enqueue kernel



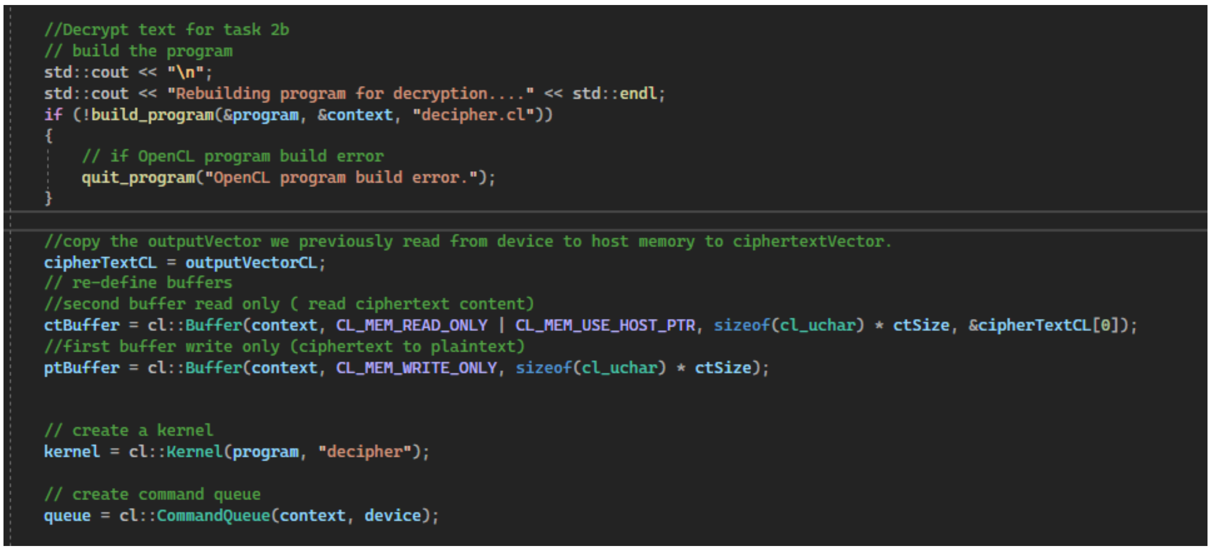
```
// set modifier and work-items
cl::NDRange offset(0);
cl::NDRange globalSize(ctSize);

// enqueue Kernel
queue.enqueueNDRangeKernel(kernel, offset, globalSize);
```

Enqueue command and saving encrypted text to myfileThree



```
// enqueue command to read from device to host memory
queue.enqueueReadBuffer(ctBuffer, CL_TRUE, 0, sizeof(cl_uchar) * ctSize, &outputVectorCL[0]);

//Saving encrypted text for task 2b.
std::ofstream myfileThree;
myfileThree.open(outputCipherTextCL);
for (int i = 0; i < outputVectorCL.size(); i++) {
    myfileThree << outputVectorCL[i];
}
std::cout << "Output Encrypted Text from OpenCL Device to: " << outputCipherTextCL << std::endl;
std::cout << "--------------------" << std::endl;
myfileThree.close();
```

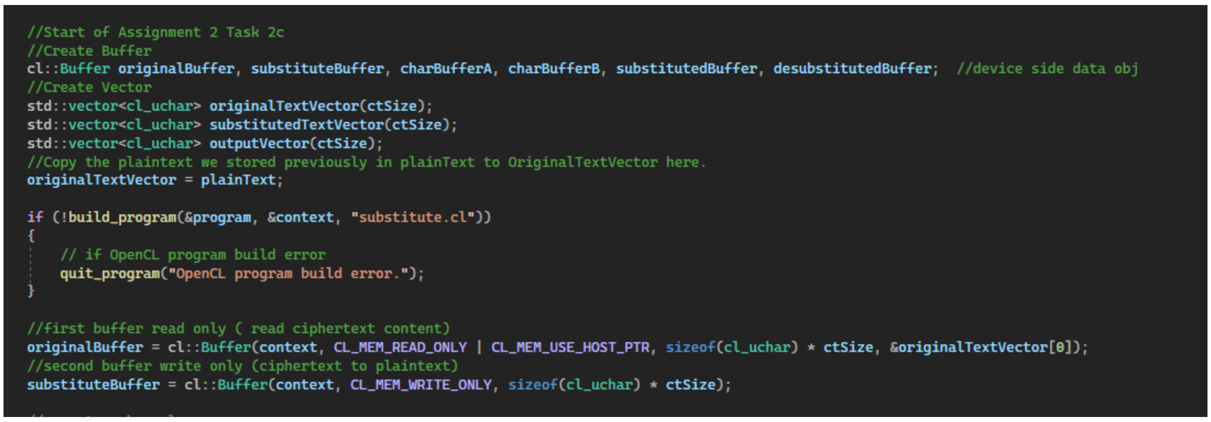
Build program using decipher.cl, copy content of encrypted text to cipherTextCL and redefine buffers.



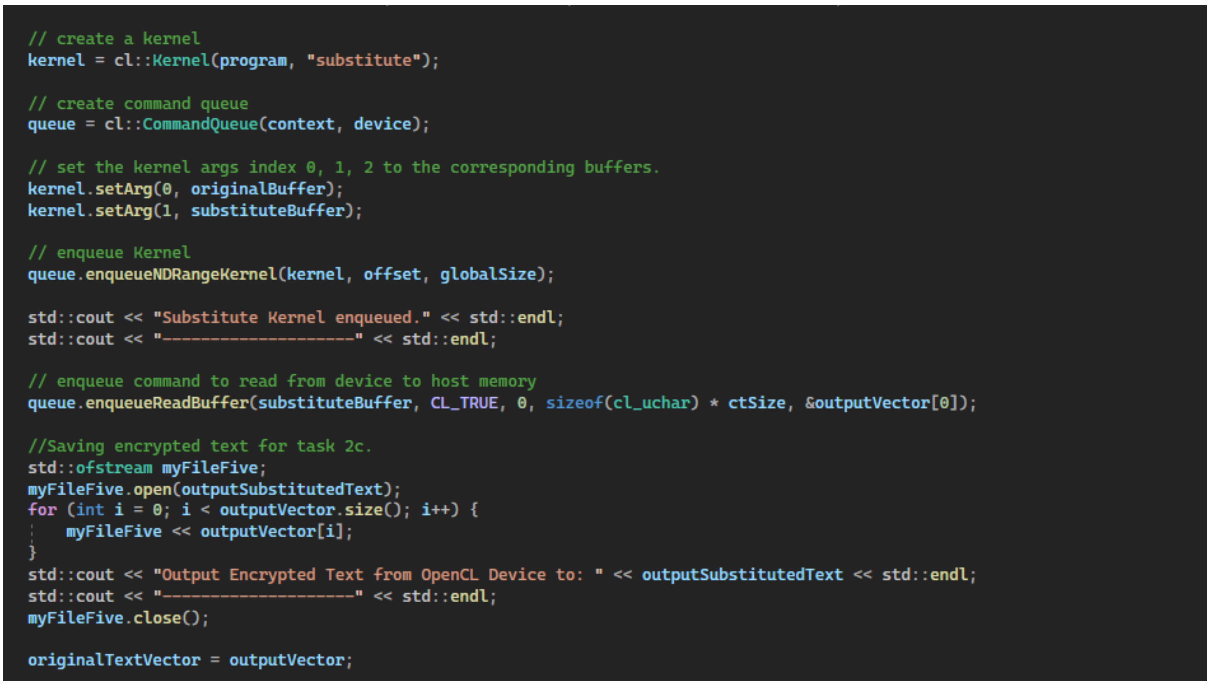
```
//Decrypt text for task 2b
// build the program
std::cout << "\n";
std::cout << "Rebuilding program for decryption...." << std::endl;
if (!build_program(&program, &context, "decipher.cl"))
{
    // if OpenCL program build error
    quit_program("OpenCL program build error.");
}

//copy the outputVector we previously read from device to host memory to ciphertextVector.
cipherTextCL = outputVectorCL;
// re-define buffers
//second buffer read only ( read ciphertext content)
ctBuffer = cl::Buffer(context, CL_MEM_READ_ONLY | CL_MEM_USE_HOST_PTR, sizeof(cl_uchar) * ctSize, &cipherTextCL[0]);
//first buffer write only (ciphertext to plaintext)
ptBuffer = cl::Buffer(context, CL_MEM_WRITE_ONLY, sizeof(cl_uchar) * ctSize);


// create a kernel
kernel = cl::Kernel(program, "decipher");

// create command queue
queue = cl::CommandQueue(context, device);
```

Task 2c

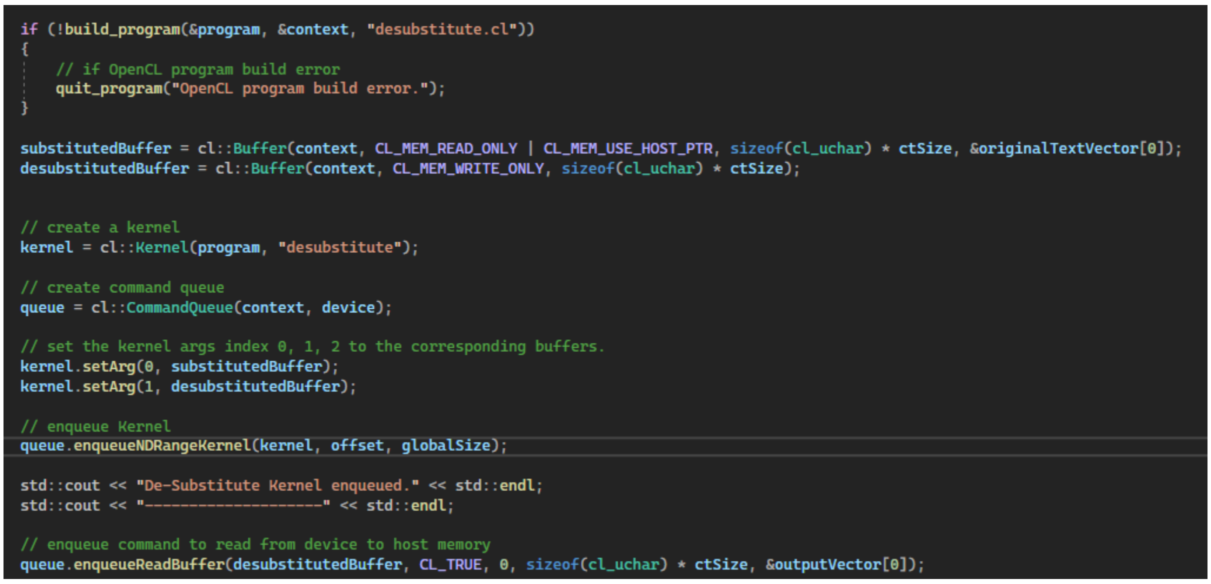
Similar to 2a and 2b



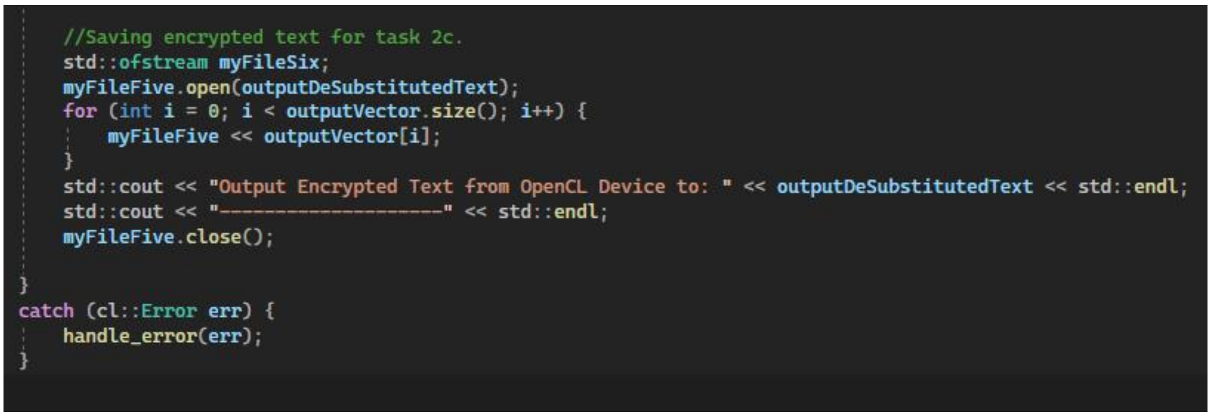
```
//Start of Assignment 2 Task 2c
//Create Buffer
cl::Buffer originalBuffer, substituteBuffer, charBufferA, charBufferB, substitutedBuffer, desubstitutedBuffer;  //device side data obj
//Create Vector
std::vector<cl_uchar> originalTextVector(ctSize);
std::vector<cl_uchar> substitutedTextVector(ctSize);
std::vector<cl_uchar> outputVector(ctSize);
//Copy the plaintext we stored previously in plainText to OriginalTextVector here.
originalTextVector = plainText;

if (!build_program(&program, &context, "substitute.cl"))
{
    // if OpenCL program build error
    quit_program("OpenCL program build error.");
}

//first buffer read only ( read ciphertext content)
originalBuffer = cl::Buffer(context, CL_MEM_READ_ONLY | CL_MEM_USE_HOST_PTR, sizeof(cl_uchar) * ctSize, &originalTextVector[0]);
//second buffer write only (ciphertext to plaintext)
substituteBuffer = cl::Buffer(context, CL_MEM_WRITE_ONLY, sizeof(cl_uchar) * ctSize);
```



```
// create a kernel
kernel = cl::Kernel(program, "substitute");

// create command queue
queue = cl::CommandQueue(context, device);

// set the kernel args index 0, 1, 2 to the corresponding buffers.
kernel.setArg(0, originalBuffer);
kernel.setArg(1, substituteBuffer);

// enqueue Kernel
queue.enqueueNDRangeKernel(kernel, offset, globalSize);

std::cout << "Substitute Kernel enqueued." << std::endl;
std::cout << "--------------------" << std::endl;

// enqueue command to read from device to host memory
queue.enqueueReadBuffer(substituteBuffer, CL_TRUE, 0, sizeof(cl_uchar) * ctSize, &outputVector[0]);

//Saving encrypted text for task 2c.
std::ofstream myFileFive;
myFileFive.open(outputSubstitutedText);
for (int i = 0; i < outputVector.size(); i++) {
    myFileFive << outputVector[i];
}
std::cout << "Output Encrypted Text from OpenCL Device to: " << outputSubstitutedText << std::endl;
std::cout << "--------------------" << std::endl;
myFileFive.close();

originalTextVector = outputVector;
```



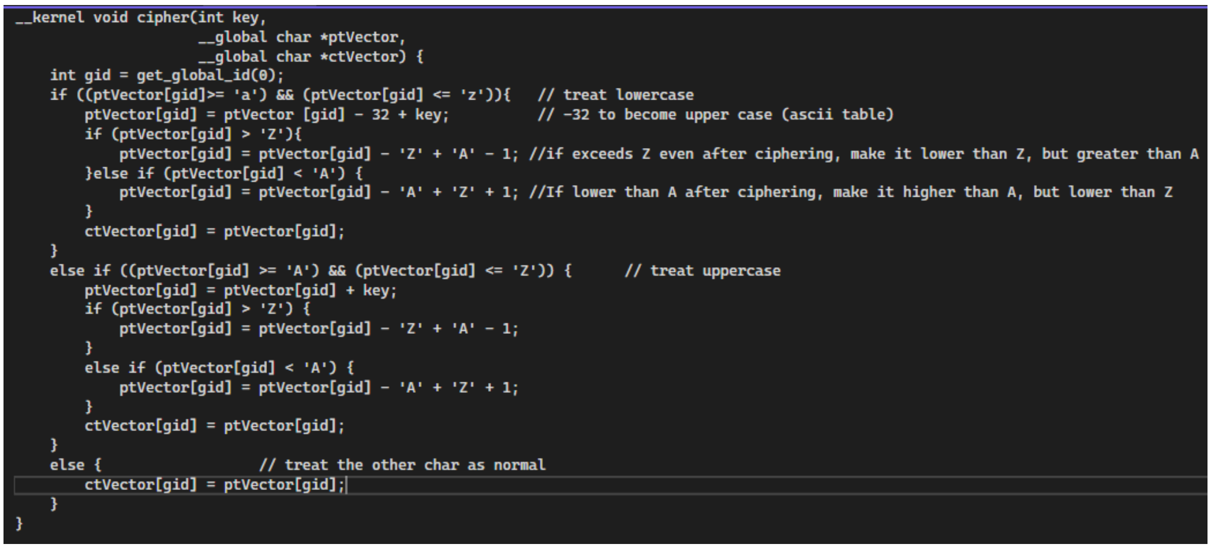
```
if (!build_program(&program, &context, "desubstitute.cl"))
{
    // if OpenCL program build error
    quit_program("OpenCL program build error.");
}

substitutedBuffer = cl::Buffer(context, CL_MEM_READ_ONLY | CL_MEM_USE_HOST_PTR, sizeof(cl_uchar) * ctSize, &originalTextVector[0]);
desubstitutedBuffer = cl::Buffer(context, CL_MEM_WRITE_ONLY, sizeof(cl_uchar) * ctSize);


// create a kernel
kernel = cl::Kernel(program, "desubstitute");

// create command queue
queue = cl::CommandQueue(context, device);

// set the kernel args index 0, 1, 2 to the corresponding buffers.
kernel.setArg(0, substitutedBuffer);
kernel.setArg(1, desubstitutedBuffer);

// enqueue Kernel
queue.enqueueNDRangeKernel(kernel, offset, globalSize);

std::cout << "De-Substitute Kernel enqueued." << std::endl;
std::cout << "--------------------" << std::endl;

// enqueue command to read from device to host memory
queue.enqueueReadBuffer(desubstitutedBuffer, CL_TRUE, 0, sizeof(cl_uchar) * ctSize, &outputVector[0]);
```



```cpp
    //Saving encrypted text for task 2c.
    std::ofstream myFileSix;
    myFileFive.open(outputDeSubstitutedText);
    for (int i = 0; i < outputVector.size(); i++) {
        myFileFive << outputVector[i];
    }
    std::cout << "Output Encrypted Text from OpenCL Device to: " << outputDeSubstitutedText << std::endl;
    std::cout << "--------------------" << std::endl;
    myFileFive.close();

}
catch (cl::Error err) {
    handle_error(err);
}
```

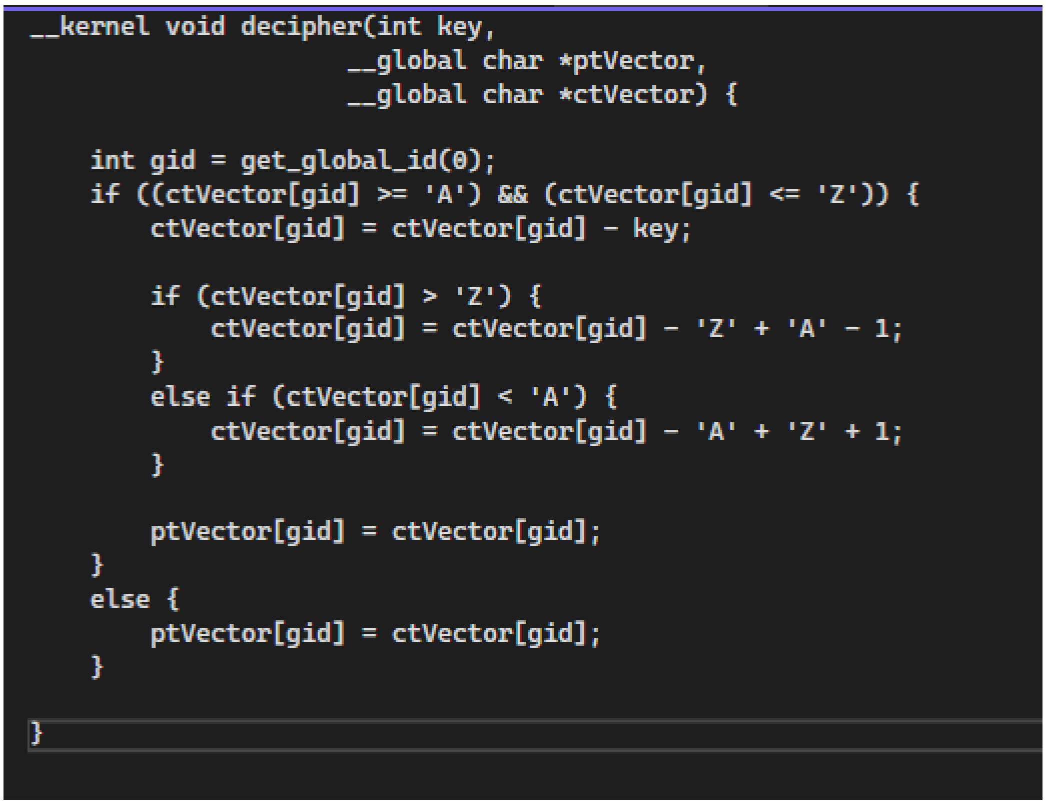
Kernels

Cipher.cl, if plain text is lowercase,make it upper case. If it is upper case, shift it normally by n(key) save ciphered char into ctVector[gid]



```c
__kernel void cipher(int key,
                     __global char *ptVector,
                     __global char *ctVector) {
    int gid = get_global_id(0);
    if ((ptVector[gid]>= 'a') && (ptVector[gid] <= 'z')){   // treat lowercase
        ptVector[gid] = ptVector [gid] - 32 + key;          // -32 to become upper case (ascii table)
        if (ptVector[gid] > 'Z'){
            ptVector[gid] = ptVector[gid] - 'Z' + 'A' - 1; //if exceeds Z even after ciphering, make it lower than Z, but greater than A
        }else if (ptVector[gid] < 'A') {
            ptVector[gid] = ptVector[gid] - 'A' + 'Z' + 1; //If lower than A after ciphering, make it higher than A, but lower than Z
        }
        ctVector[gid] = ptVector[gid];
    }
    else if ((ptVector[gid] >= 'A') && (ptVector[gid] <= 'Z')) {      // treat uppercase
        ptVector[gid] = ptVector[gid] + key;
        if (ptVector[gid] > 'Z') {
            ptVector[gid] = ptVector[gid] - 'Z' + 'A' - 1;
        }
        else if (ptVector[gid] < 'A') {
            ptVector[gid] = ptVector[gid] - 'A' + 'Z' + 1;
        }
        ctVector[gid] = ptVector[gid];
    }
    else {                  // treat the other char as normal
        ctVector[gid] = ptVector[gid];
    }
}
```

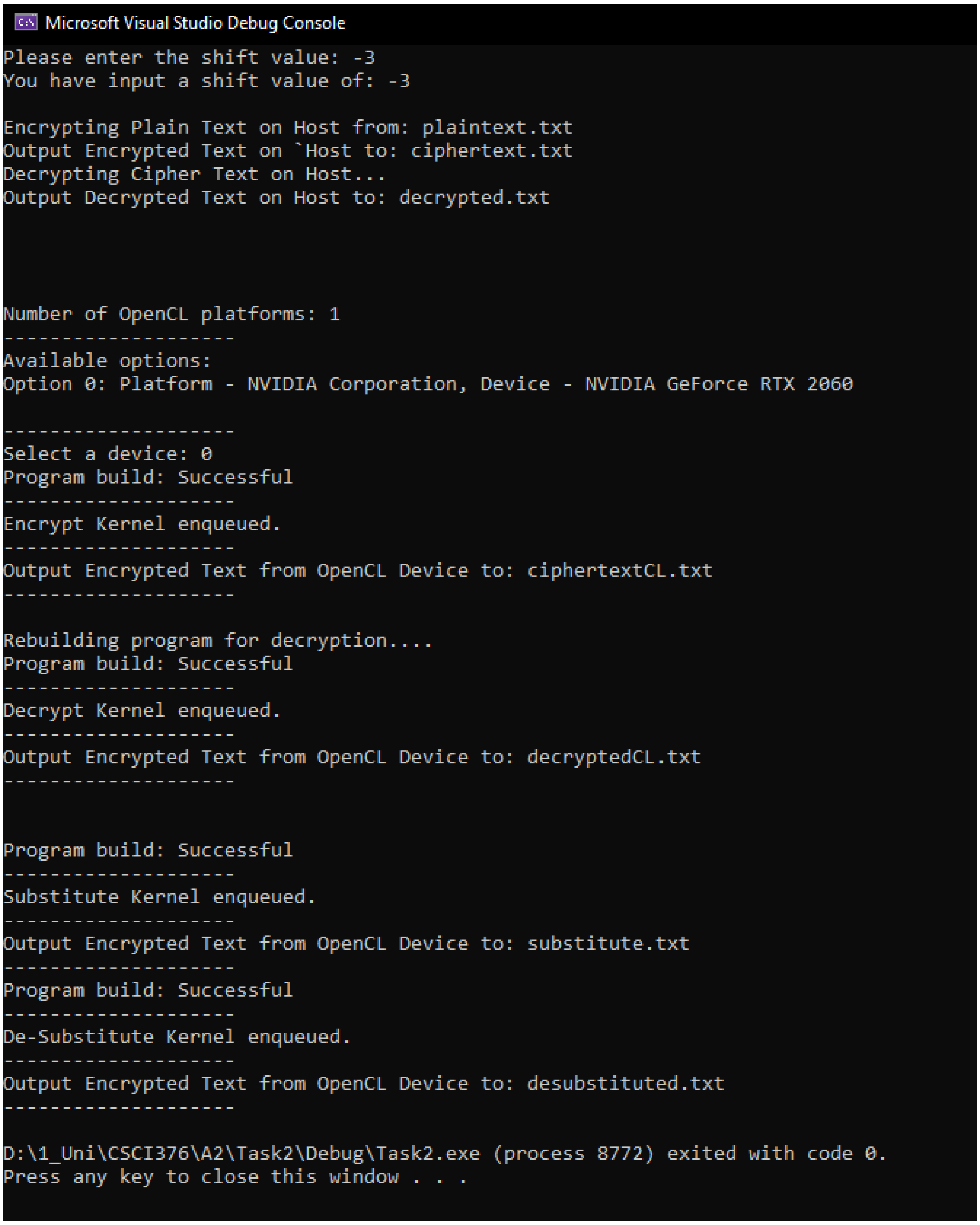
Deciper.cl revert back the cipher using the same key.



```
__kernel void decipher(int key,
                     __global char *ptVector,
                     __global char *ctVector) {

    int gid = get_global_id(0);
    if ((ctVector[gid] >= 'A') && (ctVector[gid] <= 'Z')) {
        ctVector[gid] = ctVector[gid] - key;

        if (ctVector[gid] > 'Z') {
            ctVector[gid] = ctVector[gid] - 'Z' + 'A' - 1;
        }
        else if (ctVector[gid] < 'A') {
            ctVector[gid] = ctVector[gid] - 'A' + 'Z' + 1;
        }

        ptVector[gid] = ctVector[gid];
    }
    else {
        ptVector[gid] = ctVector[gid];
    }

}
```

Substitute and desubstitute are basically switch case to change it based on the table given

Display Output

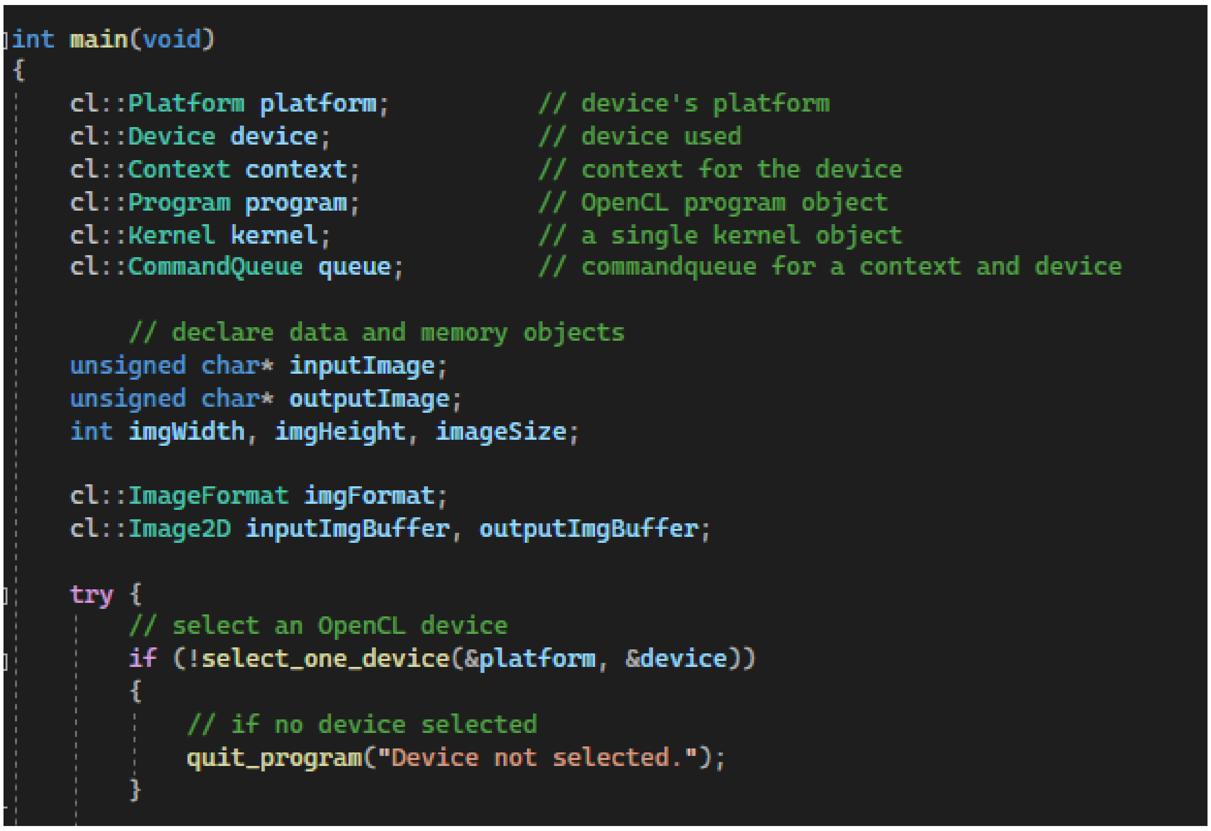


```
Please enter the shift value: -3
You have input a shift value of: -3

Encrypting Plain Text on Host from: plaintext.txt
Output Encrypted Text on `Host to: ciphertext.txt
Decrypting Cipher Text on Host...
Output Decrypted Text on Host to: decrypted.txt




Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 2060

--------------------
Select a device: 0
Program build: Successful
--------------------
Encrypt Kernel enqueued.
--------------------
Output Encrypted Text from OpenCL Device to: ciphertextCL.txt
--------------------

Rebuilding program for decryption....
Program build: Successful
--------------------
Decrypt Kernel enqueued.
--------------------
Output Encrypted Text from OpenCL Device to: decryptedCL.txt
--------------------


Program build: Successful
--------------------
Substitute Kernel enqueued.
--------------------
Output Encrypted Text from OpenCL Device to: substitute.txt
--------------------
Program build: Successful
--------------------
De-Substitute Kernel enqueued.
--------------------
Output Encrypted Text from OpenCL Device to: desubstituted.txt
--------------------

D:\1_Uni\CSCI376\A2\Task2\Debug\Task2.exe (process 8772) exited with code 0.
Press any key to close this window . . .
```

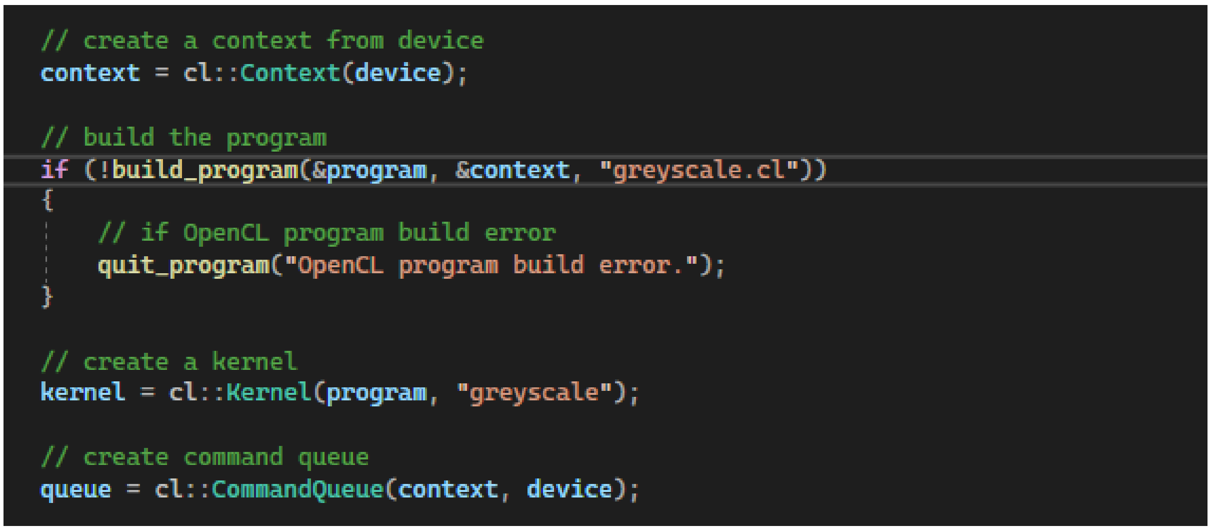
Task3

Task3a

Standard OpenCL procedures

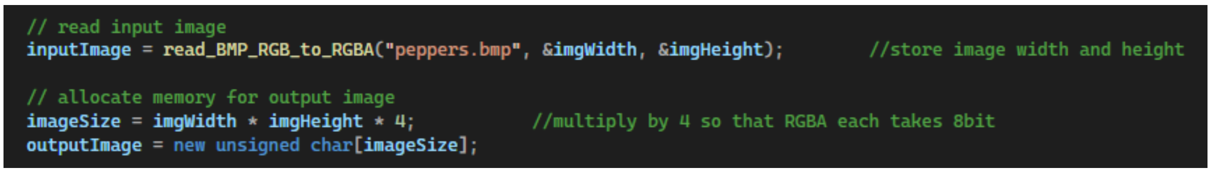


```
int main(void)
{
    cl::Platform platform;          // device's platform
    cl::Device device;              // device used
    cl::Context context;            // context for the device
    cl::Program program;            // OpenCL program object
    cl::Kernel kernel;              // a single kernel object
    cl::CommandQueue queue;         // commandqueue for a context and device

        // declare data and memory objects
    unsigned char* inputImage;
    unsigned char* outputImage;
    int imgWidth, imgHeight, imageSize;

    cl::ImageFormat imgFormat;
    cl::Image2D inputImgBuffer, outputImgBuffer;

    try {
        // select an OpenCL device
        if (!select_one_device(&platform, &device))
        {
            // if no device selected
            quit_program("Device not selected.");
        }
```



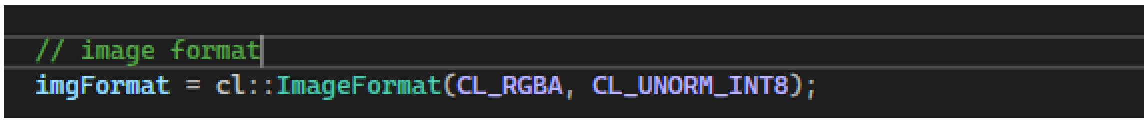
```
  // create a context from device
  context = cl::Context(device);

  // build the program
  if (!build_program(&program, &context, "greyscale.cl"))
  {
      // if OpenCL program build error
      quit_program("OpenCL program build error.");
  }

  // create a kernel
  kernel = cl::Kernel(program, "greyscale");

  // create command queue
  queue = cl::CommandQueue(context, device);
```

Read input image as well as preallocate memory space for image



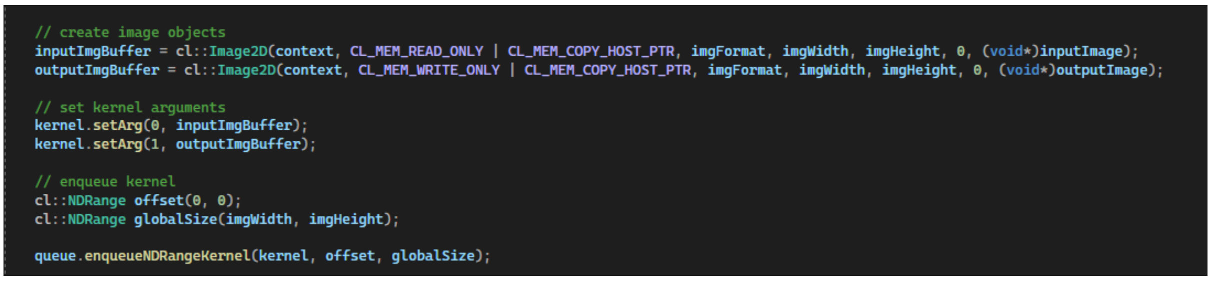
```
// read input image
inputImage = read_BMP_RGB_to_RGBA("peppers.bmp", &imgWidth, &imgHeight);        //store image width and height

// allocate memory for output image
imageSize = imgWidth * imgHeight * 4;           //multiply by 4 so that RGBA each takes 8bit
outputImage = new unsigned char[imageSize];
```

Specifying image format



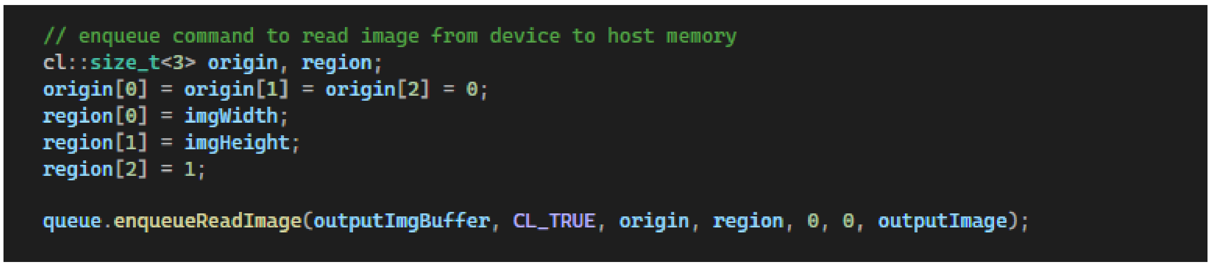
```
// image format
imgFormat = cl::ImageFormat(CL_RGBA, CL_UNORM_INT8);
```

Create image objects, kernel arguments, and enqueueing kernel



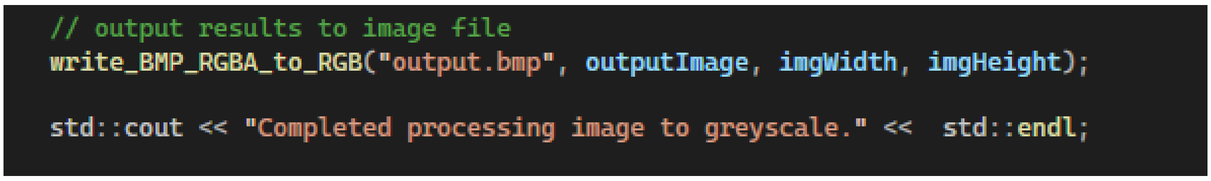
```
// create image objects
inputImgBuffer = cl::Image2D(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)inputImage);
outputImgBuffer = cl::Image2D(context, CL_MEM_WRITE_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)outputImage);

// set kernel arguments
kernel.setArg(0, inputImgBuffer);
kernel.setArg(1, outputImgBuffer);

// enqueue kernel
cl::NDRange offset(0, 0);
cl::NDRange globalSize(imgWidth, imgHeight);

queue.enqueueNDRangeKernel(kernel, offset, globalSize);
```

Enqueue command to read image from device to host memory



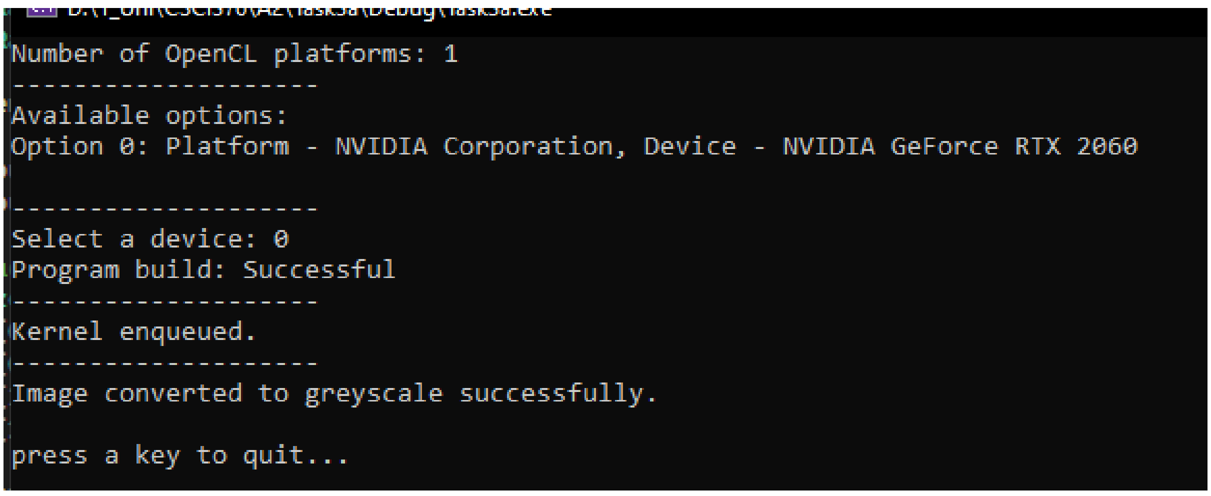
```
// enqueue command to read image from device to host memory
cl::size_t<3> origin, region;
origin[0] = origin[1] = origin[2] = 0;
region[0] = imgWidth;
region[1] = imgHeight;
region[2] = 1;

queue.enqueueReadImage(outputImgBuffer, CL_TRUE, origin, region, 0, 0, outputImage);
```

Output result as “Output.bmp” using method write_BMP_RGBA_to_RGB

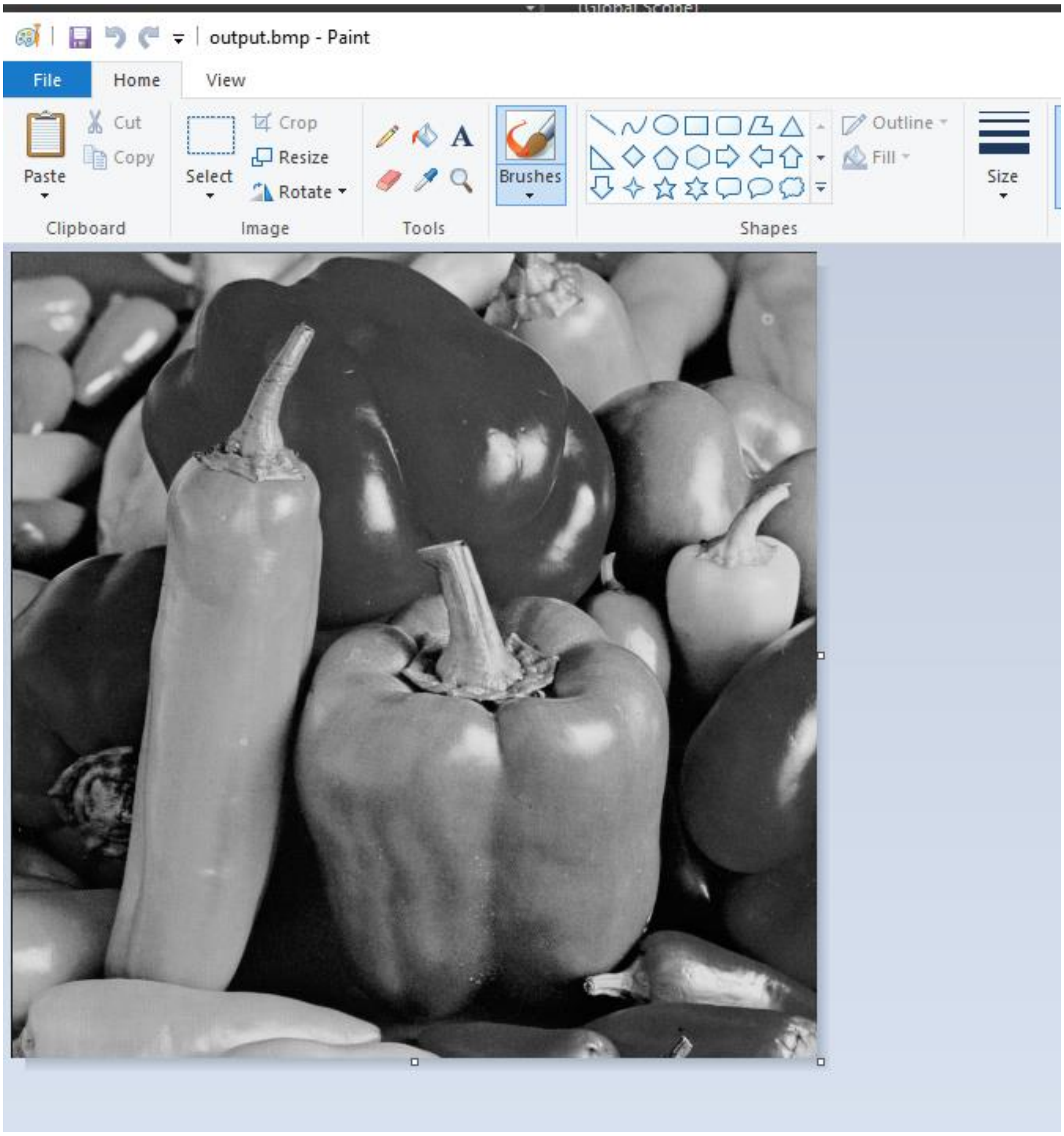


```
// output results to image file
write_BMP_RGBA_to_RGB("output.bmp", outputImage, imgWidth, imgHeight);

std::cout << "Completed processing image to greyscale." <<  std::endl;
```

Display Output

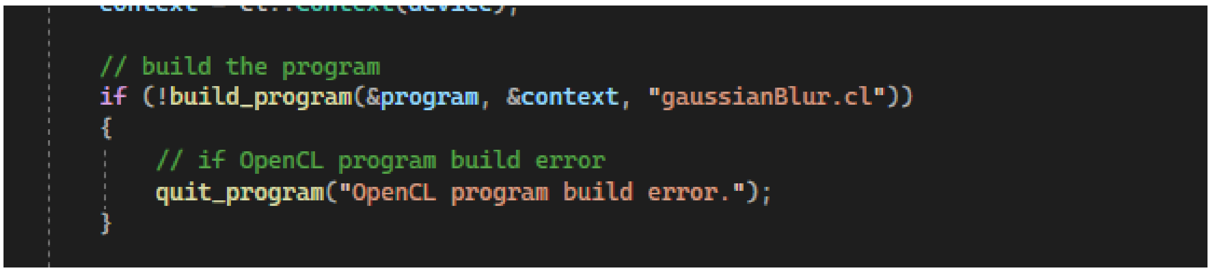


```
Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 2060

--------------------
Select a device: 0
Program build: Successful
--------------------
Kernel enqueued.
--------------------
Image converted to greyscale successfully.

press a key to quit...
```


output.bmp - Paint
File
Home
View
Paste
Cut
Copy
Clipboard
Select
Crop
Resize
Rotate
Image
Tools
Brushes
Shapes
Outline
Fill
Size

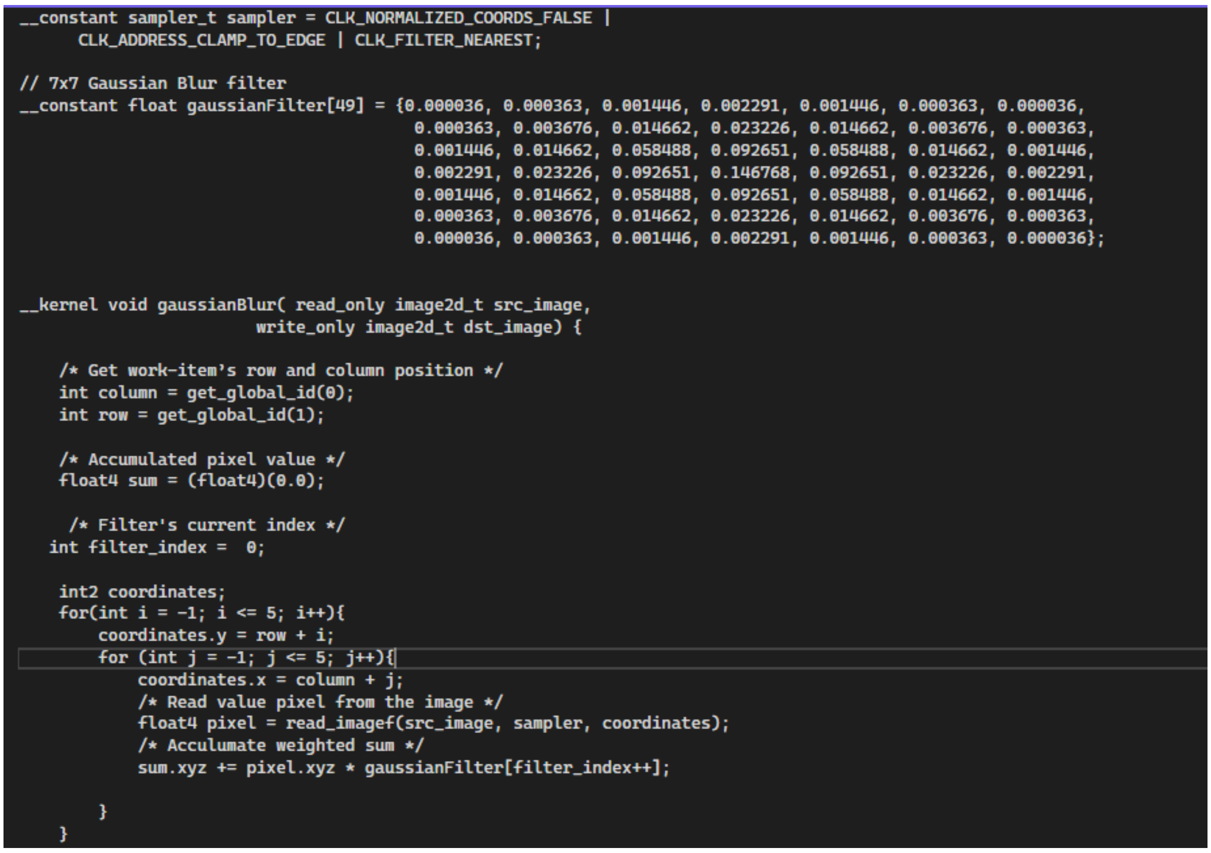
Task3b

Build program gaussianBLur.cl

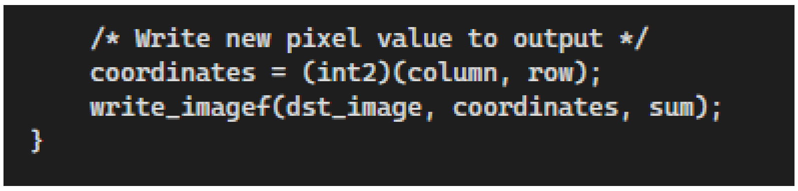


```
// build the program
if (!build_program(&program, &context, "gaussianBlur.cl"))
{
    // if OpenCL program build error
    quit_program("OpenCL program build error.");
}
```

gaussianBLur.cl

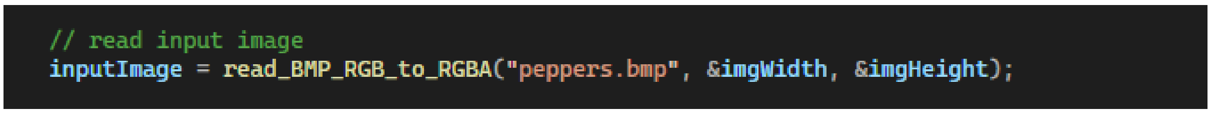


```
__constant sampler_t sampler = CLK_NORMALIZED_COORDS_FALSE |
      CLK_ADDRESS_CLAMP_TO_EDGE | CLK_FILTER_NEAREST;

// 7x7 Gaussian Blur filter
__constant float gaussianFilter[49] = {0.000036, 0.000363, 0.001446, 0.002291, 0.001446, 0.000363, 0.000036,
                                        0.000363, 0.003676, 0.014662, 0.023226, 0.014662, 0.003676, 0.000363,
                                        0.001446, 0.014662, 0.058488, 0.092651, 0.058488, 0.014662, 0.001446,
                                        0.002291, 0.023226, 0.092651, 0.146768, 0.092651, 0.023226, 0.002291,
                                        0.001446, 0.014662, 0.058488, 0.092651, 0.058488, 0.014662, 0.001446,
                                        0.000363, 0.003676, 0.014662, 0.023226, 0.014662, 0.003676, 0.000363,
                                        0.000036, 0.000363, 0.001446, 0.002291, 0.001446, 0.000363, 0.000036};


__kernel void gaussianBlur( read_only image2d_t src_image,
                        write_only image2d_t dst_image) {

    /* Get work-item's row and column position */
    int column = get_global_id(0);
    int row = get_global_id(1);

    /* Accumulated pixel value */
    float4 sum = (float4)(0.0);

     /* Filter's current index */
   int filter_index =  0;

    int2 coordinates;
    for(int i = -1; i <= 5; i++){
        coordinates.y = row + i;
        for (int j = -1; j <= 5; j++){
            coordinates.x = column + j;
            /* Read value pixel from the image */
            float4 pixel = read_imagef(src_image, sampler, coordinates);
            /* Acculumate weighted sum */
            sum.xyz += pixel.xyz * gaussianFilter[filter_index++];

        }
    }
```



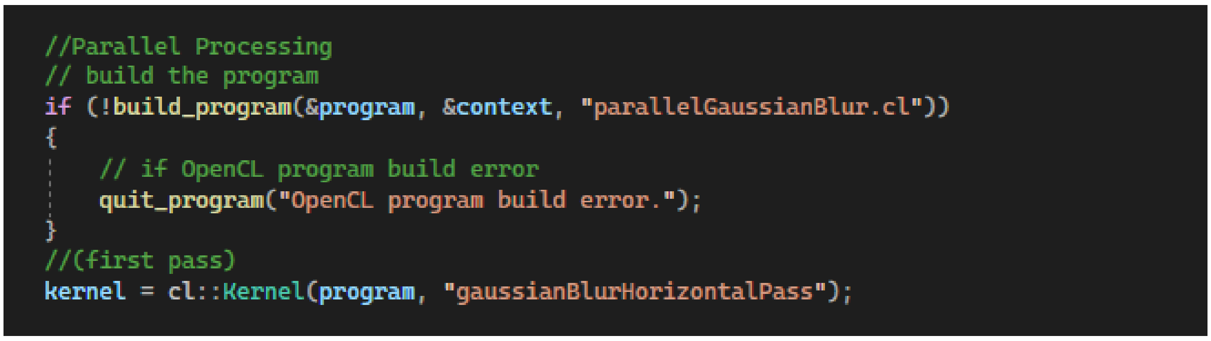
```
    /* Write new pixel value to output */
    coordinates = (int2)(column, row);
    write_imagef(dst_image, coordinates, sum);
}
```

Read input image



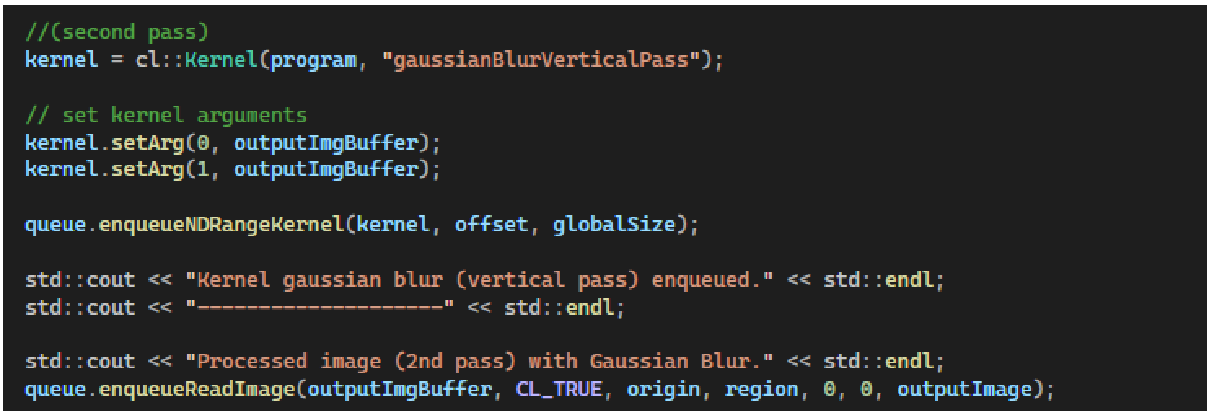
```
// read input image
inputImage = read_BMP_RGB_to_RGBA("peppers.bmp", &imgWidth, &imgHeight);
```

Naïve gaussian blur via parallel processing (horizontal pass)



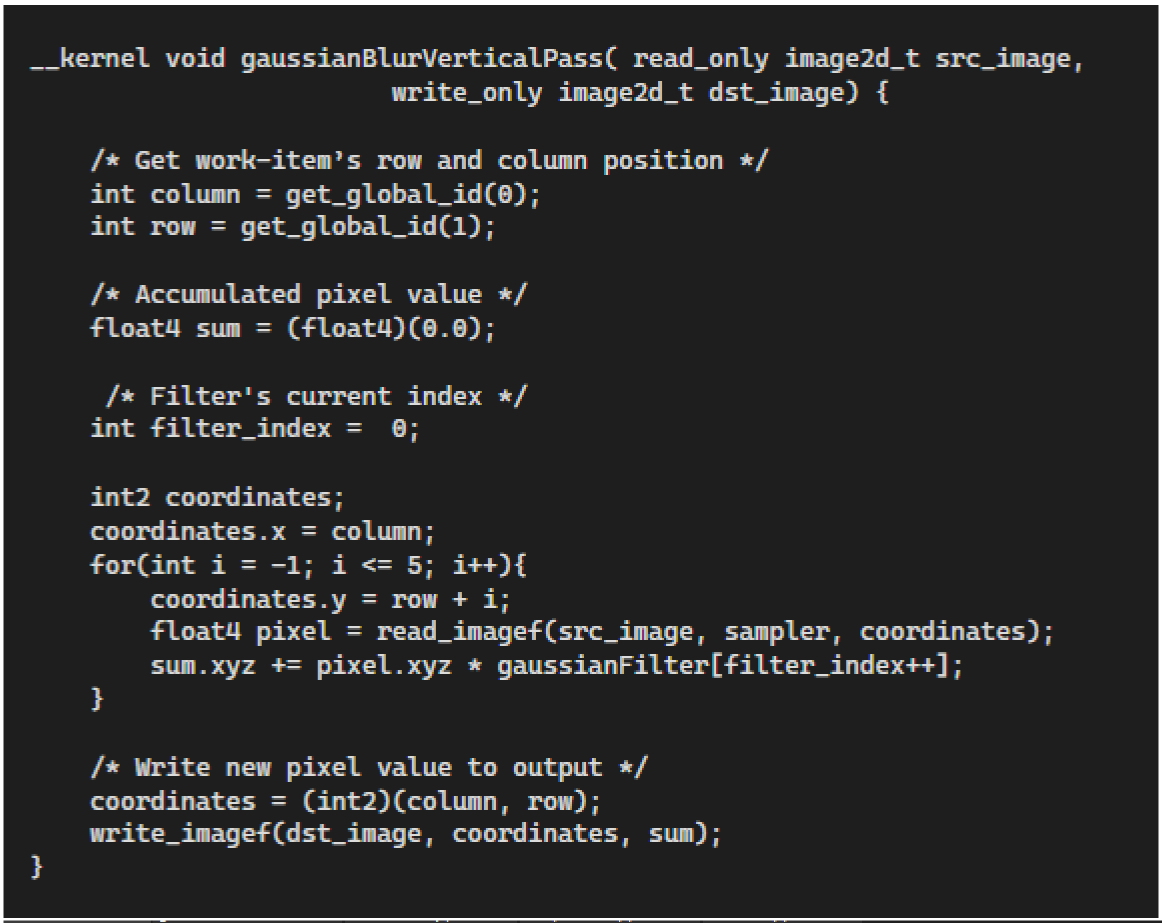
```
//Parallel Processing
// build the program
if (!build_program(&program, &context, "parallelGaussianBlur.cl"))
{
    // if OpenCL program build error
    quit_program("OpenCL program build error.");
}
//(first pass)
kernel = cl::Kernel(program, "gaussianBlurHorizontalPass");
```

Vertical pass

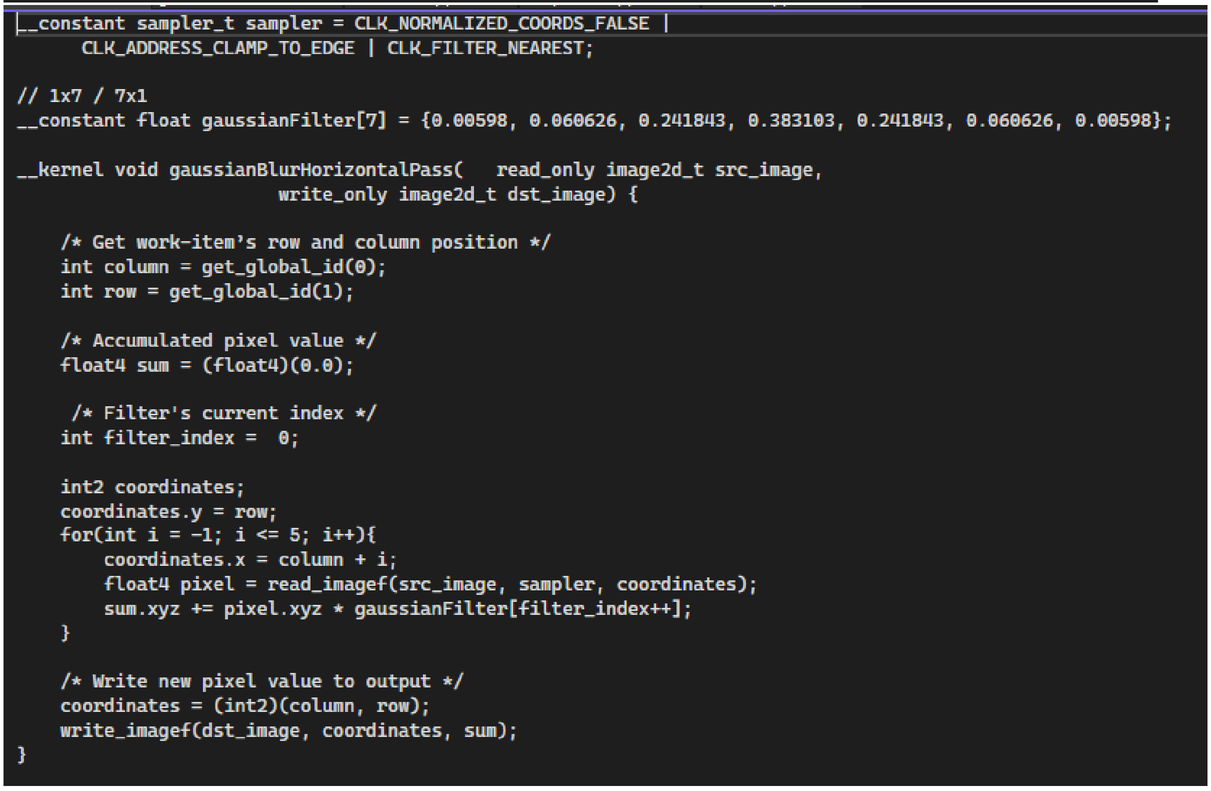


```
//(second pass)
kernel = cl::Kernel(program, "gaussianBlurVerticalPass");

// set kernel arguments
kernel.setArg(0, outputImgBuffer);
kernel.setArg(1, outputImgBuffer);

queue.enqueueNDRangeKernel(kernel, offset, globalSize);

std::cout << "Kernel gaussian blur (vertical pass) enqueued." << std::endl;
std::cout << "--------------------" << std::endl;

std::cout << "Processed image (2nd pass) with Gaussian Blur." << std::endl;
queue.enqueueReadImage(outputImgBuffer, CL_TRUE, origin, region, 0, 0, outputImage);
```

parallelGaussianBlur.cl (vertical pass and horizontal pass both using the same 1 * 7 filter)

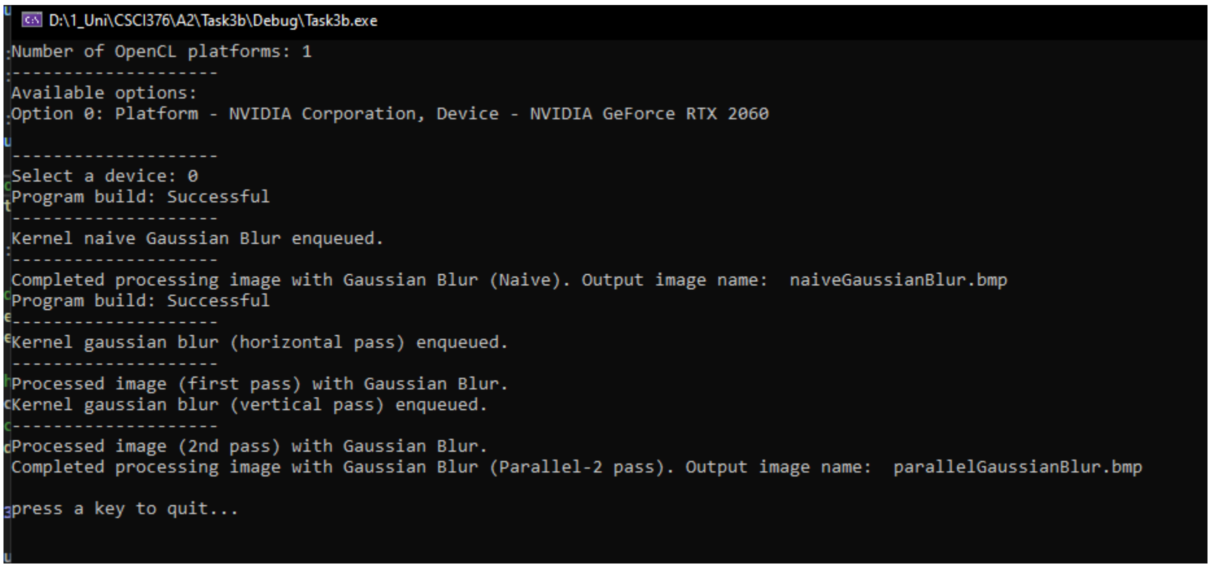


```
__kernel void gaussianBlurVerticalPass( read_only image2d_t src_image,
                        write_only image2d_t dst_image) {

    /* Get work-item's row and column position */
    int column = get_global_id(0);
    int row = get_global_id(1);

    /* Accumulated pixel value */
    float4 sum = (float4)(0.0);

     /* Filter's current index */
    int filter_index =  0;

    int2 coordinates;
    coordinates.x = column;
    for(int i = -1; i <= 5; i++){
        coordinates.y = row + i;
        float4 pixel = read_imagef(src_image, sampler, coordinates);
        sum.xyz += pixel.xyz * gaussianFilter[filter_index++];
    }

    /* Write new pixel value to output */
    coordinates = (int2)(column, row);
    write_imagef(dst_image, coordinates, sum);
}
```



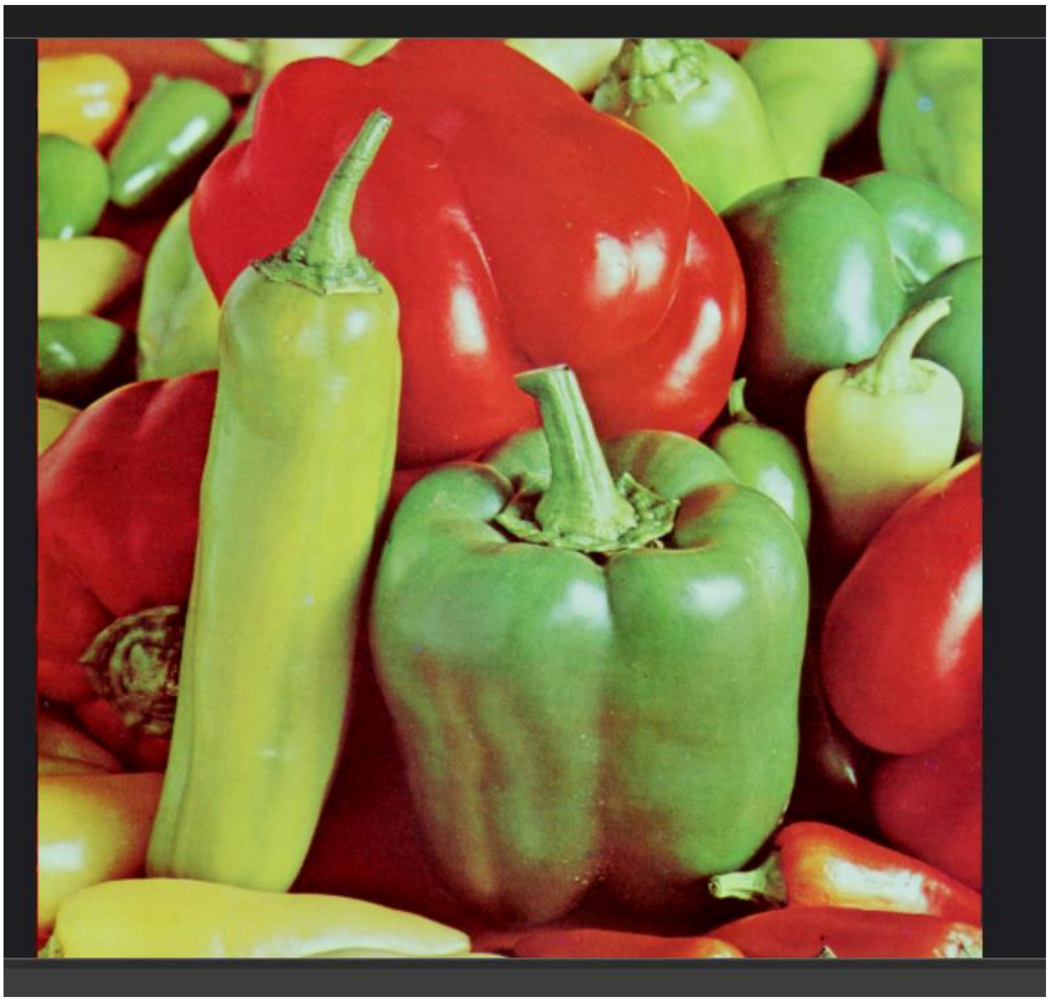
```
__constant sampler_t sampler = CLK_NORMALIZED_COORDS_FALSE |
      CLK_ADDRESS_CLAMP_TO_EDGE | CLK_FILTER_NEAREST;

// 1x7 / 7x1
__constant float gaussianFilter[7] = {0.00598, 0.060626, 0.241843, 0.383103, 0.241843, 0.060626, 0.00598};

__kernel void gaussianBlurHorizontalPass(   read_only image2d_t src_image,
                        write_only image2d_t dst_image) {

    /* Get work-item's row and column position */
    int column = get_global_id(0);
    int row = get_global_id(1);

    /* Accumulated pixel value */
    float4 sum = (float4)(0.0);

     /* Filter's current index */
    int filter_index =  0;

    int2 coordinates;
    coordinates.y = row;
    for(int i = -1; i <= 5; i++){
        coordinates.x = column + i;
        float4 pixel = read_imagef(src_image, sampler, coordinates);
        sum.xyz += pixel.xyz * gaussianFilter[filter_index++];
    }

    /* Write new pixel value to output */
    coordinates = (int2)(column, row);
    write_imagef(dst_image, coordinates, sum);
}
```

Display output

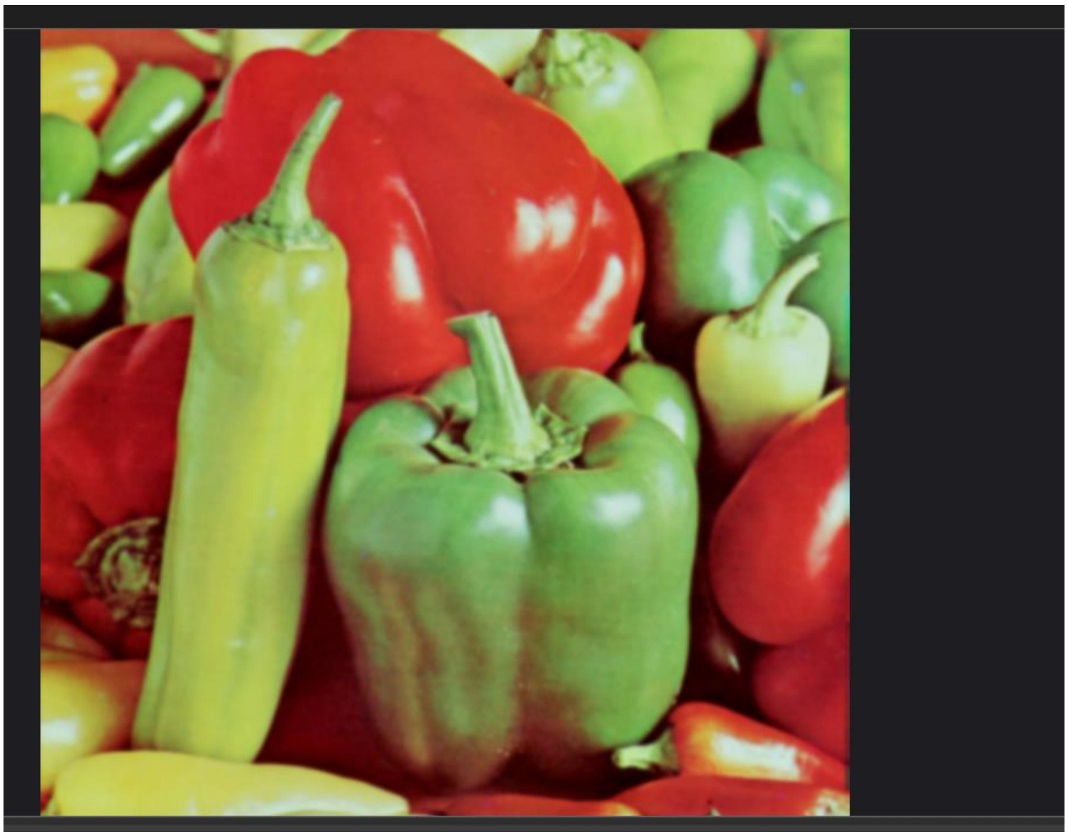


```
D:\1_Uni\CSCI376\A2\Task3b\Debug\Task3b.exe
Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 2060

--------------------
Select a device: 0
Program build: Successful
--------------------
Kernel naive Gaussian Blur enqueued.
--------------------
Completed processing image with Gaussian Blur (Naive). Output image name:  naiveGaussianBlur.bmp
Program build: Successful
--------------------
Kernel gaussian blur (horizontal pass) enqueued.
--------------------
Processed image (first pass) with Gaussian Blur.
Kernel gaussian blur (vertical pass) enqueued.
--------------------
Processed image (2nd pass) with Gaussian Blur.
Completed processing image with Gaussian Blur (Parallel-2 pass). Output image name:  parallelGaussianBlur.bmp

press a key to quit...
```

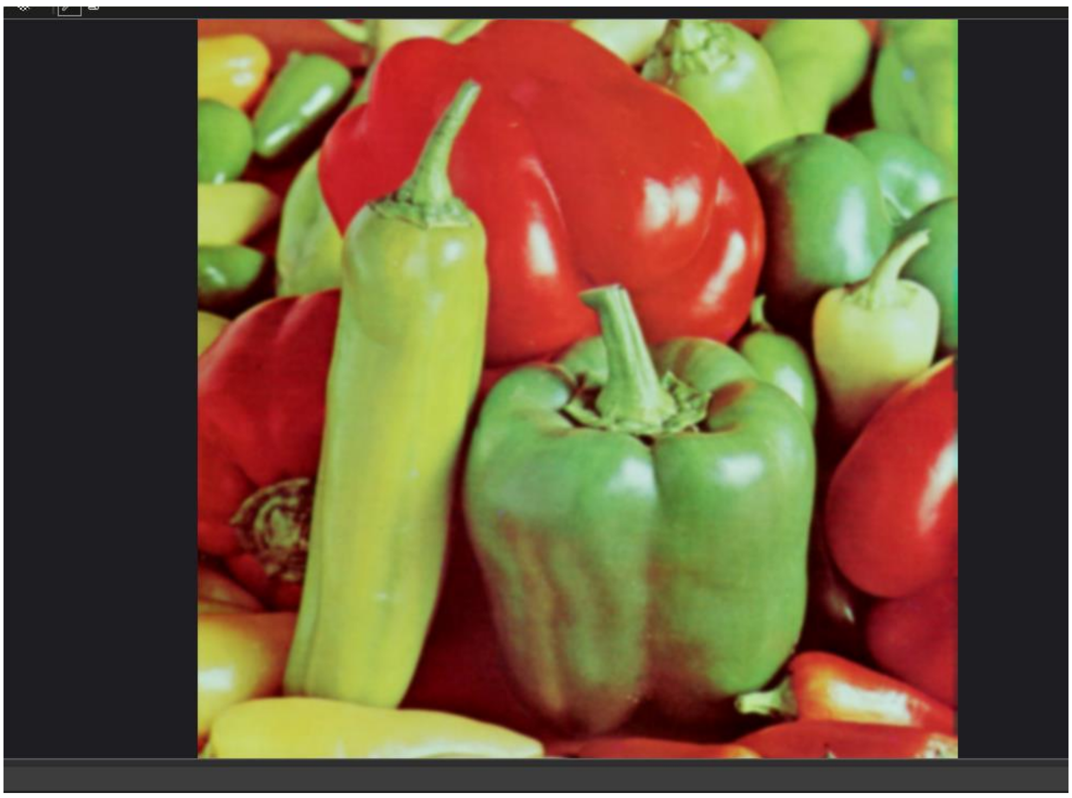
Original peppers.bmp



naiveGaussianBlur.bmp



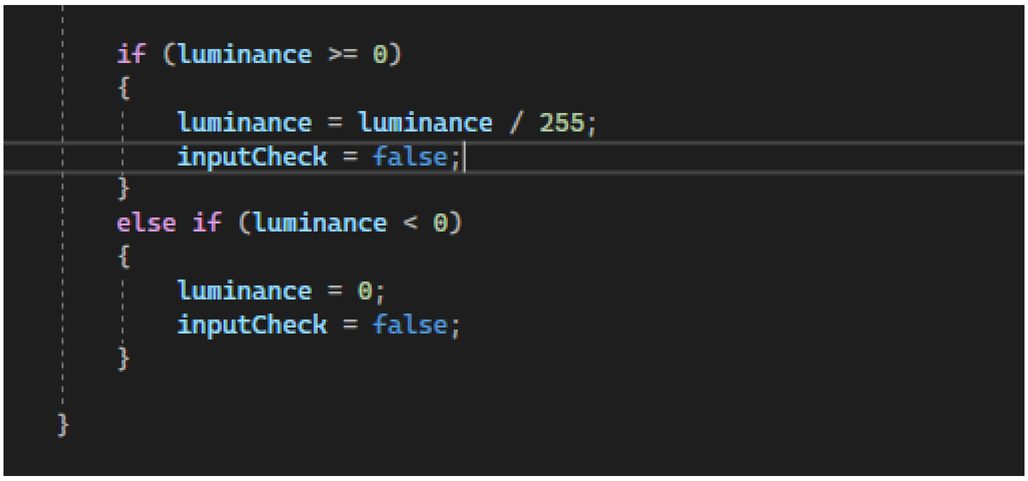
parallelGaussianBlur.bmp



Task3c

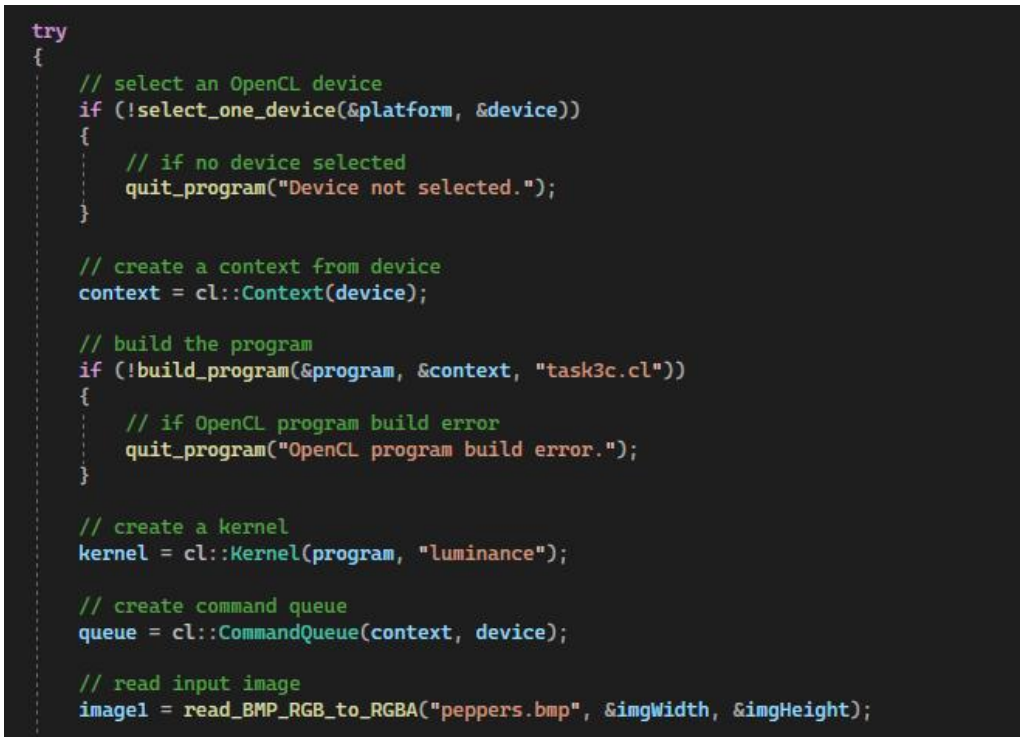
Used 3a and 3b to set for luminance as well as parallel processing(vertical and horizontal)

Checking user input as threshold luminance value of 0 - 255 (if above, keep same, however, output will be black, if below, keep it as 0)

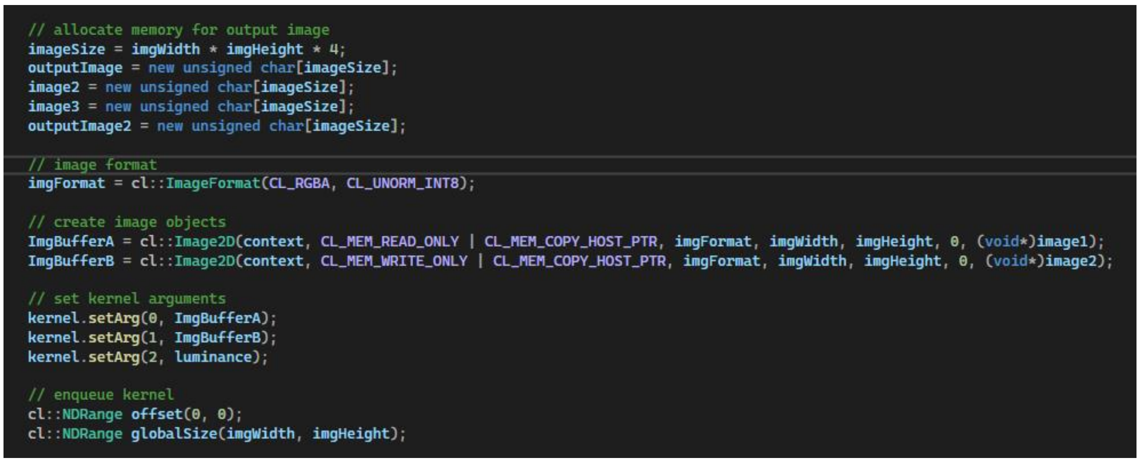


```
        if (luminance >= 0)
        {
            luminance = luminance / 255;
            inputCheck = false;
        }
        else if (luminance < 0)
        {
            luminance = 0;
            inputCheck = false;
        }

    }
```

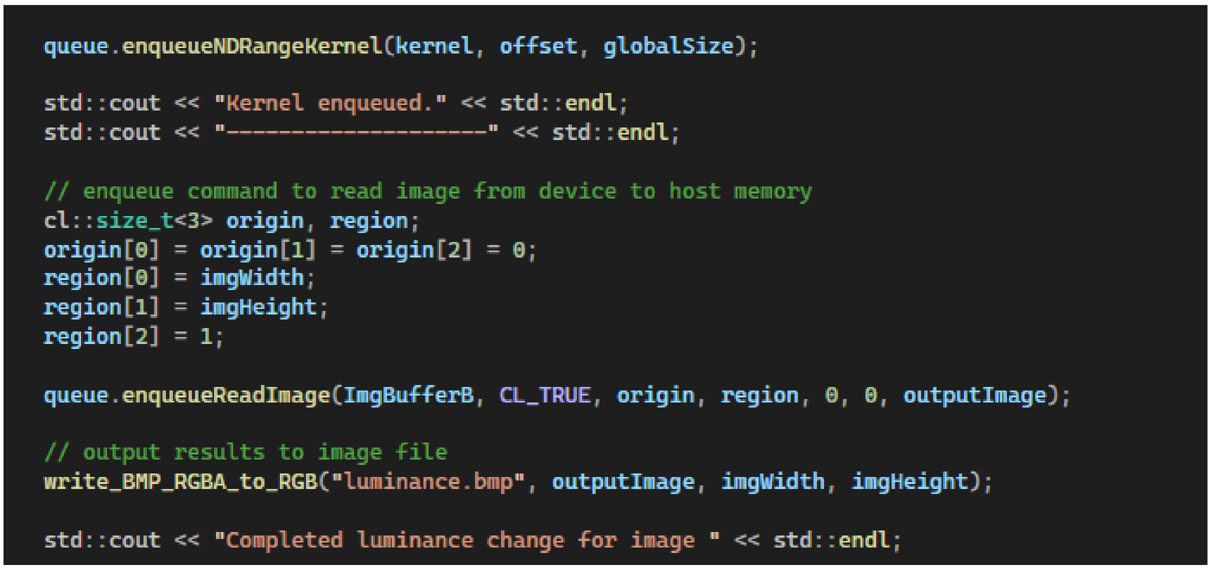
Standard OpcnCL procedure,choose device, create context, build program task3c.cl, create kernel, command queue, read in image, allocate memory for output, set image format, create image objects, set kernel args, enqueue kernel, etc



```
try
{
    // select an OpenCL device
    if (!select_one_device(&platform, &device))
    {
        // if no device selected
        quit_program("Device not selected.");
    }

    // create a context from device
    context = cl::Context(device);

    // build the program
    if (!build_program(&program, &context, "task3c.cl"))
    {
        // if OpenCL program build error
        quit_program("OpenCL program build error.");
    }

    // create a kernel
    kernel = cl::Kernel(program, "luminance");

    // create command queue
    queue = cl::CommandQueue(context, device);

    // read input image
    image1 = read_BMP_RGB_to_RGBA("peppers.bmp", &imgWidth, &imgHeight);
```

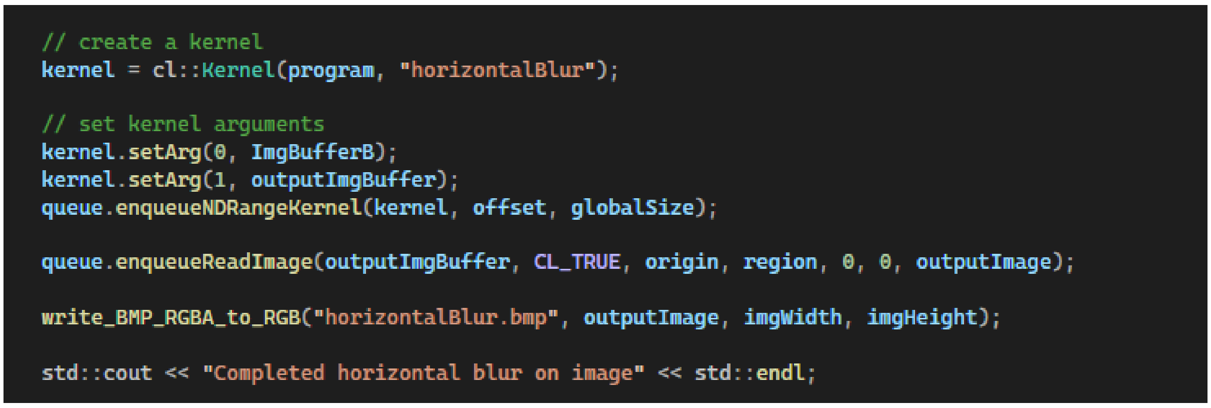


```
// allocate memory for output image
imageSize = imgWidth * imgHeight * 4;
outputImage = new unsigned char[imageSize];
image2 = new unsigned char[imageSize];
image3 = new unsigned char[imageSize];
outputImage2 = new unsigned char[imageSize];

// image format
imgFormat = cl::ImageFormat(CL_RGBA, CL_UNORM_INT8);

// create image objects
ImgBufferA = cl::Image2D(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)image1);
ImgBufferB = cl::Image2D(context, CL_MEM_WRITE_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)image2);

// set kernel arguments
kernel.setArg(0, ImgBufferA);
kernel.setArg(1, ImgBufferB);
kernel.setArg(2, luminance);

// enqueue kernel
cl::NDRange offset(0, 0);
cl::NDRange globalSize(imgWidth, imgHeight);
```

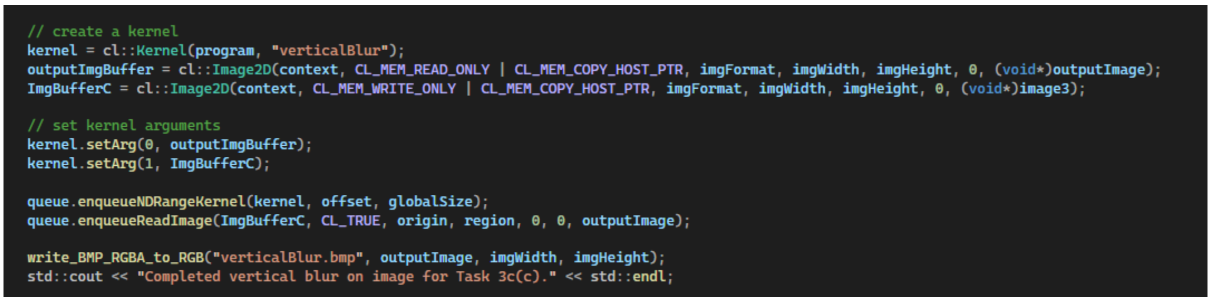


```
queue.enqueueNDRangeKernel(kernel, offset, globalSize);

std::cout << "Kernel enqueued." << std::endl;
std::cout << "--------------------" << std::endl;

// enqueue command to read image from device to host memory
cl::size_t<3> origin, region;
origin[0] = origin[1] = origin[2] = 0;
region[0] = imgWidth;
region[1] = imgHeight;
region[2] = 1;

queue.enqueueReadImage(ImgBufferB, CL_TRUE, origin, region, 0, 0, outputImage);

// output results to image file
write_BMP_RGBA_to_RGB("luminance.bmp", outputImage, imgWidth, imgHeight);

std::cout << "Completed luminance change for image " << std::endl;
```

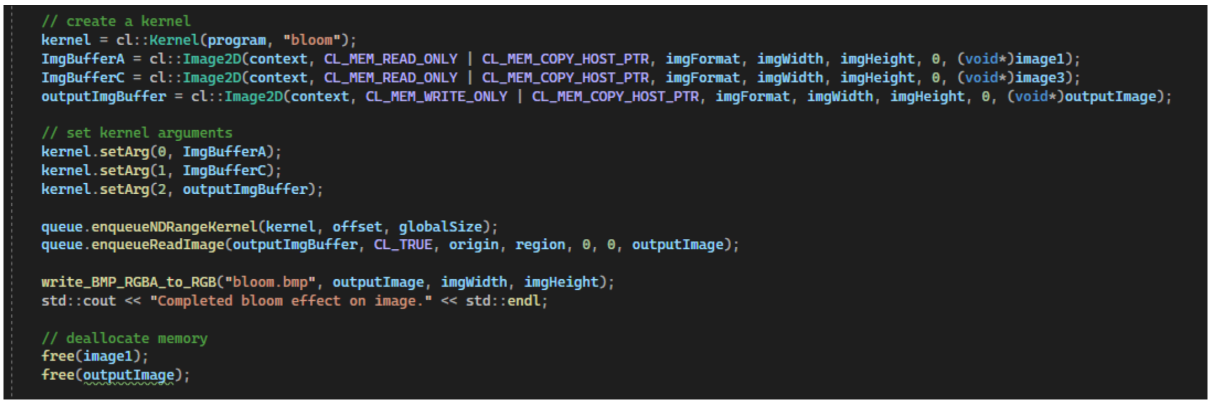
Next, repeat it for horizontal processing, vertical processing and finally blooming



```
// create a kernel
kernel = cl::Kernel(program, "horizontalBlur");

// set kernel arguments
kernel.setArg(0, ImgBufferB);
kernel.setArg(1, outputImgBuffer);
queue.enqueueNDRangeKernel(kernel, offset, globalSize);

queue.enqueueReadImage(outputImgBuffer, CL_TRUE, origin, region, 0, 0, outputImage);

write_BMP_RGBA_to_RGB("horizontalBlur.bmp", outputImage, imgWidth, imgHeight);

std::cout << "Completed horizontal blur on image" << std::endl;
```



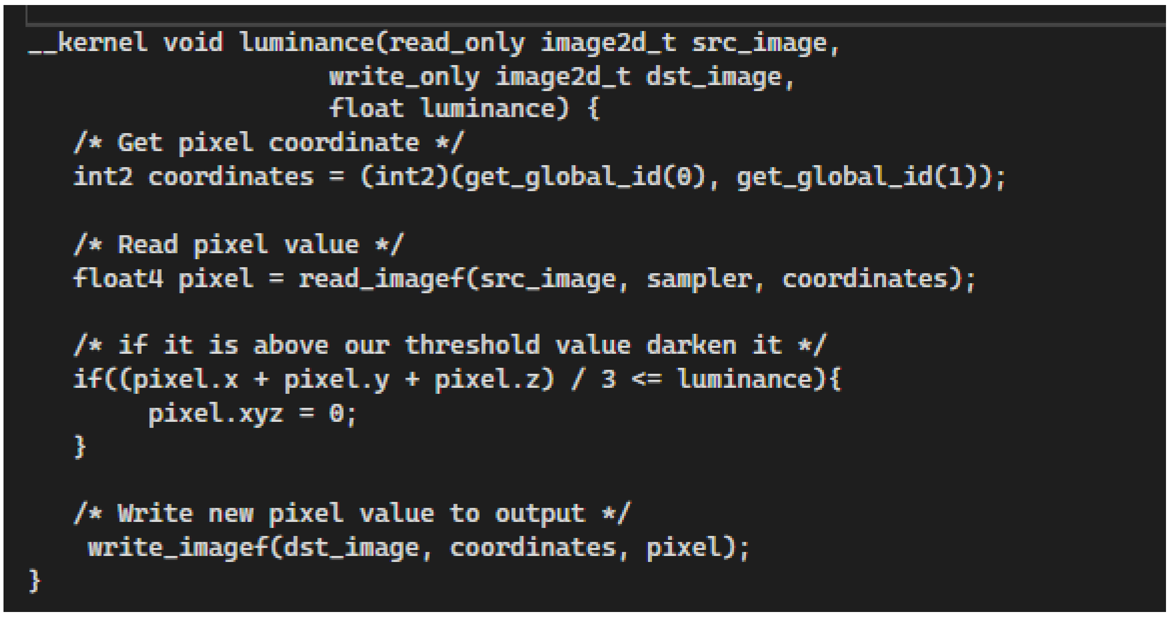
```
// create a kernel
kernel = cl::Kernel(program, "verticalBlur");
outputImgBuffer = cl::Image2D(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)outputImage);
ImgBufferC = cl::Image2D(context, CL_MEM_WRITE_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)image3);

// set kernel arguments
kernel.setArg(0, outputImgBuffer);
kernel.setArg(1, ImgBufferC);

queue.enqueueNDRangeKernel(kernel, offset, globalSize);
queue.enqueueReadImage(ImgBufferC, CL_TRUE, origin, region, 0, 0, outputImage);

write_BMP_RGBA_to_RGB("verticalBlur.bmp", outputImage, imgWidth, imgHeight);
std::cout << "Completed vertical blur on image for Task 3c(c)." << std::endl;
```



```
// create a kernel
kernel = cl::Kernel(program, "bloom");
ImgBufferA = cl::Image2D(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)image1);
ImgBufferC = cl::Image2D(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)image3);
outputImgBuffer = cl::Image2D(context, CL_MEM_WRITE_ONLY | CL_MEM_COPY_HOST_PTR, imgFormat, imgWidth, imgHeight, 0, (void*)outputImage);

// set kernel arguments
kernel.setArg(0, ImgBufferA);
kernel.setArg(1, ImgBufferC);
kernel.setArg(2, outputImgBuffer);

queue.enqueueNDRangeKernel(kernel, offset, globalSize);
queue.enqueueReadImage(outputImgBuffer, CL_TRUE, origin, region, 0, 0, outputImage);

write_BMP_RGBA_to_RGB("bloom.bmp", outputImage, imgWidth, imgHeight);
std::cout << "Completed bloom effect on image." << std::endl;

// deallocate memory
free(image1);
free(outputImage);
```

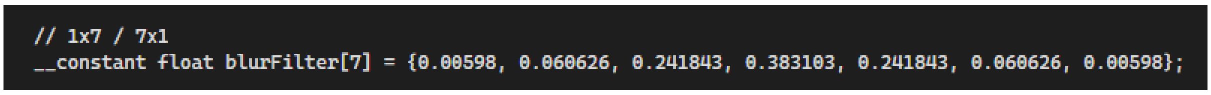
Task3c.cl

kernel for luminance, get pixel coordinate of image, as well as pixel value from source image, make sure to darken image if above threshold value, and finally write new pixel value to output.



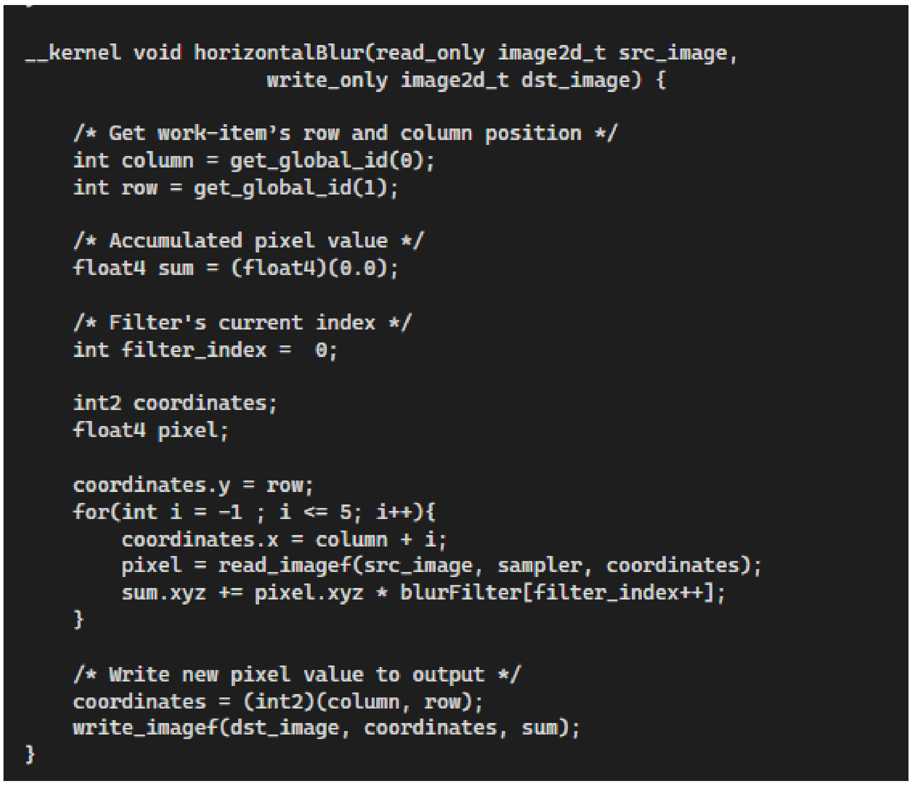
```
__kernel void luminance(read_only image2d_t src_image,
                    write_only image2d_t dst_image,
                    float luminance) {
   /* Get pixel coordinate */
   int2 coordinates = (int2)(get_global_id(0), get_global_id(1));

   /* Read pixel value */
   float4 pixel = read_imagef(src_image, sampler, coordinates);

   /* if it is above our threshold value darken it */
   if((pixel.x + pixel.y + pixel.z) / 3 <= luminance){
        pixel.xyz = 0;
   }

   /* Write new pixel value to output */
    write_imagef(dst_image, coordinates, pixel);
}
```

Filter used for horizontal and vertical bluring



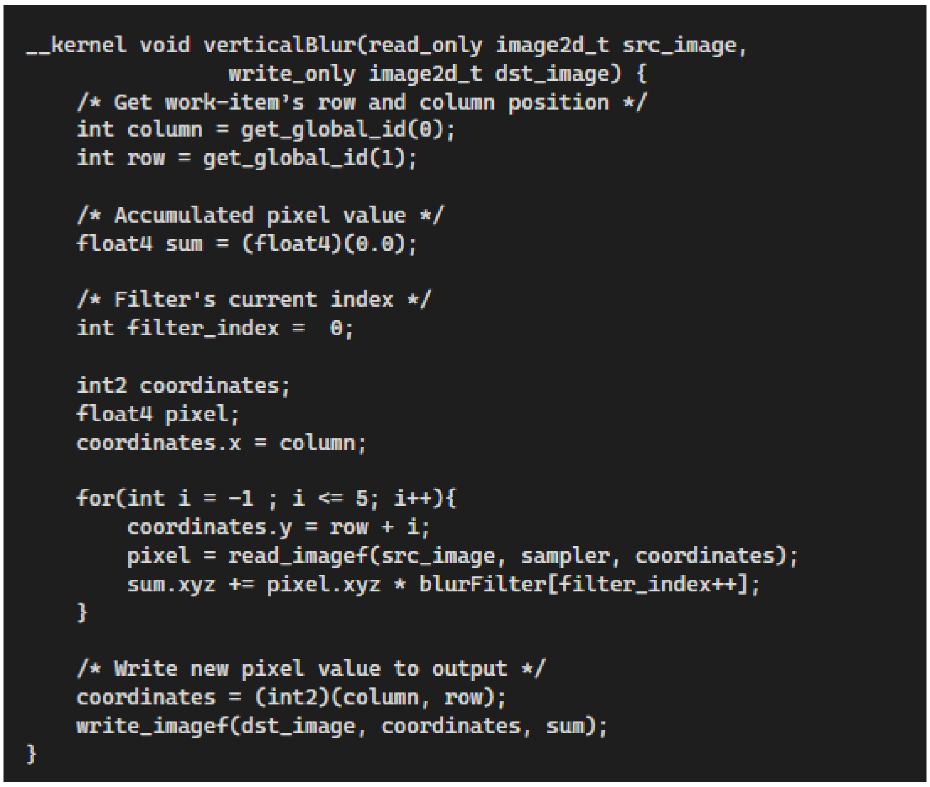
```
// 1x7 / 7x1
__constant float blurFilter[7] = {0.00598, 0.060626, 0.241843, 0.383103, 0.241843, 0.060626, 0.00598};
```

Get work item’s row and column, initialize needed variables, for each column, read in the image pixel values, and multiply it to the filter value. Then overwrite the image with new values



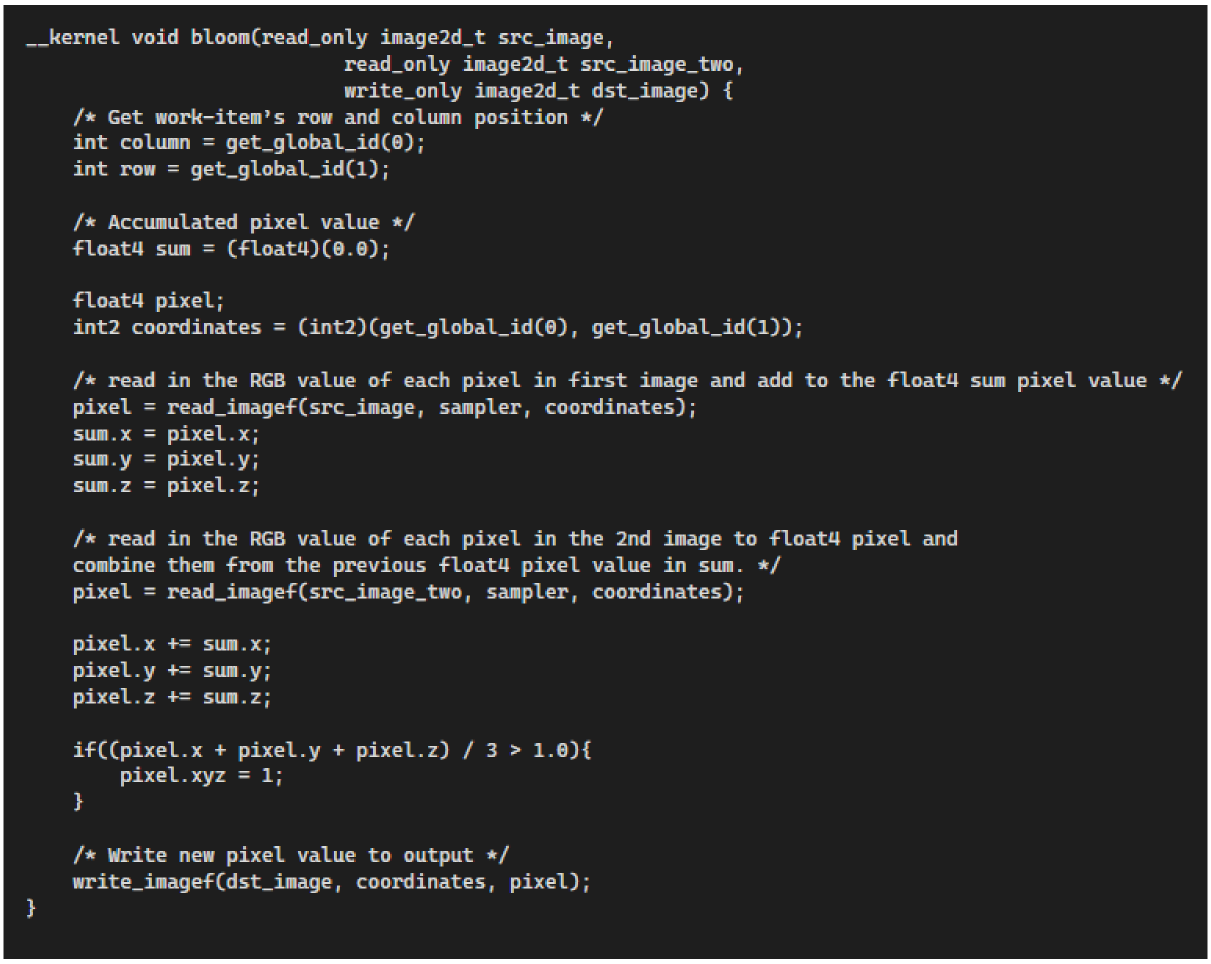
```
__kernel void horizontalBlur(read_only image2d_t src_image,
                    write_only image2d_t dst_image) {

    /* Get work-item’s row and column position */
    int column = get_global_id(0);
    int row = get_global_id(1);

    /* Accumulated pixel value */
    float4 sum = (float4)(0.0);

    /* Filter's current index */
    int filter_index =  0;

    int2 coordinates;
    float4 pixel;

    coordinates.y = row;
    for(int i = -1 ; i <= 5; i++){
        coordinates.x = column + i;
        pixel = read_imagef(src_image, sampler, coordinates);
        sum.xyz += pixel.xyz * blurFilter[filter_index++];
    }

    /* Write new pixel value to output */
    coordinates = (int2)(column, row);
    write_imagef(dst_image, coordinates, sum);
}
```

Repeat for vertical blur



```
__kernel void verticalBlur(read_only image2d_t src_image,
                  write_only image2d_t dst_image) {
    /* Get work-item’s row and column position */
    int column = get_global_id(0);
    int row = get_global_id(1);

    /* Accumulated pixel value */
    float4 sum = (float4)(0.0);

    /* Filter's current index */
    int filter_index =  0;

    int2 coordinates;
    float4 pixel;
    coordinates.x = column;

    for(int i = -1 ; i <= 5; i++){
        coordinates.y = row + i;
        pixel = read_imagef(src_image, sampler, coordinates);
        sum.xyz += pixel.xyz * blurFilter[filter_index++];
    }

    /* Write new pixel value to output */
    coordinates = (int2)(column, row);
    write_imagef(dst_image, coordinates, sum);
}
```

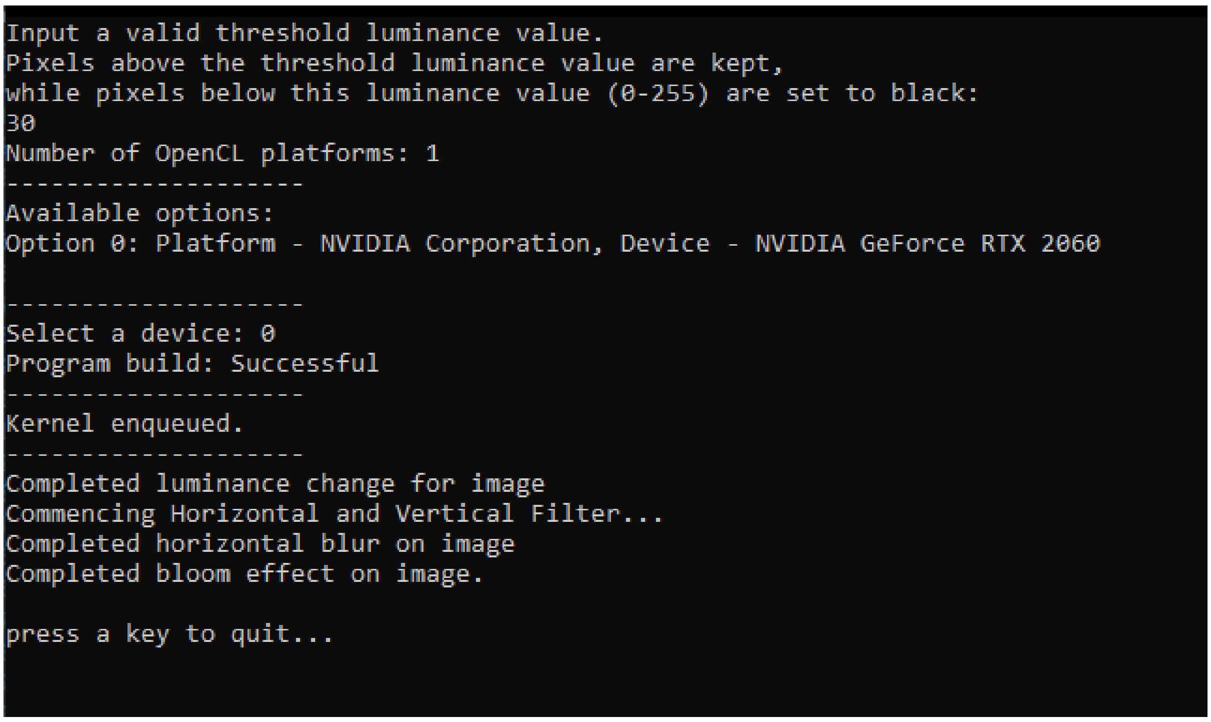
Kernel for Bloom, get work item’s row and column position, initialize variables, read in RGB values from first image, and add it to the sum pixel value, next, read in the second image value and add that to the pixel value, if value exceeds maximum threshold, change it to the max threshold. Finally, Overwrite the pixel value to output image.



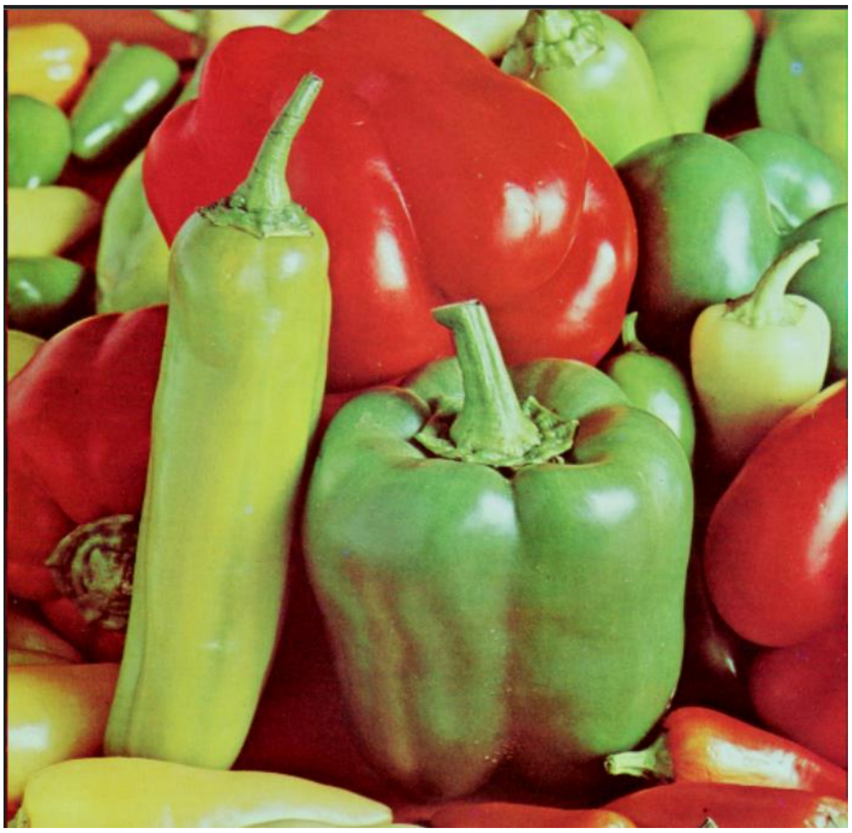
```
__kernel void bloom(read_only image2d_t src_image,
                           read_only image2d_t src_image_two,
                           write_only image2d_t dst_image) {
    /* Get work-item’s row and column position */
    int column = get_global_id(0);
    int row = get_global_id(1);

    /* Accumulated pixel value */
    float4 sum = (float4)(0.0);

    float4 pixel;
    int2 coordinates = (int2)(get_global_id(0), get_global_id(1));

    /* read in the RGB value of each pixel in first image and add to the float4 sum pixel value */
    pixel = read_imagef(src_image, sampler, coordinates);
    sum.x = pixel.x;
    sum.y = pixel.y;
    sum.z = pixel.z;

    /* read in the RGB value of each pixel in the 2nd image to float4 pixel and
    combine them from the previous float4 pixel value in sum. */
    pixel = read_imagef(src_image_two, sampler, coordinates);

    pixel.x += sum.x;
    pixel.y += sum.y;
    pixel.z += sum.z;

    if((pixel.x + pixel.y + pixel.z) / 3 > 1.0){
        pixel.xyz = 1;
    }

    /* Write new pixel value to output */
    write_imagef(dst_image, coordinates, pixel);
}
```

Display output

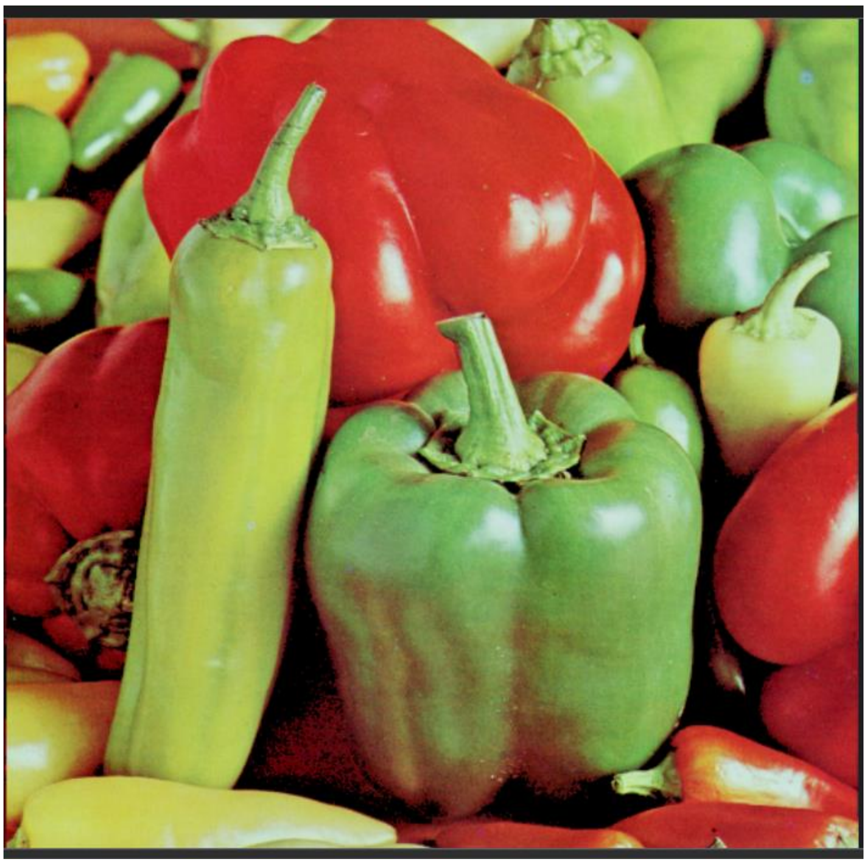
Luminance value : 30


```
Input a valid threshold luminance value.
Pixels above the threshold luminance value are kept,
while pixels below this luminance value (0-255) are set to black:
30
Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 2060

--------------------
Select a device: 0
Program build: Successful
--------------------
Kernel enqueued.
--------------------
Completed luminance change for image
Commencing Horizontal and Vertical Filter...
Completed horizontal blur on image
Completed bloom effect on image.

press a key to quit...
```

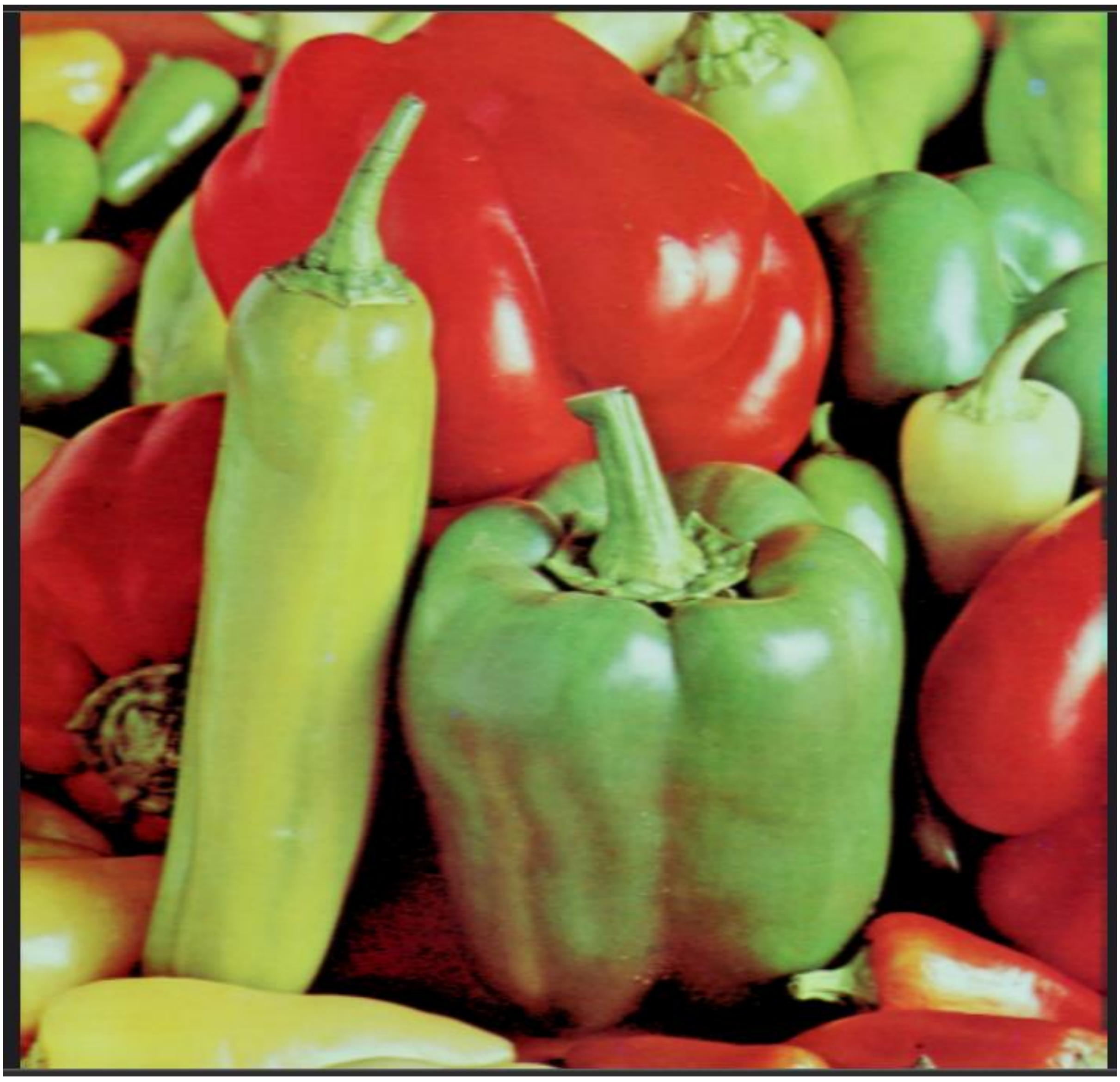
Original image



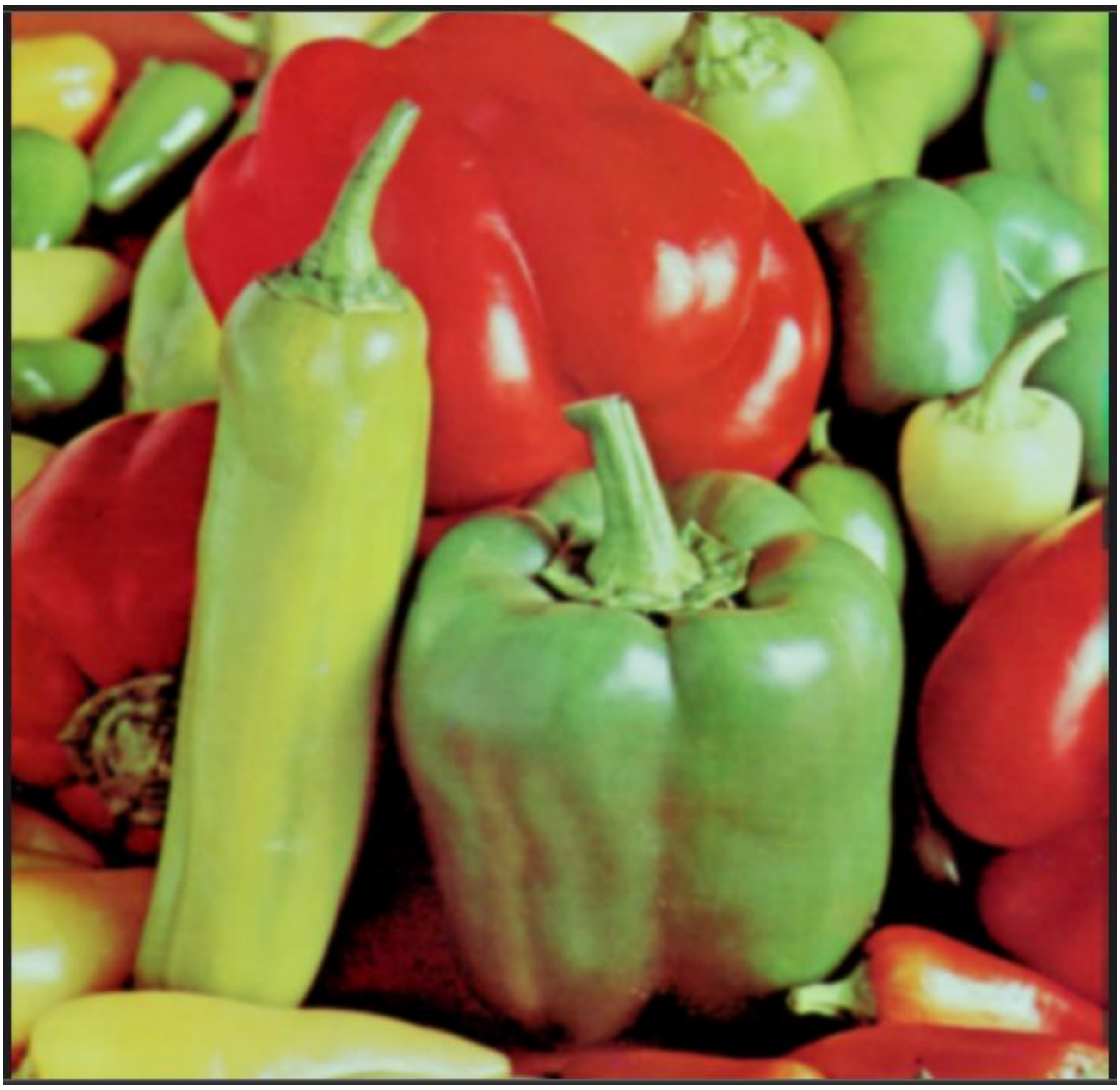
After changing color (brighter)



After horizontal blurring



After vertical blurring



Final image