# BBM 203 - PA 1 - Saving Dr Elara

*An adventure into the cosmos to uncover secrets.*

# BBM 203: Software Laboratory I - PROGRAMMING ASSIGNMENT 1 - FALL 2023

**Subject: Arrays and Matrices**
**Instructors:** Assoc. Prof. Dr. Adnan ÖZSOY, Asst. Prof. Dr. Engin DEMİR, Assoc. Prof. Dr. Hacer YALIM KELEŞ
**TAs:** Alperen ÇAKIN, **Ardan YILMAZ***, Dr. Selma DİLEK
**Programming Language:** C++11
**Due Date:** 03.11.2023 (23:59:59)
**Note:** TA marked with * is primarily responsible for this assignment.

# Contents

# 1 Introduction

## 1.1 Objective

Welcome to your first assignment! This assignment is to familiarize you with C++ and basic data structures - high-dimensional matrices. Here, you will also gain hands-on experience with classes, constructors, destructors, and various matrix operations. Additionally, you are asked to implement these for a basic image processing task, melding theoretical knowledge with real-world application.

## 1.2 Prerequisites

If you're unfamiliar with image processing, no worries! We will walk through the terminology and operations you are to implement step by step together. Let's touch on a few key concepts you'll be delving into:

### 1.2.1 Image Matrix Representation:

Grayscale images can be represented as two-dimensional (2D) matrices, where each element of the matrix corresponds to the intensity of a pixel, ranging from 0 (black) to 255 (white). For colored images, typically in RGB format, a three-dimensional (3D) matrix is used. Here, the third dimension is the channel, corresponding to the intensity values of the Red, Green, and Blue color channels respectively. Thus, an element $[i, j, k]$ gives the intensity of the $k$-th color channel at the pixel located at $(i, j)$ in the image.

### 1.2.2 Convolution:

Convolution is a mathematical operation that blends two functions (or signals). In image processing, convolution is employed to modify the appearance of an image by using a filter (or kernel), which is another matrix. The kernel is an operator used in image processing tasks, such as blurring, sharpening, edge detection, and more. The convolution operation involves placing the kernel at a pixel, multiplying the neighboring pixels by the corresponding values in the kernel, and then summing up those results to produce a new pixel in the output image.

The convolution operation $I * K$ involves sliding the kernel $K$ over the image $I$ and at each position, calculating the sum of the element-wise product of the overlapped numbers, assigning the result to the corresponding position in a new matrix. An example of

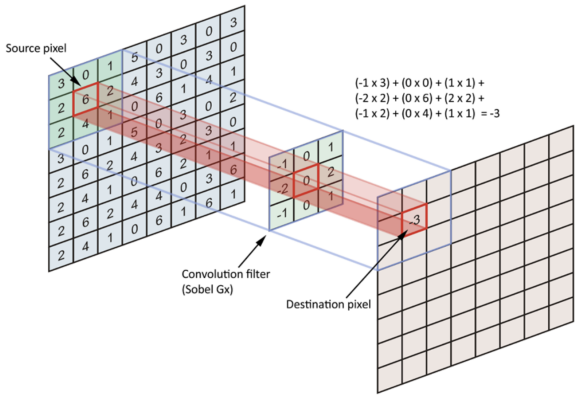



Figure 1: Convolution on Images.

convolution can be seen in Figure 1. Also, a basic convolution example is provided below for your convenience.

$$(I * K)_{i,j} = \sum_m \sum_n I_{i-m,j-n} \cdot K_{m,n}$$

Where:

- $I$ is the input image.
- $K$ is the kernel.
- $m$ and $n$ are the row and column indices to slide the kernel $K$ across the image $I$.
- The summation $\sum$ runs over all possible valid $m$ and $n$ values where the kernel and image overlap.

**The $(i, j)$th entry of the resulting image, $I * K$, is the sum of the element-wise product of the overlapped values in $I$ and $K$ at position $(i, j)$.**

- **Abuse of Terminology:** Technically, the operation described here is ***cross-correlation***, not convolution. True convolution involves rotating the kernel by 180 degrees before applying. However, they result in the same matrix given symmetric kernels. Throughout this assignment, the operation referred to as convolution is in fact cross-correlation. But be aware of the distinction in broader contexts.
- **Parameters of convolution**
  - **Kernel size:** The size of the kernel.
  - **Padding:** Adding extra pixels around the input image to deal with the edge cases. This is also quite of use to handle the shape of the output image. Padding can be "valid" (no padding) or "same" (pad to keep output size same as input), generally image is pad with 0's.

    **In the context of this homework, we will only deal with 0 padding of size 1 or no padding at all.**

Consider an image I without padding and I' with padding.

$$I = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{and} \quad I' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

  - **Stride:** The number of pixel shifts over the input matrix. Stride of 1 moves the kernel one pixel at a time, while stride of 2 moves it two pixels at a time.
  - **When striding the kernel, if a portion of it passes through the image boundaries, move on with the next row.**
  - **Source:** You can check the operation via various sources. Here's an example source.

- **Shape of the matrix after convolution:** Padding is either 0 or 1.

$$\textbf{Output height} = \left\lfloor \frac{\text{Input height} - \text{Kernel height} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$
$$\textbf{Output width} = \left\lfloor \frac{\text{Input width} - \text{Kernel width} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

- **Example convolution operation:** Here, the operation on the edges are avoided (no padding) and stride is 1, resulting in a lower-dimensional matrix. Given an image matrix $I$ and a kernel $K$ as below:

$$I = \begin{bmatrix} \mathbf{1} & \mathbf{2} & 3 \\ \mathbf{4} & \mathbf{5} & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad K = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{bmatrix}, \quad (I * K)_{0,0} = \mathbf{12}$$

$$I = \begin{bmatrix} 1 & \mathbf{2} & \mathbf{3} \\ 4 & \mathbf{5} & \mathbf{6} \\ 7 & 8 & 9 \end{bmatrix}, \quad K = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{bmatrix}, \quad (I * K)_{0,1} = \mathbf{16}$$

$$I = \begin{bmatrix} 1 & 2 & 3 \\ \mathbf{4} & \mathbf{5} & 6 \\ \mathbf{7} & \mathbf{8} & 9 \end{bmatrix}, \quad K = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{bmatrix}, \quad (I * K)_{1,0} = \mathbf{24}$$

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & \mathbf{5} & \mathbf{6} \\ 7 & \mathbf{8} & \mathbf{9} \end{bmatrix}, \quad K = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{bmatrix}, \quad (I * K)_{1,1} = \mathbf{28}$$

$$(I * K) = \begin{bmatrix} \mathbf{12} & \mathbf{16} \\ \mathbf{24} & \mathbf{28} \end{bmatrix}$$

### 1.2.3 Sobel Operator:

The Sobel Operator is a discrete differentiation operator, computing an approximate of the gradient of the image intensity function. It is utilized for edge detection and operates by convolving the image with a pair of $3 \times 3$ kernels which are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid. Below are the Sobel Operators:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Where:

- $G_x$ is used for detecting edges that are vertically oriented.
- $G_y$ is used for detecting edges that are horizontally oriented.

Given an image $I$, convolutions are performed for both $G_x$ and $G_y$:

$$I_x = I * G_x \quad \text{and} \quad I_y = I * G_y$$

Where:

- $I_x$ and $I_y$ are the images that result after convolution, representing the horizontal and vertical derivative approximations respectively.

Then, the gradient magnitude $G$ and direction $\Theta$ can be determined as follows:

$$G = \sqrt{I_x^2 + I_y^2}$$

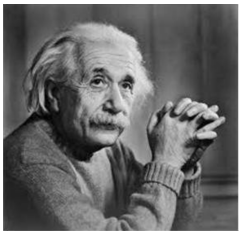
$$\Theta = \arctan\left(\frac{I_y}{I_x}\right)$$

Where:

- $G$ gives the magnitude or intensity of the gradient at each point in the image. It signifies the amount of variation or edge content in the local neighborhood of a pixel.
- $\Theta$ gives the direction of the gradient, pointing in the direction of the most rapid increase in intensity.

The approximate magnitude of the gradient is sometimes computed using:

$$G \approx |I_x| + |I_y|$$


(a) Original Image


(b) The horizontal response


(c) The vertical response

Figure 2: Original image vs horizontal and vertical responses after applying the corresponding sobel operators.

### 1.2.4 Image Sharpening:

Below is a basic image sharpening algorithm, where we initially arrive at an image with highlighted the edge information by subtracting the blurred image from the original one. Then, adding this highlighted edge information back to the original matrix results in the sharpened image. For this operation, steps are:

1. **Generate Noisy (Blurred) Image:** Given the original image, $I_{\text{orig}}$, a blurring operation is applied to generate a noisy image, $I_{\text{noisy}}$. Though there are other kernels that can be employed for this operation, **in the context of this assignment, we will opt for a simple 3x3 averaging kernel for blurring:**

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

   - **To arrive at the blurred image, convolve the original image with this averaging kernel.**
   - **Preserving the Image Dimensions:**
     Using the above given $3\times3$ kernel with padding and a stride of value 1 preserves the image dimensions as seen below. The given equation for height also applies to width.

$$\textbf{Output Height} = \left\lfloor \frac{\textbf{Input Height} - 3 + 2}{1} \right\rfloor + 1 = \textbf{Input Height}$$

2. **Sharp Image Calculation:** The sharpened image, $I_{\text{sharp}}$, is derived using the formula:

$$I_{\text{sharp}} = I_{\text{orig}} + k \cdot (I_{\text{orig}} - I_{\text{noisy}})$$

   where $k$ is a sharpening factor that scales the contribution of the high-frequency components extracted from the original image. Adjusting $k$ allows for control over the degree of sharpening applied.


Figure 3: Blurring operation

3. **Clipping:** Ensure that pixel intensity values in $I_{\mathbf{sharp}}$ lie within acceptable bounds - [0, 255]. The multiplication with the sharpening factor, k, may cause overflow. **Values that fall outside this range should be clipped** to ensure visual coherence, ie, **clip the values greater than 255 back to 255.**

#### 1.2.5 Edge Detection

- Define the Sobel Operators as above: $G_x$ and $G_y$
- Use zero padding to preserve the image shape.
- **Convolution**
  1. Convolve the original image matrix with $G_x$ to produce a new image matrix representing the change rates in the x-direction (horizontal changes). Let's call this matrix $I_x$.
  2. Convolve the original image matrix with $G_y$ to produce a new image matrix representing the change rates in the y-direction (vertical changes). Let's call this matrix $I_y$.
- **Gradient Magnitude Calculation:**
  - For each pixel position $(i, j)$, calculate the gradient magnitude:

$$G(i,j) = \sqrt{(I_x(i,j))^2 + (I_y(i,j))^2}$$

- **Gradient Direction Calculation:**
  - [**Omitted in this assignment**] Normally, the gradient direction is calculated to provide information about the edge orientation:

$$\Theta(i,j) = \arctan\left(\frac{I_y(i,j)}{I_x(i,j)}\right)$$

  - **We are omitting this step, as we will consider all pixels with a gradient magnitude greater than a certain threshold as edge pixels, regardless of their orientation for simplicity.**

- **Thresholding:**
  - Determine a threshold value. Any pixel $G(i, j)$ with a gradient magnitude larger than this threshold is considered an edge. Various strategies can be employed to determine this threshold.
  - **The thresholding policy:** Though there are more advanced methods to determine the threshold which yield better results, we will opt for a rather naive policy where **we will use the average of the gradient magnitudes across all pixels as the threshold** for the sake of simplicity and determinism. It's important to note that, in practice, this method is unlikely to yield optimal edge detection, especially for images with large uniform regions or varied intensity profiles.
  - Those pixels with a gradient greater than the threshold are on the edges.
- **Non-maximum Suppression:**
  - [**Omitted in this assignment**] Typically, to obtain thin and well-defined edges, non-maximum suppression is applied. However, for simplicity and to maintain a deterministic outcome, we will skip this step in the assignment.
- **Online resource:** You can check this algorithm via various sources. Here's an example source. **However, stick to the given Algorithm below for any details!**
- **Task:** Implement the Sobel edge detection based on the steps outlined (omitting the gradient direction and non-maximum suppression steps). Subsequently, you will use the pixels on these edges to decypher a hidden message. Details are below!

# 2 Your Quest

In an era where interstellar communication was a far-fetched dream, Dr. Elara, a pioneering astrophysicist, dedicated her existence to decoding the mysteries of the universe. Rooted in the belief that the cosmos concealed secret messages, she stood staunchly at the intersection of rigorous science and the mystical unknown.

Despite being recognized for her brilliance, her speculative methods and fearless explorations into undiscovered realms were often met with skepticism from her contemporaries. Undeterred, she assembled a covert group, a unique blend of daring scientists, astute codebreakers, and skilled astrophotographers, operating secretly and free from the doubting glances of the scientific community.

After perceiving a mysterious "ripple from the cosmos" she disappeared, leaving behind a perplexing celestial image — what she referred to as the "Celestial Tapestry" — and her team, now enveloped in a cloud of mystery and resolute determination.

The celestial tapestry was not merely a message but potentially a medium of communication, a conduit through which Elara might be reaching out, possibly navigating through dimensions unknown. She always believed that if humans managed to traverse through

time or alternate dimensions, they would likely leave messages hidden in plain sight, layered within the familiar, waiting to be uncovered by those keen enough to look deeper.

## 2.1 Task 1: Unlocking the Celestial Code

**The steps to solve the assignment are explained below in a structured way, presented as pseudo-codes. Please follow them thoroughly. For any details about specific operations in the provided pseudo-code, please refer to the Prerequisites Section.**

### 2.1.1 Restoring the Astral Clarity

Knowing Elara's penchant for hidden details and layered meanings, the first step is to reveal the details embedded within the cosmic tapestry. You believe a message is concealed in the edges of the image, as Elara often mused about the edges of the cosmos, suggesting that the path to other civilizations was woven into the edges of our perception. However, as edges are noisy, you first need to sharpen the image. **Notice that we will use zero-padding and a stride of value 1 for convolution for this algorithm.**

**Algorithm 1** Image Sharpening Algorithm

**Require:** Input Image: $InputImg$, Sharpening parameter: $k$
**Ensure:** Sharpened Image: $SharpImg$

1: $Kernel \leftarrow \frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ {Define the blurring kernel}
2: $BlurredImg \leftarrow CONVOLVE(InputImg, Kernel, stride = 1, padding = True)$ {Convolve with Kernel, applying zero padding with a stride of 1}
3: $SharpImg \leftarrow InputImg + k \times (InputImg - BlurredImg)$
4: **return** $SharpImg$

### 2.1.2 Tracing the Stellar Pathways

After sharpening the image, you need to detect the pixels on the edges. For this, you will be running an edge detection algorithm using Sobel Operators. The details are explained above and the pseudo-code is below. **Notice that the we will use fixed zero-padding and a stride of value 1 for convolution for this algorithm, which preserves the output dimensions as per the kernel size and other parameters (See Preserving the Image Dimensions bullet in 1.2.4 Image Sharpening).**

**Algorithm 2** Stellar Pathways: Edge Detection using Sobel Operators

**Require:** Image: $Img$
**Ensure:** List of edge pixels: $EdgePixels$
$G_x, G_y \leftarrow$ initialize the Sobel operators
$I_x \leftarrow CONVOLVE(Img, G_x, stride = 1, padding = True)$ {Horizontal edge response}
$I_y \leftarrow CONVOLVE(Img, G_y, stride = 1, padding = True)$ {Vertical edge response}

**for all** pixel (i,j) in $Img$ **do**
    $magnitude[i, j] \leftarrow \sqrt{(I_x[i, j])^2 + (I_y[i, j])^2}$
**end for**
$threshold \leftarrow$ average of all values in $magnitude$
Initialize $EdgePixels \leftarrow []$
Append to $EdgePixels$ all pixels with gradient above the $threshold$
**return** $EdgePixels$

**Note:** Processing each pixel sequentially (starting from the top left corner to bottom right). The order of pixels to be decoded matters!

### 2.1.3 Decoding the Alien Message

Elara, while a visionary, was constrained by the technology of her time and thus, resorted to a simple yet ingenious method of encoding: LSB steganography. In this method, the message is hidden in the least significant bits (lsb) of the pixels on the image's edges. These, when concatenated, form a binary string. Each segment of 7 bits in this binary string corresponds to a character in the ASCII table, which can then be converted into a string to reveal the hidden message

**Below are the steps of how the process works in detail:**

1. **Extracting LSBs from Edge Pixels:** First, we gather a list of edge pixels from the image. For each pixel, we extract its least significant bit (lsb). These lsb values are collected and concatenated to form a binary string.

2. **Converting Binary to ASCII:** The binary string obtained in the previous step might not have segments of exactly 7 bits. To ensure proper conversion, we **pad this binary string with leading zeros to form complete 7-bit bytes.** We then proceed to convert each 7-bit segment into its corresponding decimal value. Note that not all elements in the ASCII table are printable, namely, those with indices $\leq 32$ and $\geq 127$. To ensure all characters are printable, if the resulting ASCII value is **less than or equal to 32, we adjust it by adding 33** and if the ASCII value **is 127, clip it back to 126**.

3. **Revealing the Message:** The final step is to convert the decimal ASCII values into their corresponding characters. These characters are concatenated to form the decoded message, revealing the hidden information.

Elara's legacy lies within these celestial codes, hidden in plain sight. Your task is to decrypt these codes, navigate through the starlit pathways, and carry forth her wisdom

into the great cosmic beyond.

**Note 1:** The order in which you decode the pixels matters. Start from the top-left corner of the image and proceed to the right and then down.

**Note 2:** Padding the binary string with leading zeros ensures consistent 7-bit segments for accurate decoding. Additionally, adjustments are made to ASCII values to ensure printable characters are obtained during the conversion process.

**In short, you'll extract the least significant bit from each pixel along the detected edges, decode the binary to ASCII, and reveal the messages concealed within.**

## 2.2 Task 2: Re-Formulating and Embedding the Response

In decoding Elara's message, the team becomes the stewards of a potent secret, a conduit through space and possibly time. To forge a connection, they must navigate through the metaphysical and scientific, encoding their message back into the celestial image without diminishing its original integrity. Having decoded Elara's message, your duty now evolves from mere interpretation to active participation in this celestial dialogue, crafting a message embedded seamlessly back into the original image.

### 2.2.1 Character Transformation

For each character of the extracted message in a position indexed with a prime number, adjust its ASCII value by adding the Fibonacci number corresponding to that index, i.e., the ith position means adding the ith Fibonacci number. Ensure that the transformed character is printable within the ASCII range of 33 to 126. **Apply the same adjustment in the Decoding step, ie, add 33 to those** $\leq 32$ **and clip those** $\geq 127$ **back to** 126**.**

### 2.2.2 Right Circular Shifting

Apply right circular shifting to the manipulated message by $\lfloor len(message)/2 \rfloor$ characters. For example, shifting "HELLO" by 2 results in "LOHEL".

### 2.2.3 Message Embedding

The restructured binary message must be delicately embedded back into the celestial tapestry. To maintain the visual salience of the image **while encoding the modified message, begin the encoding process at the top-left edge pixel and continue in a sequential order down to the bottom-right edge pixel. (Use only the pixels on the edge).** Each bit should be placed into the least significant bit (LSB) of the edge pixel, preserving the original sequence while ensuring the seamless integration of the modified message.

# 3 Problem Statements and Classes

To tackle with the above problem, you are developing a software with the classes introduced in the following section. However, as responsible scientists, you opt for a structured code with classes, which are to be unit tested, ie, each method in each class will be tested and evaluated.

In this assignment, you will create several classes to handle different aspects of the task. For each class, you will provide both the declarations (in a '.h' header file) and implementations (in a '.cpp' file).

**You are required to implement the given functions in the template files, as they will be graded. DO NOT CHANGE THEIR NAMES! Other than these, you can define as many functions, variables as you wish.**

**For the ImageMatrix and Convolution constructors, you have to dynamically allocate memory using the given matrix, ie, you cannot use STL library - vectors, in particular.** However, you are allowed to use them in the member functions.

## 3.1 ImageLoader Class

An `ImageLoader class` is available to simplify image loading from a specified path. You can utilize the `ImageMatrix(const std::string &filepath)` constructor of the ImageMatrix class to create an ImageMatrix directly from a file. Please be aware that this template employs PNG images for loading, which involves non-standard library dependencies. To accommodate, a Python script (`convert_pil_to_txt.py` under the data directory) is also shared that converts PNG images into text files. You can use these text files as input for the constructors of both the ImageLoader and ImageMatrix classes. Additionally, note that the Python script relies on the PIL library, which must also be installed. For your convenience, each PNG image is accompanied by a corresponding text file, eliminating the need for manual conversion. However, you are free to use the provided Python script if preferred.

## 3.2 ImageMatrix Class

The `ImageMatrix` class encapsulates the pixel data of an image and provides functionalities for basic matrix operations.

- **Constructors and Destructor**:
  - `ImageMatrix()`: Default constructor
  - `ImageMatrix(const std::string &filepath)`: Parametrized constructor that accepts a file path as an argument, loads the image from the specified path, and initializes the matrix with the image data. **The implementation is provided.**
  - `ImageMatrix(int imgHeight, int imgWidth)`: Parametrized constructor for **creating a blank image** of the given size.

- `ImageMatrix(const double** inputMatrix, int imgHeight, int imgWidth)` Parameterized constructor that accepts a 2D matrix and directly initializes the matrix with the provided data.
- `ImageMatrix(const ImageMatrix &other)`: Copy constructor
- `ImageMatrix & operator=(const ImageMatrix &other)`: Copy Assignment Operator
- `ImageMatrix()`: Destructor

- **Overloaded Operators**:
  - `ImageMatrix operator+(const ImageMatrix &other) const` : Overloaded addition operartor that performs element-wise addition of matrices.
  - `ImageMatrix operator-(const ImageMatrix &other) const` : Overloaded subtraction operartor that performs element-wise subtraction of matrices.
  - `ImageMatrix operator*(const double &scalar) const` : Overloaded scalar multiplication operator.

- **Getter Functions**:
  - `double **get_data() const`: Retrieve the private data
  - `double get_data(int i, int j) const`: Retrieve the specific element at the given index.

- **Private Data Members**:
  - `double **data`: 2D array to store image data.

## 3.3 Class Implementation: Convolution

The `Convolution` class provides functionalities related to the convolution operation on an image with the parameters: kernel, stride, and padding.

- **Constructors and Destructors**
  - `Convolution()`: Default constructor
  - `Convolution(double** customKernel, int kernelHeight, int kernelWidth, int stride, bool padding)`: Parametrized constructor for a custom kernel and other parameters.
  - `Convolution(const Convolution &other)`: Copy constructor
  - `Convolution &operator=(const Convolution &other)`: Copy assignment operator
  - ∼`Convolution()`: Destructor

- **Public Member Functions**
  - `ImageMatrix convolve(const ImageMatrix &input_image) const`: Responsible for convolving the input image with the kernel and returning the convolved image.

**Note: Padding is a boolean value, where 'true' indicates that zero padding should be applied and 'false' indicates no padding.**

**Note: For the sake of simplicity, in the context of this PA, you can assume all kernels are square matrices.**

## 3.4 Class Implementation: ImageSharpening

The `ImageSharpening` class is designed to apply a basic image sharpening algorithm to an input image using a predefined kernel.

- **Constructors and Destructors**
  - `ImageSharpening()`: Default constructor
  - ∼`ImageSharpening()`: Destructor
- **Public Member Functions**
  - `ImageMatrix sharpen(const ImageMatrix &input_image, double k)`: Performs image sharpening on the input image using the specified sharpening factor (`k`) and returns the sharpened image.
- **Private Data Members**
  - `double** blurring_kernel`: Kernel for blurring
  - `int kernel_height`: Height of the kernel
  - `int kernel_width`: Width of the kernel

**Use the above given Algorithm 1, strictly abiding by the specs!**

## 3.5 Class Implementation: EdgeDetector

The `EdgeDetector` class is designed to detect edges in an image using Sobel operators.

- **Constructors and Destructors**
  - `EdgeDetector()`: Default constructor
  - ∼`EdgeDetector()`: Destructor
- **Public Member Functions**
  - `std::vector<std::pair<int, int>> detectEdges(const ImageMatrix &input_image)`: Detects edges in the input image using the Sobel operators and returns a vector of edge pixel positions as pairs of (row, column) indices.

**Use the above given Algorithm 2, strictly abiding by the specs!**

## 3.6 Class Implementation: DecodeMessage

The `DecodeMessage` class is designed to decode a binary string into its ASCII representation using the above algorithm.

- **Constructors and Destructors**
  - `DecodeMessage()`: The default constructor.
  - $\sim$ `DecodeMessage()`: Destructor to manage any resources or memory the class might allocate.
- **Public Member Functions**
  - `std::string binaryToASCII(const std::string& binaryString)`: This function takes a binary string and decodes it into its ASCII representation and returns the final string.

## 3.7 Class Implementation: EncodeMessage

The `EncodeMessage` class is devised to embed a binary message back into an image matrix, making sure that the bits of the message are stored in the least significant bit (LSB) of the pixel values from which the original message was extracted. **The encoded bits must correspond to the pixel locations they are originally extracted from!**

## 3.8 Class Implementation: EncodeMessage

The `EncodeMessage` is designed to embed a binary message after applying the above operations on the extracted one back to the image. **The updated message needs to be embedded in the image starting from the top left edge pixel to the bottom right.**

- **Constructors and Destructors**
  - `EncodeMessage()`: The default constructor.
  - $\sim$`EncodeMessage()`: Destructor
- **Public Member Functions**
  - `ImageMatrix encodeMessageToImage(const ImageMatrix &img, const std::string &message, const std::vector<std::pair<int, int>>& positions)`: This function accepts an image matrix, a message string, and a vector of pixel positions as inputs. It returns a new image matrix wherein the input message is embedded into the positions specified.

## 3.9 ImageProcessor Class

The `ImageProcessor Class` is to be used as the final step, where a message decoded and the updated message is encoded.

- **Constructors and Destructors**

- `ImageProcessor()`: The default constructor.
- `~ImageProcessor()`: Destructor

- **Public Member Functions**
  - `std::string decodeHiddenMessage(const ImageMatrix &img)`: Given an image of type ImageMatrix, it decodes and returns the message embedded within.
  - `ImageMatrix encodeHiddenMessage(const ImageMatrix &img, const std::string &message)`: Encodes the given message back to the image and returns the image.

# 4 Must-Use Starter Codes and Sample Input/Output

You MUST use **this starter code** for this assignment. Under src folder, you will find the template .h and .cpp files. You need to implement the functions in the .cpp files. And, under the data directory, you will find a .png image, its corresponding .txt conversion, and the python script to convert a png image to txt.

Your implementations need to work both with images loaded from a path and direct matrix initialization.

- **Loading from a file:** An **ImageLoader class** to load the image in the given path is provided for your convenience. You can use the constructor that takes file path argument `ImageMatrix(const std::string &filepath)` of the ImageMatrix class to create an ImageMatrix from the file directly as is. Please note that this template uses a PNG image for loading, which is not a trivial task as it requires libraries that are not part of the standard library. To simplify the process, a Python script (`convert_pil_to_txt.py`) that loads a PNG image and converts it into a text file is provided. You can then use this text file as an argument for the constructors of the ImageLoader and ImageMatrix classes. Additionally, please be aware that the Python script relies on the PIL library, which also needs to be installed. For each PNG image provided, you will also find the corresponding text file. That is, you do not need to perform the conversion manually, although you are welcome to use the provided Python script if you prefer. All files regarding images are provided in a separate folder: data, including the .PNG, .txt files and the python script for the conversion.
- **Loading image data from the main function:** You will find a main function provided with several ImageMatrix data examples for testing your classes. However, a comprehensive testing platform, **Tur$^3$Bo Grader**, is available at `https://test-grader.cs.hacettepe.edu.tr/`. Please use it, as it will also be used for grading.

# 5 How to Run Your Code

All classes will be unit tested. An example compile and run is as follows (those for the other classes are of the same structure):

```
g++ -std=c++11 -o test TestImageMatrixConstructor.cpp ImageMatrix
    .cpp ImageLoader.cpp

./test
```

# 6 Notes:

- **Note on Constructors and Destructor:** In C++, constructors and destructor play a crucial role in managing resources and ensuring that objects are correctly initialized and cleaned up. Here's a general guideline on when to implement your own:
  - **Default Constructor**: If your class has members that need to be initialized to specific values (other than zero or default-constructed), or if you need to perform some setup (like allocating memory or initializing resources), provide a *default constructor*.
  - **Copy Constructor**: If your class manages resources, a shallow copy of pointers does not suffice - provide a *copy constructor* that performs a deep copy. Please further check deep vs shallow copy: one example resource.
  - **Destructor**: If your class acquires resources (like memory, file handles, or network connections) or holds ownership to resources, provide a destructor to release those resources and prevent resource leaks. If your class does not manage resources or if all member variables automatically manage their own resources (like 'std::vector' or 'std::string'), the compiler-generated destructor is sufficient.
- For the sake of simplicity, in the context of this PA, we will stick to the **grayscale images only**. Also, you can assume **all kernels are square matrices**.
- For edge detection, we will **omit** the **gradient direction calculation and non-maximum suppression steps.**
- **Abuse of Terminology:** Technically, the operation described here is ***cross-correlation***, not convolution. True convolution involves rotating the kernel by 180 degrees before applying. Yet, their results are the same if the kernels are symmetrical. Throughout this assignment, the operation referred to as convolution is in fact cross-correlation. But be aware of the distinction in broader contexts.
- You must test your code via **Tur$^3$Bo Grader** `https://test-grader.cs.hacettepe.edu.tr/` (**does not count as submission!**).
- You must submit your work via `https://submit.cs.hacettepe.edu.tr/`

# 7 Submission Guidelines

Create a .zip file including all the implementation files. The given classes and the function prints are to be tested. **Do not change the given class names and function names to be tested!** You must submit your work via `https://submit.cs.hacettepe.edu.tr/`. Do not upload your main functions or other any other file. The submissions should be in the follow the hiearchy below:

- **b<studentID>.zip**
  - ImageMatrix.h
  - ImageMatrix.cpp
  - Convolution.h
  - Convolution.cpp
  - ImageSharpening.h
  - ImageSharpening.cpp
  - EdgeDetector.h
  - EdgeDetector.cpp
  - DecodeMessage.h
  - DecodeMessage.cpp
  - EncodeMessage.h
  - EncodeMessage.cpp
  - ImageProcessor.h
  - ImageProcessor.cpp

# 8 Grading Criteria

Each class will be evaluated using unit testing, so each method you implement, you gain points. Here's a breakdown of the grading criteria:

| Task / Requirement | Points |
|---|---|
| Implementing `ImageMatrix` Class | 10 |
| Implementing `Convolution` Class | 10 |
| Implementing `ImageSharpening` Class | 10 |
| Implementing `EdgeDetector` Class | 10 |
| Implementing `DecodeMessage` Class | 15 |
| Implementing `EncodeMessage` Class | 15 |
| Implementing `ImageProcessor` Class | 10 |
| No STL Vector Usage in `ImageMatrix` Class | 10 |
| No Memory Leaks/Errors | 10 |

# 9 Plagiarism Policy

All work on assignments **must be done individually**. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning

in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former conditions also hold for the material attained using AI tools, including ChatGPT, GitHub Copilot, etc.

**The submissions will be subjected to a similarity check. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in the suspension of the involved students.**

**Instead of resorting to such practices, remember we are here to help!**

# 10 ACKNOWLEDGEMENTS


- The overall story of Dr Elera in this homework is AI-Generated and the overall writing is AI-assisted. In particular, **ChatGPT** requires a special mention both in creating the story and polishing the language use. Also, the image in the cover is generated by **DALL·E 3**.
- Gratitude is also extended to the course staff, especially TAs, who contributed their insights and expertise to this assignment.