**You can use the same software for Project 1 or 2.**

**Question: Implement the Sobel and Canny Edge Detectors**

In a first step, write a function that performs a convolution on a given image and filter. Both inputs and the output are of the Matrix type, which means that you first have to convert the input image into a matrix. This seems cumbersome but is more efficient in the long run, because sometimes we want to perform multiple successive convolutions on an image, and we would also like to be able to use real-valued convolution inputs. The function should thus have the following signature:

```
Matrix convolve(Matrix m1, Matrix m2)
```

As you know, `m1` and `m2` are in principle exchangeable, but only if they are padded with zeroes outside of their borders so that all overlaps of elements from `m1` and `m2` are considered. For our computer vision applications, however, we expect the filter to be smaller than the input image, and during convolution we never want any elements of the filter to be outside of the image. Therefore, we require that `m1` contains the values of the input image, and `m2` those of the filter. The resulting matrix is always of the same size as the input image, with zeroes in those positions that could not be reached by the center element of the filter. For example, if the input image has 7 by 7 entries and the filter has 3 by 3 entries, then the output is a 7 by 7 matrix with zeroes in its first and last rows and its first and last columns. The function assumes that we use the center of the filter as the indicator of where to store the output value. If the height or width of the filter is an even number of elements, the center value is rounded down. For example, if `m2` is a 2 by 2 filter, we use its upper-left position as its "anchor" for storing output values.

Having implemented the `convolve` function, it is now easy to write a `sobel` function that takes an input image and outputs a grayscale image of the same size, indicating the edge strength at each position in the input image. The weakest edge pixel in the image has intensity 0 (black), and the strongest one intensity 255 (white), which is accomplished by linearly scaling the Sobel filter edge strength output. Use the following signature:

```
Image sobel(Image img)
```

You are now in a good position to implement the canny function, using the same signature:

```
Image canny(Image img)
```

Acknowledgement: The homework was designed and written by Marc Pomplun. Edited by Nada Attar

Include all operations as discussed in class, including initial smoothing and final thresholding by determining a threshold that works for most real-world images. Again, the output should be scaled to use of the full range of intensity values from 0 to 255, or only the values 0 and 255 if you choose to binarized the image when thresholding it. Finally, write a function `edgeDetection` that reads an image and stores its Sobel and Canny output images onto the hard drive:

```
void edgeDetection(char *inputFilename, char *sobelFilename, char *cannyFilename)
```

You do not have to worry about catching exceptions or similar things. This is not a software engineering class.

Find an image online or from your computer that clearly demonstrates the difference in results between the Sobel and Canny detectors. Include the image that you chose in your submission.

**Submitting Information:**

- Use the code I provided
- You should have all code in one file
- Submit your work on Canvas.
- Deadline Oct 10th at 6:00PM

Acknowledgement: The homework was designed and written by Marc Pomplun. Edited by Nada Attar