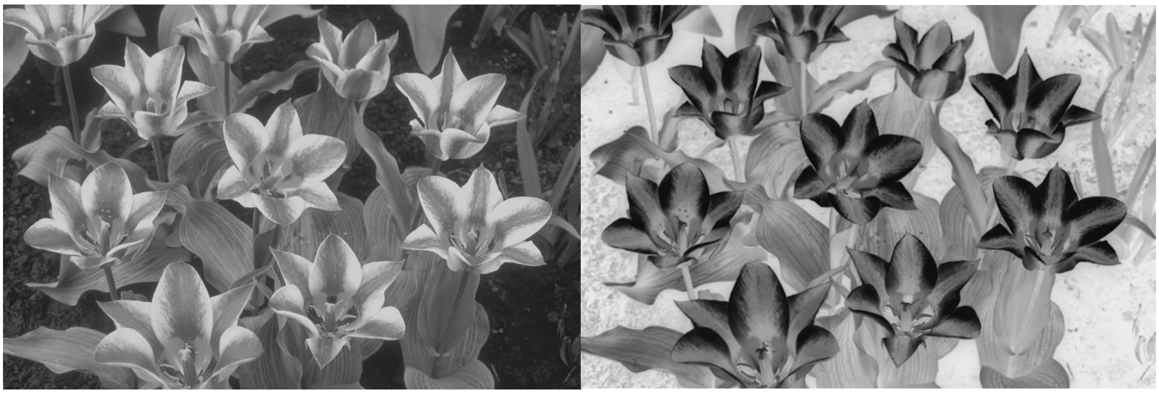
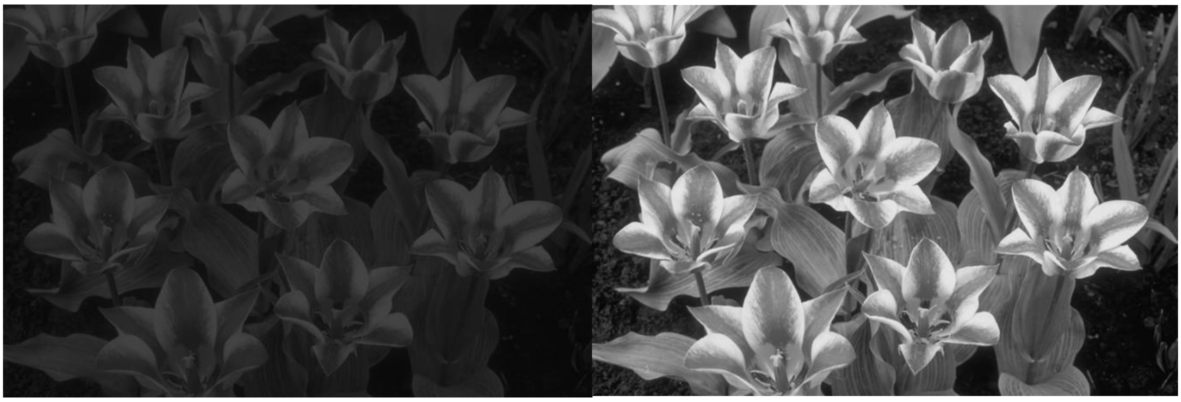
Blending

The blending function combines two images into a gif, where the first image slowly transitions to the second over 5 seconds. This works by creating 50 blended frames (except for the first and last, which are unaltered from the original images) where, in this instance, each frame is increasingly shifted from the positively coloured image, to the negative.



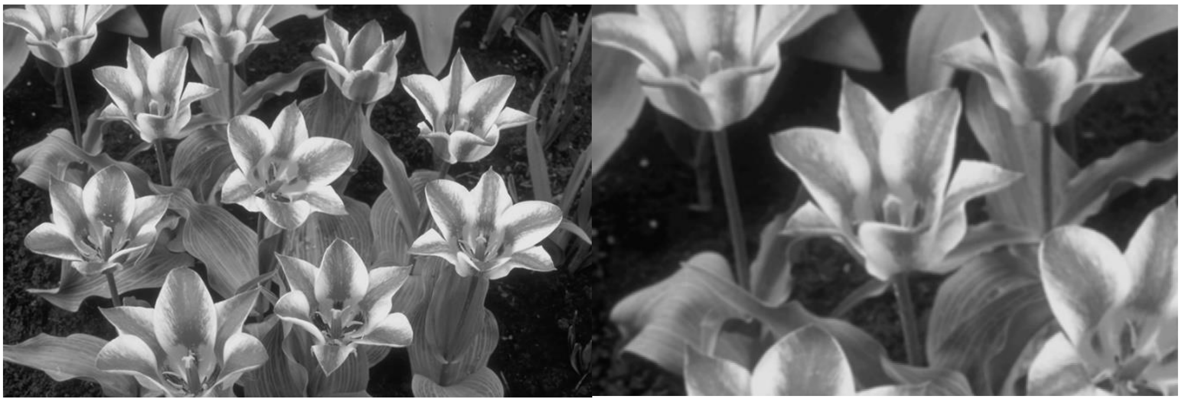
Contrast enhancement

Contrast enhancement works by taking away the minimum values of the poor contrast image, dividing by the maximum values of the image, and then multiplying all by 255. The resulting image has improved contrast over the original.



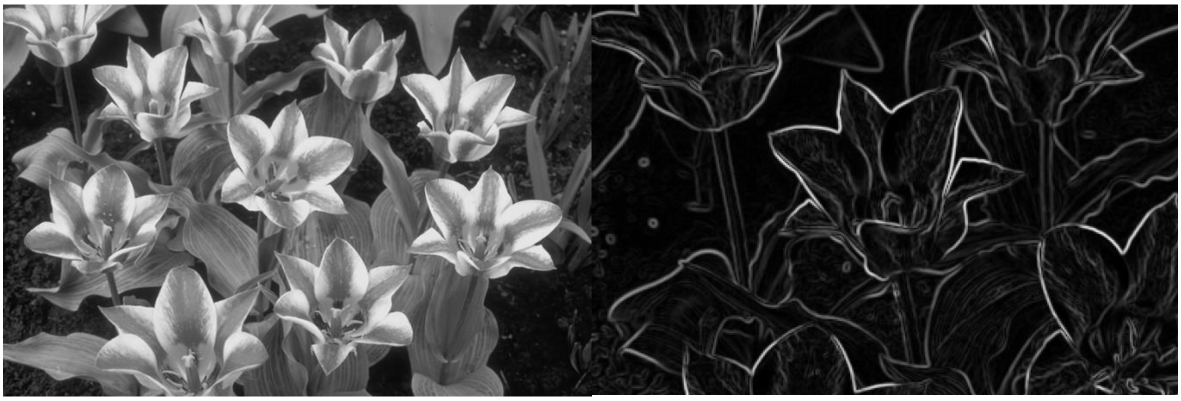
Gaussian filter

A gaussian filter applies a filter to an image that makes the image blurred. It uses a normalised image kernel applied onto the image to create the blurred effect. It also uses the cropping function to produce the output image.



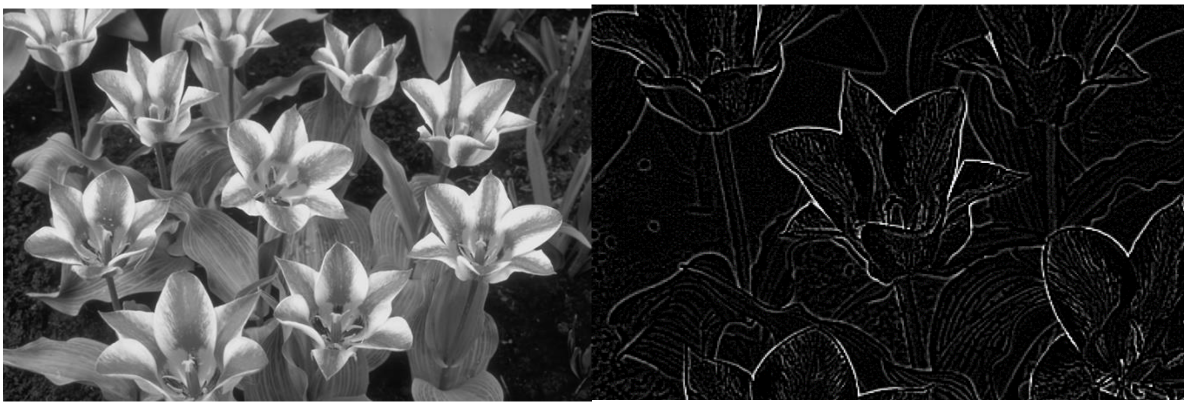
Gradient magnitude

The method I chose for producing the gradient magnitude involved creating two separate copies of the starting image and running them through a unique kernel, one vertical kernel and one horizontal. Then I squared each of the resulting images, added them together and found the square root of the sum to produce the final output image.



Laplacian filter

The Laplacian filter runs the image through a specific kernel before cropping and normalising it.

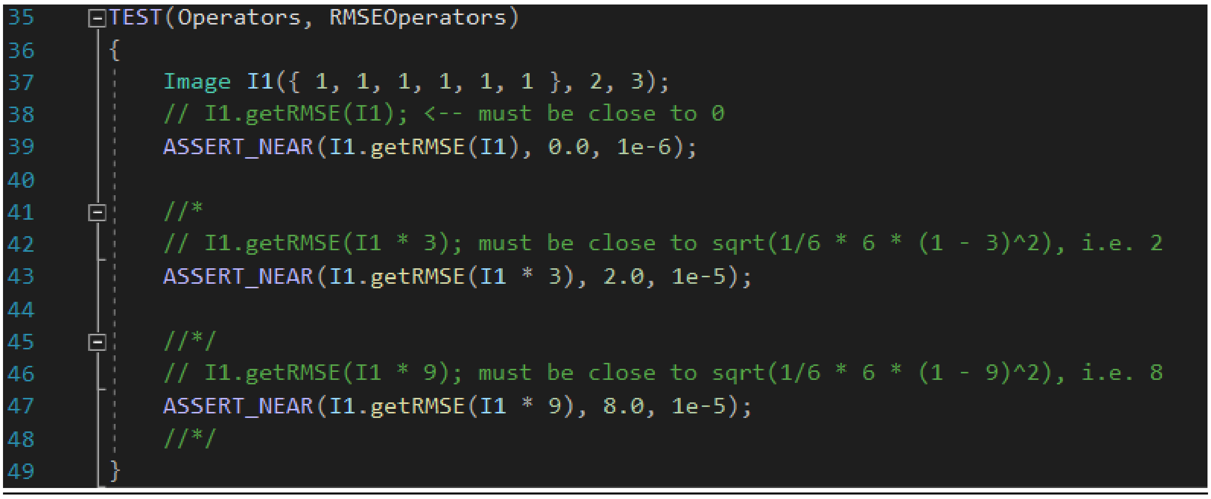


Negative

The negative filter follows a similar process to contrast enhancement, however in the final stage before saving, all values of the image are reversed (0 -> 255, 1 -> 254, etc.).

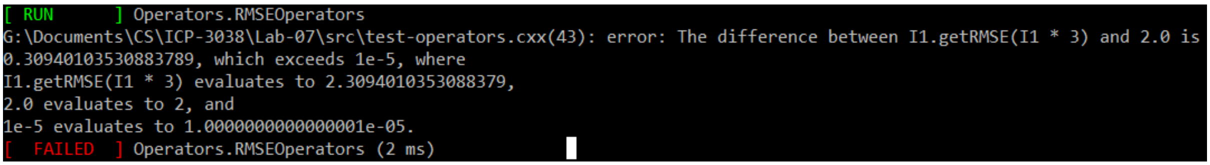
RMSE

The following tests were run:



```
35  TEST(Operators, RMSEOperators)
36  {
37      Image I1({ 1, 1, 1, 1, 1, 1 }, 2, 3);
38      // I1.getRMSE(I1); <-- must be close to 0
39      ASSERT_NEAR(I1.getRMSE(I1), 0.0, 1e-6);
40
41      //*
42      // I1.getRMSE(I1 * 3); must be close to sqrt(1/6 * 6 * (1 - 3)^2), i.e. 2
43      ASSERT_NEAR(I1.getRMSE(I1 * 3), 2.0, 1e-5);
44
45      //*/
46      // I1.getRMSE(I1 * 9); must be close to sqrt(1/6 * 6 * (1 - 9)^2), i.e. 8
47      ASSERT_NEAR(I1.getRMSE(I1 * 9), 8.0, 1e-5);
48      //*/
49  }
```

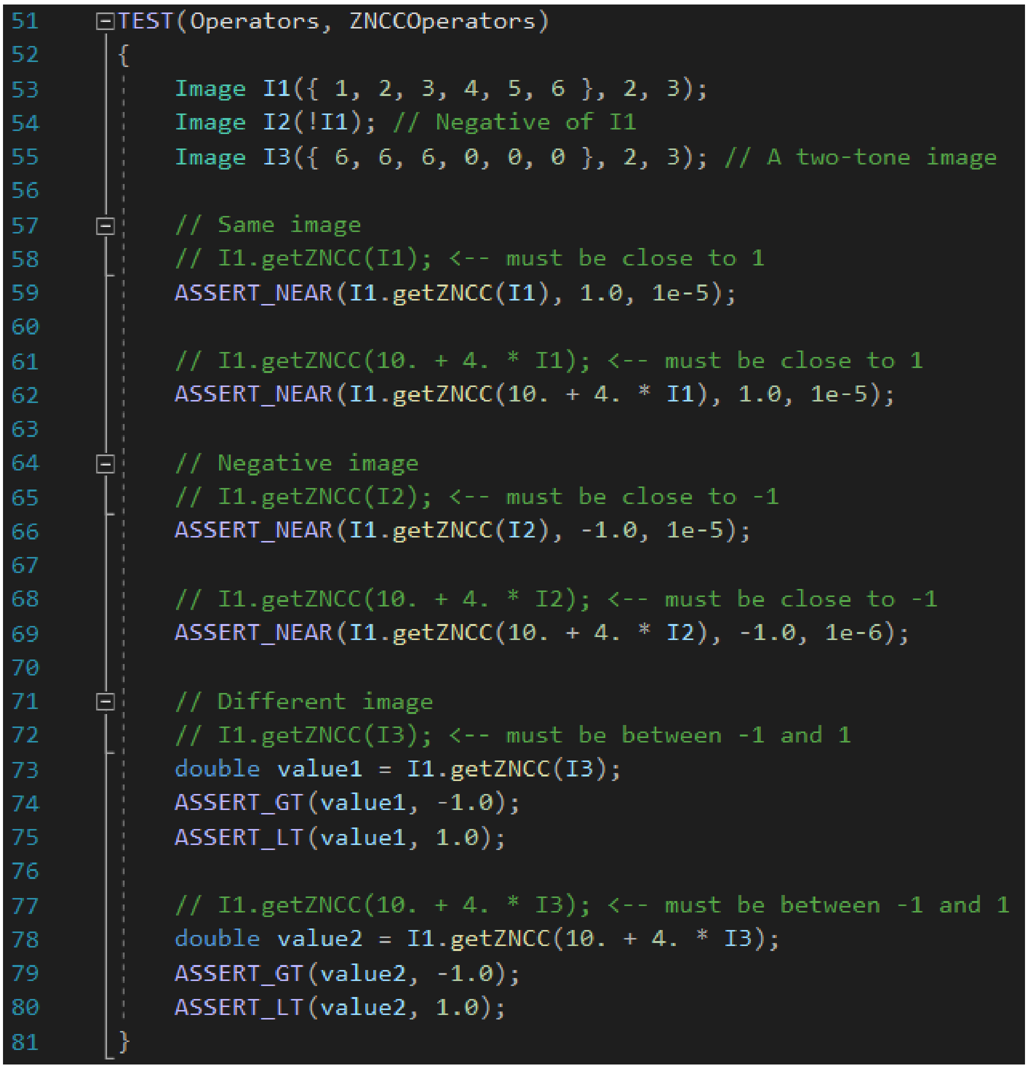
The first test to hit an error gave this output:



```
[ RUN      ] Operators.RMSEOperators
G:\Documents\CS\ICP-3038\Lab-07\src\test-operators.cxx(43): error: The difference between I1.getRMSE(I1 * 3) and 2.0 is
0.30940103530883789, which exceeds 1e-5, where
I1.getRMSE(I1 * 3) evaluates to 2.30940103530883879,
2.0 evaluates to 2, and
1e-5 evaluates to 1.0000000000000001e-05.
[  FAILED  ] Operators.RMSEOperators (2 ms)
```

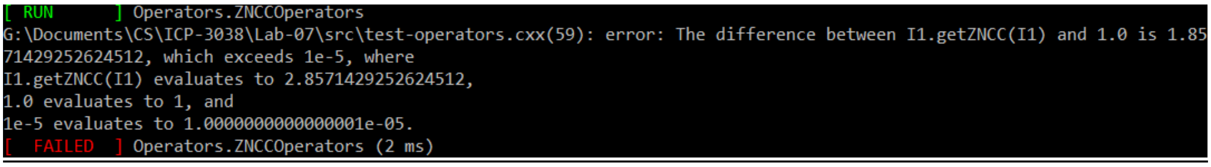
ZNCC

The following tests were run:



```
51  TEST(Operators, ZNCCOperators)
52  {
53      Image I1({ 1, 2, 3, 4, 5, 6 }, 2, 3);
54      Image I2(!I1); // Negative of I1
55      Image I3({ 6, 6, 6, 0, 0, 0 }, 2, 3); // A two-tone image
56  
57      // Same image
58      // I1.getZNCC(I1); <-- must be close to 1
59      ASSERT_NEAR(I1.getZNCC(I1), 1.0, 1e-5);
60  
61      // I1.getZNCC(10. + 4. * I1); <-- must be close to 1
62      ASSERT_NEAR(I1.getZNCC(10. + 4. * I1), 1.0, 1e-5);
63  
64      // Negative image
65      // I1.getZNCC(I2); <-- must be close to -1
66      ASSERT_NEAR(I1.getZNCC(I2), -1.0, 1e-5);
67  
68      // I1.getZNCC(10. + 4. * I2); <-- must be close to -1
69      ASSERT_NEAR(I1.getZNCC(10. + 4. * I2), -1.0, 1e-6);
70  
71      // Different image
72      // I1.getZNCC(I3); <-- must be between -1 and 1
73      double value1 = I1.getZNCC(I3);
74      ASSERT_GT(value1, -1.0);
75      ASSERT_LT(value1, 1.0);
76  
77      // I1.getZNCC(10. + 4. * I3); <-- must be between -1 and 1
78      double value2 = I1.getZNCC(10. + 4. * I3);
79      ASSERT_GT(value2, -1.0);
80      ASSERT_LT(value2, 1.0);
81  }
```

The first test to hit an error produced this output:

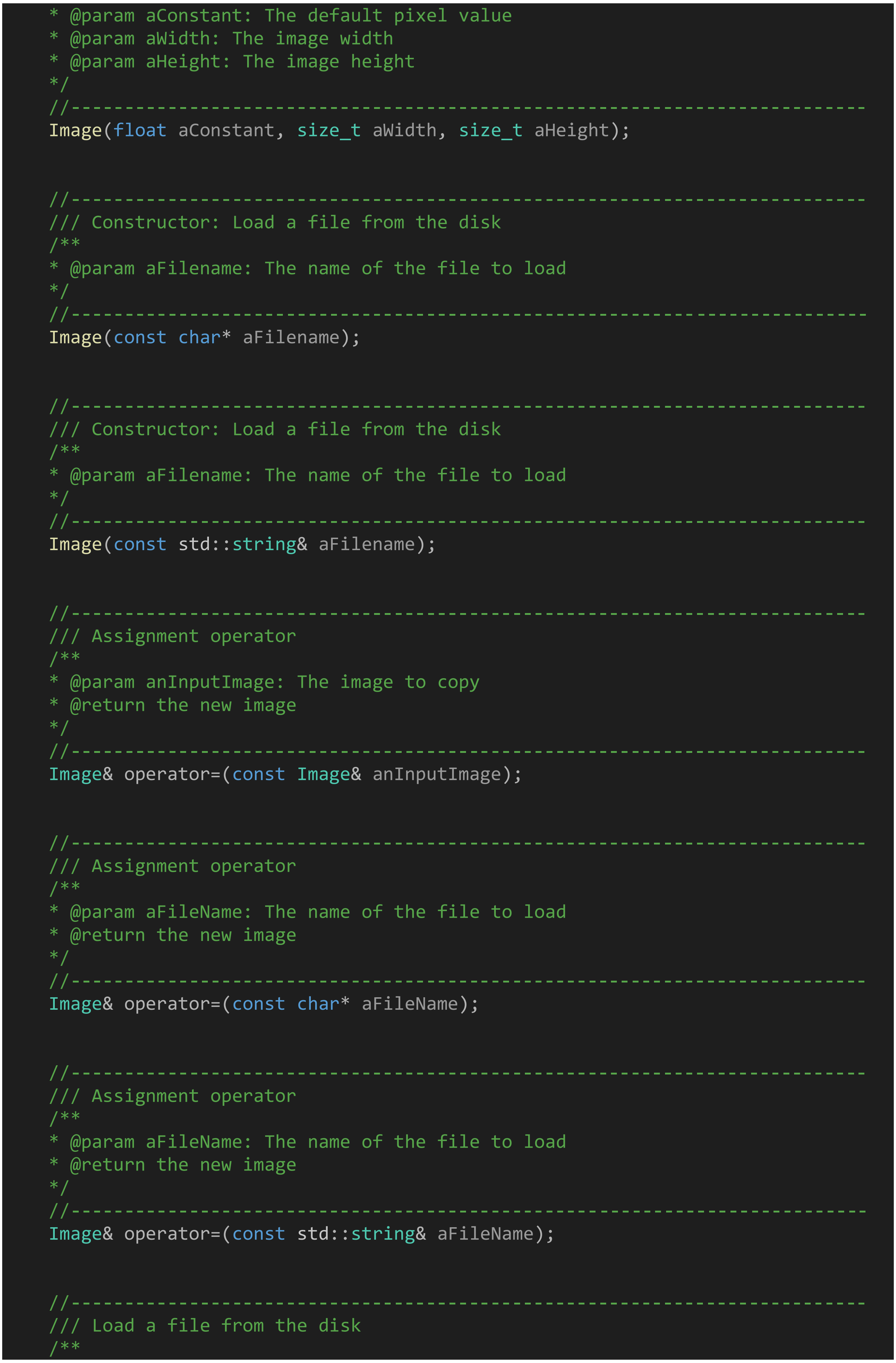


```
[ RUN      ] Operators.ZNCCOperators
G:\Documents\CS\ICP-3038\Lab-07\src\test-operators.cxx(59): error: The difference between I1.getZNCC(I1) and 1.0 is 1.85
71429252624512, which exceeds 1e-5, where
I1.getZNCC(I1) evaluates to 2.8571429252624512,
1.0 evaluates to 1, and
1e-5 evaluates to 1.0000000000000001e-05.
[  FAILED  ] Operators.ZNCCOperators (2 ms)
```

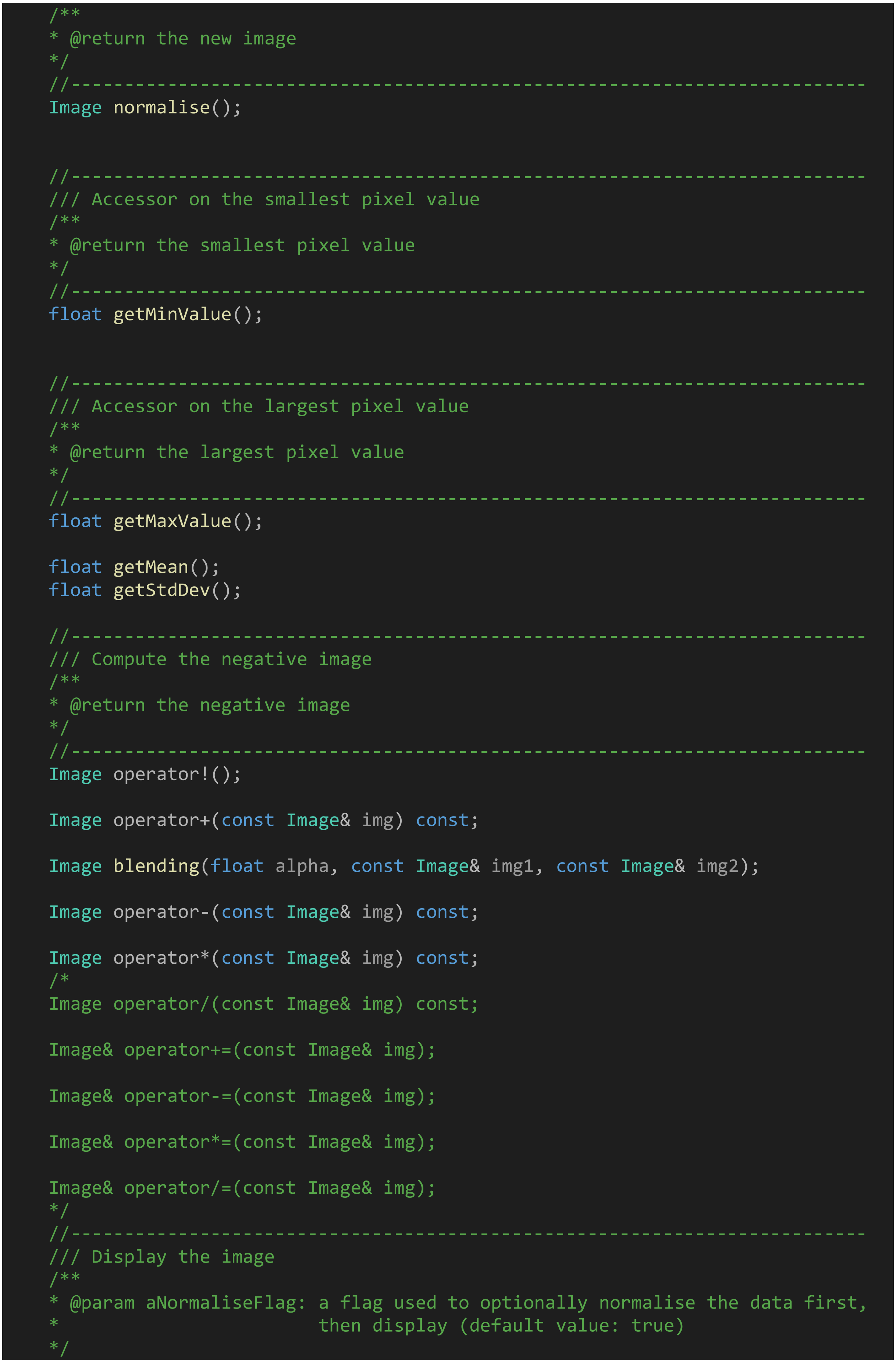
Appendix

Image.h

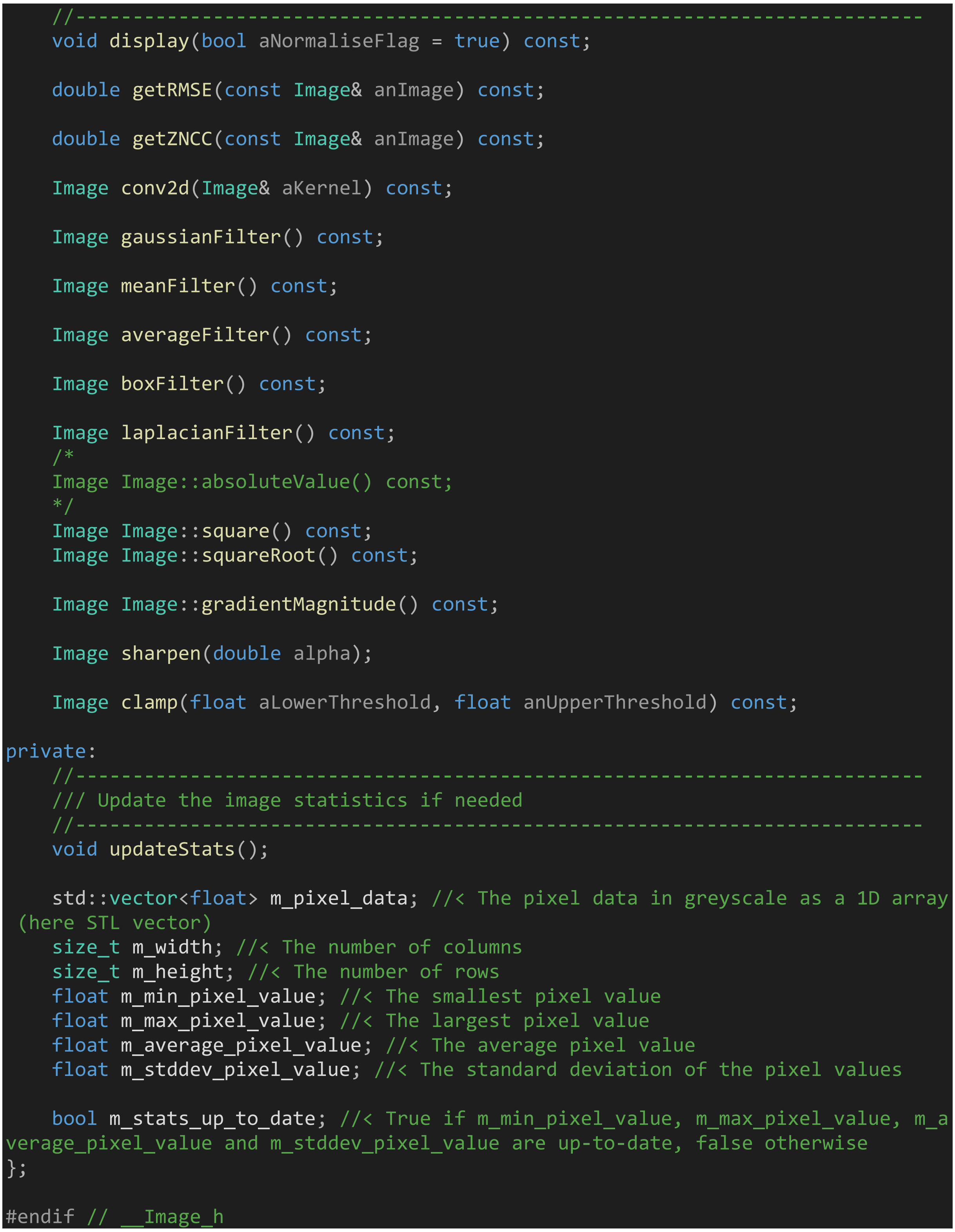
```
#ifndef __Image_h
#define __Image_h

#include <vector>
#include <string>
#include <iostream>

class Image;
std::ostream& operator<<(std::ostream& anOutputStream, const Image& anImage);
Image operator*(float aValue, const Image&);
Image operator+(float aValue, const Image&);

class Image
{
public:
    //--------------------------------------------------------------------------
    /// Default constructor: Create an empty image
    //--------------------------------------------------------------------------
    Image();


    //--------------------------------------------------------------------------
    /// Copy constructor: Copy an existing image
    /**
    * @param anImage: The image to copy
    */
    //--------------------------------------------------------------------------
    Image(const Image& anImage);


    //--------------------------------------------------------------------------
    /// Constructor: Copy an 1D array
    /**
    * @param anImage: The pixel data
    * @param aWidth: The image width
    * @param aHeight: The image height
    */
    //--------------------------------------------------------------------------
    Image(const float* anImage, size_t aWidth, size_t aHeight);


    //--------------------------------------------------------------------------
    /// Constructor: Copy an 1D array
    /**
    * @param anImage: The pixel data
    * @param aWidth: The image width
    * @param aHeight: The image height
    */
    //--------------------------------------------------------------------------
    Image(const std::vector<float>& anImage, size_t aWidth, size_t aHeight);


    //--------------------------------------------------------------------------
    /// Constructor: Copy a uniform image from a constant value
    /**
```



```
    * @param aConstant: The default pixel value
    * @param aWidth: The image width
    * @param aHeight: The image height
    */
    //--------------------------------------------------------------------------
    Image(float aConstant, size_t aWidth, size_t aHeight);


    //--------------------------------------------------------------------------
    /// Constructor: Load a file from the disk
    /**
    * @param aFilename: The name of the file to load
    */
    //--------------------------------------------------------------------------
    Image(const char* aFilename);


    //--------------------------------------------------------------------------
    /// Constructor: Load a file from the disk
    /**
    * @param aFilename: The name of the file to load
    */
    //--------------------------------------------------------------------------
    Image(const std::string& aFilename);


    //--------------------------------------------------------------------------
    /// Assignment operator
    /**
    * @param anInputImage: The image to copy
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image& operator=(const Image& anInputImage);


    //--------------------------------------------------------------------------
    /// Assignment operator
    /**
    * @param aFileName: The name of the file to load
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image& operator=(const char* aFileName);


    //--------------------------------------------------------------------------
    /// Assignment operator
    /**
    * @param aFileName: The name of the file to load
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image& operator=(const std::string& aFileName);


    //--------------------------------------------------------------------------
    /// Load a file from the disk
    /**
    * @param aFilename: The name of the file to load
    */
    //--------------------------------------------------------------------------
    void load(const char* aFilename);


    //--------------------------------------------------------------------------
    /// Load a file from the disk
    /**
    * @param aFilename: The name of the file to load
    */
    //--------------------------------------------------------------------------
    void load(const std::string& aFilename);


    //--------------------------------------------------------------------------
    /// Save a JPEG file on the disk
    /**
    * @param aFilename: The name of the file to save
    */
    //--------------------------------------------------------------------------
    void save(const char* aFilename);


    //--------------------------------------------------------------------------
    /// Save a JPEG file on the disk
    /**
    * @param aFilename: The name of the file to save
    */
    //--------------------------------------------------------------------------
    void save(const std::string& aFilename);


    //--------------------------------------------------------------------------
    /// Accessor on a given pixel
    /**
    * @param col: coordinate of the pixel along the horizontal axis
    * @param row: coordinate of the pixel along the vertical axis
    * @return the corresponding pixel value
    */
    //--------------------------------------------------------------------------
    const float& operator()(size_t col, size_t row) const;


    //--------------------------------------------------------------------------
    /// Accessor on a given pixel
    /**
    * @param col: coordinate of the pixel along the horizontal axis
    * @param row: coordinate of the pixel along the vertical axis
    * @return the corresponding pixel value
    */
    //--------------------------------------------------------------------------
    float& operator()(size_t col, size_t row);


    //--------------------------------------------------------------------------
    /// Accessor on the image width in number of pixels
    /**
    * @return the width of the image in number of pixels
    */
    //--------------------------------------------------------------------------
    size_t getWidth() const;


    //--------------------------------------------------------------------------
    /// Accessor on the image height in number of pixels
    /**
    * @return the height of the image in number of pixels
    */
    //--------------------------------------------------------------------------
    size_t getHeight() const;


    //--------------------------------------------------------------------------
    /// Accessor on the raw pixel data
    /**
    * @return the pointer on the raw pixel data
    */
    //--------------------------------------------------------------------------
    const float* getPixelPointer() const;


    //--------------------------------------------------------------------------
    /// Accessor on the raw pixel data
    /**
    * @return the pointer on the raw pixel data
    */
    //--------------------------------------------------------------------------
    float* getPixelPointer();


    //--------------------------------------------------------------------------
    /// Add a constant value to all the pixels of an image
    /**
    * @param aValue: the value to add
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image operator+(float aValue) const;


    //--------------------------------------------------------------------------
    /// Subtract a constant value to all the pixels of an image
    /**
    * @param aValue: the value to subtract
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image operator-(float aValue) const;


    //--------------------------------------------------------------------------
    /// Multiply all the pixels of an image with a constant value
    /**
    * @param aValue: the value to multiply
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image operator*(float aValue) const;


    //--------------------------------------------------------------------------
    /// Divide all the pixels of an image by a constant value
    /**
    * @param aValue: the divisor
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image operator/(float aValue) const;


    //--------------------------------------------------------------------------
    /// Add a constant value to all the pixels of an image.
    /// This method actually change the pixel values of the instance.
    /**
    * @param aValue: the value to add
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image& operator+=(float aValue);


    //--------------------------------------------------------------------------
    /// Subtract a constant value to all the pixels of an image.
    /// This method actually change the pixel values of the instance.
    /**
    * @param aValue: the value to subtract
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image& operator-=(float aValue);


    //--------------------------------------------------------------------------
    /// Multiply all the pixels of an image with a constant value.
    /// This method actually change the pixel values of the instance.
    /**
    * @param aValue: the value to multiply
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image& operator*=(float aValue);


    //--------------------------------------------------------------------------
    /// Divide all the pixels of an image by a constant value.
    /// This method actually change the pixel values of the instance.
    /**
    * @param aValue: the divisor
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image& operator/=(float aValue);


    //--------------------------------------------------------------------------
    /// Histogram stretching (also known as normalisation).
```



```
    /**
    * @return the new image
    */
    //--------------------------------------------------------------------------
    Image normalise();


    //--------------------------------------------------------------------------
    /// Accessor on the smallest pixel value
    /**
    * @return the smallest pixel value
    */
    //--------------------------------------------------------------------------
    float getMinValue();


    //--------------------------------------------------------------------------
    /// Accessor on the largest pixel value
    /**
    * @return the largest pixel value
    */
    //--------------------------------------------------------------------------
    float getMaxValue();

    float getMean();
    float getStdDev();

    //--------------------------------------------------------------------------
    /// Compute the negative image
    /**
    * @return the negative image
    */
    //--------------------------------------------------------------------------
    Image operator!();

    Image operator+(const Image& img) const;

    Image blending(float alpha, const Image& img1, const Image& img2);

    Image operator-(const Image& img) const;

    Image operator*(const Image& img) const;
    /*
    Image operator/(const Image& img) const;

    Image& operator+=(const Image& img);

    Image& operator-=(const Image& img);

    Image& operator*=(const Image& img);

    Image& operator/=(const Image& img);
    */
    //--------------------------------------------------------------------------
    /// Display the image
    /**
    * @param aNormaliseFlag: a flag used to optionally normalise the data first,
    *                        then display (default value: true)
    */
```

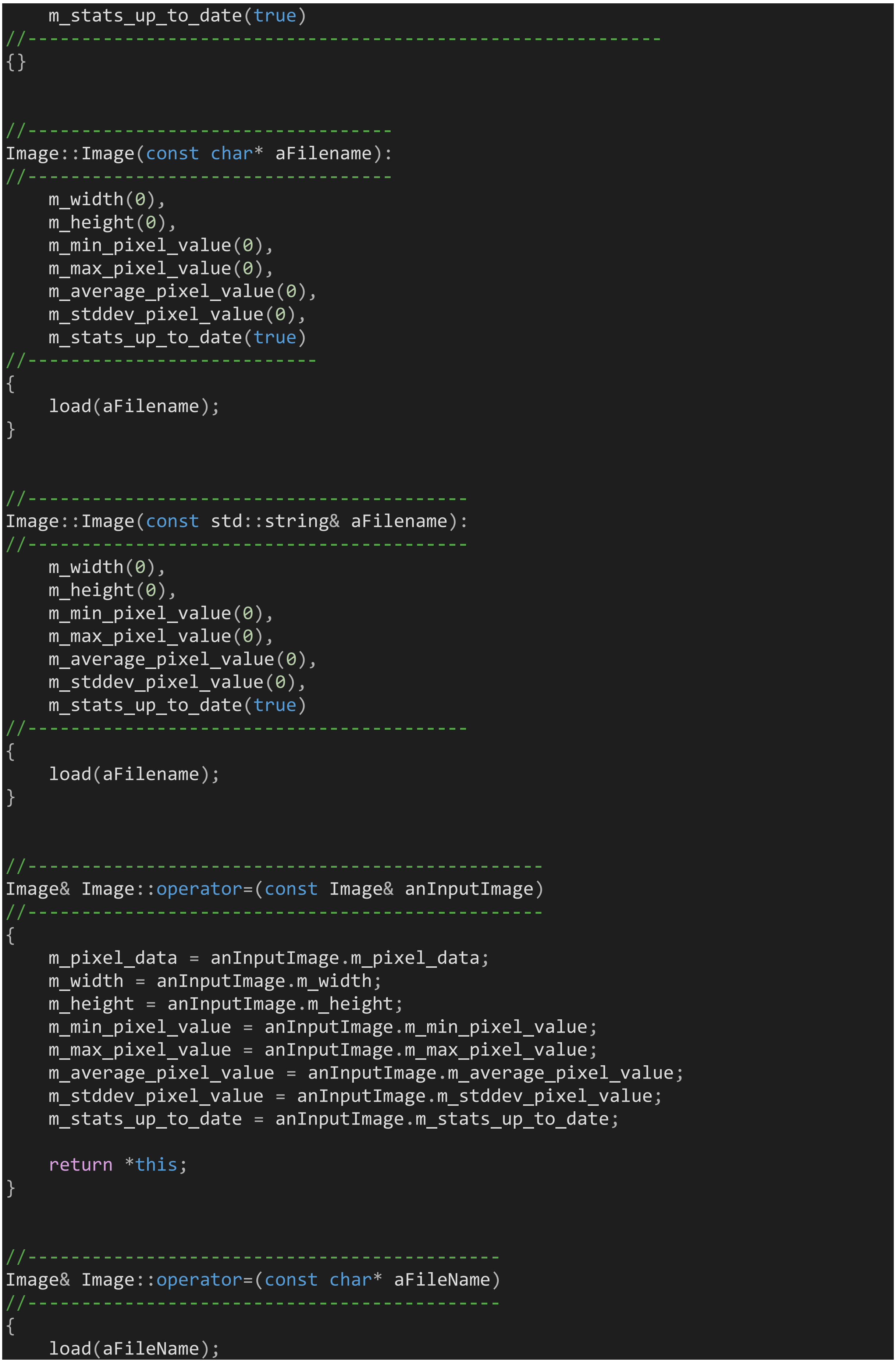


```
    //--------------------------------------------------------------------------
    void display(bool aNormaliseFlag = true) const;

    double getRMSE(const Image& anImage) const;

    double getZNCC(const Image& anImage) const;

    Image conv2d(Image& aKernel) const;

    Image gaussianFilter() const;

    Image meanFilter() const;

    Image averageFilter() const;

    Image boxFilter() const;

    Image laplacianFilter() const;
    /*
    Image Image::absoluteValue() const;
    */
    Image Image::square() const;
    Image Image::squareRoot() const;

    Image Image::gradientMagnitude() const;

    Image sharpen(double alpha);

    Image clamp(float aLowerThreshold, float anUpperThreshold) const;

private:
    //--------------------------------------------------------------------------
    /// Update the image statistics if needed
    //--------------------------------------------------------------------------
    void updateStats();

    std::vector<float> m_pixel_data; //< The pixel data in greyscale as a 1D array
 (here STL vector)
    size_t m_width; //< The number of columns
    size_t m_height; //< The number of rows
    float m_min_pixel_value; //< The smallest pixel value
    float m_max_pixel_value; //< The largest pixel value
    float m_average_pixel_value; //< The average pixel value
    float m_stddev_pixel_value; //< The standard deviation of the pixel values

    bool m_stats_up_to_date; //< True if m_min_pixel_value, m_max_pixel_value, m_a
verage_pixel_value and m_stddev_pixel_value are up-to-date, false otherwise
};

#endif // __Image_h
```

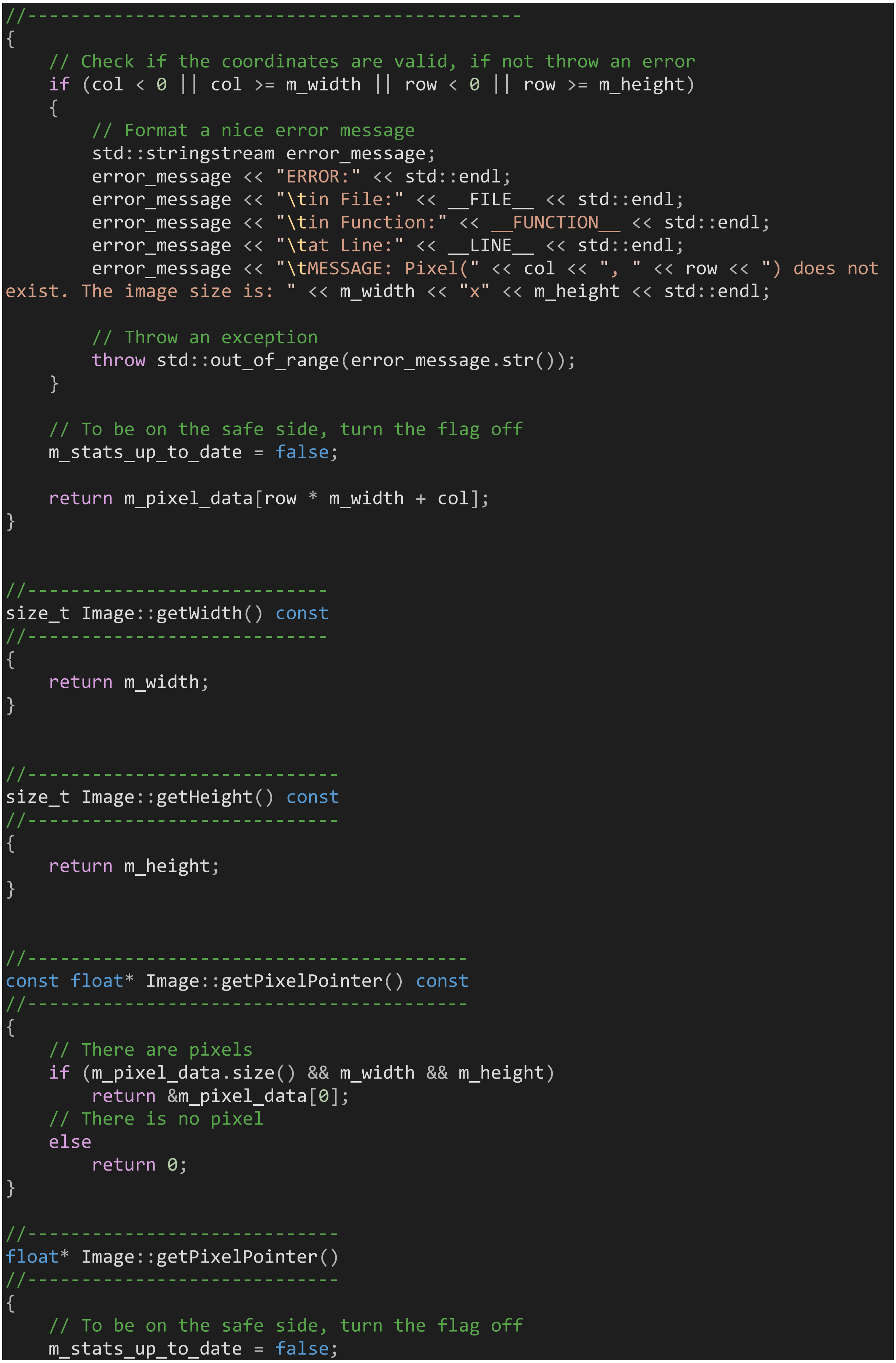
Image.cxx

```
#include <sstream>
#include <stdexcept>      // std::out_of_range
#include <cmath>

#include <algorithm> // For std::min
#include <cstring> // For toupper
#include <fstream> // For ofstream and ifstream

#ifdef HAS_OPENCV
#include <opencv2/opencv.hpp>
#endif

#include "Image.h"


//------------------------------------------------------
Image operator*(float aValue, const Image& anInputImage)
//------------------------------------------------------
{
    Image temp = anInputImage;

    float* p_data = temp.getPixelPointer();
    size_t number_of_pixels = temp.getWidth() * temp.getHeight();

    for (size_t i = 0; i < number_of_pixels; ++i)
    {
        *p_data++ *= aValue;
    }

    return temp;
}


//------------------------------------------------------
Image operator+(float aValue, const Image& anInputImage)
//------------------------------------------------------
{
    Image temp = anInputImage;

    float* p_data = temp.getPixelPointer();
    size_t number_of_pixels = temp.getWidth() * temp.getHeight();

    for (size_t i = 0; i < number_of_pixels; ++i)
    {
        *p_data++ += aValue;
    }

    return temp;
}

//-------------------------------------------------
Image operator-(float aValue, const Image& anImage)
//-------------------------------------------------
{
    // Create a black image of the right size:
    Image temp(0.0, anImage.getWidth(), anImage.getHeight());
    // Create two pointers on the raw pixel data of the input and output respectiv
ely:
    const float* p_input_data = anImage.getPixelPointer();
    float* p_output_data = temp.getPixelPointer();

    // Process all the pixels in a for loop:
    size_t number_of_pixels = temp.getWidth() * temp.getHeight();
    for (size_t i = 0; i < number_of_pixels; ++i)
    {
        *p_output_data++ += aValue - *p_input_data++;
    }

    // Return the new image
    return temp;
}


//--------------------------------------------------------------------------
std::ostream& operator<<(std::ostream& anOutputStream, const Image& anImage)
//--------------------------------------------------------------------------
{
    // Output all the rows
    for (unsigned row = 0; row < anImage.getHeight(); ++row)
    {
        // Output all the columns
        for (unsigned col = 0; col < anImage.getWidth(); ++col)
        {
            // Output the corresponding pixel
            anOutputStream << anImage(col, row);

            // Add a space charater if needed
            if (col < anImage.getWidth() - 1)
            {
                anOutputStream << " ";
            }
        }

        // Add a end of line if needed
        if (row < anImage.getHeight() - 1)
        {
            anOutputStream << std::endl;
        }
    }

    // Return the stream
    return anOutputStream;
}


//---------------------------
Image::Image():
//---------------------------
    m_width(0),
    m_height(0),
    m_min_pixel_value(0),
    m_max_pixel_value(0),
    m_average_pixel_value(0),
    m_stddev_pixel_value(0),
    m_stats_up_to_date(true)
```

```
//----------------------------
{}


//---------------------------------------------------------
Image::Image(const Image& anImage):
//---------------------------------------------------------
    m_pixel_data(anImage.m_pixel_data),
    m_width(anImage.m_width),
    m_height(anImage.m_height),
    m_min_pixel_value(anImage.m_min_pixel_value),
    m_max_pixel_value(anImage.m_max_pixel_value),
    m_average_pixel_value(anImage.m_average_pixel_value),
    m_stddev_pixel_value(anImage.m_stddev_pixel_value),
    m_stats_up_to_date(anImage.m_stats_up_to_date)
//---------------------------------------------------------
{}


//------------------------------------------------------------------
Image::Image(const float* anImage, size_t aWidth, size_t aHeight):
//------------------------------------------------------------------
    m_pixel_data(anImage, anImage + aWidth * aHeight),
    m_width(aWidth),
    m_height(aHeight),
    m_min_pixel_value(0),
    m_max_pixel_value(0),
    m_average_pixel_value(0),
    m_stddev_pixel_value(0),
    m_stats_up_to_date(false)
//------------------------------------------------------------------
{}


//-------------------------------------------------------------------------------
Image::Image(const std::vector<float>& anImage, size_t aWidth, size_t aHeight):
//-------------------------------------------------------------------------------
    m_pixel_data(anImage),
    m_width(aWidth),
    m_height(aHeight),
    m_min_pixel_value(0),
    m_max_pixel_value(0),
    m_average_pixel_value(0),
    m_stddev_pixel_value(0),
    m_stats_up_to_date(false)
//-------------------------------------------------------------------------------
{}


//-------------------------------------------------------------
Image::Image(float aConstant, size_t aWidth, size_t aHeight):
//-------------------------------------------------------------
    m_pixel_data(aWidth * aHeight, aConstant),
    m_width(aWidth),
    m_height(aHeight),
    m_min_pixel_value(aConstant),
    m_max_pixel_value(aConstant),
    m_average_pixel_value(aConstant),
    m_stddev_pixel_value(0),
```

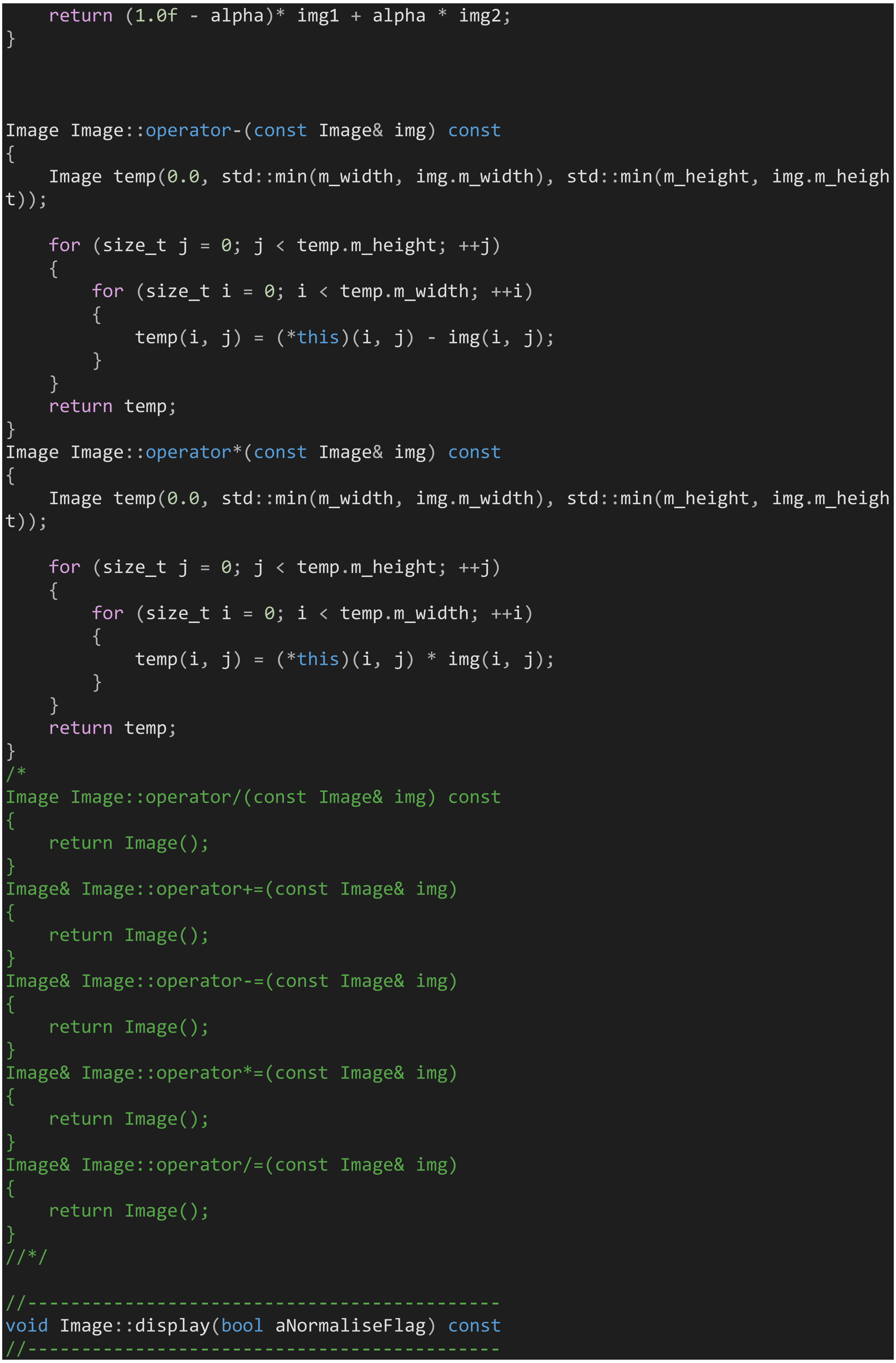


```
    m_stats_up_to_date(true)
//-----------------------------------------------------------
{}


//----------------------------------
Image::Image(const char* aFilename):
//----------------------------------
    m_width(0),
    m_height(0),
    m_min_pixel_value(0),
    m_max_pixel_value(0),
    m_average_pixel_value(0),
    m_stddev_pixel_value(0),
    m_stats_up_to_date(true)
//---------------------------
{
    load(aFilename);
}


//-----------------------------------------
Image::Image(const std::string& aFilename):
//-----------------------------------------
    m_width(0),
    m_height(0),
    m_min_pixel_value(0),
    m_max_pixel_value(0),
    m_average_pixel_value(0),
    m_stddev_pixel_value(0),
    m_stats_up_to_date(true)
//-----------------------------------------
{
    load(aFilename);
}


//------------------------------------------------
Image& Image::operator=(const Image& anInputImage)
//------------------------------------------------
{
    m_pixel_data = anInputImage.m_pixel_data;
    m_width = anInputImage.m_width;
    m_height = anInputImage.m_height;
    m_min_pixel_value = anInputImage.m_min_pixel_value;
    m_max_pixel_value = anInputImage.m_max_pixel_value;
    m_average_pixel_value = anInputImage.m_average_pixel_value;
    m_stddev_pixel_value = anInputImage.m_stddev_pixel_value;
    m_stats_up_to_date = anInputImage.m_stats_up_to_date;

    return *this;
}


//--------------------------------------------
Image& Image::operator=(const char* aFileName)
//--------------------------------------------
{
    load(aFileName);
    return *this;
}


//--------------------------------------------------
Image& Image::operator=(const std::string& aFileName)
//--------------------------------------------------
{
    load(aFileName);
    return *this;
}


//------------------------------------
void Image::load(const char* aFilename)
//------------------------------------
{
    std::string temp_filename = aFilename;
    std::string capital_filename;

    // Capitalise
    for (int i = 0; i < temp_filename.size(); ++i)
        capital_filename += std::toupper(temp_filename[i]);

    if (std::string(aFilename).size() > 4)
    {
        // Load a text file
        if(capital_filename.substr( capital_filename.length() - 4 ) == ".TXT")
        {
            // Open the file
            std::ifstream input_file (aFilename);

            // The file is not open
            if (!input_file.is_open())
            {
                // Format a nice error message
                std::stringstream error_message;
                error_message << "ERROR:" << std::endl;
                error_message << "\tin File:" << __FILE__ << std::endl;
                error_message << "\tin Function:" << __FUNCTION__ << std::endl;
                error_message << "\tat Line:" << __LINE__ << std::endl;
                error_message << "\tMESSAGE: Can't open " << aFilename << std::end
l;

                throw std::runtime_error(error_message.str());
            }

            // Empty the image
            m_pixel_data.clear();
            m_width = 0;
            m_height = 0;

            // Load the data into a vector
            std::string line;
            int number_of_rows(0);
            int number_of_columns(0);

            // Read evely line
            while (std::getline(input_file, line))
            {
                number_of_columns = 0;
                float intensity;
                std::stringstream line_parser;
                line_parser << line;
                while (line_parser >> intensity)
                {
                 m_pixel_data.push_back(intensity);
                    ++number_of_columns;
                }
                ++number_of_rows;
            }

            // Wrong number of pixels
            float size = number_of_rows * number_of_columns;
            if (size != m_pixel_data.size())
            {
                // Format a nice error message
                std::stringstream error_message;
                error_message << "ERROR:" << std::endl;
                error_message << "\tin File:" << __FILE__ << std::endl;
                error_message << "\tin Function:" << __FUNCTION__ << std::endl;
                error_message << "\tat Line:" << __LINE__ << std::endl;
                error_message << "\tMESSAGE: The file " << aFilename << " is inval
id" << std::endl;
                throw std::runtime_error(error_message.str());
            }

            // Allocate memory for file content
            m_width = number_of_columns;
            m_height = number_of_rows;
            m_stats_up_to_date = false;
        }
        // Use OpenCV
        else
        {
#ifdef HAS_OPENCV
            // Open the image in greyscale
            cv::Mat temp_image = cv::imread( aFilename, cv::IMREAD_GRAYSCALE);

            // The image did not load
            if ( !temp_image.data )
            {
                // Format a nice error message
                std::stringstream error_message;
                error_message << "ERROR:" << std::endl;
                error_message << "\tin File:" << __FILE__ << std::endl;
                error_message << "\tin Function:" << __FUNCTION__ << std::endl;
                error_message << "\tat Line:" << __LINE__ << std::endl;
                error_message << "\tMESSAGE: Can't open " << aFilename << std::end
l;

                throw std::runtime_error(error_message.str());
            }

            // Save the size of the image
            m_width = temp_image.cols;
            m_height = temp_image.rows;
            m_pixel_data.resize(m_width * m_height);

            // Copy the pixel data
            cv::Mat img_float;
            temp_image.convertTo(img_float, CV_32F);
            for (int i = 0; i < m_width * m_height; ++i)
            {
                int x = i / m_width;
                int y = i % m_width;

                m_pixel_data[i] = img_float.at<float>(x, y);
            }


            // The statistics is not up-to-date
            m_stats_up_to_date = false;
#else
            // Format a nice error message
            std::stringstream error_message;
            error_message << "ERROR:" << std::endl;
            error_message << "\tin File:" << __FILE__ << std::endl;
            error_message << "\tin Function:" << __FUNCTION__ << std::endl;
            error_message << "\tat Line:" << __LINE__ << std::endl;
            error_message << "\tMESSAGE: OpenCV not supported" << std::endl;
            throw std::runtime_error(error_message.str());
#endif
        }
    }
    // Don't know the file type
    else
    {
        // Format a nice error message
        std::stringstream error_message;
        error_message << "ERROR:" << std::endl;
        error_message << "\tin File:" << __FILE__ << std::endl;
        error_message << "\tin Function:" << __FUNCTION__ << std::endl;
        error_message << "\tat Line:" << __LINE__ << std::endl;
        error_message << "\tMESSAGE: Can't open " << aFilename << ", I don't under
stand the file type." << std::endl;
        throw std::runtime_error(error_message.str());
    }
}


//-------------------------------------------
void Image::load(const std::string& aFilename)
//-------------------------------------------
{
    load(aFilename.c_str());
}


//------------------------------------------
void Image::save(const char* aFilename)
//------------------------------------------
{
    std::string temp_filename = aFilename;
    std::string capital_filename;

    // Capitalise
    for (int i = 0; i < temp_filename.size(); ++i)
        capital_filename += std::toupper(temp_filename[i]);
    if (std::string(aFilename).size() > 4)
    {
        // Load a text file
        if(capital_filename.substr( capital_filename.length() - 4 ) == ".TXT")
        {
            // Open the file
            std::ofstream output_file (aFilename);

            // The file is not open
            if (!output_file.is_open())
            {
                // Format a nice error message
                std::stringstream error_message;
                error_message << "ERROR:" << std::endl;
                error_message << "\tin File:" << __FILE__ << std::endl;
                error_message << "\tin Function:" << __FUNCTION__ << std::endl;
                error_message << "\tat Line:" << __LINE__ << std::endl;
                error_message << "\tMESSAGE: Can't open " << aFilename << std::end
l;

                throw std::runtime_error(error_message.str());
            }

            // Write content to file
            float* p_data(getPixelPointer());
            for (unsigned int j(0); j < m_height; ++j)
            {
                for (unsigned int i(0); i < m_width; ++i)
                {
                    output_file << *p_data++;

                    // This is not the last pixel of the line
                    if (i < m_width - 1)
                    {
                        output_file << " ";
                    }
                }

                // This is not the last line
                if (j < m_height - 1)
                {
                    output_file << std::endl;
                }
            }
        }
        // Use OpenCV
        else
        {
#ifdef HAS_OPENCV
            // Convert the data into an OpenCV Mat instance.
            cv::Mat temp_image(m_height, m_width, CV_32FC1, (float*)getPixelPointe
r());

            // Write the data
            cv::imwrite(aFilename, temp_image);

#else
            // Format a nice error message
            std::stringstream error_message;
            error_message << "ERROR:" << std::endl;
            error_message << "\tin File:" << __FILE__ << std::endl;
            error_message << "\tin Function:" << __FUNCTION__ << std::endl;
            error_message << "\tat Line:" << __LINE__ << std::endl;
            error_message << "\tMESSAGE: OpenCV not supported" << std::endl;
            throw std::runtime_error(error_message.str());
#endif
        }
    }
    // Don't know the file type
    else
    {
        // Format a nice error message
        std::stringstream error_message;
        error_message << "ERROR:" << std::endl;
        error_message << "\tin File:" << __FILE__ << std::endl;
        error_message << "\tin Function:" << __FUNCTION__ << std::endl;
        error_message << "\tat Line:" << __LINE__ << std::endl;
        error_message << "\tMESSAGE: Can't save " << aFilename << ", I don't under
stand the file type." << std::endl;
        throw std::runtime_error(error_message.str());
    }
}


//------------------------------------------------
void Image::save(const std::string& aFilename)
//------------------------------------------------
{
    save(aFilename.c_str());
}


//----------------------------------------------------------
const float& Image::operator()(size_t col, size_t row) const
//----------------------------------------------------------
{
    // Check if the coordinates are valid, if not throw an error
    if (col < 0 || col >= m_width || row < 0 || row >= m_height)
    {
        // Format a nice error message
        std::stringstream error_message;
        error_message << "ERROR:" << std::endl;
        error_message << "\tin File:" << __FILE__ << std::endl;
        error_message << "\tin Function:" << __FUNCTION__ << std::endl;
        error_message << "\tat Line:" << __LINE__ << std::endl;
        error_message << "\tMESSAGE: Pixel(" << col << ", " << row << ") does not
exist. The image size is: " << m_width << "x" << m_height << std::endl;

        // Throw an exception
        throw std::out_of_range(error_message.str());
    }

    return m_pixel_data[row * m_width + col];
}


//----------------------------------------------
float& Image::operator()(size_t col, size_t row)
```



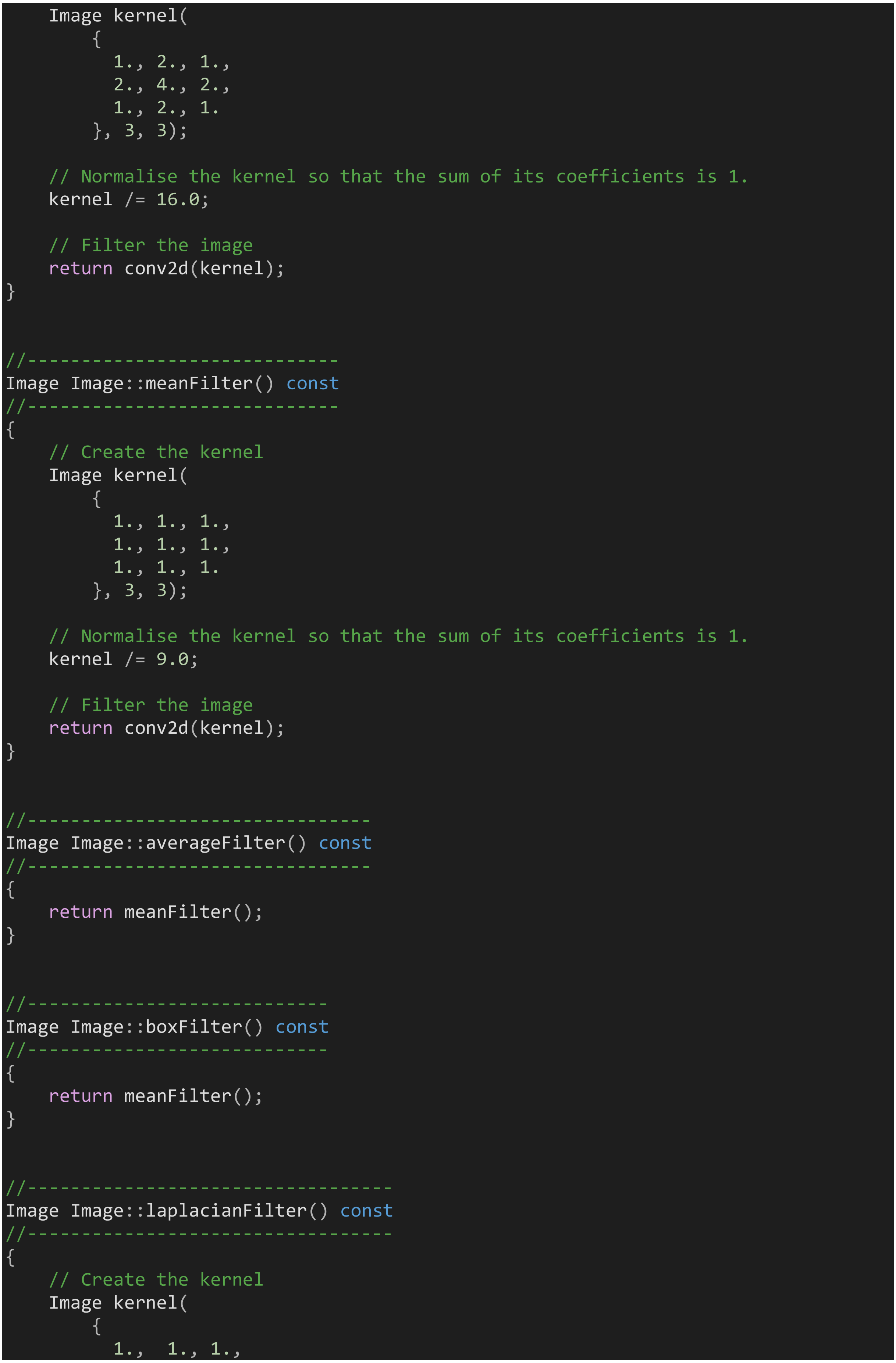
```
//----------------------------------------------
{
    // Check if the coordinates are valid, if not throw an error
    if (col < 0 || col >= m_width || row < 0 || row >= m_height)
    {
        // Format a nice error message
        std::stringstream error_message;
        error_message << "ERROR:" << std::endl;
        error_message << "\tin File:" << __FILE__ << std::endl;
        error_message << "\tin Function:" << __FUNCTION__ << std::endl;
        error_message << "\tat Line:" << __LINE__ << std::endl;
        error_message << "\tMESSAGE: Pixel(" << col << ", " << row << ") does not
exist. The image size is: " << m_width << "x" << m_height << std::endl;

        // Throw an exception
        throw std::out_of_range(error_message.str());
    }

    // To be on the safe side, turn the flag off
    m_stats_up_to_date = false;

    return m_pixel_data[row * m_width + col];
}


//----------------------------
size_t Image::getWidth() const
//----------------------------
{
    return m_width;
}


//-----------------------------
size_t Image::getHeight() const
//-----------------------------
{
    return m_height;
}


//-----------------------------------------
const float* Image::getPixelPointer() const
//-----------------------------------------
{
    // There are pixels
    if (m_pixel_data.size() && m_width && m_height)
        return &m_pixel_data[0];
    // There is no pixel
    else
        return 0;
}

//-----------------------------
float* Image::getPixelPointer()
//-----------------------------
{
    // To be on the safe side, turn the flag off
    m_stats_up_to_date = false;
    // There are pixels
    if (m_pixel_data.size() && m_width && m_height)
        return &m_pixel_data[0];
    // There is no pixel
    else
        return 0;
}


//----------------------------------------
Image Image::operator+(float aValue) const
//----------------------------------------
{
    Image temp = *this;

    for (size_t i = 0; i < m_width * m_height; ++i)
    {
        temp.m_pixel_data[i] += aValue;
    }

    return temp;
}


//----------------------------------------
Image Image::operator-(float aValue) const
//----------------------------------------
{
    Image temp = *this;

    for (size_t i = 0; i < m_width * m_height; ++i)
    {
        temp.m_pixel_data[i] -= aValue;
    }

    return temp;
}


//----------------------------------------
Image Image::operator*(float aValue) const
//----------------------------------------
{
    Image temp = *this;

    for (size_t i = 0; i < m_width * m_height; ++i)
    {
        temp.m_pixel_data[i] *= aValue;
    }

    return temp;
}


//----------------------------------------
Image Image::operator/(float aValue) const
//----------------------------------------
{
    Image temp = *this;

    for (size_t i = 0; i < m_width * m_height; ++i)
    {
        temp.m_pixel_data[i] /= aValue;
    }

    return temp;
}


//------------------------------------
Image& Image::operator+=(float aValue)
//------------------------------------
{
    *this = *this + aValue;
    return *this;
}


//------------------------------------
Image& Image::operator-=(float aValue)
//------------------------------------
{
    *this = *this - aValue;
    return *this;
}


//------------------------------------
Image& Image::operator*=(float aValue)
//------------------------------------
{
    *this = *this * aValue;
    return *this;
}


//------------------------------------
Image& Image::operator/=(float aValue)
//------------------------------------
{
    *this = *this / aValue;
    return *this;
}


//----------------------
Image Image::normalise()
//----------------------
{
    return (*this - getMinValue()) / (getMaxValue() - getMinValue());
}

//Image Image::operator!() const;


//------------------------
float Image::getMinValue()
```

```
//------------------------
{
    if (!m_stats_up_to_date) updateStats();

    return m_min_pixel_value;
}


//------------------------
float Image::getMaxValue()
//------------------------
{
    if (!m_stats_up_to_date) updateStats();

    return m_max_pixel_value;
}

//------------------------
float Image::getMean()
//------------------------
{
    if (!m_stats_up_to_date) updateStats();

    return m_average_pixel_value;
}

//------------------------
float Image::getStdDev()
//------------------------
{
    if (!m_stats_up_to_date) updateStats();

    return m_stddev_pixel_value;
}

Image Image::operator!()
{
    return getMinValue() + getMaxValue() - *this;
}

//-------------------------------------------
Image Image::operator+(const Image& img) const
//-------------------------------------------
{
    Image temp(0.0, std::min(m_width, img.m_width), std::min(m_height, img.m_heigh
t));

    for (size_t j = 0; j < temp.m_height; ++j)
    {
        for (size_t i = 0; i < temp.m_width; ++i)
        {
            temp(i, j) = (*this)(i, j) + img(i, j);
        }
    }
    return temp;
}

Image Image::blending(float alpha, const Image& img1, const Image& img2)
{
```



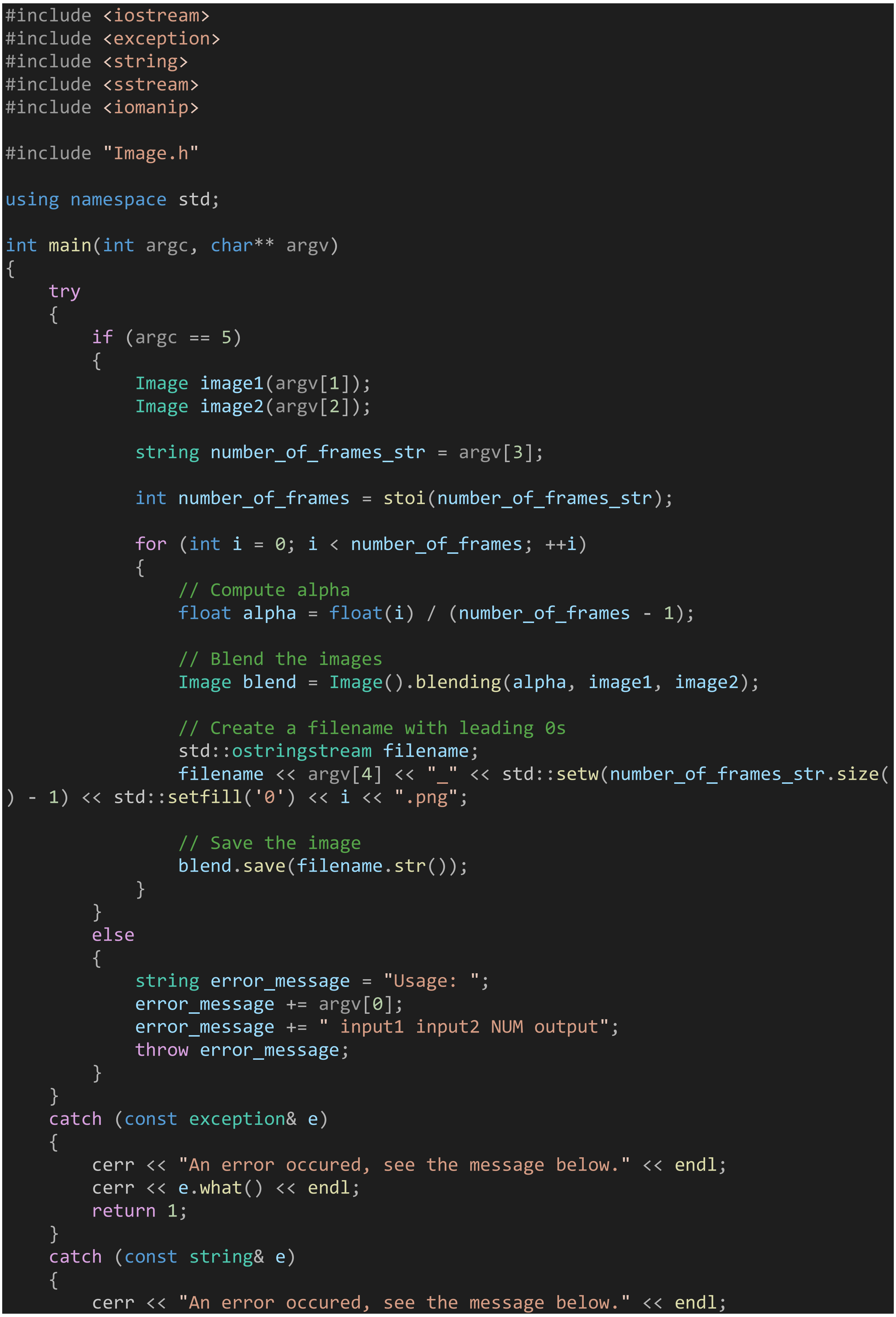
```
    return (1.0f - alpha)* img1 + alpha * img2;
}



Image Image::operator-(const Image& img) const
{
    Image temp(0.0, std::min(m_width, img.m_width), std::min(m_height, img.m_heigh
t));

    for (size_t j = 0; j < temp.m_height; ++j)
    {
        for (size_t i = 0; i < temp.m_width; ++i)
        {
            temp(i, j) = (*this)(i, j) - img(i, j);
        }
    }
    return temp;
}
Image Image::operator*(const Image& img) const
{
    Image temp(0.0, std::min(m_width, img.m_width), std::min(m_height, img.m_heigh
t));

    for (size_t j = 0; j < temp.m_height; ++j)
    {
        for (size_t i = 0; i < temp.m_width; ++i)
        {
            temp(i, j) = (*this)(i, j) * img(i, j);
        }
    }
    return temp;
}
/*
Image Image::operator/(const Image& img) const
{
    return Image();
}
Image& Image::operator+=(const Image& img)
{
    return Image();
}
Image& Image::operator-=(const Image& img)
{
    return Image();
}
Image& Image::operator*=(const Image& img)
{
    return Image();
}
Image& Image::operator/=(const Image& img)
{
    return Image();
}
//*/

//---------------------------------------------
void Image::display(bool aNormaliseFlag) const
//---------------------------------------------
```

```
{
#ifdef HAS_OPENCV
    Image display_image = *this;

    // Normalise the image if needed
    if (aNormaliseFlag)
        display_image = display_image.normalise();

    // Convert the data into an OpenCV Mat instance.
    cv::Mat cv_image(m_height, m_width, CV_32FC1, (float*)display_image.getPixelPo
inter());

    // Display the image
    cv::imshow("Display window", cv_image);

    // Wait for a keystroke in the window
    cv::waitKey(0);
#endif // HAS_OPENCV
}

double Image::getRMSE(const Image& anImage) const
{
    float wid = anImage.m_width;
    float hei = anImage.m_height;
    double temp = 0.0;
    if (wid == this->getWidth() && hei == this->getHeight()) {
        for (size_t j = 0; j < hei; ++j)
        {
            for (size_t i = 0; i < wid; ++i)
            {
                temp = sqrt((1/(wid*hei))*((j*2)*(i*2))*((anImage(i, j) - (*this)(
i, j))*(anImage(i, j) - (*this)(i, j))));
            }
        }
        return temp;
    }
    else {
        return 743697157;
    }
}

double Image::getZNCC(const Image& anImage) const
{
    float wid = anImage.m_width;
    float hei = anImage.m_height;
    float temp = 0.0;
    if (wid == this->getWidth() && hei == this->getHeight()) {
        for (float j = 0; j < hei; ++j)
        {
            for (float i = 0; i < wid; ++i)
            {

                temp = ((float)1.0 / (wid * hei)) *((j * (float)2.0) * (i * (float
)2.0)) * (((anImage(i, j) - anImage.m_average_pixel_value)*((*this)(i, j) - this-
>m_average_pixel_value))/(anImage.m_stddev_pixel_value*this-
>m_stddev_pixel_value));
            }
        }
        return temp;
    }
    else {
        return 743697157;
    }
}

//---------------------------------------------
Image Image::conv2d(Image& aKernel) const
//---------------------------------------------
{
    // Initialise the output image so that it's black and it has the right size
    Image output_image(0.0, m_width, m_height);
    int imw = output_image.m_width;
    int imh = output_image.m_height;
    int kew = aKernel.m_width;
    int keh = aKernel.m_height;

    for (int ir = 0; ir < imw; ir++) {
        for (int ic = 0; ic < imh; ic++) {
            int acc = 0;
            for (int kr = 0; kr < kew; kr++) {
                for (int kc = 0; kc < keh; kc++) {
                    float imx = 0;
                    float imy = 0;
                    if (((ir - kew) / 2 + kr) < 0) {
                        imx = 0;
                    }
                    else if (((ir - kew) / 2 + kr) > imw) {
                        imx = imw - 1;
                    }
                    else {
                        imx = ((ir - kew) / 2 + kr);
                    }
                    if (((ic - keh) / 2 + kc) < 0) {
                        imy = 0;
                    }
                    else if (((ic - keh) / 2 + kc) > imh) {
                        imy = imh - 1;
                    }
                    else {
                        imy = ((ic - keh) / 2 + kc);
                    }
                    acc += (*this)(imx,imy) * aKernel(kr,kc);
                }
            }
            output_image(ir, ic) = acc;
        }
    }

    // Return the output
    return output_image;
}


//-----------------------------------
Image Image::gaussianFilter() const
//-----------------------------------
{
    // Create the kernel
```



```
    Image kernel(
        {
          1., 2., 1.,
          2., 4., 2.,
          1., 2., 1.
        }, 3, 3);

    // Normalise the kernel so that the sum of its coefficients is 1.
    kernel /= 16.0;

    // Filter the image
    return conv2d(kernel);
}


//-----------------------------
Image Image::meanFilter() const
//-----------------------------
{
    // Create the kernel
    Image kernel(
        {
          1., 1., 1.,
          1., 1., 1.,
          1., 1., 1.
        }, 3, 3);

    // Normalise the kernel so that the sum of its coefficients is 1.
    kernel /= 9.0;

    // Filter the image
    return conv2d(kernel);
}


//--------------------------------
Image Image::averageFilter() const
//--------------------------------
{
    return meanFilter();
}


//----------------------------
Image Image::boxFilter() const
//----------------------------
{
    return meanFilter();
}


//----------------------------------
Image Image::laplacianFilter() const
//----------------------------------
{
    // Create the kernel
    Image kernel(
        {
          1.,  1., 1.,
          1., -8., 1.,
          1.,  1., 1.
        }, 3, 3);

    // Filter the image
    return conv2d(kernel);
}
/*
Image Image::absoluteValue() const
{
    return Image();
}
*/
Image Image::square() const
{
    Image temp(0.0, this->m_width, this->m_height);

    for (size_t j = 0; j < temp.m_height; ++j)
    {
        for (size_t i = 0; i < temp.m_width; ++i)
        {
            temp(i, j) = (*this)(i, j) * (*this)(i, j);
        }
    }
    return temp;
}

Image Image::squareRoot() const
{
    Image temp(0.0, this->m_width, this->m_height);

    for (size_t j = 0; j < temp.m_height; ++j)
    {
        for (size_t i = 0; i < temp.m_width; ++i)
        {
            temp(i, j) = sqrt((*this)(i, j));
        }
    }
    return temp;
}


Image Image::gradientMagnitude() const
{
    // Create the kernels
    Image kernelgx(
        {
          1., 0., -1.,
          2., 0., -2.,
          1., 0., -1.
        }, 3, 3);
    Image kernelgy(
        {
          1., 2., 1.,
          0., 0., 0.,
          -1., -2., -1.
        }, 3, 3);

    // Filter the image
    Image vert = conv2d(kernelgx);
    Image hori = conv2d(kernelgy);

    return (vert.square() + hori.square()).squareRoot();
}

Image Image::sharpen(double alpha)
{
    Image gaussian_5x5_kernel(
        {
            1.,  4.,  7.,  4., 1.,
            4., 16., 26., 16., 4.,
            7., 26., 41., 26., 7.,
            4., 16., 26., 16., 4.,
            1.,  4.,  7.,  4., 1.
        }, 5, 5);
    gaussian_5x5_kernel /= 273.;

    Image blur = conv2d(gaussian_5x5_kernel);
    Image details = *this - blur;

    Image output = *this + alpha * details;

    return output.clamp(getMinValue(), getMaxValue());
}

Image Image::clamp(float aLowerThreshold, float anUpperThreshold) const
{
    Image input = *this;
    Image output;
    for (int j = 0; j < input.m_height; ++j) {
        for (int i = 0; i < input.m_width; ++i) {
            if (input(i, j) < aLowerThreshold) {
                output(i, j) = aLowerThreshold;
            }
            else if (input(i, j) > anUpperThreshold) {
                output(i, j) = anUpperThreshold;
            }
            else {
                output(i, j) = input(i, j);
            }
        }
    }
    return output;
}

//-----------------------
void Image::updateStats()
//-----------------------
{
    // Need to udate the stats
    if (!m_stats_up_to_date && m_width * m_height)
    {
        m_min_pixel_value = m_pixel_data[0];
        m_max_pixel_value = m_pixel_data[0];
        m_average_pixel_value = m_pixel_data[0];

        for (size_t i = 1; i < m_width * m_height; ++i)
        {
            if (m_min_pixel_value > m_pixel_data[i]) m_min_pixel_value = m_pixel_data[i];
            if (m_max_pixel_value < m_pixel_data[i]) m_max_pixel_value = m_pixel_data[i];

            m_average_pixel_value += m_pixel_data[i];
        }
        m_average_pixel_value /= m_width * m_height;

        m_stddev_pixel_value = 0;
        for (size_t i = 0; i < m_width * m_height; ++i)
        {
            m_stddev_pixel_value += (m_pixel_data[i] - m_average_pixel_value) * (m_pixel_data[i] - m_average_pixel_value);
        }
        m_stddev_pixel_value /= m_width * m_height;
        m_stddev_pixel_value = sqrt(m_stddev_pixel_value);

        m_stats_up_to_date = true;
    }
}
```

blending.cxx



```
#include <iostream>
#include <exception>
#include <string>
#include <sstream>
#include <iomanip>

#include "Image.h"

using namespace std;

int main(int argc, char** argv)
{
    try
    {
        if (argc == 5)
        {
            Image image1(argv[1]);
            Image image2(argv[2]);

            string number_of_frames_str = argv[3];

            int number_of_frames = stoi(number_of_frames_str);

            for (int i = 0; i < number_of_frames; ++i)
            {
                // Compute alpha
                float alpha = float(i) / (number_of_frames - 1);

                // Blend the images
                Image blend = Image().blending(alpha, image1, image2);

                // Create a filename with leading 0s
                std::ostringstream filename;
                filename << argv[4] << "_" << std::setw(number_of_frames_str.size(
) - 1) << std::setfill('0') << i << ".png";

                // Save the image
                blend.save(filename.str());
            }
        }
        else
        {
            string error_message = "Usage: ";
            error_message += argv[0];
            error_message += " input1 input2 NUM output";
            throw error_message;
        }
    }
    catch (const exception& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e.what() << endl;
        return 1;
    }
    catch (const string& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 2;
    }
    catch (const char* e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 3;
    }

    return 0;
}
```

contrastEnhancement.cxx

```
#include <iostream>
#include <exception>
#include <string>

#include "Image.h"
#include "contrastEnhancement.h"

using namespace std;

int main(int argc, char** argv)
{
    try
    {
        if (argc == 3)
        {
        Image input(argv[1]);

        input -= input.getMinValue();
        input /= input.getMaxValue();
        input *= 255.0;

        // Same as
        // input = 255.0 * (input - input.getMinValue()) /
        //      (input.getMaxValue() - input.getMinValue());


        // Same as
        // input = 255.0 * input.normalise();

        //image.save(argv[2]);
        input.save(argv[2]);
        }
        else
        {
            string arg = argv[0];
            string error_message = "Usage: " + arg + " input_image output_image";
            throw error_message;
        }
    }
    catch (const exception& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e.what() << endl;
        return 1;
    }
    catch (const string& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 2;
    }
    catch (const char* e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 3;
    }

    return 0;
}
```

gaussianFilter.cxx

```
#include <iostream>
#include <exception>
#include <string>

#include "Image.h"

using namespace std;

int main(int argc, char** argv)
{
    try
    {
        if (argc == 3)
        {
            Image input(argv[1]);
            Image output(input.gaussianFilter());
            output.save(argv[2]);
        }
        else
        {
            string arg = argv[0];
            string error_message = "Usage: " + arg + " input_image output_image";
            throw error_message;
        }
    }
    catch (const exception& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e.what() << endl;
        return 1;
    }
    catch (const string& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 2;
    }
    catch (const char* e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 3;
    }

    return 0;
}
```

gradientMagnitude.cxx

```
#include <iostream>
#include <exception>
#include <string>

#include "Image.h"

using namespace std;

int main(int argc, char** argv)
{
    try
    {
        if (argc == 3 || argc == 4)
        {
            Image input(argv[1]);
            Image output(input.gradientMagnitude());

            // Save the image
            std::string output_filename = argv[2];
            std::string capital_filename;

            // Capitalise
            for (int i = 0; i < output_filename.size(); ++i)
                capital_filename += std::toupper(output_filename[i]);

            // There are enough characters for a file extension
            if (std::string(output_filename).size() > 4)
            {
                // Save an ASCII image file: Do not normalise
                if (capital_filename.substr(capital_filename.length() - 4) == ".TX
T")
                {
                    output.save(output_filename);
                }
                // Save the data using an image file format: Normalise
                else
                {
                    (output.normalise() * 255).save(output_filename);
                }
            }

            // Display the image
            if (argc == 4)
            {
                output.display();
            }
        }
        else
        {
            string error_message = string("Usage: ") + argv[0] + " input_image out
put_image [-display]";
            throw error_message;
        }
    }
    catch (const exception& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e.what() << endl;
        return 1;
    }
    catch (const string& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 2;
    }
    catch (const char* e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 3;
    }

    return 0;
}
```

laplacianFilter.cxx

```
#include <iostream>
#include <exception>
#include <string>

#include "Image.h"

using namespace std;

int main(int argc, char** argv)
{
    try
    {
        if (argc == 3)
        {
            Image input(argv[1]);
            Image output(input.laplacianFilter());
            output.save(argv[2]);
        }
        else
        {
            string arg = argv[0];
            string error_message = "Usage: " + arg + " input_image output_image";
            throw error_message;
        }
    }
    catch (const exception& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e.what() << endl;
        return 1;
    }
    catch (const string& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 2;
    }
    catch (const char* e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 3;
    }

    return 0;
}
```

negative.cxx

```
#include <iostream>
#include <exception>
#include <string>

#include "Image.h"
#include "contrastEnhancement.h"

using namespace std;

int main(int argc, char** argv)
{
    try
    {
        if (argc == 3)
        {
            Image input(argv[1]);

            input -= input.getMinValue();
            input /= input.getMaxValue();
            input *= 255.0;

            // Same as
            // input = 255.0 * (input - input.getMinValue()) /
            //     (input.getMaxValue() - input.getMinValue());


            // Same as
            // input = 255.0 * input.normalise();

            //image.save(argv[2]);
            input.save(argv[2]);

            Image image(argv[1]);
            (!image).save(argv[2]);
        }
        else
        {
            string arg = argv[0];
            string error_message = "Usage: " + arg + " input_image output_image";
            throw error_message;
        }
    }
    catch (const exception& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e.what() << endl;
        return 1;
    }
    catch (const string& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 2;
    }
    catch (const char* e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 3;
    }

    return 0;
}
```

sharpen.cxx

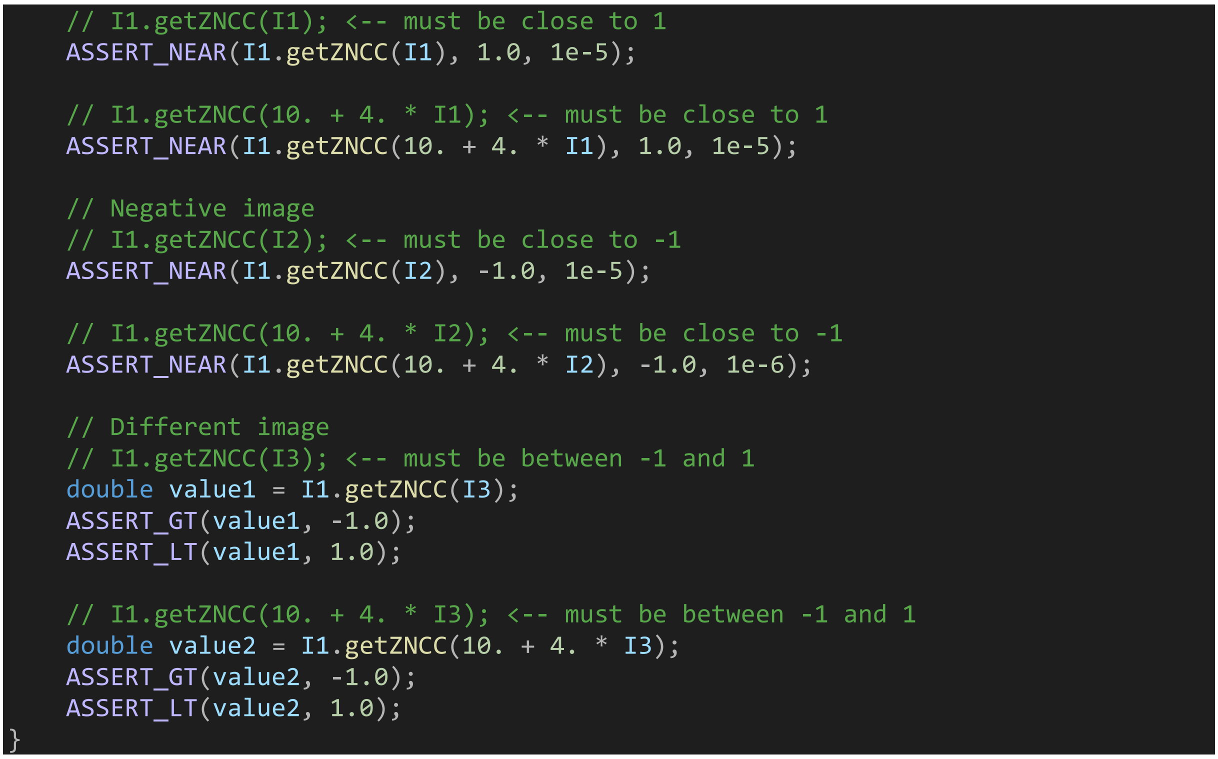
```
#include <iostream>
#include <exception>
#include <string>

#ifdef HAS_OPENCV
#include <opencv2/opencv.hpp>
#endif

#include "Image.h"

using namespace std;


void onTrackbar(int, void*);


Image g_input;
Image g_output;
double g_alpha = 1.0;

const int g_alpha_slider_int_max = 100;
const double g_alpha_slider_double_max = 10;
int g_alpha_slider = 1;


//----------------------------
int main(int argc, char** argv)
//----------------------------
{
    try
    {

        if (argc == 3 || argc == 4 || argc == 5)
        {
            // Load the input image
            g_input = argv[1];

            // Retrieve alpha
            string argv_3 = argv[3];
            if (argc >= 4)
            {
                if (argv_3 != "-display")
                {
                    g_alpha = stof(argv_3);
                }
            }

            // Filter the image
            g_output = g_input.sharpen(g_alpha);

#ifdef HAS_OPENCV
            // Display the image
            if (argv_3 == "-display" || argc == 5)
            {
                cv::namedWindow("Sharpening", cv::WINDOW_AUTOSIZE); // Create Wind
ow
```

```
                cv::createTrackbar("Alpha", "Sharpening", &g_alpha_slider, g_alpha
_slider_int_max, onTrackbar);

                onTrackbar(g_alpha_slider, 0);

                cv::waitKey(0);
            }
#endif // HAS_OPENCV


            // Save the output
            g_output.save(argv[2]);
        }
        else
        {
            string error_message = string("Usage: ") + argv[0] + " input_image out
put_image [alpha] [-display]";
            throw error_message;
        }
    }
    catch (const exception& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e.what() << endl;
        return 1;
    }
    catch (const string& e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 2;
    }
    catch (const char* e)
    {
        cerr << "An error occured, see the message below." << endl;
        cerr << e << endl;
        return 3;
    }

    return 0;
}


//---------------------------
void onTrackbar(int, void*)
//---------------------------
{
    // Compute the value of alpha from the slider
    g_alpha = g_alpha_slider_double_max * g_alpha_slider / g_alpha_slider_int_max;

    // Sharpen
    g_output = g_input.sharpen(g_alpha);

    // Normalise for floating-point numbers
    Image display_image = g_output / 255.0;

    // Convert the data into an OpenCV Mat instance.
    cv::Mat cv_image(display_image.getHeight(), display_image.getWidth(), CV_32FC1
, (float*)display_image.getPixelPointer());
```

```
    // Display the image
    cv::imshow("Sharpening", cv_image);
}
```

test-operators.cxx

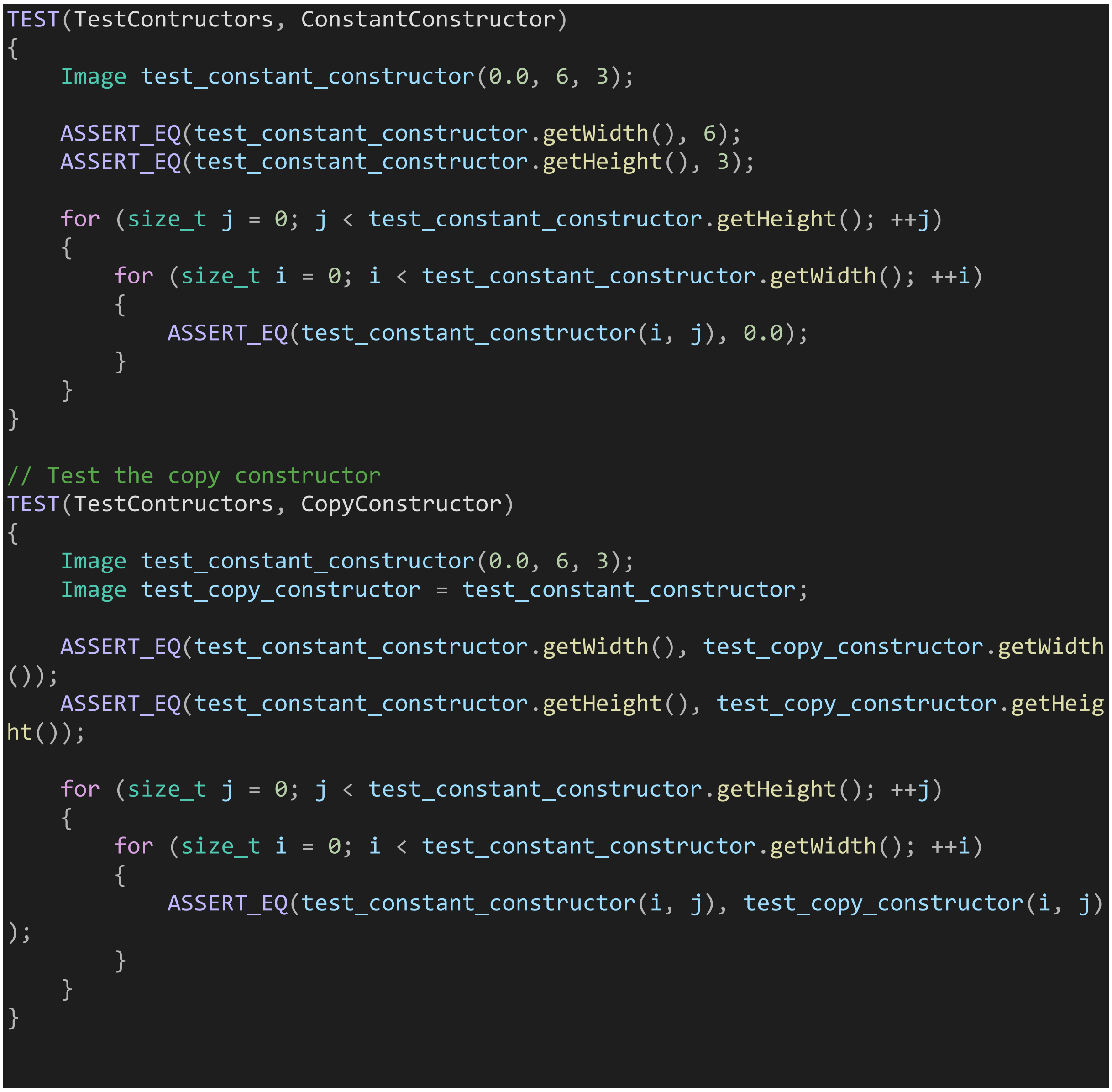
```
#include <iostream>

#include "Image.h"
#include "gtest/gtest.h"


using namespace std;
//*
// Test the default constructor
TEST(Operators, FloatImageOperators)
{
    Image input_image({0, 1, 2, 3, 4, 5, 6 , 7}, 4, 2);

    Image image_product = 2 * input_image;
    ASSERT_EQ(image_product.getWidth(), input_image.getWidth());
    ASSERT_EQ(image_product.getHeight(), input_image.getHeight());

    Image image_sum = 2 + input_image;
    ASSERT_EQ(image_sum.getWidth(), input_image.getWidth());
    ASSERT_EQ(image_sum.getHeight(), input_image.getHeight());

    cout << image_sum << endl;
    size_t k = 0;
    for (size_t j = 0; j < image_product.getHeight(); ++j)
    {
        for (size_t i = 0; i < image_product.getWidth(); ++i, ++k)
        {
            ASSERT_NEAR(image_product(i, j), k * 2, 1e-6);
            ASSERT_NEAR(image_sum(i, j), k + 2, 1e-6);
        }
    }
}
//*/

TEST(Operators, RMSEOperators)
{
    Image I1({ 1, 1, 1, 1, 1, 1 }, 2, 3);
    // I1.getRMSE(I1); <-- must be close to 0
    ASSERT_NEAR(I1.getRMSE(I1), 0.0, 1e-6);

    //*
    // I1.getRMSE(I1 * 3); must be close to sqrt(1/6 * 6 * (1 - 3)^2), i.e. 2
    ASSERT_NEAR(I1.getRMSE(I1 * 3), 2.0, 1e-5);

    //*/
    // I1.getRMSE(I1 * 9); must be close to sqrt(1/6 * 6 * (1 - 9)^2), i.e. 8
    ASSERT_NEAR(I1.getRMSE(I1 * 9), 8.0, 1e-5);
    //*/
}

TEST(Operators, ZNCCOperators)
{
    Image I1({ 1, 2, 3, 4, 5, 6 }, 2, 3);
    Image I2(!I1); // Negative of I1
    Image I3({ 6, 6, 6, 0, 0, 0 }, 2, 3); // A two-tone image

    // Same image
```



```
    // I1.getZNCC(I1); <-- must be close to 1
    ASSERT_NEAR(I1.getZNCC(I1), 1.0, 1e-5);

    // I1.getZNCC(10. + 4. * I1); <-- must be close to 1
    ASSERT_NEAR(I1.getZNCC(10. + 4. * I1), 1.0, 1e-5);

    // Negative image
    // I1.getZNCC(I2); <-- must be close to -1
    ASSERT_NEAR(I1.getZNCC(I2), -1.0, 1e-5);

    // I1.getZNCC(10. + 4. * I2); <-- must be close to -1
    ASSERT_NEAR(I1.getZNCC(10. + 4. * I2), -1.0, 1e-6);

    // Different image
    // I1.getZNCC(I3); <-- must be between -1 and 1
    double value1 = I1.getZNCC(I3);
    ASSERT_GT(value1, -1.0);
    ASSERT_LT(value1, 1.0);

    // I1.getZNCC(10. + 4. * I3); <-- must be between -1 and 1
    double value2 = I1.getZNCC(10. + 4. * I3);
    ASSERT_GT(value2, -1.0);
    ASSERT_LT(value2, 1.0);
}
```

test-constructors.cxx

```
#include <iostream>

#include "Image.h"
#include "gtest/gtest.h"


using namespace std;

// Test the default constructor
TEST(TestContructors, DefaultConstructor)
{
    Image test_default_constructor;

    ASSERT_EQ(test_default_constructor.getWidth(), 0);
    ASSERT_EQ(test_default_constructor.getHeight(), 0);
    EXPECT_TRUE(test_default_constructor.getPixelPointer() == NULL);
}

// Test the constructor from a C array
TEST(TestContructors, CArrayConstructor)
{
    float p_c_array[] = {1, 2, 3, 4, 5, 6, 7, 8};
    Image test_c_array_constructor(p_c_array, 4, 2);

    ASSERT_EQ(test_c_array_constructor.getWidth(), 4);
    ASSERT_EQ(test_c_array_constructor.getHeight(), 2);

    size_t k = 0;
    for (size_t j = 0; j < test_c_array_constructor.getHeight(); ++j)
    {
        for (size_t i = 0; i < test_c_array_constructor.getWidth(); ++i, ++k)
        {
            ASSERT_EQ(test_c_array_constructor(i, j), p_c_array[k]);
        }
    }
}

// Test the constructor from a C++ STL vector
TEST(TestContructors, CXXArrayConstructor)
{
    vector<float> p_cxx_array = {1, 2, 3, 4, 5, 6, 7, 8};
    Image test_cxx_array_constructor(p_cxx_array, 2, 4);

    ASSERT_EQ(test_cxx_array_constructor.getWidth(), 2);
    ASSERT_EQ(test_cxx_array_constructor.getHeight(), 4);

    size_t k = 0;
    for (size_t j = 0; j < test_cxx_array_constructor.getHeight(); ++j)
    {
        for (size_t i = 0; i < test_cxx_array_constructor.getWidth(); ++i, ++k)
        {
            ASSERT_EQ(test_cxx_array_constructor(i, j), p_cxx_array[k]);
        }
    }
}

// Test the constructor from a constant
```



```
TEST(TestContructors, ConstantConstructor)
{
    Image test_constant_constructor(0.0, 6, 3);

    ASSERT_EQ(test_constant_constructor.getWidth(), 6);
    ASSERT_EQ(test_constant_constructor.getHeight(), 3);

    for (size_t j = 0; j < test_constant_constructor.getHeight(); ++j)
    {
        for (size_t i = 0; i < test_constant_constructor.getWidth(); ++i)
        {
            ASSERT_EQ(test_constant_constructor(i, j), 0.0);
        }
    }
}

// Test the copy constructor
TEST(TestContructors, CopyConstructor)
{
    Image test_constant_constructor(0.0, 6, 3);
    Image test_copy_constructor = test_constant_constructor;

    ASSERT_EQ(test_constant_constructor.getWidth(), test_copy_constructor.getWidth
());
    ASSERT_EQ(test_constant_constructor.getHeight(), test_copy_constructor.getHeig
ht());

    for (size_t j = 0; j < test_constant_constructor.getHeight(); ++j)
    {
        for (size_t i = 0; i < test_constant_constructor.getWidth(); ++i)
        {
            ASSERT_EQ(test_constant_constructor(i, j), test_copy_constructor(i, j)
);
        }
    }
}
```

test-load-save.cxx

```
#include <iostream>

#include "Image.h"
#include "gtest/gtest.h"


using namespace std;

// Test the load method
TEST(LoadSave, SavePNG)
{
    Image input_image("tulips.png");
    input_image.save("output.jpg");

    Image output_image("output.jpg");

    ASSERT_EQ(output_image.getWidth(), 768);
    ASSERT_EQ(output_image.getHeight(), 512);
    ASSERT_EQ(output_image.getWidth() * output_image.getHeight(), 393216);

    ASSERT_NEAR(output_image.getMean(), 101.540, 2);
    ASSERT_NEAR(output_image.getStdDev(), 59.634, 2);
    ASSERT_NEAR(output_image.getMinValue(), 3, 2);
    ASSERT_NEAR(output_image.getMaxValue(), 242, 2);
}

// Test ASCII files
TEST(LoadSave, LoadSaveASCII)
{
    // Create a 3x2 image
    Image input_image({ 1, 2, 3, 4, 5, 6 }, 3, 2);

    // Save the image in a text file
    input_image.save("output.txt");

    // Load the text file
    Image output_image("output.txt");

    // Check the image size
    ASSERT_EQ(output_image.getWidth(), 3);
    ASSERT_EQ(output_image.getHeight(), 2);
    ASSERT_EQ(output_image.getWidth() * output_image.getHeight(), 6);

    // Check the image stats
    ASSERT_NEAR(output_image.getMean(), (1 + 2 + 3 + 4 + 5 + 6) / 6.0, 1.0 / 6.0);
    ASSERT_NEAR(output_image.getMinValue(), 1, 1.0 / 6.0);
    ASSERT_NEAR(output_image.getMaxValue(), 6, 1.0 / 6.0);

    // Check all the pixel values
    ASSERT_NEAR(output_image(0, 0), 1, 1.0 / 6.0);
    ASSERT_NEAR(output_image(1, 0), 2, 1.0 / 6.0);
    ASSERT_NEAR(output_image(2, 0), 3, 1.0 / 6.0);
    ASSERT_NEAR(output_image(0, 1), 4, 1.0 / 6.0);
    ASSERT_NEAR(output_image(1, 1), 5, 1.0 / 6.0);
    ASSERT_NEAR(output_image(2, 1), 6, 1.0 / 6.0);

}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)

PROJECT (ICP3038-img-class VERSION 0.2)


# Use C++ 11
set(CMAKE_CXX_STANDARD 11) # C++11
set(CMAKE_CXX_STANDARD_REQUIRED ON) # C++11 is required (i.e. not optional)
set(CMAKE_CXX_EXTENSIONS OFF) # without compiler extensions like gnu++11

# Find OpenCV

# Add where OpenCV might be installed (look in D: first, then in C:)
IF (WIN32)
    SET (CMAKE_PREFIX_PATH ${CMAKE_PREFIX_PATH} "C:/opencv/build")
ENDIF (WIN32)
SET (CMAKE_PREFIX_PATH ${CMAKE_PREFIX_PATH} "C:/opencv/build")


FIND_PACKAGE(OpenCV REQUIRED)
IF(OpenCV_FOUND)
    add_definitions(-DHAS_OPENCV)

    # If windows is used, copy the dlls into the project directory
    SET (CV_VERSION_STRING
${OpenCV_VERSION_MAJOR}${OpenCV_VERSION_MINOR}${OpenCV_VERSION_PATCH})
    IF (WIN32)
        IF ( ${OpenCV_VERSION_MAJOR} EQUAL 4)
            IF (EXISTS
"${OpenCV_DIR}/x64/vc15/bin/opencv_videoio_ffmpeg${CV_VERSION_STRING}_64.dll")
                FILE (COPY
"${OpenCV_DIR}/x64/vc15/bin/opencv_videoio_ffmpeg${CV_VERSION_STRING}_64.dll"
                      DESTINATION "${CMAKE_CURRENT_BINARY_DIR}/")
            ELSE ()
                         MESSAGE (WARNING
"opencv_videoio_ffmpeg${CV_VERSION_STRING}_64.dll is not in
${OpenCV_DIR}/x64/vc15/bin/, you have to make sure is it in the PATH or to copy it
manually in your project binary directory")
            ENDIF ()
        ELSE ()
            IF (EXISTS
"${OpenCV_DIR}/x64/vc15/bin/opencv_ffmpeg${CV_VERSION_STRING}_64.dll")
                FILE (COPY
"${OpenCV_DIR}/x64/vc15/bin/opencv_ffmpeg${CV_VERSION_STRING}_64.dll"
                      DESTINATION "${CMAKE_CURRENT_BINARY_DIR}/")
            ELSE ()
                         MESSAGE (WARNING
"opencv_ffmpeg${CV_VERSION_STRING}_64.dll is not in ${OpenCV_DIR}/x64/vc15/bin/,
you have to make sure is it in the PATH or to copy it manually in your project
binary directory")
            ENDIF ()
        ENDIF ()

        IF (EXISTS
"${OpenCV_DIR}/x64/vc15/bin/opencv_world${CV_VERSION_STRING}.dll")
            FILE (COPY
"${OpenCV_DIR}/x64/vc15/bin/opencv_world${CV_VERSION_STRING}.dll"
                  DESTINATION "${CMAKE_CURRENT_BINARY_DIR}/")
        ELSE ()
            MESSAGE (WARNING "opencv_world${CV_VERSION_STRING}.dll is not in
${OpenCV_DIR}/x64/vc15/bin/, you have to make sure is it in the PATH or to copy it
manually in your project binary directory")
        ENDIF ()

        IF (EXISTS
"${OpenCV_DIR}/x64/vc15/bin/opencv_world${CV_VERSION_STRING}d.dll")
            FILE (COPY
"${OpenCV_DIR}/x64/vc15/bin/opencv_world${CV_VERSION_STRING}d.dll"
                  DESTINATION "${CMAKE_CURRENT_BINARY_DIR}/")
        ELSE ()
            MESSAGE (WARNING "opencv_world${CV_VERSION_STRING}d.dll is not in
${OpenCV_DIR}/x64/vc15/bin/, you have to make sure is it in the PATH or to copy it
manually in your project binary directory")
        ENDIF ()
    ENDIF (WIN32)
ELSE(OpenCV_FOUND)
    MESSAGE(WARNING "OpenCV not found.")
ENDIF(OpenCV_FOUND)


# Build GoogleTest
INCLUDE(cmake/External_GTest.cmake)

# Enable unit testing
enable_testing()


# Compilation
ADD_EXECUTABLE(test-constructors
    include/Image.h
    src/Image.cxx
    src/test-constructors.cxx)

# Add dependency
ADD_DEPENDENCIES(test-constructors googletest)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(test-constructors PUBLIC include)
target_include_directories(test-constructors PUBLIC ${GTEST_INCLUDE_DIRS})

IF(OpenCV_FOUND)
    target_include_directories(test-constructors PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_directories(test-constructors PUBLIC ${GTEST_LIBS_DIR})
target_link_libraries(test-constructors ${GTEST_LIBRARIES} ${OpenCV_LIBS})

# Add the unit test
add_test (Constructors test-constructors)


# Compilation
ADD_EXECUTABLE(test-operators
    include/Image.h
    src/Image.cxx
    src/test-operators.cxx)
```

```
# Add dependency
ADD_DEPENDENCIES(test-operators googletest)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(test-operators PUBLIC include)
target_include_directories(test-operators PUBLIC ${GTEST_INCLUDE_DIRS})

IF(OpenCV_FOUND)
    target_include_directories(test-operators PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_directories(test-operators PUBLIC ${GTEST_LIBS_DIR})
target_link_libraries(test-operators ${GTEST_LIBRARIES} ${OpenCV_LIBS})

# Add the unit test
add_test (Operators test-operators)


# The documentation build is an option. Set it to ON by default
option(BUILD_DOC "Build documentation" ON)

# Check if Doxygen is installed
find_package(Doxygen)
if (DOXYGEN_FOUND)

    SET (PROJECT_NAME ICP3038-img-class)
    SET (PROJECT_DESCRIPTION "An Image class used in ICP3038 at Bangor
University")

    # set input and output files
    set(DOXYGEN_IN ${CMAKE_CURRENT_SOURCE_DIR}/docs/Doxyfile.in)
    set(DOXYGEN_OUT ${CMAKE_CURRENT_BINARY_DIR}/Doxyfile)

    # Configure the file
    configure_file(${DOXYGEN_IN} ${DOXYGEN_OUT} @ONLY)

    # Add a custom target
    add_custom_target( doc_doxygen ALL
        COMMAND ${DOXYGEN_EXECUTABLE} ${DOXYGEN_OUT}
        WORKING_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}
        COMMENT "Generating API documentation with Doxygen"
        VERBATIM )
else (DOXYGEN_FOUND)
    message(WARNING "Doxygen need to be installed to generate the doxygen
documentation")
endif (DOXYGEN_FOUND)

FILE(COPY ${CMAKE_CURRENT_SOURCE_DIR}/docs/tulips.png DESTINATION
${CMAKE_CURRENT_BINARY_DIR})


# Compilation
ADD_EXECUTABLE(test-load-save
    include/Image.h
    src/Image.cxx
    src/test-load-save.cxx)
```

```
# Add dependency
ADD_DEPENDENCIES(test-load-save googletest)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(test-load-save PUBLIC include)
target_include_directories(test-load-save PUBLIC ${GTEST_INCLUDE_DIRS})

IF(OpenCV_FOUND)
    target_include_directories(test-load-save PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_directories(test-load-save PUBLIC ${GTEST_LIBS_DIR})
target_link_libraries(test-load-save ${GTEST_LIBRARIES} ${OpenCV_LIBS})

# Add the unit test
add_test (Constructors test-load-save)


# Compilation
ADD_EXECUTABLE(contrastEnhancement
    include/Image.h
    src/Image.cxx
    src/contrastEnhancement.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(contrastEnhancement PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(contrastEnhancement PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(contrastEnhancement ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(negative
    include/Image.h
    src/Image.cxx
    src/negative.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(negative PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(negative PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(negative ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(blending
    include/Image.h
    src/Image.cxx
    src/blending.cxx)
```

```
# Add include directories
TARGET_INCLUDE_DIRECTORIES(blending PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(blending PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(blending ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(display
    include/Image.h
    src/Image.cxx
    src/display.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(display PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(display PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(display ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(gaussianFilter
    include/Image.h
    src/Image.cxx
    src/gaussianFilter.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(gaussianFilter PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(gaussianFilter PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(gaussianFilter ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(boxFilter
    include/Image.h
    src/Image.cxx
    src/boxFilter.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(boxFilter PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(boxFilter PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
```

```
target_link_libraries(boxFilter ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(laplacianFilter
    include/Image.h
    src/Image.cxx
    src/laplacianFilter.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(laplacianFilter PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(laplacianFilter PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(laplacianFilter ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(gradientMagnitude
    include/Image.h
    src/Image.cxx
    src/gradientMagnitude.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(gradientMagnitude PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(gradientMagnitude PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(gradientMagnitude ${OpenCV_LIBS})


# Compilation
ADD_EXECUTABLE(sharpen
    include/Image.h
    src/Image.cxx
    src/sharpen.cxx)

# Add include directories
TARGET_INCLUDE_DIRECTORIES(sharpen PUBLIC include)

IF(OpenCV_FOUND)
    target_include_directories(sharpen PUBLIC ${OpenCV_INCLUDE_DIRS})
ENDIF(OpenCV_FOUND)

# Add linkage
target_link_libraries(sharpen ${OpenCV_LIBS})
```