```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <opencv2/opencv.hpp>

////////////////////////////////////////////

using namespace cv;

//courtesy of stack overflow :)
namespace patch { //std not registering as namespace for
to_string(int)
    template <typename T> std::string to_string(const T& n) {
        std::ostringstream stm;
        stm << n;
        return stm.str();
    }
}

////////////////////////////////////////////

void display_image(const Mat&, std::string);

void save_image(const Mat&, std::string);

bool is_grayscale(const Mat&);

void color2grey(Mat*);

void hist_make(const Mat&, Mat*);

void enhance(Mat*, std::string);

////////////////////////////////////////////

int main(int argc, char** argv) {
    Mat image = imread(argv[1],1);
    int n = 1;
    std::string argstr = "nature.JPG";

    display_image(image, argstr);

    enhance(&image, argstr);

    return 0;
}

////////////////////////////////////////////
```

```
void display_image(const Mat& image, std::string imname) {
    namedWindow(imname, WINDOW_AUTOSIZE);
    imshow(imname, image);
    waitKey(0);
}

void save_image(const Mat& image, std::string imname) {
    imwrite(imname, image); //save grayscale to file
    std::cout << imname << std::endl;
}

bool is_grayscale(const Mat& image) {
    if(image.type() == CV_8UC1) { //CV_8UC1 is enumerated 8 bit single
channel unsigned matrix
        return true;
    } else {
        return false;
    }
}

void color2grey(Mat* image) { //using luminosity method
    Mat grayscale = Mat(image->rows, image->cols, CV_8UC1); //Mat
constructor
    for(int r = 0; r < image->rows; r++) {
        for(int c = 0; c < image->cols; c++) { //Each pixel is an
array of 3. We weight each color based on human eye sensitivity.
            int tmp = (image->at<Vec3b>(r, c)[0] * .11) + (image-
>at<Vec3b>(r, c)[1] * .59)+ (image->at<Vec3b>(r, c)[2] * .33);
            grayscale.at<uchar>(r,c) = tmp;
        }
    }
    *image = grayscale;
}

void hist_make(const Mat& image, Mat* hist) { //with special thanks to
stack overflow!
    int bins = 256;
    float range[] = {0, 256};
    const float* hist_range = {range};

    calcHist(&image, 1, 0, Mat(), *hist, 1, &bins, &hist_range, true,
false);

    int hist_w = 512;
    int hist_h = 400;
    int bin_w = cvRound( (double) hist_w/bins );

    Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );

    /// Normalize the result to [ 0, histImage.rows ]
```

```
    normalize(*hist, *hist, 0, histImage.rows, NORM_MINMAX, -1,
Mat() );

    /// Draw for each channel
    for( int i = 1; i < bins; i++ ) {
        line( histImage, Point( bin_w*(i-1), hist_h - cvRound(hist-
>at<float>(i-1)) ) ,
                Point( bin_w*(i), hist_h - cvRound(hist-
>at<float>(i)) ),
                Scalar( 255, 0, 0), 2, 8, 0
            );
    }

    *hist = histImage;
}

void enhance(Mat* image, std::string imname) {
    if(!is_grayscale(*image)) {
        color2grey(image);
        //std::cout << is_grayscale(*image) << std::endl;
    }

    Mat OG_hist, NEW_hist;

    hist_make(*image, &OG_hist);
    display_image(OG_hist, "OG_hist.png");
    save_image(OG_hist, imname.substr(0,imname.length()-4) +
"_OG_hist.png");

    equalizeHist(*image, *image);
    display_image(*image, "equalized_" + imname);
    save_image(*image, "equalized_" + imname);

    hist_make(*image, &NEW_hist);
    display_image(NEW_hist, "NEW_hist.png");
    save_image(NEW_hist, imname.substr(0,imname.length()-4) +
"_NEW_hist.png");
}
```