```cpp
#include <iostream>

#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>

using namespace cv;

void filter(Mat* input, Mat* result) {
  Size s = input->size();
  long h, w;
  long sum;

  //std::cout << "Input type was : " << input->type() << std::endl;
  uint8_t out[s.height][s.width];

  std::cout << s.height << " " << s.width << std::endl;

  for (w=0; w<s.width; w++){ //loop over the image,
    for(h=0; h<s.height; h++){
      sum = 0;
      std::vector<uint8_t> median;

      for (int _x = -1; _x < 2; ++_x) //for every pixel, loop over every pixel in a 3x3 kernel
      {
        for (int _y = -1; _y < 2; ++_y)
        {
          int idx_y = h + _y; //sum the incrementor with the kernel's, so we can identify borders
          int idx_x = w + _x;

          if (idx_x < 0 || idx_x > s.width) //the kernel goes outside of the image, therefore break and ignore that pixel.
            break;

          if (idx_y < 0 || idx_y > s.height)
            break;

          median.push_back(input->at<uint8_t>(idx_y,idx_x)); // Add all the pixels from the kernel into a vector
        }
      }
      std::sort(std::begin(median), std::end(median)); //sort all pixel values from high to low

      for (auto it = median.begin(); it != median.end(); ++it) {
        sum = median.at(median.size()/2); //The pixel at the y,x coordinate is now the median from our 3x3 sliding window
      }
      out[h][w]=(uint8_t)sum;
    }
  }
  // this did not work, since the out array was never copied to memory, causing image data pointing to nowhere!
  //result = Mat(s.height, s.width, CV_8U, out);
  // Instead, the data from out needs to be copied directly to Mat result
```

```
with the correct size
  std::memcpy(result->data, out, s.height*s.width*sizeof(uint8_t));
}

void gammaCorrection(Mat* input, Mat* output, float gamma) {
  Size s = input->size();
  long h, w;
  float sum;

  std::cout << "Correcting gamma" << std::endl;
  uint8_t out[s.height][s.width];
  uint8_t lut[256];

  for (int i = 0; i < 256; i++) { //create a lookup table, with the gamma
correction curve.
    lut[i] = saturate_cast<uint8_t>(pow((float)(i / 255.0), gamma) * 255.0f);
//saturate cast negative values to 0, and higher values to 255 (uint8_t or
unsigned char)
    //std::cout << unsigned(lut[i]) << " "; //print the function for testing.
cout prints uint8_t as chars so we cast it.
  }


  for (w=0; w<s.width; w++){ //loop over the image,
    for(h=0; h<s.height; h++){
      sum = lut[(input->at<uint8_t>(h,w))]; //the original output value will
be scaled to the value in the LUT.
      out[h][w]=(uint8_t)sum;
    }
  }
  std::memcpy(output->data, out, s.height*s.width*sizeof(uint8_t)); //copy
our standard 2D array to a new buffer that OpenCV understands
}


int main() {
  // Read the image (in BGR)
    Mat img = imread("pixerror.png", IMREAD_COLOR);
    if(img.empty())
    {
        std::cout << "Could not read the image: " << std::endl;
        return 1;
    }
    Size imgsize = img.size();

    // Split the image into 3 new images for blue, green and red.
    std::cout << "Splitting channels: " << std::endl;
  Mat bands[3];
  split(img, bands);

  Mat bandsFiltered[3];
  Mat bandsCorrected[3];
  bandsFiltered[0] = Mat(imgsize.height, imgsize.width, CV_8U);
  bandsFiltered[1] = Mat(imgsize.height, imgsize.width, CV_8U);
  bandsFiltered[2] = Mat(imgsize.height, imgsize.width, CV_8U);

  bandsCorrected[0] = Mat(imgsize.height, imgsize.width, CV_8U);
  bandsCorrected[1] = Mat(imgsize.height, imgsize.width, CV_8U);
  bandsCorrected[2] = Mat(imgsize.height, imgsize.width, CV_8U);
```

```
  filter(&bands[0],&bandsFiltered[0]); //filter all channels from noise individually
  filter(&bands[1],&bandsFiltered[1]);
  filter(&bands[2],&bandsFiltered[2]);

  gammaCorrection(&bandsFiltered[0],&bandsCorrected[0],0.33);
  gammaCorrection(&bandsFiltered[1],&bandsCorrected[1],0.33);
  gammaCorrection(&bandsFiltered[2],&bandsCorrected[2],0.33);

  Mat merged;
  std::vector<Mat> channels = {bandsCorrected[0],bandsCorrected[1],bandsCorrected[2]};
  merge(channels, merged);

    // Display the image until q is pressed
    std::cout << "Displaying result: " << std::endl;
    imshow("Display window", bands[0]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bands[1]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bands[2]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bandsFiltered[0]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bandsFiltered[1]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bandsFiltered[2]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", merged);
    waitKey(0); // Wait for a keystroke in the window
    return 0;
}
```