Image Processing Application

PROGRAMMING-2-PROJECT 2023-24

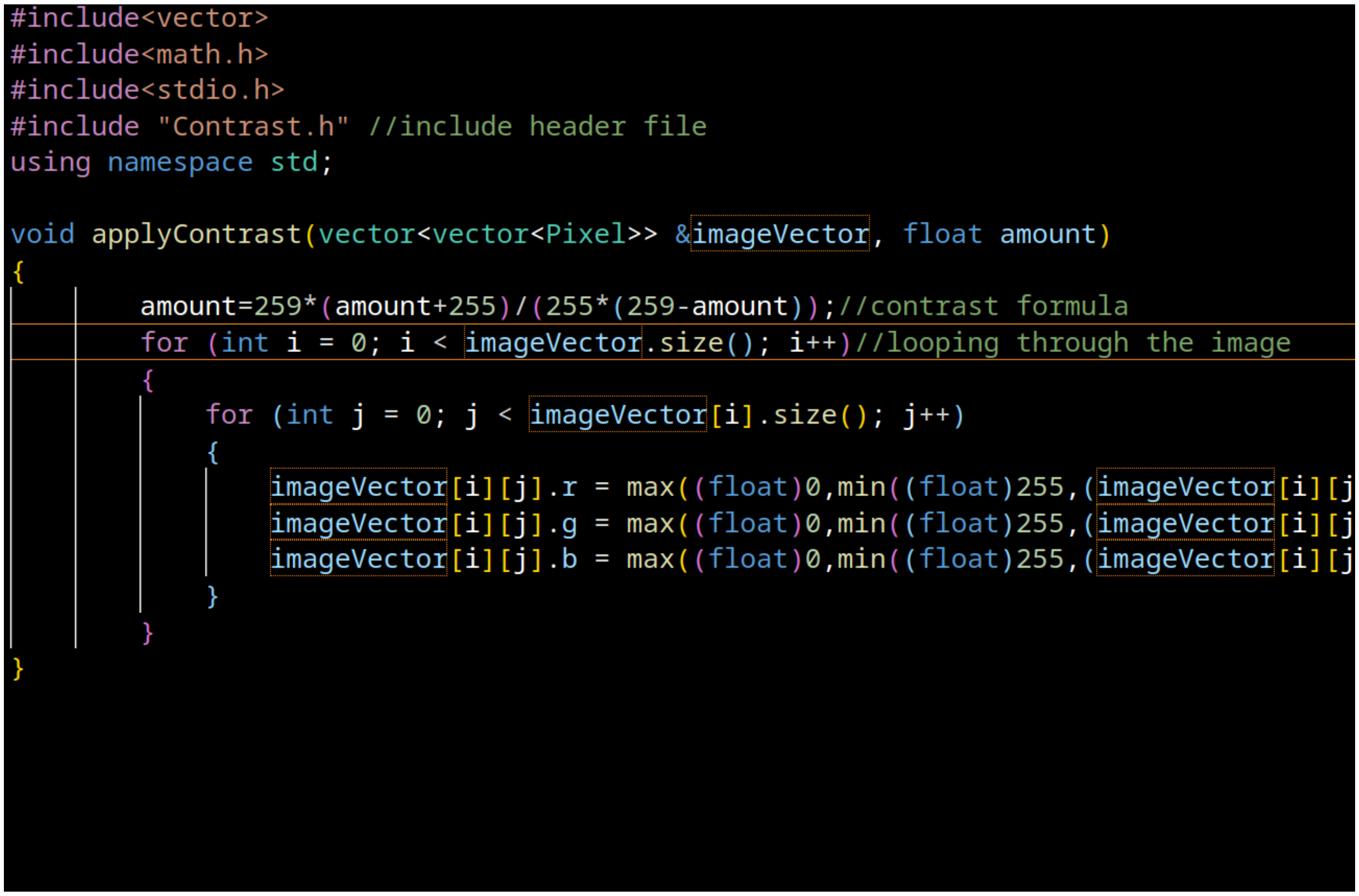
Motivation behind the Project:

This project aims to create an image processing application which caters to the needs of people who are interested in their images being more according to their tastes and preferences.

Features:

• Image processing model contains hue saturation, apply contrast, gaussian blur, image flip and rotate
• It is provided with a logging service where all your previous made changes will be saved and then you can access them clearly.
• There are several other features added in the backend.

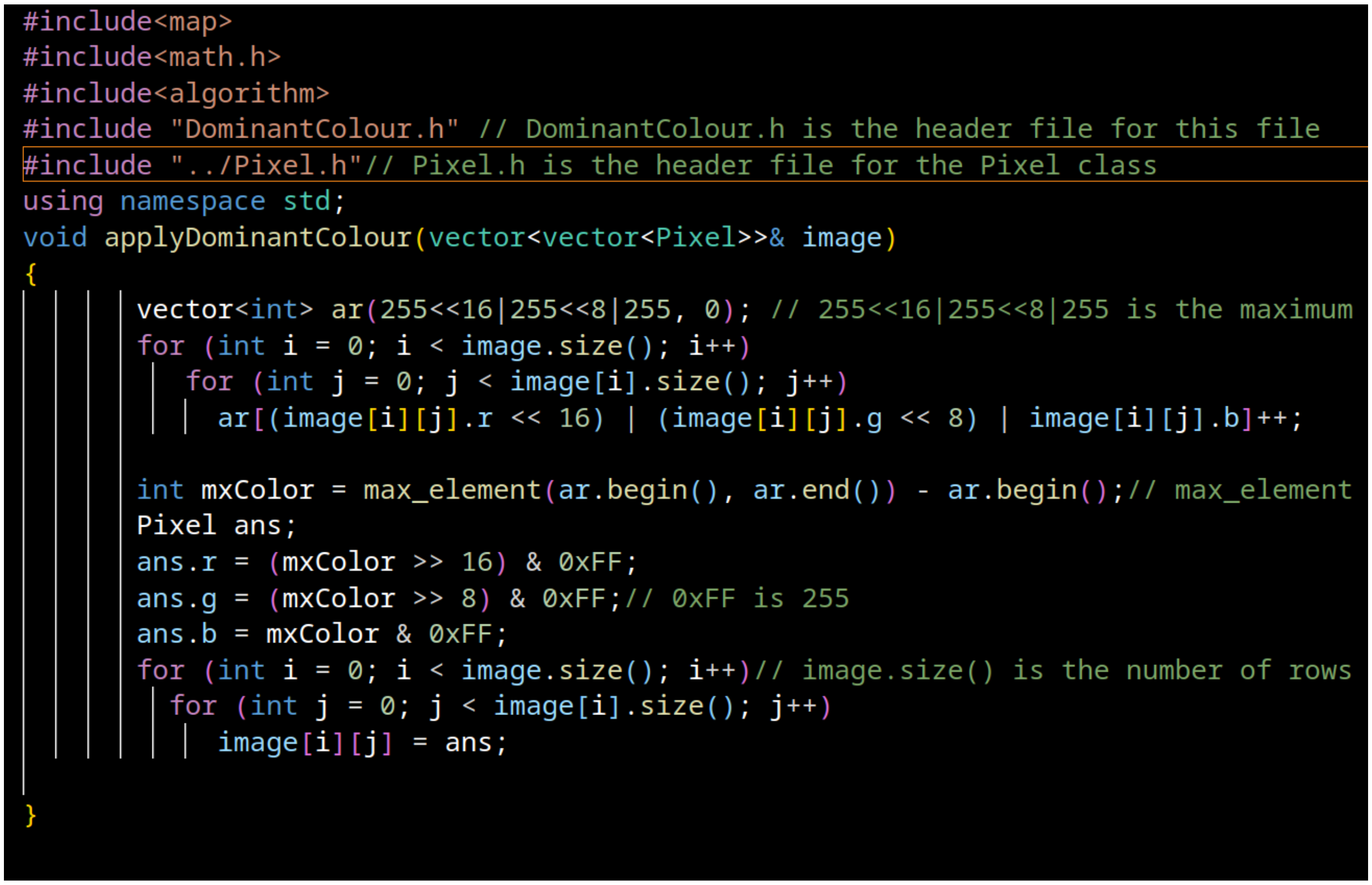
Screenshots and explanations of the backend:



```
#include<vector>
#include<math.h>
#include<stdio.h>
#include "Contrast.h" //include header file
using namespace std;

void applyContrast(vector<vector<Pixel>> &imageVector, float amount)
{
        amount=259*(amount+255)/(255*(259-amount));//contrast formula
        for (int i = 0; i < imageVector.size(); i++)//looping through the image
        {
            for (int j = 0; j < imageVector[i].size(); j++)
            {
                imageVector[i][j].r = max((float)0,min((float)255,(imageVector[i][j
                imageVector[i][j].g = max((float)0,min((float)255,(imageVector[i][j
                imageVector[i][j].b = max((float)0,min((float)255,(imageVector[i][j
            }
        }
}
```

This is the contrast file. Basically throughout our backend we are dividing our images into small pixels and matrices and then performing operations on them so that it becomes easier to understand. All the formulas used are similar to the ones given in the documentation stack.

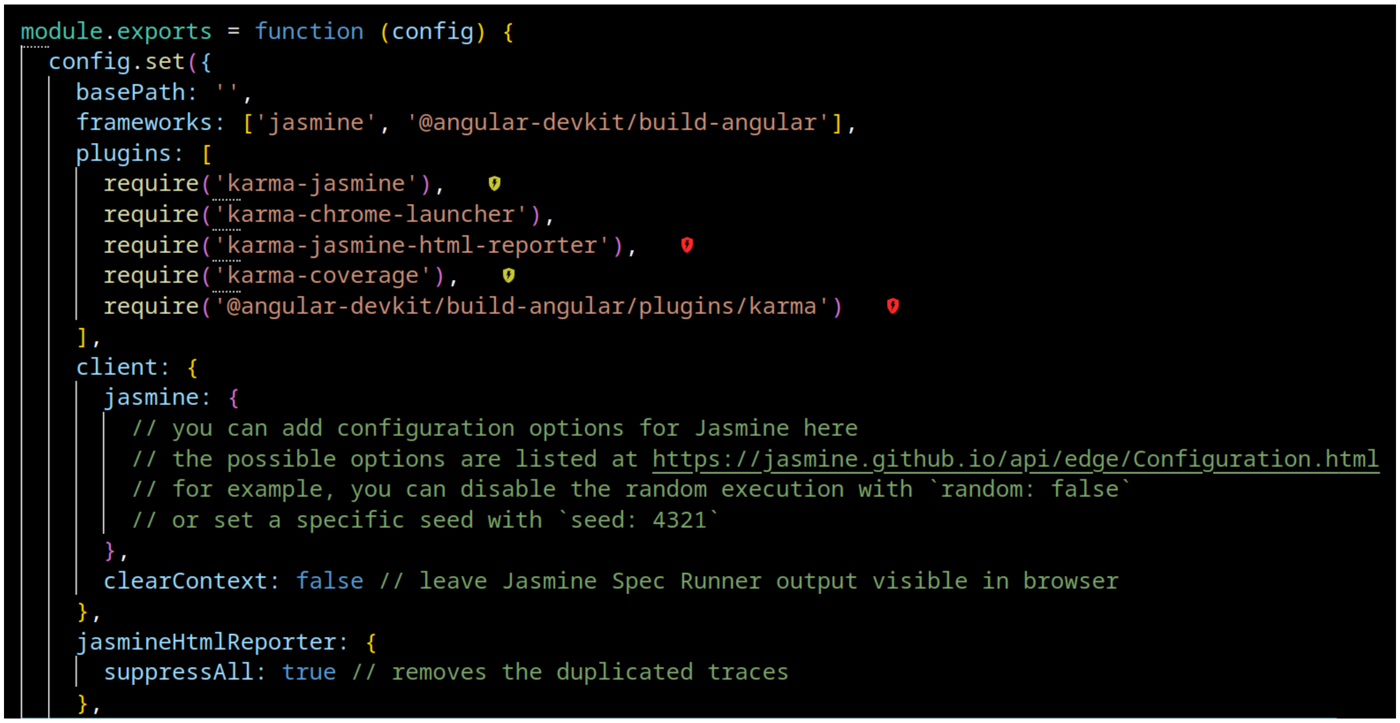


```
#include<map>
#include<math.h>
#include<algorithm>
#include "DominantColour.h" // DominantColour.h is the header file for this file
#include "../Pixel.h"// Pixel.h is the header file for the Pixel class
using namespace std;
void applyDominantColour(vector<vector<Pixel>>& image)
{
            vector<int> ar(255<<16|255<<8|255, 0); // 255<<16|255<<8|255 is the maximum
            for (int i = 0; i < image.size(); i++)
               for (int j = 0; j < image[i].size(); j++)
                 ar[(image[i][j].r << 16) | (image[i][j].g << 8) | image[i][j].b]++;

            int mxColor = max_element(ar.begin(), ar.end()) - ar.begin();// max_element
            Pixel ans;
            ans.r = (mxColor >> 16) & 0xFF;
            ans.g = (mxColor >> 8) & 0xFF;// 0xFF is 255
            ans.b = mxColor & 0xFF;
            for (int i = 0; i < image.size(); i++)// image.size() is the number of rows
              for (int j = 0; j < image[i].size(); j++)
                 image[i][j] = ans;

}
```

Like this in certain programmes we have basically divided the image into small pixels and then applied individual colors as it was easier to apply this way.



```
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      jasmine: {
        // you can add configuration options for Jasmine here
        // the possible options are listed at https://jasmine.github.io/api/edge/Configuration.html
        // for example, you can disable the random execution with `random: false`
        // or set a specific seed with `seed: 4321`
      },
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    jasmineHtmlReporter: {
      suppressAll: true // removes the duplicated traces
    },
```

The frontend was done in Angular which integrated all the c++ files and java to recompile with the backend so that the image processing application could have been running.
At the end the program is working successfully and the effects are being successfully handled.

THANK YOU FOR VIEWING OUR PROJECT