hotPhotoLab Program Structure

Description

The program is decomposed into multiple modules and header files for better organization and maintainability. The structure is as follows:

Modules and Header Files

- **PhotoLab.c**
  Contains the `main()` function, `PrintMenu()`, and `AutoTest()`.
- **FileIO.c**
  Contains the function definitions for `LoadImage()` and `SaveImage()`.
- **FileIO.h**
  Header file for FileIO.c with function declarations for `LoadImage()` and `SaveImage()`.
- **Constants.h**
  Defines constants used throughout the program.
- **DIPs.c**
  Contains function definitions for basic Digital Image Processing (DIP) operations:
  - `BlackNWhite()`
  - `Negative()`
  - `ColorFilter()`
  - `Edge()`
  - `Shuffle()`
  - `VFlip()`
  - `HMirror()`
  - `Pixelate()`
- **DIPs.h**
  Header file for DIPs.c with function declarations for the DIP operations.
- **Advanced.c**
  Contains function definitions for advanced DIP operations:
  - `FishEye()`
  - `Rotate()`
  - `Posterize()`
  - `MotionBlur()`
- **Advanced.h**
  Header file for Advanced.c with function declarations for the advanced DIP operations.
- **Makefile**
  The makefile used for building the program.

Makefile

makefile
CopyEdit

```makefile
# Makefile for PhotoLab

all: PhotoLab PhotoLabTest

clean:
    rm -f *.o *.a PhotoLab PhotoLabTest

PhotoLab.o: PhotoLab.c FileIO.h Advanced.h DIPs.h Constants.h
    gcc -Wall -std=c11 -c PhotoLab.c -o PhotoLab.o

PhotoLabTest.o: PhotoLab.c FileIO.h Advanced.h DIPs.h Constants.h
    gcc -Wall -DDEBUG -std=c11 -c PhotoLab.c -o PhotoLabTest.o

FileIO.o: FileIO.c FileIO.h Constants.h
    gcc -Wall -std=c11 -c FileIO.c -o FileIO.o

Advanced.o: Advanced.c Advanced.h Constants.h
    gcc -Wall -std=c11 -c Advanced.c -o Advanced.o

DIPs.o: DIPs.c DIPs.h Constants.h
    gcc -Wall -std=c11 -c DIPs.c -o DIPs.o

PhotoLab: PhotoLab.o FileIO.o libFilter.a
    gcc PhotoLab.o FileIO.o -L. -lFilter -lm -o PhotoLab

PhotoLabTest: FileIO.o PhotoLabTest.o libFilter.a
    gcc -g PhotoLabTest.o FileIO.o -L. -lFilter -lm -o PhotoLabTest

libFilter.a: DIPs.o Advanced.o
    ar rc libFilter.a DIPs.o Advanced.o
    ranlib libFilter.a
```

Advanced DIP Functions

FishEye()

Applies a fisheye effect to the image.

c

CopyEdit

```
void FishEye(unsigned char R[WIDTH][HEIGHT], unsigned char
G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT], double base_factor,
double k, double scaling_factor) {
    unsigned char R_out[WIDTH][HEIGHT], G_out[WIDTH][HEIGHT],
B_out[WIDTH][HEIGHT];
    int center_x = WIDTH / 2;
    int center_y = HEIGHT / 2;

    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            double dx = (x - center_x) / (double)center_x;
            double dy = (y - center_y) / (double)center_y;
            double radius = sqrt(dx * dx + dy * dy);

            double distortion = (1.0 + k * radius * radius);
            double theta = atan2(dy, dx);
            double new_radius = (radius * base_factor) / (distortion *
scaling_factor);
            new_radius = fmin(new_radius, 1.0);

            int x_src = floor(center_x + (new_radius * cos(theta) *
center_x));
            int y_src = floor(center_y + (new_radius * sin(theta) *
center_y));

            if (x_src >= 0 && x_src < WIDTH && y_src >= 0 && y_src <
HEIGHT) {
                R_out[x][y] = R[x_src][y_src];
                G_out[x][y] = G[x_src][y_src];
                B_out[x][y] = B[x_src][y_src];
            } else {
                R_out[x][y] = 0;
                G_out[x][y] = 0;
                B_out[x][y] = 0;
            }
        }
    }

    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            R[x][y] = R_out[x][y];
            G[x][y] = G_out[x][y];
            B[x][y] = B_out[x][y];
        }
    }
}
```

Posterize()

Reduces the number of bits for each color channel.

c

CopyEdit

```
void Posterize(unsigned char R[WIDTH][HEIGHT], unsigned char
G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT], unsigned int rbits,
unsigned int gbits, unsigned int bbits) {
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            R[x][y] = (R[x][y] & (~((1 << rbits) - 1))) | ((1 <<
(rbits - 1)) - 1);
            G[x][y] = (G[x][y] & (~((1 << gbits) - 1))) | ((1 <<
(gbits - 1)) - 1);
            B[x][y] = (B[x][y] & (~((1 << bbits) - 1))) | ((1 <<
(bbits - 1)) - 1);
        }
    }
}
```

Rotate()

Rotates the image by a specified angle.

c

CopyEdit

```
void Rotate(unsigned char R[WIDTH][HEIGHT], unsigned char
G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT], double Angle, double
ScaleFactor, int CenterX, int CenterY) {
    double theta = -Angle * 2 * PI / 360.0;
    unsigned char R_temp[WIDTH][HEIGHT], G_temp[WIDTH][HEIGHT],
B_temp[WIDTH][HEIGHT];

    // Copy original image to temp arrays
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            R_temp[x][y] = R[x][y];
            G_temp[x][y] = G[x][y];
            B_temp[x][y] = B[x][y];
        }
    }

    // Create new image
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            int new_x = (int)(((cos(theta) / ScaleFactor) * (x -
CenterX)) - ((sin(theta) / ScaleFactor) * (y - CenterY)) + CenterX);
            int new_y = (int)(((sin(theta) / ScaleFactor) * (x -
CenterX)) + ((cos(theta) / ScaleFactor) * (y - CenterY)) + CenterY);

            if (new_x >= 0 && new_x < WIDTH && new_y >= 0 && new_y <
HEIGHT) {
                R[x][y] = R_temp[new_x][new_y];
                G[x][y] = G_temp[new_x][new_y];
                B[x][y] = B_temp[new_x][new_y];
            } else {
                // Set to black
                R[x][y] = 0;
                G[x][y] = 0;
                B[x][y] = 0;
            }
        }
    }
}
```

MotionBlur()

Applies a horizontal motion blur effect.

c
CopyEdit

```
void MotionBlur(int BlurAmount, unsigned char R[WIDTH][HEIGHT],
unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]) {
    unsigned char R_temp[WIDTH][HEIGHT], G_temp[WIDTH][HEIGHT],
B_temp[WIDTH][HEIGHT];

    // Copy original image to temp arrays
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            R_temp[x][y] = R[x][y];
            G_temp[x][y] = G[x][y];
            B_temp[x][y] = B[x][y];
        }
    }

    // Apply horizontal motion blur
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            double sumR = R_temp[x][y] * 0.5; // 50% weight for the
original pixel
            double sumG = G_temp[x][y] * 0.5;
            double sumB = B_temp[x][y] * 0.5;
            double totalWeight = 0.5; // Start with the original
pixel's weight

            int availablePixels = (x + BlurAmount <= WIDTH) ?
BlurAmount : (WIDTH - 1 - x); // Adjust for edge cases

            for (int i = 1; i <= availablePixels; i++) {
                double weight = 0.5 / availablePixels;  // Adjust
remaining 50% weight based on available pixels
                sumR += R_temp[x + i][y] * weight;
                sumG += G_temp[x + i][y] * weight;
                sumB += B_temp[x + i][y] * weight;
                totalWeight += weight;
            }

            // Assign the weighted values
            R[x][y] = (unsigned char)(sumR / totalWeight);
            G[x][y] = (unsigned char)(sumG / totalWeight);
            B[x][y] = (unsigned char)(sumB / totalWeight);
        }
    }
}
```

DEBUG Macro

The `DEBUG` macro is used to enable or disable debugging features in the program.

Main Function

c

CopyEdit

```
#ifdef DEBUG
    AutoTest(R, G, B);
#else
    // Normal execution
#endif
```

AutoTest Function

c

CopyEdit

```
#ifdef DEBUG
    printf("Negative tested!\n\n");
#endif
```

Problems Encountered

Over 20 submissions were required to get the autograder to execute correctly. The issue was caused by incorrect file naming (DIPS.H and DIPS.c instead of DIPs.h and DIPs.c).