Report of task 1:

The code:

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

typedef long long ll;

#define WIDTH       1192

#define HEIGHT      500

#define f(i,n)      for (ll i=0;i<n;i++)

#define wo(a)       while(!feof(a))

void flipHorizontal(uint8\_t image[HEIGHT][WIDTH]){

    f(i,HEIGHT) f(j,WIDTH/2) {

        uint8\_t temp = image[i][j];

        image[i][j]=image[i][WIDTH-1-j];

        image[i][WIDTH-1-j] = temp;

    }

}

int main() {

    FILE \*inputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/100dollars.tif", "rb");

    FILE \*outputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/flipped\_image.tif", "wb");

    if (inputFile == NULL) {

        printf("Error opening input image file.\n");

        return 1;

    }

    if (outputFile == NULL) {

        printf("Error opening output image file.\n");

        return 1;

    }

    uint8\_t metadata[8];

    fread(&metadata, sizeof(uint8\_t), 8, inputFile);

    fwrite(&metadata, sizeof(uint8\_t), 8, outputFile);

    uint8\_t image[HEIGHT][WIDTH];

    fread(&image, sizeof(uint8\_t),HEIGHT\*WIDTH, inputFile);

    flipHorizontal(image);

    fwrite(&image, sizeof(uint8\_t), HEIGHT\*WIDTH, outputFile);

    uint8\_t trailing ;

    wo(inputFile){

        fread(&trailing, sizeof(uint8\_t), 1, inputFile);

        fwrite(&trailing, sizeof(uint8\_t), 1, outputFile);

    }

    fclose(inputFile);

    fclose(outputFile);

    printf("Image flipped and saved successfully.\n");

    return 0;

}

Explanation of the code:

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

- These lines include necessary header files for standard input-output operations (stdio.h), dynamic memory allocation (stdlib.h), and fixed-width integer types (stdint.h).

typedef long long ll;

- This line defines a type alias ll for long long integer type.

#define WIDTH       1192

#define HEIGHT      500

- These lines define macros for the width and height of the image.

#define f(i,n)      for (ll i=0;i<n;i++)

- This line defines a macro f for a simple for loop, which takes two arguments i (loop variable) and n (loop bound).

#define wo(a)       while(!feof(a))

- This line defines a macro wo for a loop that continues until the end of the file a is reached.

void flipHorizontal(uint8\_t image[HEIGHT][WIDTH]

- This line defines a function named flipHorizontal that takes a 2D array image as its argument. The array represents an image with the specified height and width.

f(i,HEIGHT) f(j,WIDTH/2)

- These lines start a nested loop to iterate over each pixel in the image. The outer loop iterates over the rows (HEIGHT) and the inner loop iterates over half of the columns (WIDTH/2). This loop structure ensures that only half of the columns are swapped during horizontal flipping, as flipping the entire width would result in flipping the image back to its original orientation.

uint8\_t temp = image[i][j];

        image[i][j]=image[i][WIDTH-1-j];

        image[i][WIDTH-1-j] = temp;

- These lines swap the pixel values between the current column (j) and its corresponding column from the end (WIDTH-1-j). This effectively flips the image horizontally.

int main()

- This line starts the main function, which is the entry point of the program.

FILE \*inputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/100dollars.tif", "rb");

FILE \*outputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/flipped\_image.tif", "wb");

- These lines open the input and output image files in binary mode for reading ("rb") and writing ("wb") respectively.

if (inputFile == NULL) {

        printf("Error opening input image file.\n");

        return 1;

    }

    if (outputFile == NULL) {

        printf("Error opening output image file.\n");

        return 1;

    }

- These lines check if the input and output file pointers (inputFile and outputFile) are NULL, which would indicate an error in opening the files. If either file fails to open, an error message is printed, and the program returns 1, indicating an error.

    uint8\_t metadata[8];

    fread(&metadata, sizeof(uint8\_t), 8, inputFile);

    fwrite(&metadata, sizeof(uint8\_t), 8, outputFile);

- These lines read the first 8 bytes (metadata) from the input file and write them to the output file. This is done to preserve any metadata or header information present in the input image file.

uint8\_t image[HEIGHT][WIDTH];

    fread(&image, sizeof(uint8\_t),HEIGHT\*WIDTH, inputFile);

- This line declares a 2D array image to store the pixel values of the image. It reads the pixel data from the input file into this array, with each pixel represented by a uint8\_t value. The total number of pixels read is HEIGHT \* WIDTH.

flipHorizontal(image);

- This line calls the flipHorizontal function to flip the image horizontally. The function modifies the image array in place.

fwrite(&image, sizeof(uint8\_t), HEIGHT\*WIDTH, outputFile);

- This line writes the modified pixel data (the flipped image) from the image array to the output file. It writes HEIGHT \* WIDTH bytes of data, each of size sizeof(uint8\_t).

    uint8\_t trailing ;

    wo(inputFile){

        fread(&trailing, sizeof(uint8\_t), 1, inputFile);

        fwrite(&trailing, sizeof(uint8\_t), 1, outputFile);

    }

- These lines copy any remaining data from the input file to the output file. The wo macro defines a loop that continues until the end of the input file is reached (feof(inputFile) returns non-zero). Inside the loop, one byte is read from the input file and then immediately written to the output file. This ensures that any trailing data in the input file is preserved in the output file.

    fclose(inputFile);

    fclose(outputFile);

- These lines close both the input and output files once all operations are completed. This is important for releasing system resources associated with the file handles.

The justification why of the code works:

The provided code effectively flips an image horizontally and saves the result to a new file. Here's why the code works:

1. Function flipHorizontal(): This function iterates through each row of the image (HEIGHT) and swaps the pixels from left to right, effectively flipping the image horizontally. It achieves this by traversing only up to half of the width (WIDTH/2), ensuring that each pixel is swapped only once.

2. Main Function:

- File Handling: The code opens the input image file in binary read mode ("rb") and the output image file in binary write mode ("wb"). If either file fails to open, it prints an error message and returns 1, indicating an error.

- Metadata Handling: The first 8 bytes of the input image (metadata) are read and written to the output image, ensuring that any essential information at the beginning of the file is preserved.

- Image Data Processing: The pixel data of the image is read into a 2D array image using fread(). Then, the flipHorizontal() function is called to horizontally flip the image.

- Writing to Output File: The flipped image data is written to the output file using fwrite(). The trailing data, which comes after the image data in the input file, is also copied to the output file using a while loop until the end of the input file is reached.

- File Closure and Success Message: Finally, the input and output files are closed, and a success message is printed to indicate that the image has been successfully flipped and saved.

The usage of uint8\_t variable type in this code is justified for representing pixel values. Here's why it's appropriate:

- Pixel Values: Images typically represent pixel color or intensity using a single byte. uint8\_t, being an unsigned 8-bit integer, perfectly matches this requirement, allowing values from 0 to 255 to represent pixel intensity.

- Memory Efficiency: With each pixel requiring only one byte, uint8\_t ensures memory efficiency, crucial for minimizing memory footprint, especially in large images.

- Compatibility: uint8\_t is guaranteed to be exactly 8 bits wide, ensuring consistency across platforms and compilers. This enhances code portability and avoids potential issues related to data size and representation.

In summary, the code effectively flips the image horizontally and uses uint8\_t for representing pixel values due to its compatibility, memory efficiency, and adherence to standard image formats.

Input:



Output:



Report of task 2:

The code:

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

typedef long long ll;

#define WIDTH       1192

#define HEIGHT      500

#define f(i,n)      for (ll i=0;i<n;i++)

#define f1(i,n)     for (ll i=7;i<n-7;i++)

#define f2(i,n)     for (ll i=-n;i<=n;i++)

#define wo(a)       while(!feof(a))

void blur(uint8\_t image[HEIGHT][WIDTH], int redi){

    uint8\_t new\_img[HEIGHT][WIDTH];

    f(i, HEIGHT) {

        f(j, WIDTH) new\_img[i][j] = image[i][j];

    }

    f(i, HEIGHT) f(j, WIDTH) {

        int sum = 0, cnt = 0;

        f2(di,redi) f2(dj,redi) {

            ll ddi = i + di,ddj = j + dj;

            if (ddi >= 0 && ddi < HEIGHT && ddj >= 0 && ddj < WIDTH) {

                sum += new\_img[ddi][ddj];

                cnt++;

            }

        }

        image[i][j] = sum / cnt;

    }

}

int main() {

    FILE \*inputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/100dollars.tif", "rb");

    FILE \*outputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/blurred\_image.tif", "wb");

    if (inputFile == NULL) {

        printf("Error opening input image file.\n");

        return 1;

    }

    if (outputFile == NULL) {

        printf("Error opening output image file.\n");

        return 1;

    }

    uint8\_t metadata[8];

    fread(&metadata, sizeof(uint8\_t), 8, inputFile);

    fwrite(&metadata, sizeof(uint8\_t), 8, outputFile);

    uint8\_t image[HEIGHT][WIDTH];

    fread(&image, sizeof(uint8\_t),HEIGHT\*WIDTH, inputFile);

    int r;

    printf("Enter radius for your blurring: ");

    scanf("%d",&r);

    blur(image,r);

    fwrite(&image, sizeof(uint8\_t), HEIGHT\*WIDTH, outputFile);

    uint8\_t trailing ;

    wo(inputFile){

        fread(&trailing, sizeof(uint8\_t), 1, inputFile);

        fwrite(&trailing, sizeof(uint8\_t), 1, outputFile);

    }

    fclose(inputFile);

    fclose(outputFile);

    printf("Image blurred and saved successfully.\n");

    return 0;

}

Explanation of the code:

#include <stdlib.h>

#include <stdint.h>

typedef long long ll;

#define WIDTH       1192

#define HEIGHT      500

#define f(i,n)      for (ll i=0;i<n;i++)

#define f1(i,n)     for (ll i=7;i<n-7;i++)

#define f2(i,n)     for (ll i=-n;i<=n;i++)

#define wo(a)       while(!feof(a))

- Header Inclusions: Include necessary header files for standard input/output operations (<stdio.h>), dynamic memory allocation (<stdlib.h>), and fixed-width integer types (<stdint.h>).

- Type Definitions: Define a custom type ll as a shorthand for long long. This is not used in the code and seems unnecessary.

- Macro Definitions: Define constants WIDTH and HEIGHT representing the dimensions of the image. Macros f, f1, f2, and wo are defined to simplify loop constructs.

void blur(uint8\_t image[HEIGHT][WIDTH], int redi){

    uint8\_t new\_img[HEIGHT][WIDTH];

    f(i, HEIGHT) {

        f(j, WIDTH) new\_img[i][j] = image[i][j];

    }

- Function Definition - blur(): This function takes a 2D array image representing the image pixels and an integer redi representing the blur radius. Inside the function:

- new\_img is declared as a 2D array to store a copy of the original image.

- Nested loops copy the pixel values from image to new\_img.

    f(i, HEIGHT) f(j, WIDTH) {

        int sum = 0, cnt = 0;

        f2(di,redi) f2(dj,redi) {

            ll ddi = i + di,ddj = j + dj;

            if (ddi >= 0 && ddi < HEIGHT && ddj >= 0 && ddj < WIDTH) {

                sum += new\_img[ddi][ddj];

                cnt++;

            }

        }

        image[i][j] = sum / cnt;

    }

- Blur Algorithm: The function continues with nested loops over each pixel of the image. For each pixel:

- It initializes sum and cnt to calculate the average pixel value within a square neighborhood centered at the current pixel.

- It iterates over a square neighborhood defined by the blur radius redi, summing the pixel values and counting the number of valid pixels.

- It calculates the average pixel value and assigns it to the corresponding pixel in the original image.

int main() {

    FILE \*inputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/100dollars.tif", "rb");

    FILE \*outputFile = fopen("d:/IUT accademic/CSE\_4202\_lab6/blurred\_image.tif", "wb");

- Main Function: The program starts by opening the input and output files in binary read and write modes, respectively.

if (inputFile == NULL) {

        printf("Error opening input image file.\n");

        return 1;

    }

    if (outputFile == NULL) {

        printf("Error opening output image file.\n");

        return 1;

    }

- These lines check if the input and output file pointers (inputFile and outputFile) are NULL, which would indicate an error in opening the files. If either file fails to open, an error message is printed, and the program returns 1, indicating an error.

    uint8\_t metadata[8];

    fread(&metadata, sizeof(uint8\_t), 8, inputFile);

    fwrite(&metadata, sizeof(uint8\_t), 8, outputFile);

- Metadata Handling: The first 8 bytes of the input image (metadata) are read and written to the output image to preserve essential information.

    uint8\_t image[HEIGHT][WIDTH];

    fread(&image, sizeof(uint8\_t),HEIGHT\*WIDTH, inputFile);

    int r;

    printf("Enter radius for your blurring: ");

    scanf("%d",&r);

- User Input: The user is prompted to enter the blur radius, which determines the size of the neighborhood for blurring.

    blur(image,r);

    fwrite(&image, sizeof(uint8\_t), HEIGHT\*WIDTH, outputFile);

- Image Blurring: The blur() function is called to apply the blur effect to the image based on the specified radius. The resulting blurred image is written to the output file.

    uint8\_t trailing ;

    wo(inputFile){

        fread(&trailing, sizeof(uint8\_t), 1, inputFile);

        fwrite(&trailing, sizeof(uint8\_t), 1, outputFile);

    }

- Copying Trailing Data: Any remaining data in the input file (after the image data) is copied to the output file.

    fclose(inputFile);

    fclose(outputFile);

- These lines close both the input and output files once all operations are completed. This is important for releasing system resources associated with the file handles.

Overall, the code reads an image file, applies a blur effect based on user input, and saves the blurred image to a new file, preserving any metadata and trailing data.

The justification why of the code works:

1. File Handling: The code opens an input image file for reading and an output image file for writing in binary mode. It checks if the files are successfully opened and prints an error message if not.

2. Metadata Preservation: It reads the first 8 bytes of the input image file (metadata) and writes them to the output image file to preserve any essential information.

3. Image Processing:

- The blur function is defined to apply the blur effect to the image.

- A 2D array new\_img is declared to store a copy of the original image.

- The pixel values of the original image are copied to new\_img.

- The main part of the code iterates over each pixel of the image.

- For each pixel, it calculates the sum of pixel values in a square neighborhood defined by the blur radius (redi) and counts the number of pixels included in the neighborhood.

- It then calculates the average pixel value by dividing the sum by the count and assigns it to the corresponding pixel in the original image.

4. User Interaction: The user is prompted to enter the blur radius, which determines the size of the square neighborhood for blurring.

5. File Writing: The blurred image data is written to the output file.

6. Trailing Data Copying: Any remaining data in the input file (after the image data) is copied to the output file.

7. File Closure and Success Message: Finally, both input and output files are closed, and a success message is printed.

Regarding the reason for declaring sum and cnt variables as int data type:

- sum variable needs to store the sum of pixel values, and cnt needs to store the count of pixels in the neighborhood. These values could potentially exceed the range of a uint8\_t, especially when summing multiple pixel values.

- Using int ensures that the variables have a wider range to accommodate the sum of pixel values and avoid potential overflow issues.

Input:



Output:

