Session with ChatGPT to generate starter code for CSCI 551 for cranky PPM and PGM format file input, update, and output.

I have recorded a portion of this as well – here.

Here's my first prompt:

please generate C code to read in two different PPM format images, combine them so the smaller is embedded in the center of the larger, and then write out the combined image as a new PPM named hello-world.ppm

ChatGPT

To embed a smaller PPM image at the center of a larger one, you'd first need to load the PPM images. Here's a simple C program that does this. Note that this program assumes both images use the P3 (ASCII) format, and that the larger image's dimensions are indeed larger or equal to the smaller image's dimensions in both width and height.

```
c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int r, g, b;
} Pixel;

typedef struct {
    int width, height;
    Pixel **data;
} Image;

Image *load_ppm(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    char type[3];
    fscanf(file, "%2s", type);

    if (type[0] != 'P' || type[1] != '3') {
        fprintf(stderr, "Only P3 format is supported.\n");
        exit(1);
    }

    Image *img = (Image *)malloc(sizeof(Image));
    int max_value;

    fscanf(file, "%d %d %d", &img->width, &img->height, &max_value);

    img->data = (Pixel **)malloc(img->height * sizeof(Pixel *));
    for (int i = 0; i < img->height; i++) {
        img->data[i] = (Pixel *)malloc(img->width * sizeof(Pixel));
        for (int j = 0; j < img->width; j++) {
            fscanf(file, "%d %d %d", &img->data[i][j].r, &img->data[i][j].g,
&img->data[i][j].b);
        }
    }

    fclose(file);
    return img;
}

void save_ppm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P3\n%d %d\n255\n", img->width, img->height);

    for (int i = 0; i < img->height; i++) {
        for (int j = 0; j < img->width; j++) {
            fprintf(file, "%d %d %d ", img->data[i][j].r, img->data[i][j].g,
img->data[i][j].b);
        }
        fprintf(file, "\n");
    }

    fclose(file);
}

int main() {
    Image *img1 = load_ppm("image1.ppm"); // Replace with your filenames
    Image *img2 = load_ppm("image2.ppm");

    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[i + offsetY][j + offsetX] = img2->data[i][j];
        }
    }

    save_ppm("hello-world.ppm", img1);

    // Free memory
    for (int i = 0; i < img1->height; i++) {
        free(img1->data[i]);
    }
    free(img1->data);
    free(img1);

    for (int i = 0; i < img2->height; i++) {
        free(img2->data[i]);
    }
    free(img2->data);
    free(img2);
    return 0;
}
```

Replace `image1.ppm` and `image2.ppm` with the names of your larger and smaller PPM files, respectively. This program will read both files, embed the smaller image at the center of the larger one, and then write the result to `hello-world.ppm`.

S

The above code only supports P3 format, I need code for P6 format - please regenerate the above code so that it can handle images in P6 format.

ChatGPT

Certainly! P6 format is the binary version of P3. The principle is the same, but instead of reading ASCII values, we read binary data.

Here's the updated code to handle P6 format:

```
c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    unsigned char r, g, b;
} Pixel;

typedef struct {
    int width, height;
    Pixel *data;
} Image;

Image *load_ppm(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    char type[3];
    fscanf(file, "%2s", type);
    fgetc(file);  // To consume newline after the magic number

    if (type[0] != 'P' || type[1] != '6') {
        fprintf(stderr, "Only P6 format is supported.\n");
        exit(1);
    }

    Image *img = (Image *)malloc(sizeof(Image));
    int max_value;

    fscanf(file, "%d %d %d", &img->width, &img->height, &max_value);
    fgetc(file);  // Consume the newline after max_value
    img->data = (Pixel *)malloc(img->width * img->height * sizeof(Pixel));

    fread(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
    return img;
}

void save_ppm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P6\n%d %d\n255\n", img->width, img->height);

    fwrite(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
}

int main() {
    Image *img1 = load_ppm("image1.ppm"); // Replace with your filenames
    Image *img2 = load_ppm("image2.ppm");

    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[(i + offsetY) * img1->width + (j + offsetX)] = img2-
>data[i * img2->width + j];
        }
    }

    save_ppm("hello-world.ppm", img1);

    // Free memory
    free(img1->data);
    free(img1);

    free(img2->data);
    free(img2);

    return 0;
}
```

Again, replace the file names as needed. This code reads P6 format images, embeds the smaller image into the center of the larger one, and writes the result in P6 format to `hello-world.ppm`.

S

Great, but can you make the filenames for the PPM input image1 and image2 to be any name specified on the command line by using a argument vector in main?

ChatGPT

Certainly! Using `argv` and `argc` to take command line arguments is a common way to specify inputs to a C program. Here's how you can adapt the given code to take in filenames from the command line:

```
c
// ... [rest of the code remains the same]

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <larger_image.ppm> <smaller_image.ppm>\n", 
argv[0]);
        return 1;
    }

    Image *img1 = load_ppm(argv[1]); // Load the larger image
    Image *img2 = load_ppm(argv[2]); // Load the smaller image

    // Check if img1 is indeed larger than img2
    if (img1->width < img2->width || img1->height < img2->height) {
        fprintf(stderr, "The first image should be larger than or equal in 
dimensions to the second image.\n");
        return 1;
    }

    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[(i + offsetY) * img1->width + (j + offsetX)] = img2-
>data[i * img2->width + j];
        }
    }

    save_ppm("hello-world.ppm", img1);

    // Free memory
    free(img1->data);
    free(img1);

    free(img2->data);
    free(img2);

    return 0;
}
```

To run the program, you would use a command like:

```
bash
$ ./program_name larger_image.ppm smaller_image.ppm
```

Make sure to replace `program_name` with the name of your compiled C program.

S

Give me all the code again including the supporting functions please.

ChatGPT

Of course! Here's the complete code including the modified `main` function:

```
c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    unsigned char r, g, b;
} Pixel;

typedef struct {
    int width, height;
    Pixel *data;
} Image;

Image *load_ppm(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    char type[3];
    fscanf(file, "%2s", type);
    fgetc(file);  // To consume newline after the magic number

    if (type[0] != 'P' || type[1] != '6') {
        fprintf(stderr, "Only P6 format is supported.\n");
        exit(1);
    }

    Image *img = (Image *)malloc(sizeof(Image));
    int max_value;

    fscanf(file, "%d %d %d", &img->width, &img->height, &max_value);
    fgetc(file);  // Consume the newline after max_value

    img->data = (Pixel *)malloc(img->width * img->height * sizeof(Pixel));

    fread(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
    return img;
}

void save_ppm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P6\n%d %d\n255\n", img->width, img->height);

    fwrite(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <larger_image.ppm> <smaller_image.ppm>\n",
argv[0]);
        return 1;
    }

    Image *img1 = load_ppm(argv[1]); // Load the larger image
    Image *img2 = load_ppm(argv[2]); // Load the smaller image

    // Check if img1 is indeed larger than img2
    if (img1->width < img2->width || img1->height < img2->height) {
        fprintf(stderr, "The first image should be larger than or equal in
dimensions to the second image.\n");
        return 1;
    }

    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[(i + offsetY) * img1->width + (j + offsetX)] = img2-
>data[i * img2->width + j];
        }
    }

    save_ppm("hello-world.ppm", img1);

    // Free memory
    free(img1->data);
    free(img1);

    free(img2->data);
    free(img2);

    return 0;
}
```

You can compile and run this code with the desired PPM images as command-line arguments to produce a combined image named "hello-world.ppm".

S

Your code does not work. It produced the output hello-world.ppm with no data and when I test viewed it with the tool eom on Linux it produced an error noting that it has an image width of zero. The file your code produced had an incomplete header and no data in it. Can you generate good code and test it this time?

ChatGPT

I apologize for the oversight. Let's address this step by step.

The first step is to correctly read the P6 format, including skipping potential comment lines. Once the images are correctly loaded, the next step is to overlay the smaller image onto the larger image. Finally, the merged image can be saved back in P6 format.

Let's re-write the code with these considerations:

```
c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    unsigned char r, g, b;
} Pixel;

typedef struct {
    int width, height;
    Pixel *data;
} Image;

// Function to skip comments in PPM
void skip_comments(FILE *fp) {
    char ch = fgetc(fp);
    while (ch == '#') {
        while (fgetc(fp) != '\n');
        ch = fgetc(fp);
    }
    ungetc(ch, fp);
}

Image *load_ppm(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    char type[3];
    fscanf(file, "%2s", type);
    fgetc(file);  // To consume newline

    if (type[0] != 'P' || type[1] != '6') {
        fprintf(stderr, "Only P6 format is supported.\n");
        exit(1);
    }

    skip_comments(file);
    Image *img = (Image *)malloc(sizeof(Image));
    int max_value;

    fscanf(file, "%d %d", &img->width, &img->height);
    skip_comments(file);
    fscanf(file, "%d", &max_value);
    fgetc(file);  // Consume the newline after max_value

    img->data = (Pixel *)malloc(img->width * img->height * sizeof(Pixel));

    fread(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
    return img;
}

void save_ppm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P6\n%d %d\n255\n", img->width, img->height);

    fwrite(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <larger_image.ppm> <smaller_image.ppm>\n",
argv[0]);
        return 1;
    }

    Image *img1 = load_ppm(argv[1]);
    Image *img2 = load_ppm(argv[2]);

    if (img1->width < img2->width || img1->height < img2->height) {
        fprintf(stderr, "The first image should be larger than the second
image.\n");
        return 1;
    }

    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[(i + offsetY) * img1->width + (j + offsetX)] = img2-
>data[i * img2->width + j];
        }
    }
    save_ppm("hello-world.ppm", img1);

    free(img1->data);
    free(img1);
    free(img2->data);
    free(img2);

    return 0;
}
```

This revised version considers comments in the PPM and ensures that the headers are read correctly. Compile the program and run it with two P6 format PPM files to produce the merged image.

S

It worked - good job!

ChatGPT

Thank you for your patience, and I'm glad to hear it worked! If you have any more questions or need further assistance with anything else, please don't hesitate to ask. Happy coding!

S

Now write the same code but for PGM files rather than PPM - binary PGM files please.

ChatGPT

Certainly! PGM (Portable GrayMap) format represents grayscale images, and its binary form uses the `P5` magic number. Instead of dealing with RGB pixels as we did with PPM, here we'll deal with single-byte grayscale values.

Here's the code adapted for PGM files:

```
c
#include <stdio.h>
#include <stdlib.h>

typedef unsigned char Pixel;

typedef struct {
    int width, height;
    Pixel *data;
} Image;

// Function to skip comments in PGM
void skip_comments(FILE *fp) {
    char ch = fgetc(fp);
    while (ch == '#') {
        while (fgetc(fp) != '\n');
        ch = fgetc(fp);
    }
    ungetc(ch, fp);
}

Image *load_pgm(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    char type[3];
    fscanf(file, "%2s", type);
    fgetc(file);  // To consume newline

    if (type[0] != 'P' || type[1] != '5') {
        fprintf(stderr, "Only P5 format is supported.\n");
        exit(1);
    }

    skip_comments(file);

    Image *img = (Image *)malloc(sizeof(Image));
    int max_value;

    fscanf(file, "%d %d", &img->width, &img->height);
    skip_comments(file);
    fscanf(file, "%d", &max_value);
    fgetc(file);  // Consume the newline after max_value

    img->data = (Pixel *)malloc(img->width * img->height * sizeof(Pixel));

    fread(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
    return img;
}

void save_pgm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P5\n%d %d\n255\n", img->width, img->height);

    fwrite(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <larger_image.pgm> <smaller_image.pgm>\n",
argv[0]);
        return 1;
    }
    Image *img1 = load_pgm(argv[1]);
    Image *img2 = load_pgm(argv[2]);

    if (img1->width < img2->width || img1->height < img2->height) {
        fprintf(stderr, "The first image should be larger than the second 
image.\n");
        return 1;
    }

    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[(i + offsetY) * img1->width + (j + offsetX)] = img2-
>data[i * img2->width + j];
        }
    }

    save_pgm("hello-world.pgm", img1);

    free(img1->data);
    free(img1);
    free(img2->data);
    free(img2);

    return 0;
}
```

This program will read two binary PGM images (`P5` format), place the smaller one in the center of the larger one, and save the result to `hello-world.pgm`.

S

Write the same code to merge 2 different PGM files but now add a command line argument so I can specify the ouput file name. Please regenerate all of the code.

ChatGPT

Of course! I'll integrate the output file name as a command line argument and provide you the full code:

```
c
#include <stdio.h>
#include <stdlib.h>

typedef unsigned char Pixel;

typedef struct {
    int width, height;
    Pixel *data;
} Image;

// Function to skip comments in PGM
```

```
void skip_comments(FILE *fp) {
    char ch = fgetc(fp);
    while (ch == '#') {
        while (fgetc(fp) != '\n');
        ch = fgetc(fp);
    }
    ungetc(ch, fp);
}

Image *load_pgm(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    char type[3];
    fscanf(file, "%2s", type);
    fgetc(file);  // To consume newline

    if (type[0] != 'P' || type[1] != '5') {
        fprintf(stderr, "Only P5 format is supported.\n");
        exit(1);
    }

    skip_comments(file);

    Image *img = (Image *)malloc(sizeof(Image));
    int max_value;

    fscanf(file, "%d %d", &img->width, &img->height);
    skip_comments(file);
    fscanf(file, "%d", &max_value);
    fgetc(file);  // Consume the newline after max_value

    img->data = (Pixel *)malloc(img->width * img->height * sizeof(Pixel));

    fread(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
    return img;
}

void save_pgm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P5\n%d %d\n255\n", img->width, img->height);

    fwrite(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);
}
```

```
int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <larger_image.pgm> <smaller_image.pgm>
<output_image.pgm>\n", argv[0]);
        return 1;
    }

    Image *img1 = load_pgm(argv[1]);
    Image *img2 = load_pgm(argv[2]);

    if (img1->width < img2->width || img1->height < img2->height) {
        fprintf(stderr, "The first image should be larger than the second
image.\n");
        return 1;
    }

    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[(i + offsetY) * img1->width + (j + offsetX)] = img2-
>data[i * img2->width + j];
        }
    }

    save_pgm(argv[3], img1); // Saving the result to the output file
specified by the user

    free(img1->data);
    free(img1);
    free(img2->data);
    free(img2);

    return 0;
}
```

With this version, you can now specify the names of the larger and smaller PGM files along with the desired output file name when running the program.

S

Can you please test this code for me with some example PGM input files and show that it produces a valid PGM output file?

ChatGPT

I'm sorry, but I cannot directly execute or test code in this environment. However, I can guide you on how to test it yourself or help troubleshoot if you run into any issues.

Here's a brief guide on how you can test it:

1. **Compile the Code:** Use a C compiler (like `gcc`) to compile the code:

- `gcc your_program_name.c -o your_program_name`

- **Find or Create PGM Images:** You can find sample PGM images online or create them using image manipulation tools like GIMP or ImageMagick.

- **Run the Program:** If your compiled executable is named `your_program_name`, and you have two PGM files named `larger.pgm` and `smaller.pgm`, run the program as:

bash

3. `./your_program_name larger.pgm smaller.pgm output.pgm`
4.
5. **Verify the Output:** You can open the resulting `output.pgm` using any image viewer that supports PGM (like GIMP, eog, etc.) to see if the smaller image is embedded at the center of the larger image.

If you encounter any errors or the output isn't as expected, please let me know the error message or the problem you're facing, and I'll assist you further.

S

Ok, I tested the code, so can you please comment it for me.

ChatGPT

Certainly! Here's the commented version of the code:

c

```
#include <stdio.h>
#include <stdlib.h>

// Define a Pixel as an unsigned char since PGM only deals with grayscale
values (0-255)
typedef unsigned char Pixel;

// Image structure to hold the width, height, and pixel data of a PGM image
typedef struct {
    int width, height;
    Pixel *data;
} Image;

// Function to skip comments in a PGM file. PGM files might have comments
starting with #
void skip_comments(FILE *fp) {
    char ch = fgetc(fp);
    while (ch == '#') {
        while (fgetc(fp) != '\n');  // Skip the entire line of comment
        ch = fgetc(fp);
    }
    ungetc(ch, fp);  // Push back the read character to the file stream
}

// Function to load a PGM file given its filename
```

```
Image *load_pgm(const char *filename) {
    FILE *file = fopen(filename, "rb");  // Open the file in binary read mode
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    char type[3];
    fscanf(file, "%2s", type);  // Read the magic number (e.g., P5)
    fgetc(file);  // Consume the newline after the magic number

    if (type[0] != 'P' || type[1] != '5') {
        fprintf(stderr, "Only P5 format is supported.\n");
        exit(1);
    }

    skip_comments(file);  // Skip any comments in the PGM file

    Image *img = (Image *)malloc(sizeof(Image));
    int max_value;

    // Read the image dimensions and maximum pixel value
    fscanf(file, "%d %d", &img->width, &img->height);
    skip_comments(file);
    fscanf(file, "%d", &max_value);
    fgetc(file);  // Consume the newline after max_value

    // Allocate memory for pixel data
    img->data = (Pixel *)malloc(img->width * img->height * sizeof(Pixel));

    // Read pixel data from the file into the allocated memory
    fread(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);  // Close the file
    return img;    // Return the loaded image
}

// Function to save an Image structure as a PGM file
void save_pgm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");  // Open the file in binary write
mode
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    // Write the PGM header
    fprintf(file, "P5\n%d %d\n255\n", img->width, img->height);

    // Write the pixel data to the file
    fwrite(img->data, sizeof(Pixel), img->width * img->height, file);

    fclose(file);  // Close the file
}

int main(int argc, char *argv[]) {
    // Check for correct number of command line arguments
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <larger_image.pgm> <smaller_image.pgm>
<output_image.pgm>\n", argv[0]);
        return 1;
    }

    // Load the two PGM images from the provided file paths
    Image *img1 = load_pgm(argv[1]);
    Image *img2 = load_pgm(argv[2]);

    // Check if the first image is indeed larger than the second
    if (img1->width < img2->width || img1->height < img2->height) {
        fprintf(stderr, "The first image should be larger than the second
image.\n");
        return 1;
    }

    // Calculate the starting position to embed the smaller image into the
center of the larger image
    int offsetX = (img1->width - img2->width) / 2;
    int offsetY = (img1->height - img2->height) / 2;

    // Embed the smaller image into the larger one
    for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
            img1->data[(i + offsetY) * img1->width + (j + offsetX)] = img2-
>data[i * img2->width + j];
        }
    }

    // Save the merged image to the specified output file
    save_pgm(argv[3], img1);

    // Free allocated memory for both images
    free(img1->data);
    free(img1);
    free(img2->data);
    free(img2);

    return 0;  // End of program
}
```

The comments should help clarify the functionality of each part of the code. If you have any additional questions or need further explanations on specific parts, feel free to ask!

S

Now generate simple code to read in a PPM by name from the command line and to output it again to a new name as specified on the command line unchanged.

ChatGPT

Certainly. Here's a simple program to read a PPM (P6 format) from a specified filename and write it out to a new file, both provided as command line arguments:

```
c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int width, height, max_val;
    unsigned char *data;
} Image;

Image* read_ppm(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    Image *img = (Image *)malloc(sizeof(Image));
    char type[3];
    fscanf(file, "%2s", type);
    if (type[0] != 'P' || type[1] != '6') {
        fprintf(stderr, "Unsupported file format.\n");
        exit(1);
    }
    fgetc(file); // consume newline

    fscanf(file, "%d %d %d", &img->width, &img->height, &img->max_val);
    fgetc(file); // consume newline

    img->data = (unsigned char *)malloc(3 * img->width * img->height *
sizeof(unsigned char));
    fread(img->data, 3, img->width * img->height, file);

    fclose(file);
    return img;
}

void write_ppm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P6\n%d %d\n%d\n", img->width, img->height, img->max_val);
    fwrite(img->data, 3, img->width * img->height, file);

    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <input.ppm> <output.ppm>\n", argv[0]);
        return 1;
    }

    Image *img = read_ppm(argv[1]);
    write_ppm(argv[2], img);
    free(img->data);
    free(img);

    return 0;
}
```

To use the program, you'd compile it, and then run it specifying an input and output filename:

```
lua
gcc program_name.c -o output_name
./output_name input.ppm output.ppm
```

It reads the input PPM file and writes it to the output file unchanged.

S

Now code the same for a PGM instead of PPM.

ChatGPT

Sure! Here's the code adapted for PGM (P5 format):

```
c
#include <stdio.h>
#include <stdlib.h>

// Structure to hold PGM image data
typedef struct {
    int width, height, max_val;
    unsigned char *data;
} Image;

// Function to read a PGM file from a given filename
Image* read_pgm(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    Image *img = (Image *)malloc(sizeof(Image));
    char type[3];
    fscanf(file, "%2s", type);
    if (type[0] != 'P' || type[1] != '5') {
        fprintf(stderr, "Unsupported file format.\n");
        exit(1);
    }
    fgetc(file); // consume newline

    fscanf(file, "%d %d %d", &img->width, &img->height, &img->max_val);
    fgetc(file); // consume newline

    img->data = (unsigned char *)malloc(img->width * img->height *
sizeof(unsigned char));
```

```
    fread(img->data, 1, img->width * img->height, file);

    fclose(file);
    return img;
}

// Function to write a PGM image to a given filename
void write_pgm(const char *filename, Image *img) {
    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Failed to open file");
        exit(1);
    }

    fprintf(file, "P5\n%d %d\n%d\n", img->width, img->height, img->max_val);
    fwrite(img->data, 1, img->width * img->height, file);

    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <input.pgm> <output.pgm>\n", argv[0]);
        return 1;
    }

    Image *img = read_pgm(argv[1]);
    write_pgm(argv[2], img);

    free(img->data);
    free(img);

    return 0;
}
```

To use this code:

1. Save it to a file, say `pgm_processor.c`.
2. Compile it using:

```
gcc pgm_processor.c -o pgm_processor
```

3. Run the program with:

```
lua
./pgm_processor input.pgm output.pgm
```

The code reads the input PGM file and writes it unchanged to the output file.

S

Can you save a log of this session as a downloadable document for me?

ChatGPT

I cannot directly provide downloadable files, but you can easily create a document yourself by following these steps:

1. **Text Selection:** Highlight and select the entire conversation or the portions you wish to save.
2. **Copy:** Use the right-click menu or press `Ctrl+C` (or `Cmd+C` on macOS) to copy the selected text.
3. **Document Creation:**
   - Open a word processing application like Microsoft Word, Google Docs, LibreOffice Writer, or any other.
   - Create a new document.
   - Paste the copied text into this document using `Ctrl+V` (or `Cmd+V` on macOS).
4. **Save or Download:** Use the "Save" or "Download" option of your word processing application to save the document to your local storage or cloud storage.

This way, you can keep a log of our conversation for your future reference.