Report

The program processes a bitmap image file specified by the user, performing two main operations: image reduction and image merging with a customizable frame. It outputs new bitmap files representing the reduced image and the merged image with a frame.

Key Features and Operations

Bitmap File Reading and Writing:

The program begins by opening a user-specified bitmap file in binary mode, reading the file header, and then the image data.

It writes the processed images (reduced and merged with frame) to new files, preserving the bitmap format.

Image Reduction:

Reduces the image's resolution by half, selecting every other pixel from the original image to form a new, smaller image.

The reduced image maintains the aspect ratio and color depth of the original.

Image Merging with Frame Addition:

Merges the reduced image into four quadrants and adds a customizable frame around the merged image based on user input for color and size.

Allows customization through user input for the frame's pixel width and RGB color.

Dynamic Memory Management:

Dynamically allocates memory for the original, reduced, and merged images' pixel data, ensuring efficient use of memory.

Implements proper memory deallocation to prevent memory leaks.

User Interaction:

Prompts the user for necessary inputs such as frame size and color, enabling customization of the output image.

Technical Details

The program uses a struct to represent the bitmap header, facilitating organized access and manipulation of the file's metadata.

It employs dynamic memory allocation (malloc) for handling image data, crucial for processing variable-sized images.

The image processing logic iterates over pixel data, requiring an understanding of image data storage and manipulation.

Error handling is implemented for file operations, ensuring the program responds gracefully to issues like missing files.

Suggestions for Improvement

Error Checking for Memory Allocation: Adding checks after each malloc call to ensure memory was successfully allocated would enhance the program's robustness.

Refactoring for Code Duplication: The frame drawing logic is repetitive; abstracting this into a separate function could reduce duplication.

Avoiding Platform-specific Functions: system("pause") is Windows-specific; a portable alternative could improve cross-platform compatibility.

Replacing Magic Numbers: Using named constants instead of hardcoded numbers for header sizes and offsets would improve readability.