# README DOCUMENT

**Overview:**

We will first read BMP Image in this task, then apply transformation to form the negative image and finally write image back into the file such that it creates a new file and the program preserves non-destructive characteristic.

We define negative image on a strip or sheet of transparent plastic film as a image, in which the lightest areas of the photographed subject appear darkest and the darkest areas appear lightest. The intensity levels vary from 0 to L-1 in a digital image. Negative Image helps finding the details from the darker regions of the image. We will see further how to perform such a image transformation.

**Execution Command:**

Unzip the files and open command prompt. Go to the directory.

To directly run the program:

./main.exe <image\_path/image\_name.bmp> <image\_path/output\_image.bmp>

To compile and run the program follow below commands:

* g++ -Wall –c main.cpp –o main.exe
* ./main.exe <image\_path/image\_name.bmp> <image\_path/output\_image.bmp>

Note 1 - Above instructions are for Windows platform.

Note 2 – To view the command instruction use the following command:

* ./main.exe --command

**Possible Error:**

**Issue**: 16 bit application is not supportable.

**Reason:** Windows (64-bit) does not support running 16-bit Windows-based applications. The primary reason is that handles have 32 significant bits on 64-bit Windows. Therefore, handles cannot be truncated and passed to 16-bit applications without loss of data. Attempts to launch 16-bit applications fail.

**Source Files**

+ bmpimagefile.h: a header file that contains useful data structure, variables and function declarations which we must implement

+ bmpimagefile.cpp: Contains source code for reading, writing the “bmp” file. It also contains function to de-allocate the image memory in case of failures.

+ main.cpp : This is the starting point for the program and it make use of read and write function from “bmpimagefile” and contains logic for negative transformation defined below:

* The pixel value vary from 0 to L-1 in the digital image. The negative transformation is defined as s=L-1-r.
* To invert an image, each of its pixel value ‘r’ is subtracted from the maximum pixel value L-1 and the original pixel is replaced with the result ‘s’.

+ images: directory of images containing a set of good and bad image :

+ images1: those images which satisfy our requirement and assumptions and produce the negative image.

+images2: those images which fails to satisfy the assumptions and doesn’t produce any negative image.

**Assumptions:**

Three main assumptions were taken into consideration while implementing the task:

* Image file is of type BMP.
* No compression was used on the image.
* Bits per pixel value needs to be 24.

The program will stop after printing the appropriate reason in case assumptions were not satisfied by the image.

**Error Handling:**

Following error-handling is in place:

1. - iostate-file: To handle issues related to file reading, writing or any issues while opening the file itself.
2. – bad-alloc: Issue while assigning the memory for the image data and image headers.
3. – image-header-issue: Image headers not readable or getting written in output image.
4. – out-of-bound: To handle issues while accessing the image data out of the defined memory space.

To get more information and helpful links related to each of these errors use the following command

* ./main.exe --error-help

Note – Program will halt after providing appropriate issue.

**Detailed Guide:**

Part 1: Reading a BMP Image File

In real-world program bugs are related to programmers not thoroughly checking the inputs to the programs. So one should think carefully about what could possibly go wrong when opening a file or reading the image file. In case, something goes wrong, we return NULL and ensure no resource leakage.

In real-world scenarios, we must check for following list of errors and if there are any errors, then leak no resource and return null.

1. Ensure you can open the image file.
2. Read the image header and perform assumption based validation.
   1. Make sure that all the info from the header is readable
   2. Ensure the magic identifier is correct in the header.
   3. Image has no compression type
   4. Bits per pixel for image is set to 24.
3. Allocate required space for image, and pixels and any other data required.
   1. Note – malloc() returns NULL if it cannot able to allocate the space.
4. Read the pixels or image data.
5. Ensure to read all width\*height RGB pixels
6. Ensure that end of the file has been reached and you have close the file pointed.

Finally, we will testing this function on both the corrupted images as well as normal images to ensure the correctness of the program.

Part 2: Writing a BMP Image File

Saving is easier than reading a BMP image file, because it can be assumed now that the BMPImage struct is valid. Thus we don’t have to worry about handling errors that one may be encounter while reading a file. In this function, we take an BMPImage struct and write it out in the appropriate format.

Part 3: Main Function for the program.

Main function reads the appropriate arguments from command line and perform initial validation. On validation check, if successful, it makes call to read function from bmpimagefile.cpp to have the all the required data in BMPImage struct.

Next, it performs the negative transformation by subtracting 255(maximum value for 8 bit pixel) from each of the RGB based pixel and collects the updated data.

Finally, it makes call to the write functionality defined in bmpimagefunc.cpp.

This program prints useful image header information and If all works well, the program creates the output negative file and closes it successfully. Otherwise it halts printing the issue.