ASSIGNMENT - 1

GROUP-7

ANKIT BANSAL | 12EC35024
AMRUT RAJKARNE | 12EC35017

ASSIGNMENT – 1 : BMP FILE

Aim: To write C/C++ modular functions to read, diagonally flip, and then write BMP image files. All functions must support at least 24-bit RGB, and 8-bit grayscale image formats.

- ReadBMP:
  a. Input: Filename of input image
  b. Output: BMP header structure, Image pixel array loaded onto memory

- ConvertFlipGrayscale:
  a. Input: Image pixel array
  b. Output: Grayscale-converted and diagonally flipped (transposed) pixel array

- WriteBMP:
  a. Input: Filename of output (grayscale) image, BMP header structure, Image pixel array
  b. Output: BMP file written on disk

BMP File Image:

The BMP file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device.

Bitmap file header:

This block of bytes is at the start of the file and is used to identify the file. A typical application reads this block first to ensure that the file is actually a BMP file and that it is not damaged.

Offset hex	Offset dec	Size	Purpose
00	0	2 bytes	The header field used to identify the BMP and DIB file is `0x42` `0x4D` in hexadecimal, same as `BM` in ASCII.
02	2	4 bytes	The size of the BMP file in bytes

Offset (hex)	Offset (dec)	Size (bytes)	Windows BITMAPINFOHEADER[2]
0E	14	4	the size of this header (40 bytes)
12	18	4	the bitmap width in pixels (signed integer)
16	22	4	the bitmap height in pixels (signed integer)
1A	26	2	the number of color planes (must be 1)
1C	28	2	the number of bits per pixel, which is the color depth of the image. Typical values are 1, 4, 8, 16, 24 and 32.
1E	30	4	the compression method being used. See the next table for a list of possible values
22	34	4	the image size. This is the size of the raw bitmap data; a dummy 0 can be given for BI_RGB bitmaps.
26	38	4	the horizontal resolution of the image. (pixel per meter, signed integer)
2A	42	4	the vertical resolution of the image. (pixel per meter, signed integer)
2E	46	4	the number of colors in the color palette, or 0 to default to 2^n
32	50	4	the number of important colors used, or 0 when every color is important; generally ignored

Color table:

The color table (palette) occurs in the BMP image file directly after the BMP file header, the DIB header (and after optional three red, green and blue bitmasks if the BITMAPINFOHEADER header with BI_BITFIELDS option is used). Therefore, its offset is the size of the BITMAPFILEHEADER plus the size of the DIB header (plus optional 12 bytes for the three bit masks).

Pixel array (bitmap data):

The pixel array is a block of 32-bit DWORDs, that describes the image pixel by pixel. Normally pixels are stored "upside-down" with respect to normal image raster scan order, starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image.[5] Unless BITMAPCOREHEADER is used, uncompressed Windows bitmaps also can be stored from the top to bottom, when the Image Height value is negative.

Algorithm

ReadBmp

Open input image in binary read mode.Read header info(first 54 bytes of the binary data) and Image Pixel Data and copies it to Image structure which has two components BITMAPHEADER bmpheader and RGB ** pixeldata.

PrintInfo

Prints the header information by using Image structure as input.

ConvertFlipGrayscale

- Convert image pixel array RGB data to grayscale according to : Grayscale Intensity= 0.30*R + 0.59*G + 0.11*B.
- Store grayscale image data in a pointer named gray
- Flip grayscale data by computing the transpose of the pixel data matrix
- Store the flipped grayscale image data in a pointer named flipped
- The gray and flipped variables will be used by WriteBmp to write the BMP image.

WriteBmp

- Open the output file in write binary mode.
- Write the BMP header structure data to the binary file(54 bytes)
- Read the height and width of image from the BMP header structure.
- Write the image pixel array data to the binary file from the current pointer to the file
- Close the file being written to.

Code

```
/*****************************************************************************/
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

#pragma pack(push,1)                                                    // push
current alignment to stack and set it to 1 byte boundary

typedef struct
{
       unsigned short signature;                                        // magic
number used to identify the BMP file: 0x42 0x4D (Hex code points for B and M).
       unsigned int filesize;                                                  //
size of the BMP file in bytes
       unsigned short reserved1, reserved2;            // reserved.
       unsigned int offset, infosize, width, height;   //offset,bytes for info
header(40),width,height in pixels
       unsigned short planes, bitperpix;                        // number of color
planes(1),no. of bits per pixel
       unsigned int compression;                                        // Type of
compression
       unsigned int ImageSize;                                                 //
Image size in bytes (including paddding)
       unsigned int xPPM;                                               //
Horizontal resolution in pixels per metre
       unsigned int yPPM;                                               // Vertical
resolution in pixels per metre
       unsigned int colors;                                             // Colors
used in BMP
       unsigned int impcolors;                                          // Important
Colors used in BMP
}BITMAPHEADER;

typedef struct                                                                  //
Struct representing B,G,R values of a single pixel
{
       unsigned char blue, green, red;
}RGB;

typedef struct                                          // Struct representing an
image with header and address pointing to 2D array of struct RB
{
       BITMAPHEADER bmpheader;
       RGB ** pixeldata;

} Image;

#pragma pack(pop)                                                       // restore
original alignment from stack
```

```
/* Function to Read BMP Image */
void ReadBmp(FILE *bmpInput,Image * orgnlImage)
{

        int i, sizeperpix, sizepad, row, col,j;
        unsigned char *padding;

         // Read BMP header
        fread(&(orgnlImage->bmpheader), 1, sizeof(BITMAPHEADER), bmpInput);

        // To verify validity of BMP file
        if (orgnlImage->bmpheader.signature !=
0x4D42)
        {
                cout<<"Not a BMP file"<<endl;
                fclose(bmpInput);
                exit(0);
        }

        sizeperpix = orgnlImage->bmpheader.bitperpix / 8;

        // Zero Padding end of each row
        sizepad = (4 - ((orgnlImage->bmpheader.width*sizeperpix) % 4)) %
4;

        // Image Height
        row = orgnlImage->bmpheader.height;

        // Image Width
        col = orgnlImage-
>bmpheader.width;


        fseek(bmpInput, orgnlImage->bmpheader.offset,
SEEK_SET);                          //set the file pointer to the beginning of pixel
data
        orgnlImage->pixeldata = (RGB
**)malloc(row*sizeof(RGB*));

        for (i = 0; i<row; i++)
        {
                orgnlImage->pixeldata[i] = (RGB *)malloc(col*sizeof(RGB));  // Dynamic
Memory Allocation
                for (j = 0; j < col; j++)
                {
                        fread(&(orgnlImage->pixeldata[i][j]), sizeperpix, 1, bmpInput);
// Pixel Data to 2D array
                }

                if (sizepad != 0){
                        padding = (unsigned char*)malloc(sizepad);
                        fread(&padding, 1, sizepad, bmpInput);
                        free(padding);
                }
        }

        fclose(bmpInput);
```

```
	cout << "Image reading done" <<endl;
	cout << "" <<endl;
}

/* Function to produce Image Information Summary */
void printInfo(Image *orgnlImage){

	cout << "Type: " << orgnlImage->bmpheader.signature << endl;
	cout << "size: " << orgnlImage->bmpheader.filesize << endl;
	cout << "offset: " << orgnlImage->bmpheader.offset << endl;
	cout << "Info header size: " << orgnlImage->bmpheader.infosize << endl;
	cout << "Bitmap width: " << orgnlImage->bmpheader.width << endl;
	cout << "Bimap height " << orgnlImage->bmpheader.height<< endl;
	cout << "Color planes: " << orgnlImage->bmpheader.planes << endl;
	cout << "BPP: " << orgnlImage->bmpheader.bitperpix << endl;
	cout << "Compression Method " << orgnlImage->bmpheader.compression << endl;
	cout << "Raw bmp data size: " << orgnlImage->bmpheader.ImageSize << endl;
	cout << "Horizontal Resolution: " << orgnlImage->bmpheader.xPPM << endl;
	cout << "Vertical Resolution: " << orgnlImage->bmpheader.yPPM << endl;
	cout << "No of Color: " << orgnlImage->bmpheader.colors << endl;
	cout << "No of imp colors: " << orgnlImage->bmpheader.impcolors << endl;
	cout << "" <<endl;
}

/* Flip Gray Scale Function */
void ConvertFlipGrayscale(RGB ** pixelarray, int row, int col, unsigned char*** gray,
unsigned char*** flip)
{
	int i, j;
	*gray = (unsigned char**)malloc(sizeof(unsigned char*)* row);
	for (i = 0; i < row; i++)
	{
		(*gray)[i] = (unsigned char*)malloc(sizeof(unsigned char)* col);
	}
	for (i = 0; i < row; i++)
	{
		for (j = 0; j < col; j++)
		{
			(*gray)[i][j] = unsigned
char((0.11*float(pixelarray[i][j].blue) + 0.59*float(pixelarray[i][j].green) +
0.3*float(pixelarray[i][j].red)));
		}
	}
	*flip = (unsigned char**)malloc(sizeof(unsigned char*)* col);
	for (i = 0; i < col; i++)
	{
		(*flip)[i] = (unsigned char*)malloc(sizeof(unsigned char)* row);
	}
	for (i = 0; i < col; i++)
	{
		for (j = 0; j < row; j++)
		{

			(*flip)[i][j] = (*gray)[row-1-j][col-1-i];  // Formula used for
transposing

		}
```

```
        }

}
/*****************************************************************************/

/* Write BMP Image */
void WriteBmp(FILE *bmpOut, BITMAPHEADER header, unsigned char** pixarray)
{
        int i, j, sizepad;
        BITMAPHEADER head;

        // Padding zeros at the end
        unsigned char *padding;
        sizepad = (4 - (header.width % 4)) % 4;

        // Set original header & make necessary changes
        head =
header;

        head.bitperpix =
8;

        head.ImageSize = header.height*(header.width + (sizepad));
        head.colors =
256;
                // Color Pallete size 256
        head.offset = header.offset +
1024;                                                   // Color Table into
account
        head.filesize = head.ImageSize + head.offset;

        // Write Modified Header to the file
        fwrite(&head, sizeof(unsigned char), 54,
bmpOut);

        // Define color table (RGBA32 format)
        unsigned char colorTable[1024];
        for (i = 0; i < 1024; i++)
        {
                colorTable[i] = unsigned char(i/4)
;
                if (i % 4 == 3)
                        colorTable[i] = 0;

        }

        // Write the color table to the file
        fwrite(colorTable, sizeof(unsigned char), 1024, bmpOut);

        // Write pixel data to the file
        for (i = 0; i<header.height; i++)
        {
                for (j = 0; j<header.width; j++)
                {
                        fwrite(&pixarray[i][j], sizeof(unsigned char), 1, bmpOut);
                }
                //padding required only when number of columns is not a multiple of 4
```

```
                if (sizepad != 0)
                        {
                                padding = (unsigned char *)calloc(sizepad,
sizeof(unsigned char));
                                fwrite(padding, 1, sizepad, bmpOut);
                        }
                        }
        fclose(bmpOut);
        cout << "Image writing done" <<endl;
        cout << "" <<endl;
}

int main()
{
        FILE *bmpip, *bmpop;
        int temp;
        char ip[50];
        Image orgnlImage;
        unsigned char** grayscale;
        unsigned char** flippedgray;

        string fname;
        string fname_write;
        string fname_fwrite;

        cout << "Please input filename" << endl;
        cin >> fname;
        fname = "TestImages/" + fname + ".bmp";
        bmpip = fopen(fname.c_str(), "rb"); // open the input BMP color image and
store the file address in fptr
        if(bmpip == NULL) cout<<"404: Not found"<<endl;

        // Read BMP - Function Call
        ReadBmp(bmpip, &orgnlImage);

        printInfo(&orgnlImage);

        cout << "Please output filename for Grayscale" << endl;
        cin >> fname_write;
        fname_write = "TestImages/" + fname_write + ".bmp";
    bmpop = fopen(fname_write.c_str(), "wb");
        if(bmpop == NULL) cout<<"404: Not found"<<endl;

        // Converts RGB pixel data into Grayscale
        ConvertFlipGrayscale(orgnlImage.pixeldata, orgnlImage.bmpheader.height,
orgnlImage.bmpheader.width, &grayscale, &flippedgray);

        // Write BMP Image - Function Call
        WriteBmp(bmpop, orgnlImage.bmpheader,grayscale);
        fclose(bmpop);

        // [Flipping] - Interchanging height with width
        temp = orgnlImage.bmpheader.height;
        orgnlImage.bmpheader.height = orgnlImage.bmpheader.width;
        orgnlImage.bmpheader.width = temp;

        cout << "Please output filename for Flipped Grayscale" << endl;
```

```
        cin >> fname_fwrite;
        fname_fwrite = "TestImages/" + fname_fwrite + ".bmp";
    bmpop = fopen(fname_fwrite.c_str(), "wb");
        if(bmpop == NULL) cout<<"404: Not found"<<endl;

        // Write BMP Image - Function Call
        WriteBmp(bmpop, orgnlImage.bmpheader, flippedgray);
        fclose(bmpop);
        _getch();
}
/*****************************************************************************/
```

Output Results

A 24-bit / 8-bit image taken as an input at the command line (as the first argument) writes a flipped grayscale image as output to the disk with the given file name at the command line (as the second argument).

Analysis

Padding bytes (not necessarily 0) must be appended to the end of the rows in order to bring up the length of the rows to a multiple of four bytes. When the pixel array is loaded into memory, each row must begin at a memory address that is a multiple of 4. This address/offset restriction is mandatory only for Pixel Arrays loaded in memory. A 24-bit bitmap with Width=1, would have 3 bytes of data per row (blue, green, red) and 1 byte of padding, while Width=2 would have 2 bytes of padding, Width=3 would have 3 bytes of padding, and Width=4 would not have any padding at all. Normally pixels in the pixel array are stored “upside-down” with respect to normal image raster scan order, starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image. Thus, while flipping the image along the diagonal, we have to read the image left-to-right and then take transpose of the pixel data matrix.