0. BMP- 16M, GIF-256, PNG - 256T, JPEG- 16M
1. GIF
2. Lossless creates orignal file exactly as it is. Lossy reomves unnecesarry bits of information.
3. JPEG
4. When the operating system erases a FAT file, the system modifies the filename’s first character in the file’s directory entry to signal that the file
    has been deleted and that the directory entry can be recycled. The system then moves all of the file’s FAT clusters to the hard drive’s list of free clusters.
    The actual file data is never touched
5. Sanitize hard-drive information by intentionally by overwriting that data with other data
6. Library that declares sets of integer types having specified widths with corresponding macros and that specify limits of integer types.
7. Specifies width or bits of data that is to be used. Eg. uint8_t means an integer that is exactly 8 bits wide.
8. BYTE 1 byte, DWORD 4 bytes, LONG 4 bytes, WORD 2 bytes.
9. The character 'B' then the character 'M' in 1-byte ASCII encoding, which specifies the file type.
10. bfSize specifies the size in bytes of the bitmap file and biSize specifies the size of the structure in bytes.
11. BMP has origin in upper left corner & is top down
12. biBitCount
13. Trying to open non existant file or due to permission errors.
14. As it specifies the number of objects to be read.
15. (4 - (bi.biWidth * sizeof(RGBTRIPLE)) % 4) % 4 = (4 - (3 * 3) % 4) % 4 = (4 - 1) % 4 = 3.
16. Set file position to the given stream
17. Offset to specify current location of file pointer
18. David Malan

```c
/**
 * recover.c
 *
 * Computer Science 50
 * Problem Set 4
 *
 * Recovers JPEGs from a forensic image.
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#define SIZE 512
typedef uint8_t  BYTE;

int main(int argc, char* argv[])
{
    FILE* card = fopen("card.raw","r");

    BYTE buffer[SIZE];
    int count = -1;//as first file shud be 000.jpg

    if (card == NULL)
    {
        printf("Could not open card.\n");
        return 1;
    }
    FILE* image = NULL;
    char* outfile = malloc(sizeof(char));

    while((fread(&buffer,SIZE*sizeof(BYTE),1,card))==1){
        //check jpeg bytes
        if(buffer[0]==255 && buffer[1]==216 && buffer[2]==255 && buffer[3] >> 4 == 14){//new jpg
            if(count > -1){
                fclose(image);
            }
            count++;
            sprintf(outfile,"%.3d.jpg",count);
            image = fopen(outfile, "w");
            if (image == NULL)
            {
                printf("Could not open %s.\n", outfile);
                return 2;
            }

            fwrite(&buffer,SIZE*sizeof(BYTE),1,image);
        }else if(count > -1){
            fwrite(&buffer,SIZE*sizeof(BYTE),1,image);
        }
```

```
49.     }
50.     free(outfile);
51.     fclose(card);
52.     fclose(image);
53.     return 0;
54. }
55.
```

```h
/**
 * bmp.h
 *
 * Computer Science 50
 * Problem Set 4
 *
 * BMP-related data types based on Microsoft's own.
 */

#include <stdint.h>

/**
 * Common Data Types
 *
 * The data types in this section are essentially aliases for C/C++
 * primitive data types.
 *
 * Adapted from http://msdn.microsoft.com/en-us/library/cc230309.aspx.
 * See http://en.wikipedia.org/wiki/Stdint.h for more on stdint.h.
 */
typedef uint8_t  BYTE;
typedef uint32_t DWORD;
typedef int32_t  LONG;
typedef uint16_t WORD;

/**
 * BITMAPFILEHEADER
 *
 * The BITMAPFILEHEADER structure contains information about the type, size,
 * and layout of a file that contains a DIB [device-independent bitmap].
 *
 * Adapted from http://msdn.microsoft.com/en-us/library/dd183374(VS.85).aspx.
 */
typedef struct
{
    WORD   bfType;
    DWORD  bfSize;
    WORD   bfReserved1;
    WORD   bfReserved2;
    DWORD  bfOffBits;
} __attribute__((__packed__))
BITMAPFILEHEADER;

/**
 * BITMAPINFOHEADER
 *
 * The BITMAPINFOHEADER structure contains information about the
 * dimensions and color format of a DIB [device-independent bitmap].
```

```
 *
 * Adapted from http://msdn.microsoft.com/en-us/library/dd183376(VS.85).aspx.
 */
typedef struct
{
    DWORD  biSize;
    LONG   biWidth;
    LONG   biHeight;
    WORD   biPlanes;
    WORD   biBitCount;
    DWORD  biCompression;
    DWORD  biSizeImage;
    LONG   biXPelsPerMeter;
    LONG   biYPelsPerMeter;
    DWORD  biClrUsed;
    DWORD  biClrImportant;
} __attribute__((__packed__))
BITMAPINFOHEADER;

/**
 * RGBTRIPLE
 *
 * This structure describes a color consisting of relative intensities of
 * red, green, and blue.
 *
 * Adapted from http://msdn.microsoft.com/en-us/library/aa922590.aspx.
 */
typedef struct
{
    BYTE  rgbtBlue;
    BYTE  rgbtGreen;
    BYTE  rgbtRed;
} __attribute__((__packed__))
RGBTRIPLE;

```

```c
/**
 * copy.c
 *
 * Computer Science 50
 * Problem Set 4
 *
 * Copies a BMP piece by piece, just because.
 */

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

#include "bmp.h"

int main(int argc, char* argv[])
{
    // ensure proper usage
    if (argc != 3)
    {
        printf("Usage: ./copy infile outfile\n");
        return 1;
    }

    // remember filenames
    char* infile = argv[1];
    char* outfile = argv[2];

    // open input file
    FILE* inptr = fopen(infile, "r");
    if (inptr == NULL)
    {
        printf("Could not open %s.\n", infile);
        return 2;
    }

    // open output file
    FILE* outptr = fopen(outfile, "w");
    if (outptr == NULL)
    {
        fclose(inptr);
        fprintf(stderr, "Could not create %s.\n", outfile);
        return 3;
    }

    // read infile's BITMAPFILEHEADER
    BITMAPFILEHEADER bf;
    fread(&bf, sizeof(BITMAPFILEHEADER), 1, inptr);

```

```
49.     // read infile's BITMAPINFOHEADER
50.     BITMAPINFOHEADER bi;
51.     fread(&bi, sizeof(BITMAPINFOHEADER), 1, inptr);
52.
53.     // ensure infile is (likely) a 24-bit uncompressed BMP 4.0
54.     if (bf.bfType != 0x4d42 || bf.bfOffBits != 54 || bi.biSize != 40 ||
55.         bi.biBitCount != 24 || bi.biCompression != 0)
56.     {
57.         fclose(outptr);
58.         fclose(inptr);
59.         fprintf(stderr, "Unsupported file format.\n");
60.         return 4;
61.     }
62.
63.     // write outfile's BITMAPFILEHEADER
64.     fwrite(&bf, sizeof(BITMAPFILEHEADER), 1, outptr);
65.
66.     // write outfile's BITMAPINFOHEADER
67.     fwrite(&bi, sizeof(BITMAPINFOHEADER), 1, outptr);
68.
69.     // determine padding for scanlines
70.     int padding =  (4 - (bi.biWidth * sizeof(RGBTRIPLE)) % 4) % 4;
71.
72.     // iterate over infile's scanlines
73.     for (int i = 0, biHeight = abs(bi.biHeight); i < biHeight; i++)
74.     {
75.         // iterate over pixels in scanline
76.         for (int j = 0; j < bi.biWidth; j++)
77.         {
78.             // temporary storage
79.             RGBTRIPLE triple;
80.
81.             // read RGB triple from infile
82.             fread(&triple, sizeof(RGBTRIPLE), 1, inptr);
83.
84.             // write RGB triple to outfile
85.             fwrite(&triple, sizeof(RGBTRIPLE), 1, outptr);
86.         }
87.
88.         // skip over padding, if any
89.         fseek(inptr, padding, SEEK_CUR);
90.
91.         // then add it back (to demonstrate how)
92.         for (int k = 0; k < padding; k++)
93.         {
94.             fputc(0x00, outptr);
95.         }
96.     }
```

```
 97.
 98.     // close infile
 99.     fclose(inptr);
100.
101.     // close outfile
102.     fclose(outptr);
103.
104.     // that's all folks
105.     return 0;
106. }
107.
```

```c
/**
 * copy.c
 *
 * Computer Science 50
 * Problem Set 4
 *
 * Copies a BMP piece by piece, just because.
 */

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

#include "bmp.h"

int main(int argc, char* argv[])
{
    // ensure proper usage
    if (argc != 3)
    {
        printf("Usage: ./copy infile outfile\n");
        return 1;
    }

    // remember filenames
    char* infile = argv[1];
    char* outfile = argv[2];

    // open input file
    FILE* inptr = fopen(infile, "r");
    if (inptr == NULL)
    {
        printf("Could not open %s.\n", infile);
        return 2;
    }

    // open output file
    FILE* outptr = fopen(outfile, "w");
    if (outptr == NULL)
    {
        fclose(inptr);
        fprintf(stderr, "Could not create %s.\n", outfile);
        return 3;
    }

    // read infile's BITMAPFILEHEADER
    BITMAPFILEHEADER bf;
    fread(&bf, sizeof(BITMAPFILEHEADER), 1, inptr);

```

```
49.     // read infile's BITMAPINFOHEADER
50.     BITMAPINFOHEADER bi;
51.     fread(&bi, sizeof(BITMAPINFOHEADER), 1, inptr);
52.
53.     // ensure infile is (likely) a 24-bit uncompressed BMP 4.0
54.     if (bf.bfType != 0x4d42 || bf.bfOffBits != 54 || bi.biSize != 40 ||
55.         bi.biBitCount != 24 || bi.biCompression != 0)
56.     {
57.         fclose(outptr);
58.         fclose(inptr);
59.         fprintf(stderr, "Unsupported file format.\n");
60.         return 4;
61.     }
62.
63.     // write outfile's BITMAPFILEHEADER
64.     fwrite(&bf, sizeof(BITMAPFILEHEADER), 1, outptr);
65.
66.     // write outfile's BITMAPINFOHEADER
67.     fwrite(&bi, sizeof(BITMAPINFOHEADER), 1, outptr);
68.
69.     // determine padding for scanlines
70.     int padding =  (4 - (bi.biWidth * sizeof(RGBTRIPLE)) % 4) % 4;
71.
72.     // iterate over infile's scanlines
73.     for (int i = 0, biHeight = abs(bi.biHeight); i < biHeight; i++)
74.     {
75.         // iterate over pixels in scanline
76.         for (int j = 0; j < bi.biWidth; j++)
77.         {
78.             // temporary storage
79.             RGBTRIPLE triple;
80.
81.             // read RGB triple from infile
82.             fread(&triple, sizeof(RGBTRIPLE), 1, inptr);
83.
84.             if(triple.rgbtRed == 0xff && triple.rgbtGreen == 0x00 && triple.rgbtBlue == 0x00){
85.                 triple.rgbtRed = 0x00;
86.             }
87.             if(triple.rgbtRed == 0x00 && triple.rgbtGreen == 0x00 && triple.rgbtBlue == 0x00){
88.                triple.rgbtRed = 0xff;
89.                triple.rgbtGreen = 0xff;
90.                triple.rgbtBlue = 0xff;
91.             }
92.             if(triple.rgbtRed == 0x00 && triple.rgbtGreen == 0x00 && triple.rgbtBlue != 0x00){
93.                triple.rgbtRed = 0x00;
94.                triple.rgbtGreen = 0x00;
95.                triple.rgbtBlue = 0x00;
96.             }
```

```
 97.             // write RGB triple to outfile
 98.             fwrite(&triple, sizeof(RGBTRIPLE), 1, outptr);
 99.         }
100.
101.         // skip over padding, if any
102.         fseek(inptr, padding, SEEK_CUR);
103.
104.         // then add it back (to demonstrate how)
105.         for (int k = 0; k < padding; k++)
106.         {
107.             fputc(0x00, outptr);
108.         }
109.     }
110.
111.     // close infile
112.     fclose(inptr);
113.
114.     // close outfile
115.     fclose(outptr);
116.
117.     // that's all folks
118.     return 0;
119. }
120.
```

```c
/**
 * copy.c
 *
 * Computer Science 50
 * Problem Set 4
 *
 * Copies a BMP piece by piece, just because.
 */

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

#include "bmp.h"

int main(int argc, char* argv[])
{
    // ensure proper usage
    if (argc != 4)
    {
        printf("Usage: ./copy n infile outfile\n");
        return 1;
    }

    int n = atoi(argv[1]);

    //ensure proper n
    if(n < 0 || n > 100){
        printf("n cannot be less than 0 or greater than 100");
        return 2;
    }

    // remember filenames
    char* infile = argv[2];
    char* outfile = argv[3];

    // open input file
    FILE* inptr = fopen(infile, "r");
    if (inptr == NULL)
    {
        printf("Could not open %s.\n", infile);
        return 2;
    }

    // open output file
    FILE* outptr = fopen(outfile, "w");
    if (outptr == NULL)
    {
        fclose(inptr);
```

```
49.         fprintf(stderr, "Could not create %s.\n", outfile);
50.         return 3;
51.     }
52.
53.     // read infile's BITMAPFILEHEADER
54.     BITMAPFILEHEADER bf;
55.     fread(&bf, sizeof(BITMAPFILEHEADER), 1, inptr);
56.
57.     // read infile's BITMAPINFOHEADER
58.     BITMAPINFOHEADER bi;
59.     fread(&bi, sizeof(BITMAPINFOHEADER), 1, inptr);
60.
61.     int padding = (4-(bi.biWidth * sizeof(RGBTRIPLE)) %4) %4;
62.     LONG inWidth = bi.biWidth;
63.     LONG inHeight =abs(bi.biHeight);
64.
65.     // ensure infile is (likely) a 24-bit uncompressed BMP 4.0
66.     if (bf.bfType != 0x4d42 || bf.bfOffBits != 54 || bi.biSize != 40 ||
67.         bi.biBitCount != 24 || bi.biCompression != 0)
68.     {
69.         fclose(outptr);
70.         fclose(inptr);
71.         fprintf(stderr, "Unsupported file format.\n");
72.         return 4;
73.     }
74.
75.      bi.biWidth = n * bi.biWidth;
76.      bi.biHeight = n * bi.biHeight;
77.
78.      int paddingOut = (4-(bi.biWidth * sizeof(RGBTRIPLE))%4)%4;
79.
80.      bi.biSizeImage = abs(bi.biHeight)*(bi.biWidth * sizeof(RGBTRIPLE) + paddingOut);
81.      bf.bfSize = bi.biSizeImage + 54;
82.
83.     // write outfile's BITMAPFILEHEADER
84.     fwrite(&bf, sizeof(BITMAPFILEHEADER), 1, outptr);
85.
86.     // write outfile's BITMAPINFOHEADER
87.     fwrite(&bi, sizeof(BITMAPINFOHEADER), 1, outptr);
88.
89.
90.     // iterate over infile's scanlines
91.     for (int i = 0; i < inHeight; i++)
92.     {
93.         long curr = ftell(inptr);
94.         // iterate over pixels in scanline
95.         for (int j = 0; j < n; j++)
96.         {
```

```
            fseek(inptr,curr,SEEK_SET);

            for(int k = 0; k < inWidth; k++){
                // temporary storage
                RGBTRIPLE triple;

                // read RGB triple from infile
                fread(&triple, sizeof(RGBTRIPLE), 1, inptr);

                for(int l = 0; l < n; l++){
                    // write RGB triple to outfile
                    fwrite(&triple, sizeof(RGBTRIPLE), 1, outptr);
                }
            }

            // skip over padding, if any
            fseek(inptr, padding, SEEK_CUR);

            // then add it back (to demonstrate how)
            for (int k = 0; k < paddingOut; k++)
            {
                fputc(0x00, outptr);
            }
        }
    }

    // close infile
    fclose(inptr);

    // close outfile
    fclose(outptr);

    // that's all folks
    return 0;
}

```