```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 infile outfile\n");
        return 1;
    }

    // remember filenames
    char* infile = argv[2];
    char* outfile = argv[3];
    int n = atoi(argv[1]);

    if (n < 1 || n > 100)
    {
        printf("The scale must be between 1 and 100\n");
        return 1;
    }

    // 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;
    BITMAPFILEHEADER bf2;
    fread(&bf, sizeof(BITMAPFILEHEADER), 1, inptr);
    bf2 = bf;

    // read infile's BITMAPINFOHEADER
    BITMAPINFOHEADER bi;
    BITMAPINFOHEADER bi2;
    fread(&bi, sizeof(BITMAPINFOHEADER), 1, inptr);
    bi2 = bi;

    // ensure infile is (likely) a 24-bit uncompressed BMP 4.0
    if (bf.bfType != 0x4d42 || bf.bfOffBits != 54 || bi.biSize != 40 ||
        bi.biBitCount != 24 || bi.biCompression != 0)
    {
        fclose(outptr);
        fclose(inptr);
        fprintf(stderr, "Unsupported file format.\n");
        return 4;
    }

    // new
    bi2.biWidth = bi.biWidth * n;
    bi2.biHeight = bi.biHeight * n;

    // determine padding for scanlines
    int padding =  (4 - (bi.biWidth * sizeof(RGBTRIPLE)) % 4) % 4;
    // new
    int newpadding =  (4 - (bi2.biWidth * sizeof(RGBTRIPLE)) % 4) % 4;

    bi2.biSizeImage = ((bi2.biWidth * sizeof(RGBTRIPLE)) + newpadding)
    * abs(bi2.biHeight);

    bf2.bfSize = bi2.biSizeImage + 54;

    // write outfile's BITMAPFILEHEADER
    fwrite(&bf2, sizeof(BITMAPFILEHEADER), 1, outptr);

    // write outfile's BITMAPINFOHEADER
    fwrite(&bi2, sizeof(BITMAPINFOHEADER), 1, outptr);

    // iterate over infile's scanlines
    for (int i = 0, biHeight = abs(bi.biHeight); i < biHeight; i++)
    {
        for (int l = 0; l < n; l++)
```

```c
        {
            // regresar scanline
            if (l != 0)
            {
                fseek(inptr, -bi.biWidth * sizeof(RGBTRIPLE), SEEK_CUR);
            }

            // iterate over pixels in scanline
            for (int j = 0; j < bi.biWidth; j++)
            {
                // temporary storage
                RGBTRIPLE triple;

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

                // write RGB triple to outfile
                for (int i2 = 0; i2 < n; i2++)
                {
                    fwrite(&triple, sizeof(RGBTRIPLE), 1, outptr);
                }
            }

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

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

    // close infile
    fclose(inptr);

    // close outfile
    fclose(outptr);

    // that's all folks
    return 0;
}
```