```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: resize scale infile outfile\n");
        return 1;
    }

    // remember scale
    int scale = atoi(argv[1]);

    // ensure n is is a positive int less than or equal to 100
    if (scale > 100 || scale < 1)
    {
        printf("Scale must be a positive int less than or equal to 100\n");
        return 1;
    }

    // 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)
```

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

```c
    // allocate storage for buffer to hold scanline
    RGBTRIPLE *buffer = malloc(sizeof(RGBTRIPLE) * (bi.biWidth));

    // iterate over infile's scanlines
    for (int i = 0, biHeight = abs(originalHeight); i < biHeight; i++)
    {
        int tracker = 0;
        // iterate over pixels in scanline
        for (int j = 0; j < originalWidth; j++)
        {
            // temporary storage
            RGBTRIPLE triple;

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

            // write pixel to buffer n times
            for(int count = 0; count < scale; count++)
            {
                *(buffer + (tracker)) = triple;
                tracker++;
            }
        }

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

        // write RGB triple to outfile
        for(int r = 0; r < scale; r++)
        {
            fwrite((buffer), sizeof(RGBTRIPLE), bi.biWidth, outptr);

                // write padding to outfile
            for (int k = 0; k < padding; k++)
                fputc(0x00, outptr);
           }
    }

    // free memory from buffer
    free(buffer);

    // close infile
    fclose(inptr);

    // close outfile
    fclose(outptr);

    // that's all folks
```

```
145.     return 0;
146. }
```