```c
/**
 * copy.c
 *
 * Computer Science 50
 * Problem Set 4
 *
 * Copies a BMP piece by piece, just because.
 *
 * Gonzalo de la Torre Amaya // A01610067
 * TecnolÃ³gico de Monterrey, SLP // CS50x
 */

#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];

    // Determinar el factor (num) como primer argumento y en int
    int num = atoi(argv[1]);

    if (num < 1 || num > 100)
    {
        printf("Valor debe ser numero positivo y menor o igual a 0");
    }
    // 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);

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

    // 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;
    }

    // Variables para el anchura
    LONG Width_old = bi.biWidth;
    bi.biWidth = bi.biWidth * num;

    // Variables para la altura
    LONG Height_old = bi.biHeight;
    bi.biHeight = bi.biHeight * num;

   // determine padding for scanlines
    int padding_new = (4 - (bi.biWidth * sizeof(RGBTRIPLE) % 4)) % 4;
    int padding_old = (4 - (Width_old * sizeof(RGBTRIPLE) % 4)) % 4;

     //
    bi.biSizeImage = abs(bi.biHeight) * (bi.biWidth * sizeof(RGBTRIPLE) + padding_new);
    bf.bfSize = bi.biSizeImage + 54;

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

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

    // iterate over infile's scanlines
    for (int i = 0, biHeight = abs(Height_old); i < biHeight; i++)
    {
        for (int g = 0; g < num; g++)
        {
            if (g != 0)
            {
                fseek(inptr, (Width_old * sizeof(RGBTRIPLE) + padding_old) * -1, SEEK_CUR);
            }
            // iterate over pixels in scanline
            for (int j = 0; j < Width_old; j++)
            {
                // temporary storage
                RGBTRIPLE triple;

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

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

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

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


    // close infile
    fclose(inptr);

    // close outfile
    fclose(outptr);

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