Table of Contents

Resize

tl;dr

Implement a program that resizes BMPs, per the below.

```
$ ./resize .25 large.bmp small.bmp
```

```
$ ./resize 4 small.bmp large.bmp
```

Background

Be sure you’re familiar with the structure of 24-bit uncompressed BMPs, as introduced in Whodunit (../../whodunit/whodunit).

Distribution

Downloading

```
$ wget http://cdn.cs50.net/2017/fall/psets/4/resize.zip (http://cdn.cs50.net/2017/f
$ unzip resize.zip
$ rm resize.zip
$ cd resize
$ ls
bmp.h  copy.c  large.bmp  small.bmp  smiley.bmp
```

Specification

Implement a program called `resize` that resizes (i.e., enlarges or shrinks) 24-bit uncompressed BMPs by a factor of `f`.

- Implement your program in a file called `resize.c` in a directory called `resize`.
- Your program should accept exactly three command-line arguments, whereby
  - the first (`f`) must be a floating-point value in (0.0, 100.0],
  - the second must be the name of a BMP to be resized, and
  - the third must be the name of the resized version to be written.

  + If your program is not executed with such, it should remind the user of correct usage, as with `fprintf` (to `stderr`), and `main` should return `1`.
- Your program, if it uses `malloc`, must not leak any memory.

Walkthrough

Usage

Your program should behave per the examples below. Assumed that the underlined text is what some user has typed.

```
$ ./resize
Usage: ./resize f infile outfile
$ echo $?
1
```

```
$ ./resize .5 large.bmp smaller.bmp
$ echo $?
0
```

```
$ ./resize 2 small.bmp larger.bmp
$ echo $?
0
```

Hints

With a program like this, we could have created `large.bmp` out of `small.bmp` by resizing the latter by a factor of 4 (i.e., by multiplying both its width and its height by 4), per the below.

```
./resize 4 small.bmp large.bmp
```

You’re welcome to get started by copying (yet again) `copy.c` and naming the copy `resize.c`. But spend some time thinking about what it means to resize a BMP, particularly if `f` is in (0.0, 1.0). (You may assume that `f` times the size of `infile` will not exceed 2^{32} - 1. As for a value of `1.0` for `f`, the result should indeed be an `outfile` with dimensions identical to `infile`'s.) How you handle floating-point imprecision and rounding is entirely up to you, as is how you handle inevitable loss of detail. Decide which of the fields in `BITMAPFILEHEADER` and `BITMAPINFOHEADER` you might need to modify. Consider whether or not you'll need to add or subtract padding to scanlines. And do be sure to support a value of `1` for `f`, the result of which should be an `outfile` with dimensions identical to `infile`'s.

If you happen to use `malloc`, be sure to use `free` so as not to leak memory. Try using `valgrind` to check for any leaks!

Testing

If you'd like to peek at, e.g., `large.bmp`'s headers (in a more user-friendly way than `xxd` allows), you may execute the below.

```
~cs50/hacker4/peek large.bmp
```

Better yet, if you'd like to compare your outfile's headers against those from the staff's solution, you might want to execute commands like the below. (Think about what each is doing.)

```
./resize 4 small.bmp student.bmp
~cs50/hacker4/resize 4 small.bmp staff.bmp
~cs50/hacker4/peek student.bmp staff.bmp
```

`check50`

```
check50 cs50/2018/x/resize/more
```

Staff's Solution

```
~cs50/hacker4/resize
```