resize

TODO

- parse float input
- update outfile’s header info
- resize horizontally
- remember padding!
- resize vertically

copy.c

- parses int input
- opens a file
- updates header info for outfile
- reads each scanline, pixel by pixel
- writes each pixel into the output file's scanline

```
cp copy.c resize.c
```

TODO

- parse float input
  - `sscanf, atof`
- update outfile's header info
- resize horizontally
- remember padding!
- resize vertically

TODO

- ☑ parse float input
- ☐ update outfile’s header info
- ☐ resize horizontally
- ☐ remember padding!
- ☐ resize vertically

bitmaps

- just an arrangement of bytes!
- how do we interpret this arrangement?
- `bmp.h`

updating header info

- new bmp → new header info
- what's changing?
  - file size
  - image size
  - width
  - height

BITMAPINFOHEADER

- `biWidth`
  - width of image (in pixels)
    - does not include padding
- `biHeight`
  - height of image (in pixels)

BITMAPINFOHEADER

- biSizeImage
  - total size of image (in bytes)
    - includes pixels and padding

```
bi.biSizeImage =
    ((sizeof(RGBTRIPLE) * bi.biWidth) + padding)
    * abs(bi.biHeight);
```

BITMAPFILEHEADER

- `bfSize`
  - total size of file (in bytes)
  - includes pixels, padding, and headers
- `bf.bfSize = bi.biSizeImage + sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);`

what's changing?

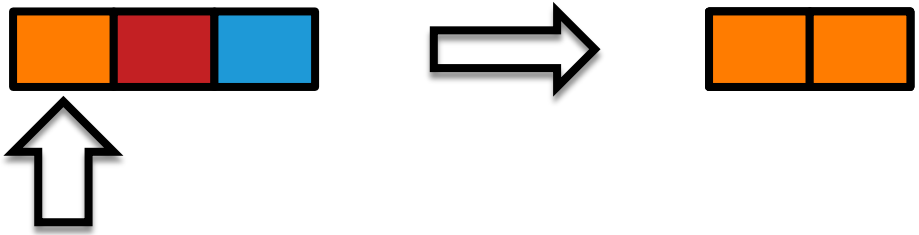
old	new
bi.biWidth	bi.biWidth *= f
bi.biHeight	bi.biHeight *= f
bi.biSizeImage	...?
bf.bfSize	...?

TODO

- ☑ parse float input
- ☑ update outfile's header info
- ☐ resize horizontally
- ☐ remember padding!
- ☐ resize vertically

resize horizontally

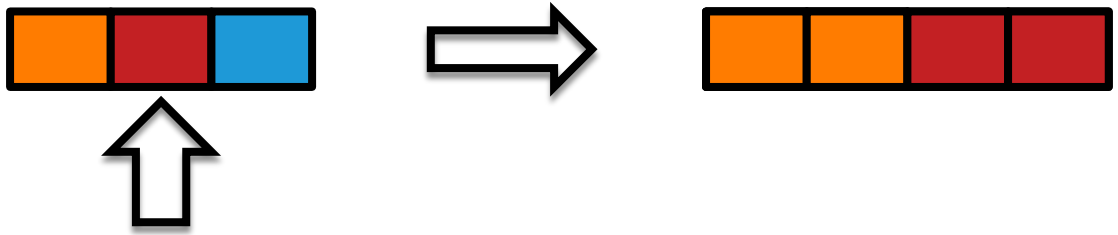
f = 2.0



```
for each pixel in row
    write f times
```

resize horizontally

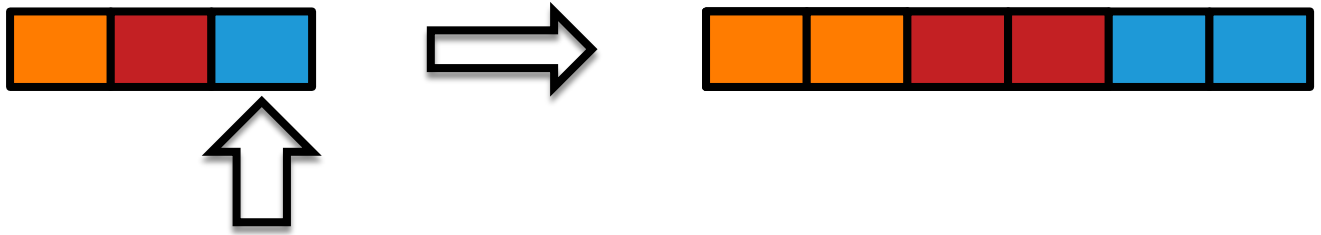
f = 2.0



```
for each pixel in row
    write f times
```

resize horizontally

f = 2.0



```
for each pixel in row
    write f times
```

reading files

```
fread(data, size, number, inptr);
```

- `data`: pointer to a struct that will contain the bytes you're reading
- `size`: size of each element to read
- `number`: number of elements to read
- `inptr`: `FILE *` to read from

writing files

```
fwrite(data, size, number, outptr);
```

- `data`: pointer to the struct that contains the bytes you're reading from
- `size`: size of each element to read
- `number`: number of elements to read
- `outptr`: `FILE *` to write to

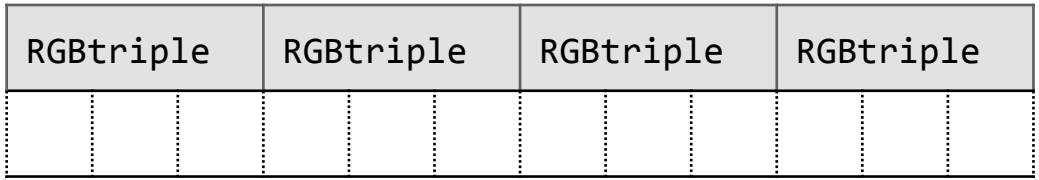
TODO

- ☑ parse float input
- ☑ update outfile's header info
- ☑ resize horizontally
- ☐ remember padding!
- ☐ resize vertically

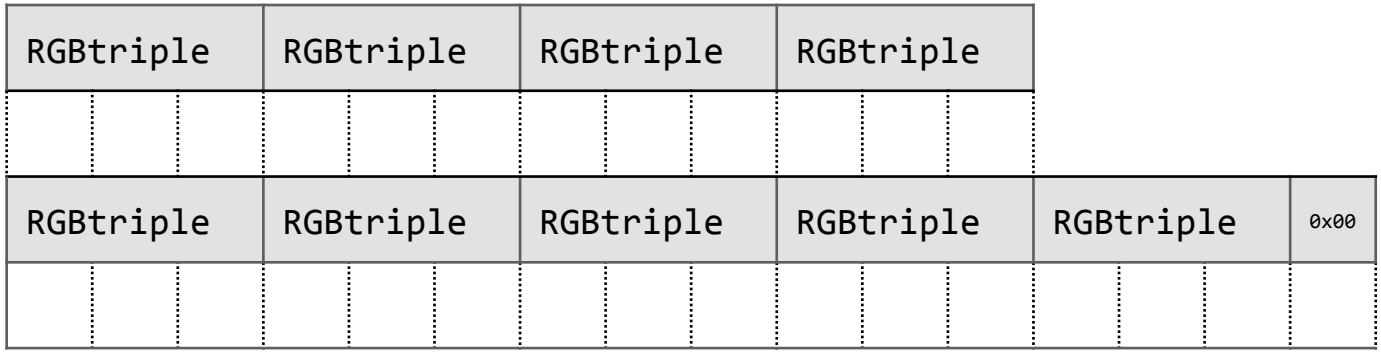
padding

- each pixel is 3 bytes
- length of each scanline must be a multiple of 4 bytes
- if the number of pixels isn't a multiple of 4, we need "padding"
  - padding is just zeros (`0x00`)

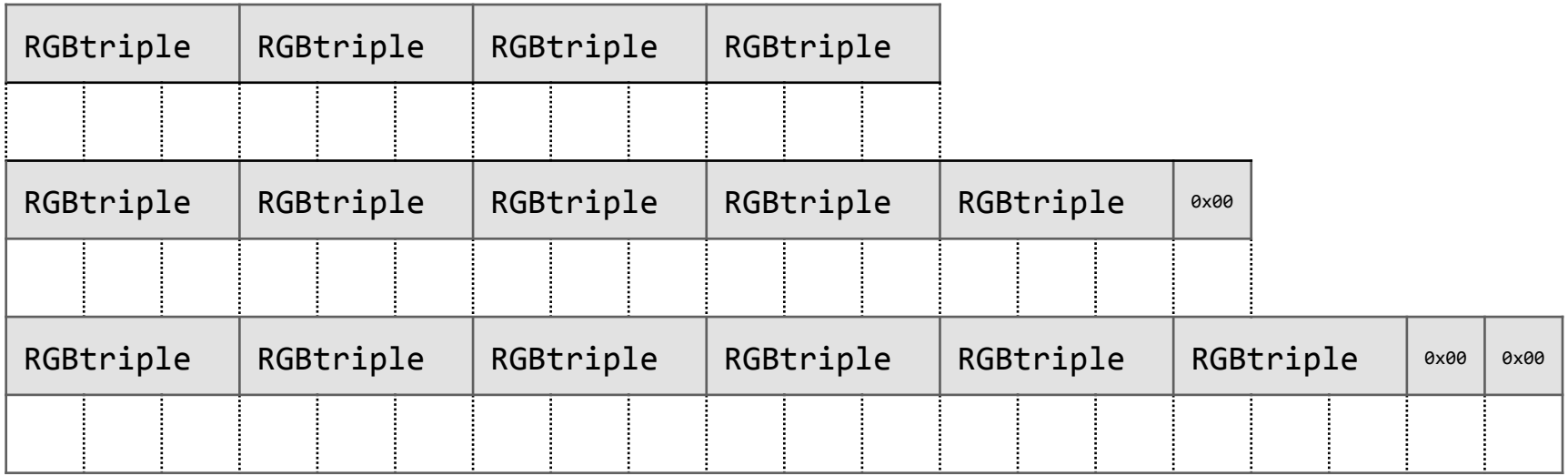
padding



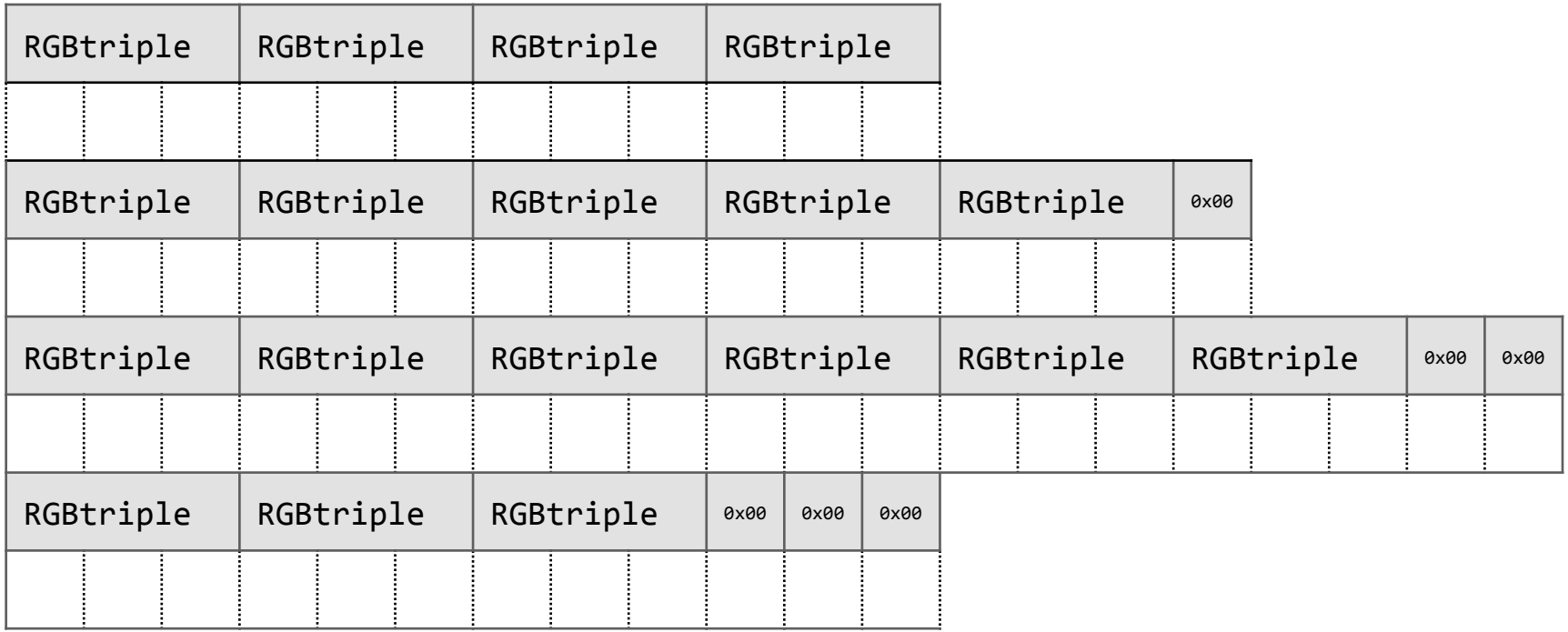
padding


RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
0x00

padding


RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
0x00
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
0x00
0x00

padding


RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
0x00
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
RGBtriple
0x00
0x00
RGBtriple
RGBtriple
RGBtriple
0x00
0x00
0x00

padding

```
padding = (4 – (bi.biWidth * sizeof(RGBTRIPLE)) % 4) % 4
```

- the outfile and infile have different widths
  - so the padding is different!

what's changing?

old	new
bi.biWidth	bi.biWidth *= f
bi.biHeight	bi.biHeight *= f
bi.biSizeImage	...?
bf.bfSize	...?

what's changing?

old	new
bi.biWidth	bi.biWidth *= n
bi.biHeight	bi.biHeight *= n
bi.biSizeImage	...?
bf.bfSize	...?
padding	...?

remember to

- skip over infile padding
  - `fseek`
- write outfile padding for each row
  - `fputc`

file position indicator

```
fseek(inptr, offset, from);
```

- inptr: `FILE *` to seek in
- offset: number of bytes to move cursor
- from:
  - `SEEK_CUR` (current position in file)
  - `SEEK_SET` (beginning of file)
  - `SEEK_END` (end of file)

writing padding

```
fputc(chr, outptr);
```

- chr: char to write
- outptr: `FILE *` to write to

```
fputc(0x00, outptr);
```

TODO:

- ☑ parse float input
- ☑ update outfile's header info
- ☑ resize horizontally
- ☑ remember padding!
- ☐ resize vertically

resize vertically

```
figure out which row should be written
seek to that row
draw pixels into outfile
write outfile row’s padding
```

TODO

- ☑ parse float input
- ☑ update outfile’s header info
- ☑ resize horizontally
- ☑ remember padding!
- ☑ resize vertically

this was resize