whodunit

TODO

- open file
- update header’s info for outfile
- read clue’s scanline, pixel by pixel
- change pixel’s color as necessary
- write verdict’s scanline, pixel by pixel

copy.c

- 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 whodunit.c
```

TODO

- open file
- update header’s info for outfile
- read clue’s scanline, pixel by pixel
- change pixel’s color as necessary
- write verdict’s scanline, pixel by pixel

opening files

```
FILE *inptr = fopen("foo.bmp", "r");
```

- opens `foo.bmp` for reading

```
FILE *outptr = fopen("bar.bmp", "w");
```

- opens `bar.bmp` for writing

TODO

- ☑ open file
- ☐ update header’s info for outfile
- ☐ read clue’s scanline, pixel by pixel
- ☐ change pixel’s color as necessary
- ☐ write verdict’s scanline, pixel by pixel

bitmaps

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

BITMAPINFOHEADER

- `biWidth`
  - width of image (in pixels)
    - does not include padding
- `biHeight`
  - height of image (in pixels)
- `biSizeImage`
  - total size of image (in bytes)
    - includes pixels and padding

BITMAPFILEHEADER

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

TODO

- ☑ open file
- ☑ update header’s info for outfile
- ☐ read clue’s scanline, pixel by pixel
- ☐ change pixel’s color as necessary
- ☐ write verdict’s scanline, pixel by pixel

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
  - `sizeof`
- `number`: number of elements to read
- `inptr`: `FILE *` to read from

TODO

- ☑ open file
- ☑ update header’s info for outfile
- ☑ read clue’s scanline, pixel by pixel
- ☐ change pixel’s color as necessary
- ☐ write verdict’s scanline, pixel by pixel

pixel color

- each color is represented by 3 bytes:
  - amount of blue
  - amount of green
  - amount of red

`ff0000` → blue

`ffffff` → white

smiley.bmp

ffffff ffffff 0000ff 0000ff 0000ff 0000ff ffffff ffffff

ffffff 0000ff ffffff ffffff ffffff ffffff 0000ff ffffff

0000ff ffffff 0000ff ffffff ffffff 0000ff ffffff 0000ff

0000ff ffffff ffffff ffffff ffffff ffffff ffffff 0000ff

0000ff ffffff 0000ff ffffff ffffff 0000ff ffffff 0000ff

0000ff ffffff ffffff 0000ff 0000ff ffffff ffffff 0000ff

ffffff 0000ff ffffff ffffff ffffff ffffff 0000ff ffffff

ffffff ffffff 0000ff 0000ff 0000ff 0000ff ffffff ffffff

RGBTRIPLE

- struct to represent pixels

```
// make a green pixel

RBGTRIPLE triple;
triple.rgbtBlue = 0x00;
triple.rgbtGreen = 0xff;
triple.rgbtRed = 0x00;
```

RGBTRIPLE

```
if (triple.rgbtBlue == 0xff)
{
    printf("I'm feeling blue!\n");
}
```

RGBTRIPLE

```
if (triple.rgbtBlue == 0xff)
{
    printf("I'm feeling blue!\n");
}

// change to pure blue
triple.rgbtGreen = 0x00;
triple.rgbtRed = 0x00;
```

remove red from each pixel

0000ff 0000ff 8000ff 8000ff 8000ff 8000ff 0000ff 0000ff

0000ff 8000ff 0000ff 0000ff 0000ff 0000ff 8000ff 0000ff

8000ff 0000ff 0090ff 0000ff 0000ff 0090ff 0000ff 8000ff

8000ff 0000ff 0000ff 0000ff 0000ff 0000ff 0000ff 8000ff

8000ff 0000ff 0090ff 0000ff 0000ff 0090ff 0000ff 8000ff

8000ff 0000ff 0000ff 0090ff 0090ff 0000ff 0000ff 8000ff

0000ff 8000ff 0000ff 0000ff 0000ff 0000ff 8000ff 0000ff

0000ff 0000ff 8000ff 8000ff 8000ff 8000ff 0000ff 0000ff

remove red from each pixel

```
000000 000000 800000 800000 800000 800000 000000 000000
000000 800000 000000 000000 000000 000000 800000 000000
800000 000000 009000 000000 000000 009000 000000 800000
800000 000000 000000 000000 000000 000000 000000 800000
800000 000000 009000 000000 000000 009000 000000 800000
800000 000000 000000 009000 009000 000000 000000 800000
000000 800000 000000 000000 000000 000000 800000 000000
000000 000000 800000 800000 800000 800000 000000 000000
```

0000ff → ffffff

0000ff 0000ff 8000ff 8000ff 8000ff 8000ff 0000ff 0000ff

0000ff 8000ff 0000ff 0000ff 0000ff 0000ff 8000ff 0000ff

8000ff 0000ff 0090ff 0000ff 0000ff 0090ff 0000ff 8000ff

8000ff 0000ff 0000ff 0000ff 0000ff 0000ff 0000ff 8000ff

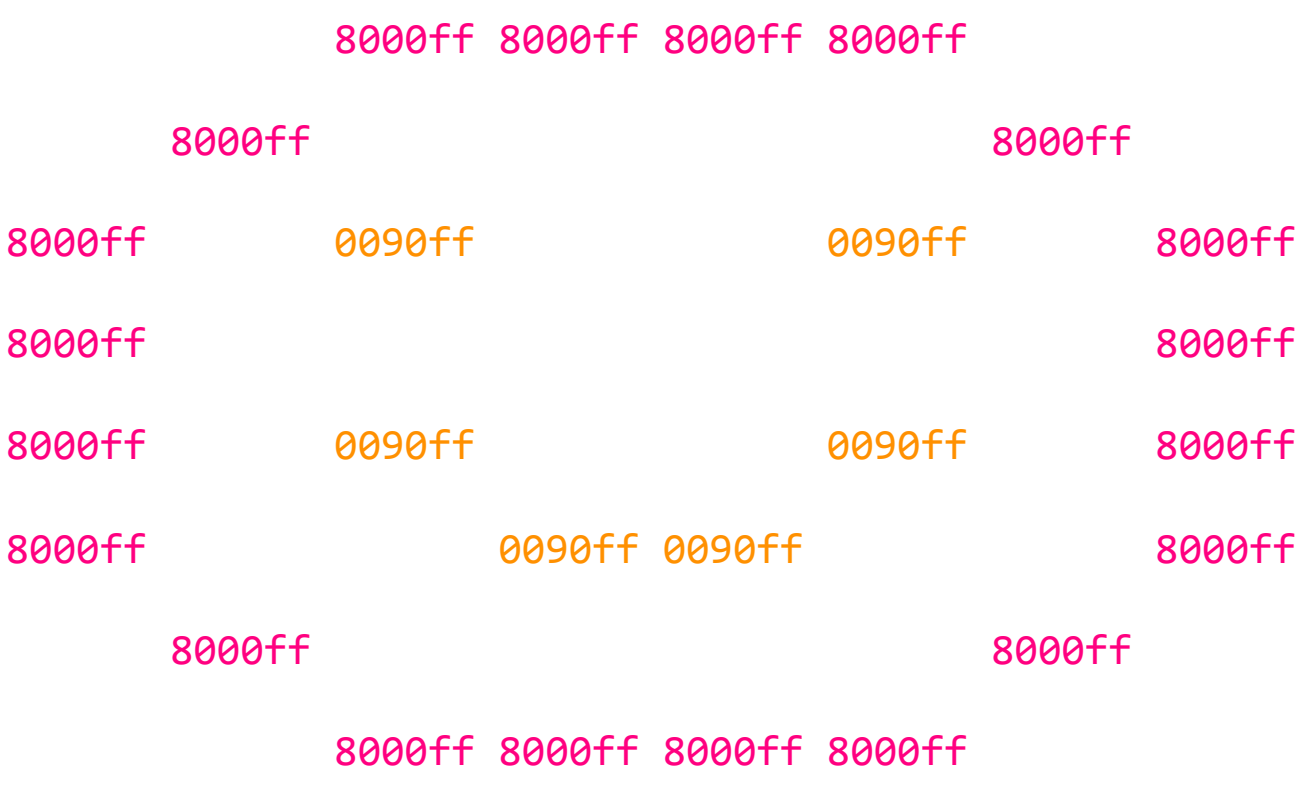
8000ff 0000ff 0090ff 0000ff 0000ff 0090ff 0000ff 8000ff

8000ff 0000ff 0000ff 0090ff 0090ff 0000ff 0000ff 8000ff

0000ff 8000ff 0000ff 0000ff 0000ff 0000ff 8000ff 0000ff

0000ff 0000ff 8000ff 8000ff 8000ff 8000ff 0000ff 0000ff

0000ff → ffffff



TODO

- ☑ open file
- ☑ update header’s info for outfile
- ☑ read clue’s scanline, pixel by pixel
- ☑ change pixel’s color as necessary
- ☐ write verdict’s scanline, pixel by pixel

writing files

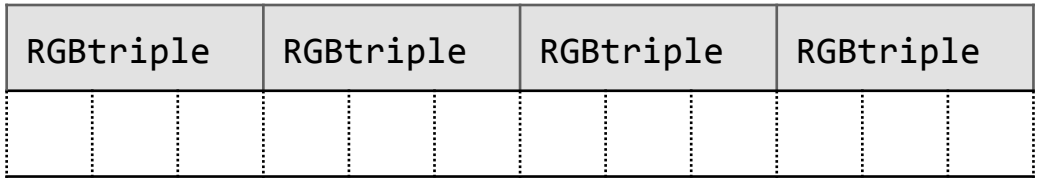
```
fwrite(data, size, number, outptr);
```

- `data`: pointer to the struct that contains the bytes you're writing
- `size`
- `number`
- `outptr`: `FILE *` to write to

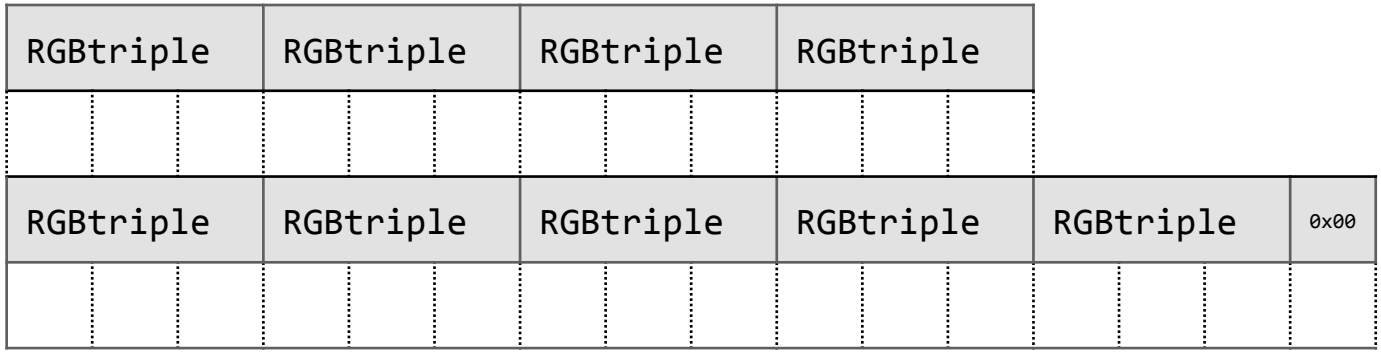
padding

- 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`)

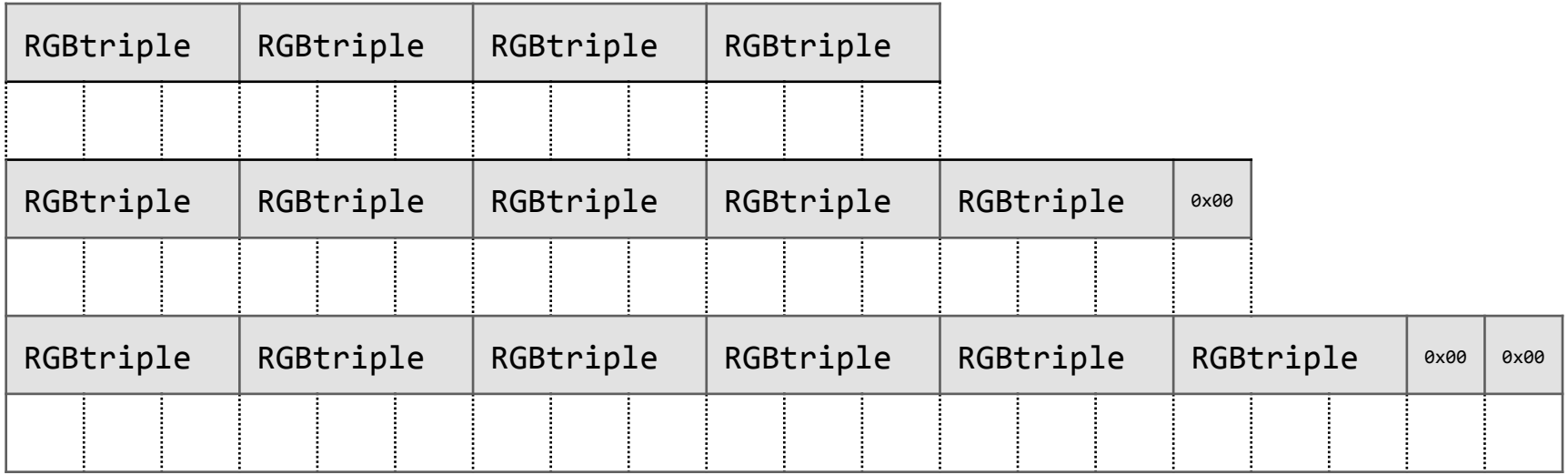
padding



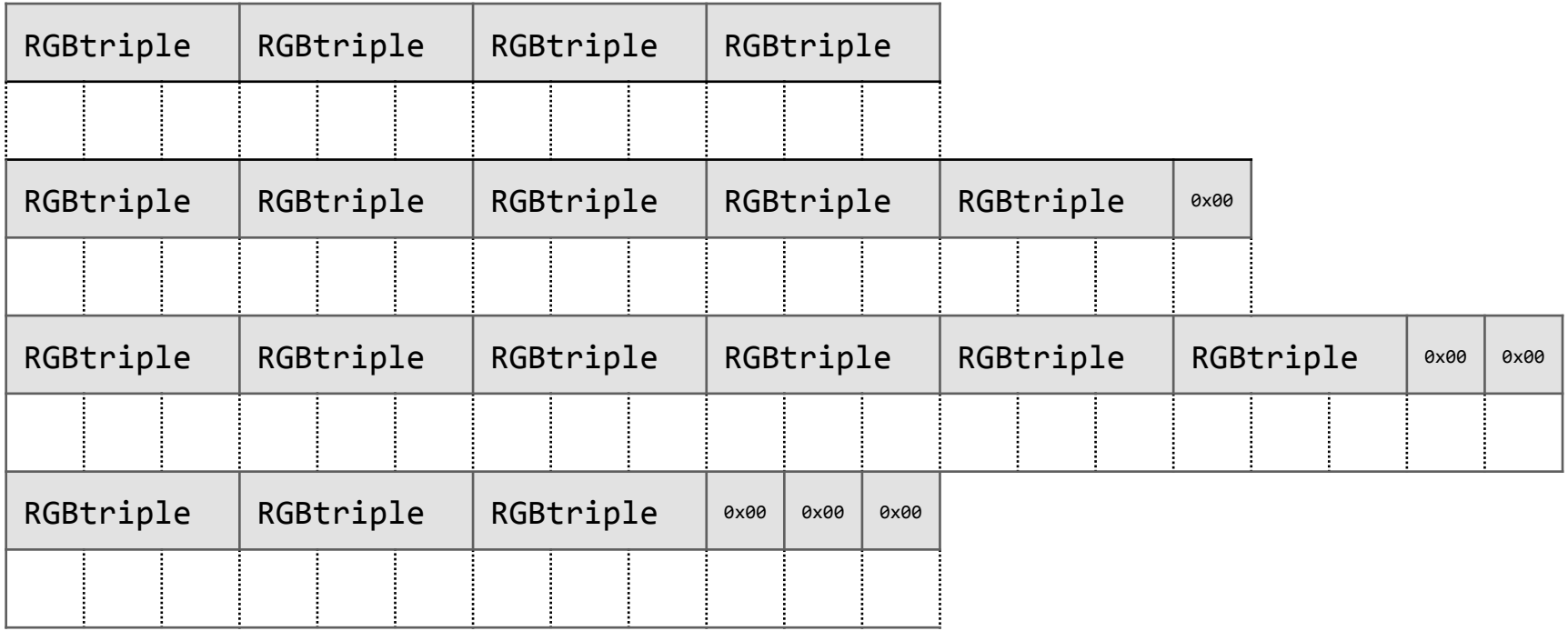
padding


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
```

- clue and verdict have the same width
  - so the padding is the same!
- padding isn’t an `RGBTRIPLE`
  - we can’t `fread` padding

writing padding

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

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

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

file position indicator

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

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

TODO

- ☑ open file
- ☑ update header’s info for outfile
- ☑ read clue’s scanline, pixel by pixel
- ☑ change pixel’s color as necessary
- ☑ write verdict’s scanline, pixel by pixel

this was whodunit