# pset 4: Forensics

Zamyla Chan | zamyla@cs50.net

# Toolbox

- □ `update50`
- □ File I/O
- □ `copy.c`
- □ bitmaps
  - ▣ padding!
- □ JPEGs

# pset 4

0. A Section of Questions
1. Whodunit
2. Resize
3. Recover

# File I/O Toolbox

- □ fopen
- □ fread
- □ fwrite
- □ fputc
- □ fseek
- □ sprintf
- □ fclose
- □ feof

# Opening Files

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

- Opens foo.bmp for reading

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

- Opens bar.bmp for writing

# Reading Files

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

- &data: pointer to a struct which 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

# Reading Files

```
fread(&data, sizeof(DOG), 2, inptr);
```

vs.

```
fread(&data, 2 * sizeof(DOG), 1, inptr);
```

# Writing Files

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

□ &data: pointer to the struct that contains the bytes you’re reading from

□ size

□ number

□ outptr: FILE* to write to

# Writing Files

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

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

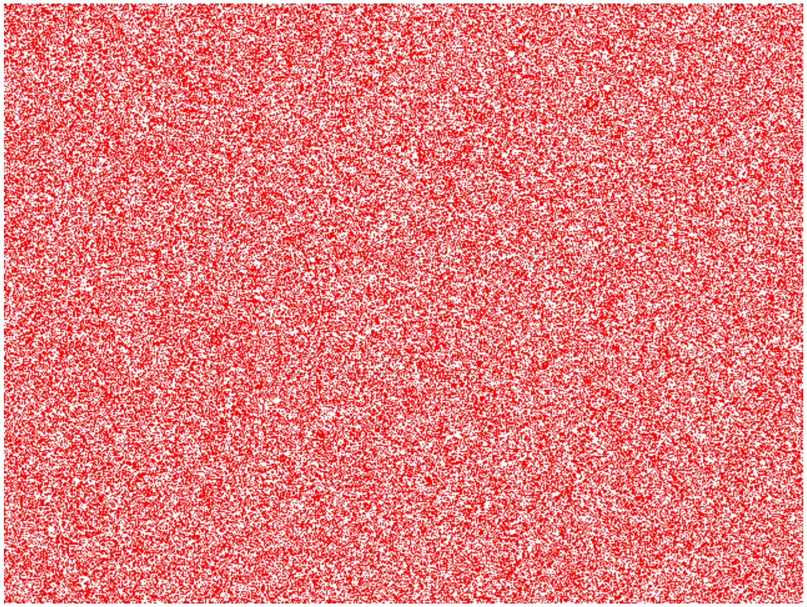
# File Position Indicator

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

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

# Whodunit

```
./whodunit clue.bmp verdict.bmp
```

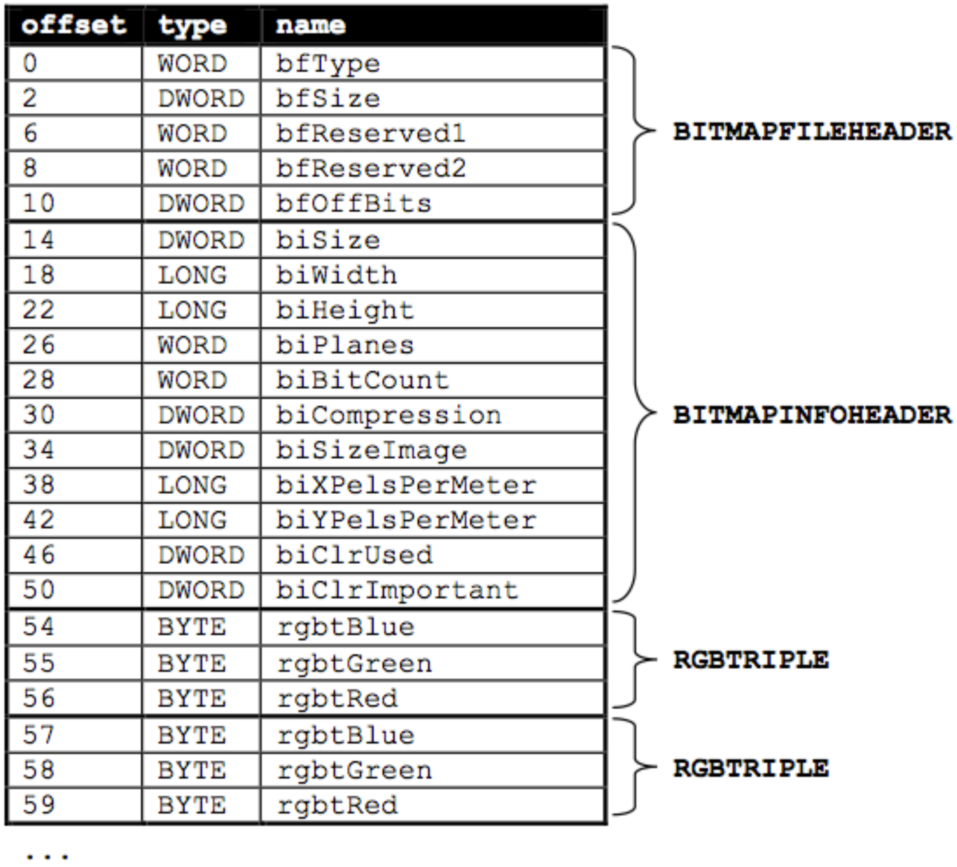


# TODO

- Understand the structure of bitmaps
  - Metadata
  - Pixel colors
  - Padding

# Bitmaps

- An arrangement of bytes



| offset | type | name | |
|---|---|---|---|
| 0 | WORD | bfType | BITMAPFILEHEADER |
| 2 | DWORD | bfSize | |
| 6 | WORD | bfReserved1 | |
| 8 | WORD | bfReserved2 | |
| 10 | DWORD | bfOffBits | |
| 14 | DWORD | biSize | BITMAPINFOHEADER |
| 18 | LONG | biWidth | |
| 22 | LONG | biHeight | |
| 26 | WORD | biPlanes | |
| 28 | WORD | biBitCount | |
| 30 | DWORD | biCompression | |
| 34 | DWORD | biSizeImage | |
| 38 | LONG | biXPelsPerMeter | |
| 42 | LONG | biYPelsPerMeter | |
| 46 | DWORD | biClrUsed | |
| 50 | DWORD | biClrImportant | |
| 54 | BYTE | rgbtBlue | RGBTRIPLE |
| 55 | BYTE | rgbtGreen | |
| 56 | BYTE | rgbtRed | |
| 57 | BYTE | rgbtBlue | RGBTRIPLE |
| 58 | BYTE | rgbtGreen | |
| 59 | BYTE | rgbtRed | |

...

- bmp.h

# Header

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

# 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
```

# Padding

- Each pixel is 3 bytes
- Size 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**)

# xxd –c 12 -g 3 -s 54 small.bmp

```
0000036: 00ff00 00ff00 00ff00 000000  ......
0000042: 00ff00 ffffff 00ff00 000000  ......
000004e: 00ff00 00ff00 00ff00 000000  ......
```

# xxd -c 36 -g 3 -s 54 large.bmp

```
0000036: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
000005a: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
000007e: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
00000a2: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
00000c6: 00ff00 00ff00 00ff00 00ff00 ffffff ffffff ffffff ffffff 00ff00 00ff00 00ff00 00ff00 ......
00000ea: 00ff00 00ff00 00ff00 00ff00 ffffff ffffff ffffff ffffff 00ff00 00ff00 00ff00 00ff00 ......
000010e: 00ff00 00ff00 00ff00 00ff00 ffffff ffffff ffffff ffffff 00ff00 00ff00 00ff00 00ff00 ......
0000132: 00ff00 00ff00 00ff00 00ff00 ffffff ffffff ffffff ffffff 00ff00 00ff00 00ff00 00ff00 ......
0000156: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
000017a: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
000019e: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
00001c2: 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 00ff00 ......
```

# RGBTRIPLE

- struct to represent pixels

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

What color is this?

# TODO

- ☑ Understand the structure of bitmaps
- □ Make the bmp readable!
  - Open `clue.bmp` file
  - Read each scanline, pixel by pixel
  - Change pixels as needed
  - Write the scanline into **`verdict.bmp`**, pixel by pixel

# `copy.c`

- Opens a file
- Read each scanline, pixel by pixel
- Copies each pixel into the output file’s scanline

```
cp copy.c whodunit.c
```

# TODO

- ☑ Understand the structure of bitmaps
- □ Make the bmp readable
  - ☑ Open `clue.bmp` file
  - ☑ Read each scanline, pixel by pixel
  - ■ **Change pixels as needed**
  - ☑ Write the scanline into `verdict.bmp`, pixel by pixel

# Changing pixel color

- For a given pixel `triple`, you can access:
  - `triple.rgbtBlue`
  - `triple.rgbtGreen`
  - `triple.rgbtRed`
- Hmm, that's handy!

# Resize

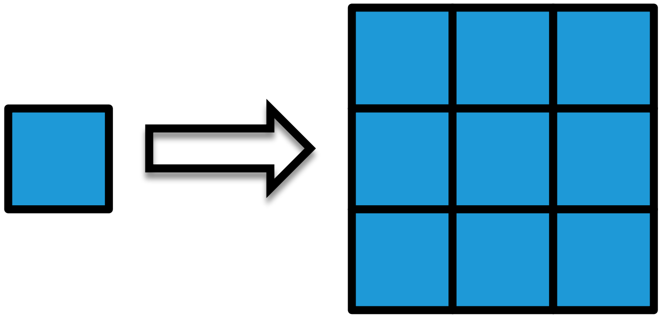
Scale bmp image by a factor of n

Usage: `./resize n infile outfile`

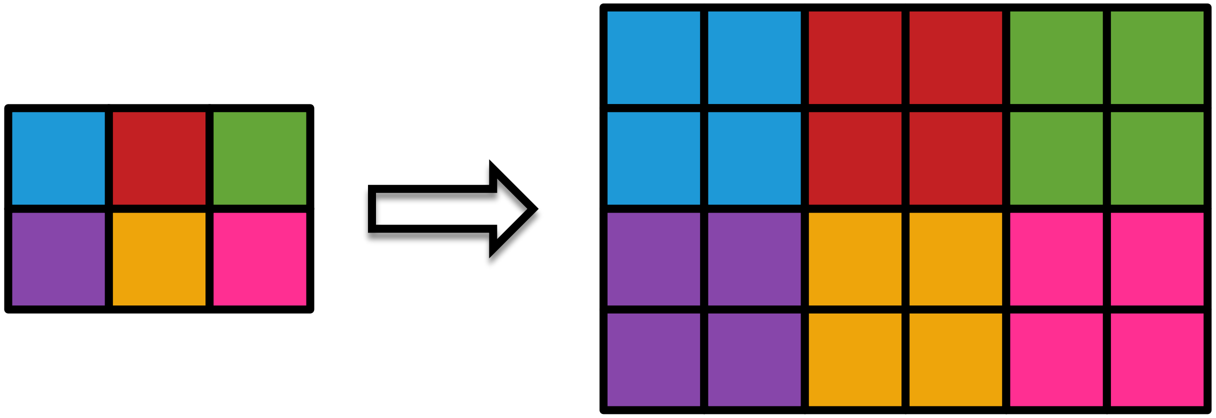
# Resize

- □ Every pixel repeated n times
- □ Every row repeated n times

n = 3



n = 2



# TODO

- □ Open file
- □ Update header info for outfile

# Update header info

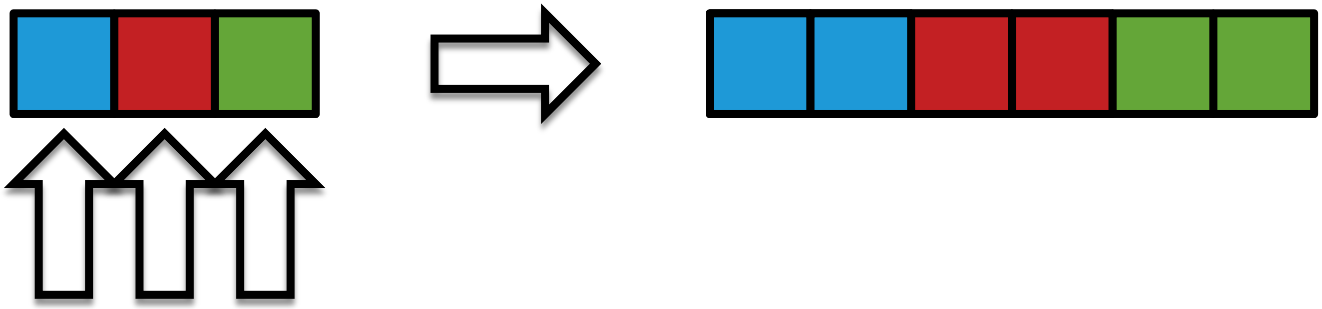
- New bmp → new header info
- What's changing?
  - file size
  - image size
  - width
  - height
- Which structs need to be changed? How do you access those variables?

# TODO

- ☑ Open file
- ☑ Update header info for outfile
- ☑ Read each scanline, pixel by pixel
- □ Resize Horizontally
- □ Padding!
- □ Resize Vertically

# Resize Horizontally

n = 2



# TODO

- ☑ Open file
- ☑ Update header info for outfile
- ☑ Read each scanline, pixel by pixel
- ☑ Resize Horizontally
- □ Padding!
- □ Resize Vertically

# Padding

- Padding isn't an `RGBTRIPLE`
  - we can't `fread` padding
- Infile image and outfile image have different padding!

# Padding

- If the number of pixels isn’t a multiple of 4, we need to **add** “padding” such that the scanline has a multiple of 4 bytes
  - Padding is just zeros (`0x00`)
- Hmm… a formula would come in handy!

# TODO

- ☑ Open file
- ☑ Update header info
- ☑ Read each scanline, pixel by pixel
- ☑ Resize Horizontally
- ☑ Padding!
- □ Resize Vertically

# Resize Vertically

- Several different ways to do this!

1. "Rewrite" methods
   - Remember pixels in an array
   - Write array as many times as needed
2. "Re-copy" methods
   - Go back to the start of the original row
   - Repeat the horizontal resizing

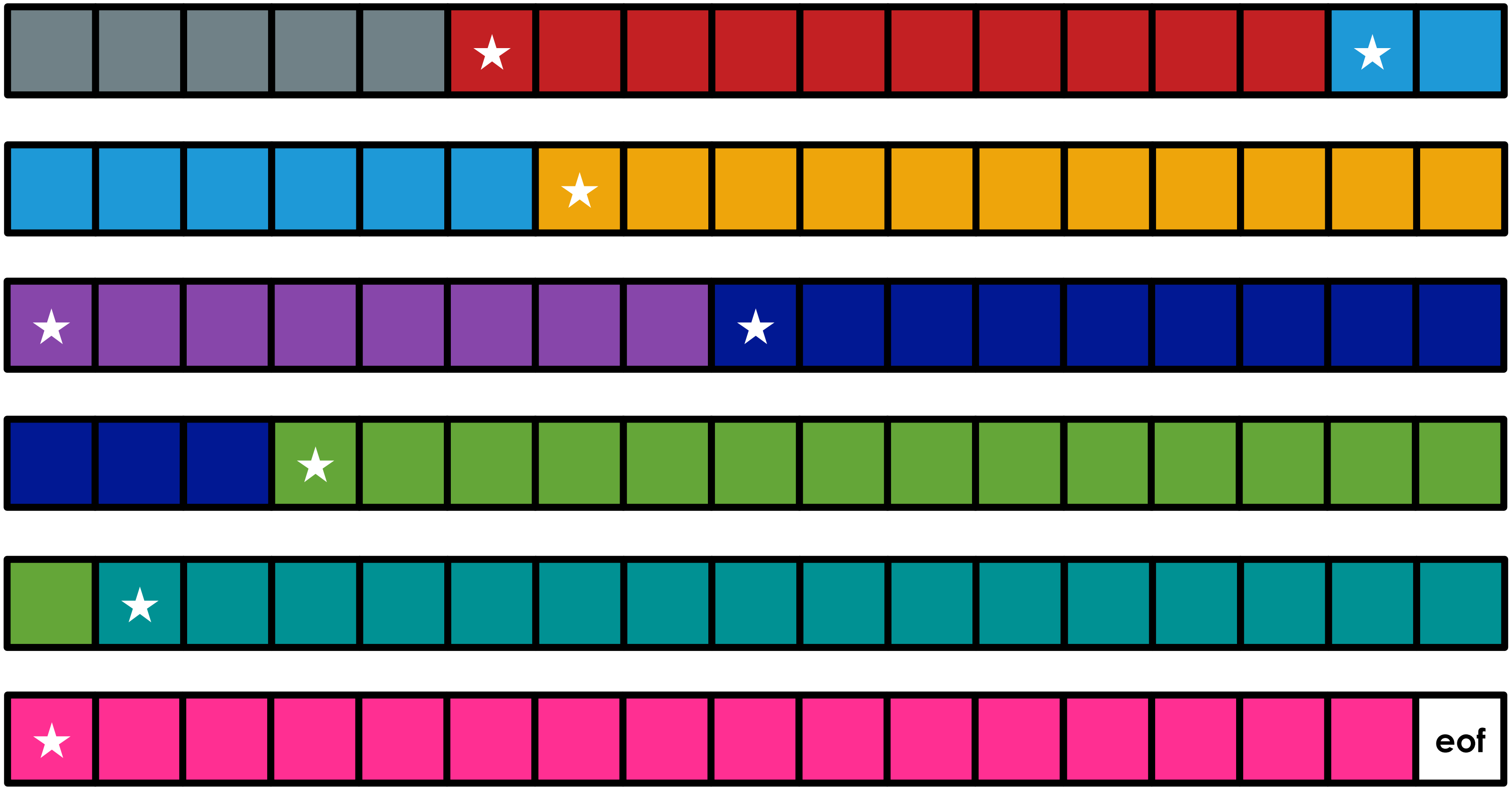
# Recover

# JPEGs

- JPEGs are also just sequences of bytes
- Each JPEG starts with either:
  - `0xff 0xd8 0xff 0xe0`
  - `0xff 0xd8 0xff 0xe1`
- JPEGs are stored side-by-side on the CF card

# Recover

Each □ represents 512 bytes



# Pseudocode

```
open card file
repeat until end of file
   read 512 bytes into a buffer
   start of a new jpg?
         yes → …
         no  → …
   already found a jpg?
         no  → …
         yes → …
close last jpg
close card file
```

# Making JPG files

- Filenames: `###.jpg`
- JPEGs named in the order in which they are found, starting at 0.
  - (So keep track!)
- `sprintf(title,"%d.jpg", 2);`
  - title: char array to store the resultant string
  - Hmm… this gives `2.jpg`, not `002.jpg`
    - How long is each array?

# Contest!

this was walkthrough 4