REC3: File I/O in C

Shinwoo Kim
Teaching Assistant
shinwookim@pitt.edu
https://sites.pitt.edu/~shk148/

Spring 2023, Term 2234
Friday 12 PM Recitation
Feb 3rd, 2023

Department of Computer Science
School of Computing & Information
University of Pittsburgh

- Standard Integer Sizes
- Reading/writing files in C
  - `fopen()`, `fread()`, `fwrite()`, `fseek()`, `fclose()`
- Project 1 discussion

Fixing sizes of data types

- **In lecture, you saw that C's data types don't have a defined size**
  - There were minimum requirements, but it was really *machine-dependent*
  - `int` must be greater than a `short` ⇒ but just how greater?
    - Who knows! The compiler does
  - This may cause issues, if we need to represent a integer as exactly N bytes
- **`libc` to the rescue!**
  - The C standard library provides a header file will allows us to do this
  - `#include <stdint.h>`
    - `int8_t, int16_t, int32_t, int64_t`
    - `uint8_t, uint16_t, uint32_t, uint64_t`
    - `int_least8_t,...`
  - https://www.gnu.org/software/libc/manual/html_node/Integers.html

Basics of File I/O

Reading and writing files in C

```
[ ~ ]$ hexdump -C binary_file_example
00000000  41 00 41 00 00 00 42 00  42 00 00 00 43 00 43 00  |A.A...B.B...C.C.|
00000010  00 00 44 00 44 00 00 00  45 00 45 00 00 00 46 00  |..D.D...E.E...F.|
00000020  46 00 00 00 47 00 47 00  00 00 48 00 48 00 00 00  |F...G.G...H.H...|
00000030  49 00 49 00 00 00 4a 00  4a 00 00 00 4b 00 4b 00  |I.I...J.J...K.K.|
00000040  00 00 4c 00 4c 00 00 00  4d 00 4d 00 00 00 4e 00  |..L.L...M.M...N.|
00000050  4e 00 00 00 4f 00 4f 00  00 00 50 00 50 00 00 00  |N...O.O...P.P...|
00000060  51 00 51 00 00 00 52 00  52 00 00 00 53 00 53 00  |Q.Q...R.R...S.S.|
00000070  00 00 54 00 54 00 00 00  55 00 55 00 00 00 56 00  |..T.T...U.U...V.|
00000080  56 00 00 00 57 00 57 00  00 00 58 00 58 00 00 00  |V...W.W...X.X...|
00000090  59 00 59 00 00 00 5a 00  5a 00 00 00              |Y.Y...Z.Z...|
0000009c
```

What we have seen so far …

- **In lab 0, you (maybe unknowingly) used command line arguments to interact with your program**
  - When you ran `./calculator` `4 5 +`
- **In lab 1, you used the standard I/O stream(s)**
  - `printf()`, `scanf()`, and other `<stdio.h>` functions
- **This week, we'll learn to read and write from files on your computer**
  - which you will need to do for the first project

What is a file?

- **In C, a file is simply a sequence (*stream*) of bytes:**
  - Text files (or ASCII file) is sequence of ASCII code, i.e., each byte is the 8 bit code of a character (*.txt, *.c, etc.)
  - Binary files contains the original binary number as stored in memory (*.pdf, *.doc, *.jpg, etc.)

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
```

A hex dump of the 318 byte Wikipedia favicon

Opening files with fopen()

```
FILE *fopen(const char * pathname, const char*mode);
```

```
> FILE* pt = fopen("E:\\PATH\program.txt","w");
```

- opens the file whose name is the string pointed to by pathname and associates a stream with it.
- returns a pointer (of type FILE) to the stream

Opening Files with `fopen()`

```
*fopen(const char * filename, const char * mode );
```

Modes:

- `r`: opens an existing file for reading.
- `w`: opens a file for writing.
  - If `filename` does not exist, new file is created.
  - starts writing at the beginning of file.
- `a`: opens a text file for writing in appending mode.
  - If `filename` does not exist, new file is created.
  - start appending content in the existing file content.
- `r+`: opens a file for both reading and writing.
- `b`: indicates file is a binary file
- and more…
  - Use **`man fopen`** to learn more

`fread()` lets us read, `fwrite()` lets us write

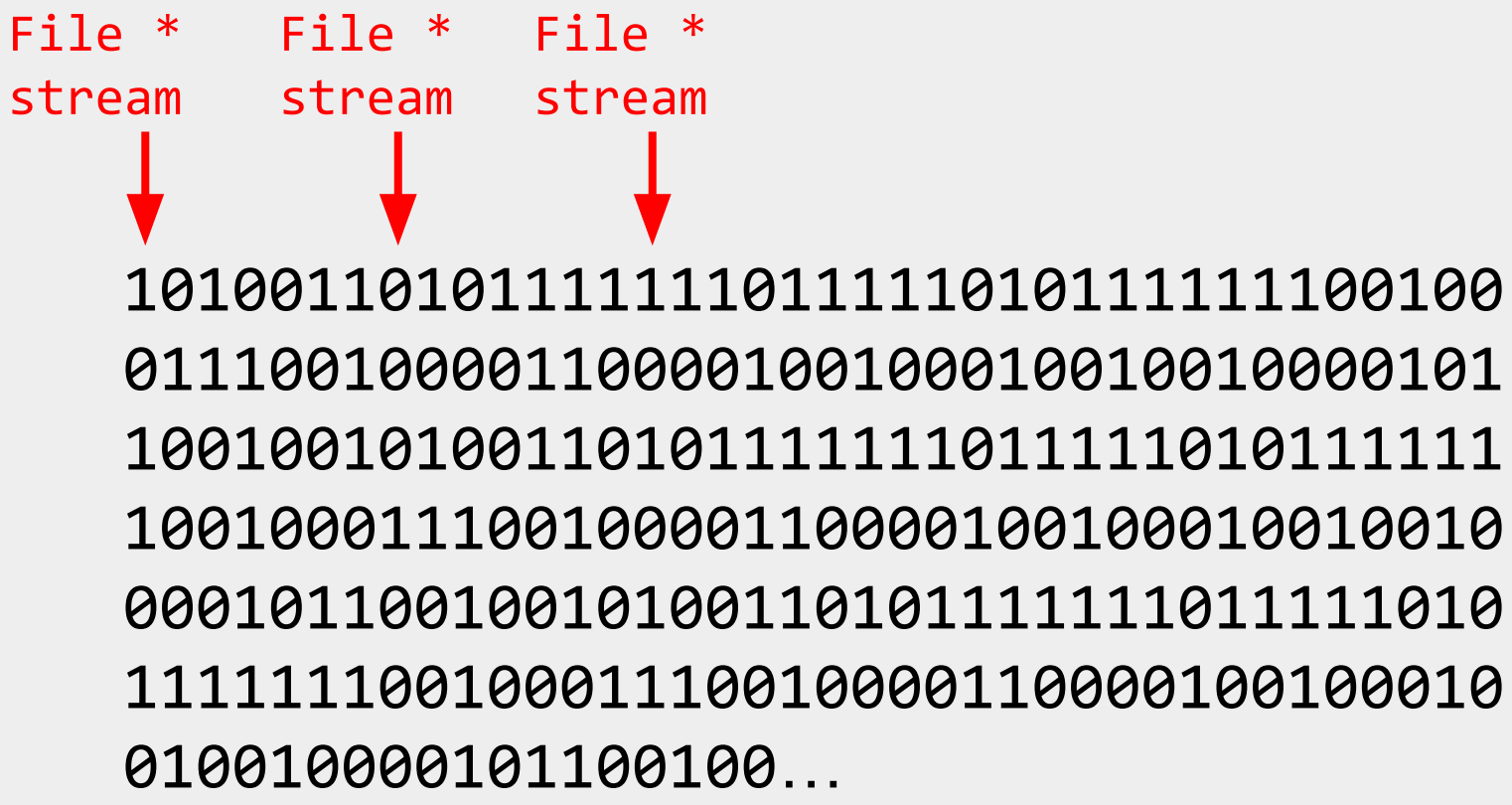
`fread(void *ptr, size_t size, size_t nmemb, FILE* stream);`

- reads `nmemb` items of data each `size` bytes long
- from `stream`
- stores them at the location given by `ptr`.

`fwrite(const void *ptr, size_t size, size_t nmemb, FILE * stream);`

- writes `nmemb` items of data each `size` bytes
- to the `stream`
- from the location given by `ptr`.

Reading and writing moves the pointer




```
> fread(ptr1, 1, 1, stream)
> fwrite(ptr1, 1, 1, stream)
```

We can rewind or fast-forward with `fseek()`

```
fseek(FILE *stream, long offset, int whence);
```

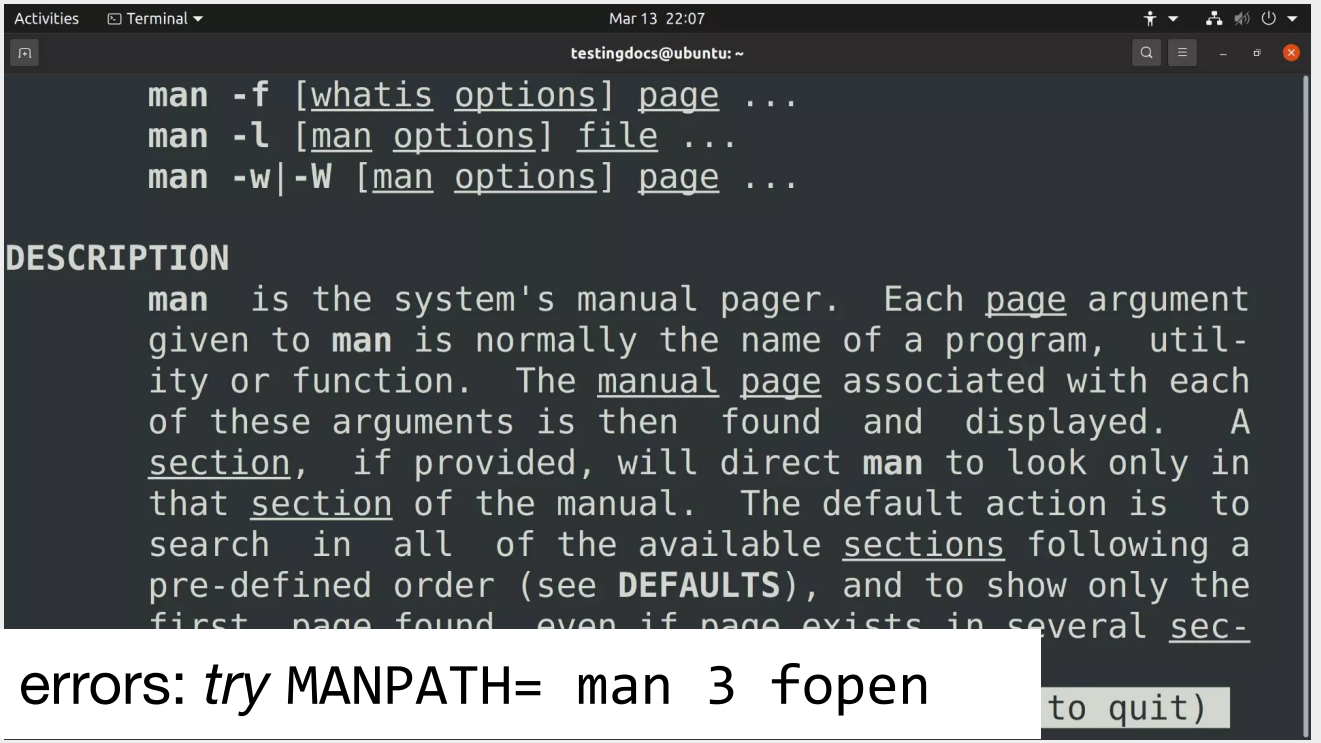
- sets the file position indicator for the stream
- new position (measured in bytes) = `offset` + `whence`.

`whence`:

- `SEEK_SET` - from start-of-file
- `SEEK_CUR` - from current position
- `SEEK_END` - from end-of-file

Always remember to save (and close) your files!

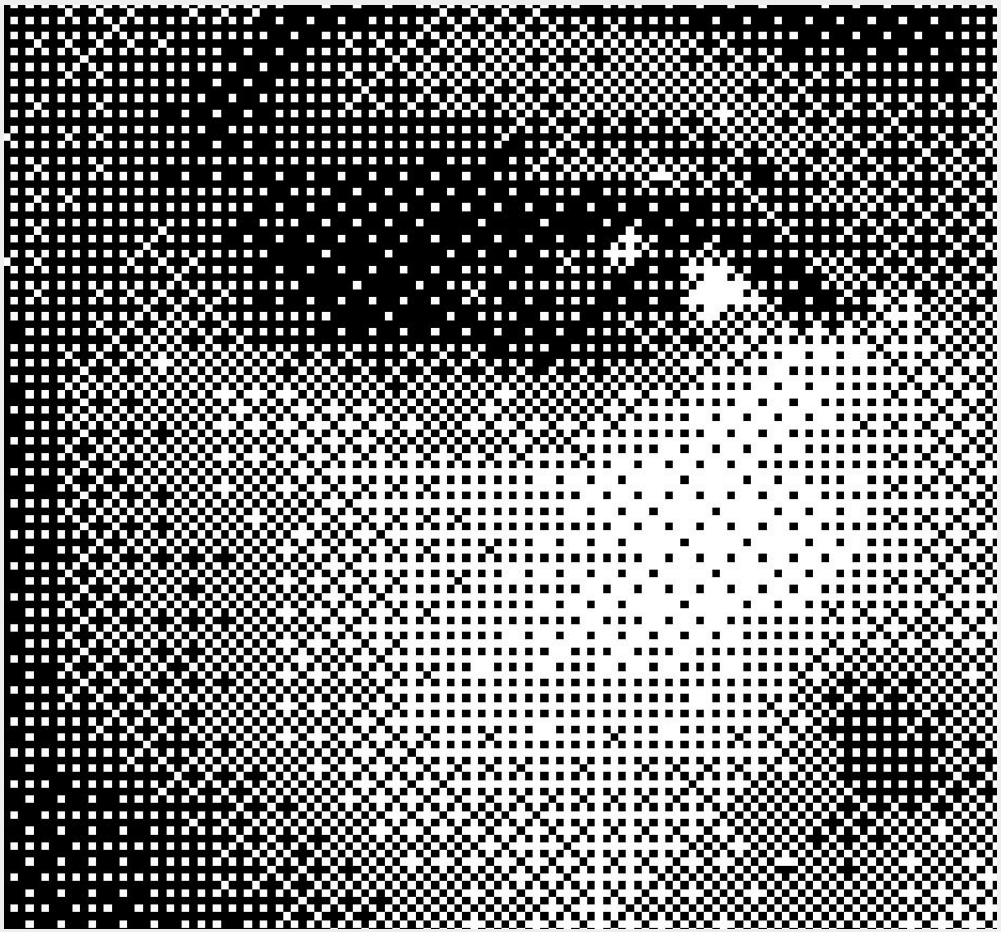
- **Just like memory leaks, you may also get file handle leaks**
  - If you use `fopen()`, always remember to `fclose()`
    - `int fclose(FILE* filePointer)`
      - returns `0` on success!
- **If you are confused about these functions → Consult the MANual**



Thoth `man` errors: *try* `MANPATH= man 3 fopen`

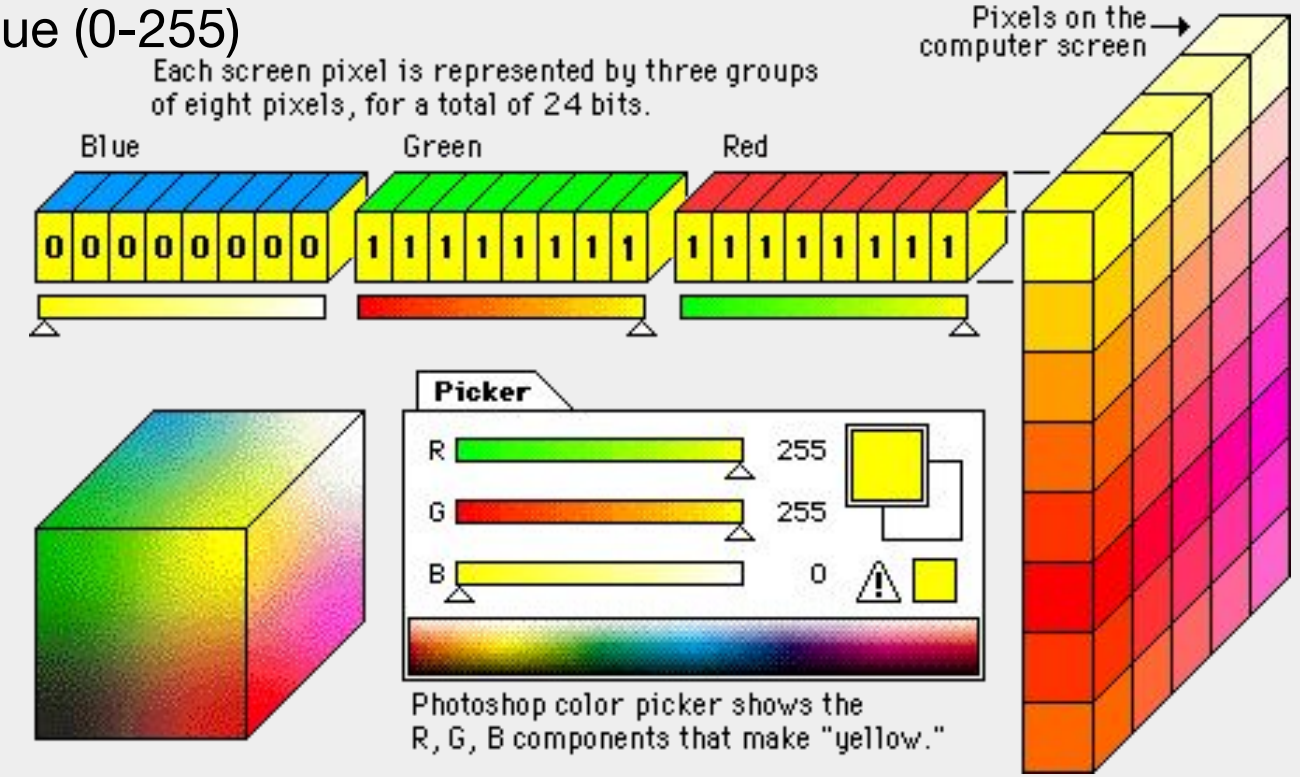
Project 1

Hiding data & reading files



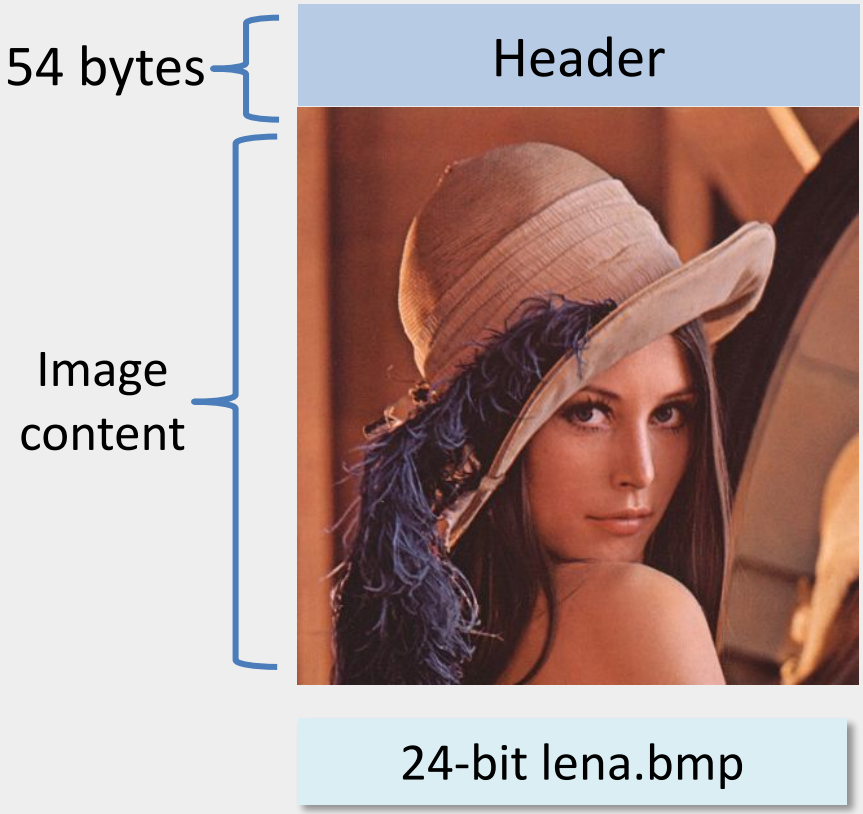
*.BMP Bitmap Pictures

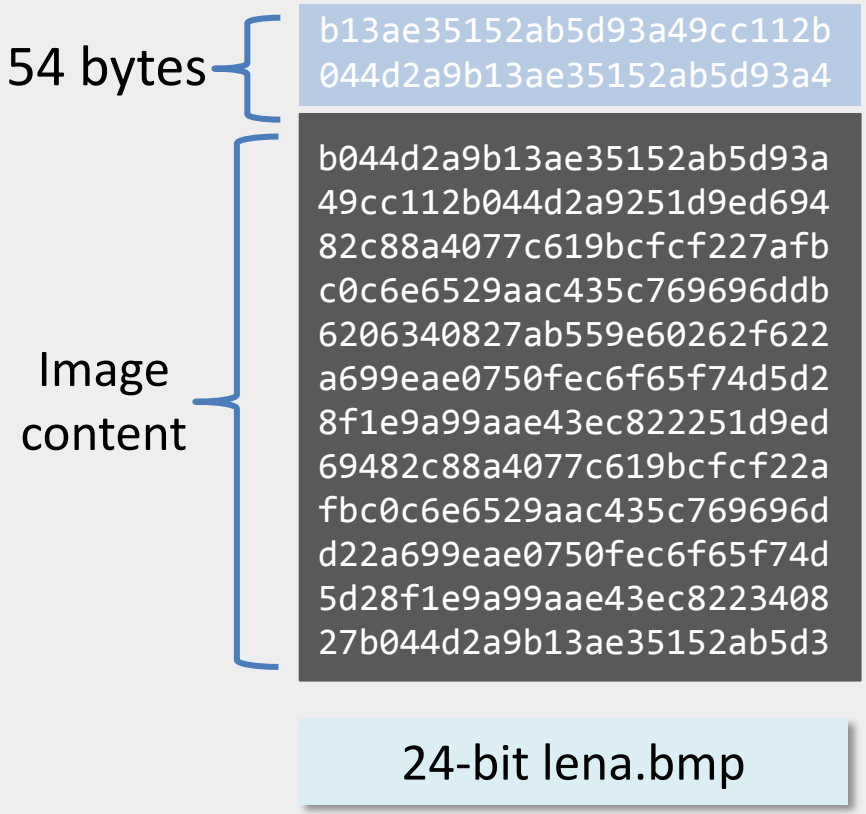
- **Bitmap Image File**
  - Container format for a big array of pixels (picture cells)
  - Many different formats; we will focus on **Windows Bitmap** (24-bit RGB color)
    - Each pixel is represented by a 24-bit number:
    - 8 bit for Red (0-255)
    - 8 bit for Green (0-255)
    - 8 bit for Blue (0-255)



TODO 1: Reading the metadata

At the beginning of the BMP is a header which contains metadata (key details about the picture)





TODO 1: Reading the metadata

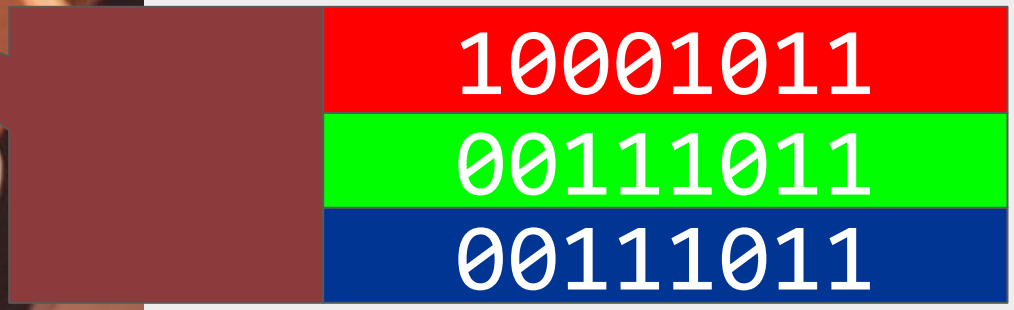
For phase 1, you are expected to read the header(s), print and validate them.

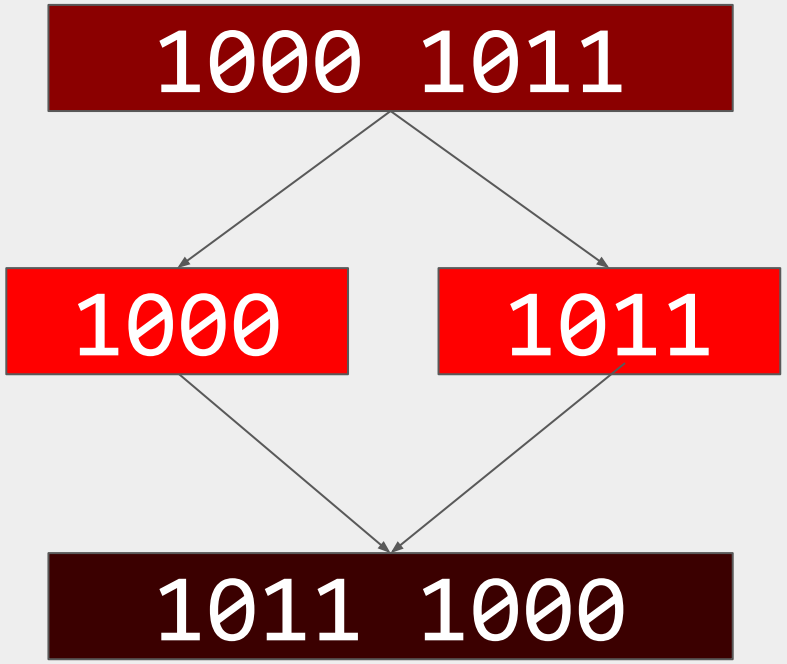
Header file (Image profile)

- File header (14 bytes)
- DIB Header (50 bytes)

Bitmap File Header	
Identifier (ID)	2
File Size	4
Reserved	4
Bitmap Data Offset	4
DIB Header	
Bitmap Header Size	4
Width	4
Height	4
Planes	2
Bits Per Pixel	2
Compression	4
Bitmap Data Size	4
H-Resolution	4
V-Resolution	4
Used Colors	4
Important Colors	4

TODO 2: Revealing the Hidden Image

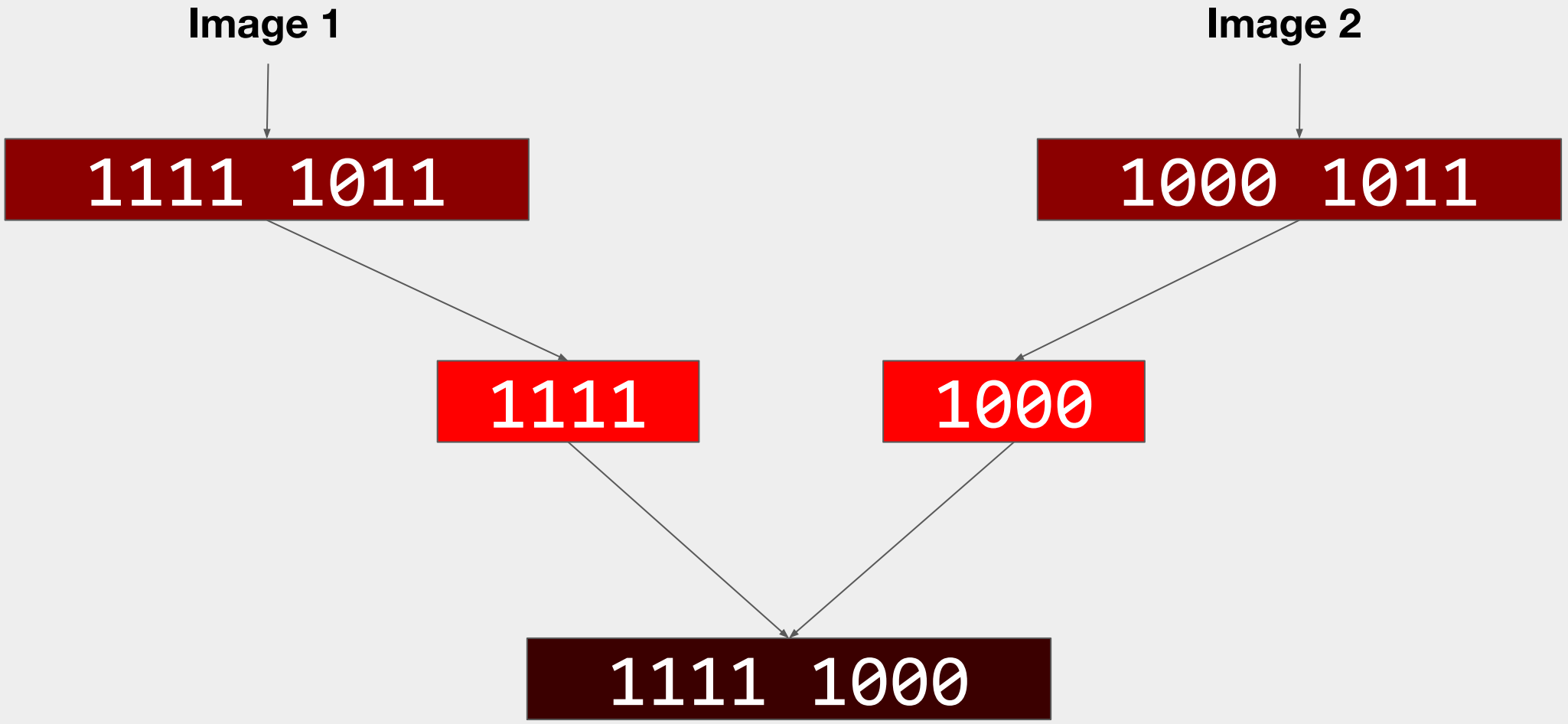




Move pixel by pixel (row-wise), flip the 4 MSB and 4 LSB in each color of each pixel

TODO 3: Hiding your own image

- **Again, move pixel by pixel (row-wise)**
  - At each pixel, grab the 4 MSB from image 1 and 4 MSB from image 2
  - Write the 4 MSB from image 2 into the 4 LSB of image 1



Tying the parts together

Your program must do the following

`./bmp_stenography --info FILENAME`

```
=== BMP Header ===
Type: BM
Size: 2073654
Reserved 1: 0
Reserved 2: 0
Image offset: 54

=== DIB Header ===
Size: 40
Width: 960
Height: 720
# color planes: 1
# bits per pixel: 24
Compression scheme: 0
Image size: 2073600
Horizontal resolution: 7559
Vertical resolution: 7559
# colors in palette: 0
# important colors: 0
```

`./bmp_stenography --reveal FILENAME`

- **Reveals the hidden picture**
- **Should overwrite `FILENAME`**

`./bmp_stenography --hide FILENAME1 FILENAME2`

- **Hides `FILENAME1` inside `FILENAME2`**

HINT:

Your `main()` function takes arguments:

```
int main(int argc, char *argv[]);
```

- `argc`: # of arguments
- `argv[]`: actual arguments
  - Note `argv[0]` is the program name