NOTICE TO PODCAST SUBSCRIBERS

The source code for CS 50's library as well as for Fall 2007's problem sets in general can be found at `http://cs50.tv/`.

Computer Science 50: Introduction to Computer Science I
Harvard College
Fall 2007

Problem Set 4: Forensics

out of 66 points

due by 7:00 P.M. on Friday, 2 November 2007

HACKER EDITION

Goals.

The goals of this problem set are to:

- Familiarize you with file I/O.
- Get you comfortable with hexadecimal, pointers, and arrays as buffers.
- Fight crime.

Recommended Reading.

Per the syllabus, no books are required for this course. If you feel that you would benefit from some supplementary reading, though, below are some recommendations.

- Section 21 – 26, 31, 32, 35, and 40 of `http://www.howstuffworks.com/c.htm`.
- Chapters 9, 11, 14, and 16 of *Programming in C*.

Collaboration.

For this Hacker Edition of Problem Set 4, you are hereby explicitly allowed to collaborate with a fellow student in CS 50 on #7 but only #7. Not only may you communicate in English, you may communicate in C and even share code for that problem. You and your partner must still submit separately, per this document's instructions, but the contents of your `~/cs50/ps4/case/` directories must be identical, as you will be graded identically for #7. It is required that both partners contribute equally to that problem's workload.

Inasmuch as #7 is, more than ever, about solving a problem that can be attacked from multiple angles, you should find it both stimulating and fun to collaborate with a peer. Feel free to solicit a partner via the course's bulletin board.

Although you are encouraged, you are not required to work with a partner. Rest assured that those who opt to work alone will be evaluated more on the basis of their work's quality than on their work's quantity.

Academic Honesty.

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed (*e.g.*, by some problem set or the final project). Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or accidentally shared in the course's virtual terminal room) or lifting material from a book, magazine, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this class that you have submitted or will submit to another. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the staff.

All forms of cheating will be dealt with harshly.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. In other words, you may communicate with classmates in English, but you may not communicate in, say, C. If in doubt as to the appropriateness of some discussion, contact the staff.

Grading Metrics.

Each question is worth the number of points specified parenthetically in line with it.

Your responses to questions requiring exposition will be graded on the basis of their clarity and correctness. Your responses to questions requiring code will be graded along the following axes.

> **Correctness.** To what extent does your code adhere to the problem's specifications?
> **Design.** To what extent is your code written clearly, efficiently, elegantly, and/or logically?
> **Style.** To what extent is your code commented and indented, your variables aptly named, *etc.*?

Rest assured that grades will be normalized across sections at term's end.

HACKER EDITION

HACKER EDITION

Getting Started.

0. SSH to `nice.fas.harvard.edu` and execute the command below.

```
cp -r ~cs50/pub/ps/distributions/ps4/hacker ~/cs50/ps4
```

Navigate your way to `~/cs50/ps4/`. If you list the contents of your current working directory, you should see the below. If you don't, don't hesitate to ask the staff for assistance.

```
case/  questions.txt  whodunit/
```

As this output implies, most of your work for this problem set will be organized within two subdirectories. Let's get started.

1. (4 points.) If you've ever seen Windows XP's default wallpaper (think rolling hills and blue skies), then you've seen a BMP. If you've ever looked at a webpage, you've probably seen a GIF. If you've ever looked at a digital photo, you've probably seen a JPEG. Read up a bit on the BMP, GIF, and JPEG file formats.[1] Then, in `ps4/questions.txt`, tell us the below.

    i. How many different colors does each format support?
    ii. Which of these formats supports animation?
    iii. What's the difference between lossy and lossless compression?
    iv. Which of these formats is lossy-compressed?

2. (2 points.) Curl up with the article from MIT below.

```
http://www.fas.harvard.edu/~cs50/pub/ps/shared/ps4/article.pdf
```

Though somewhat technical, you should find the article's language quite accessible. Once you've read the article, answer each of the following questions in a sentence or more in `ps4/questions.txt`.

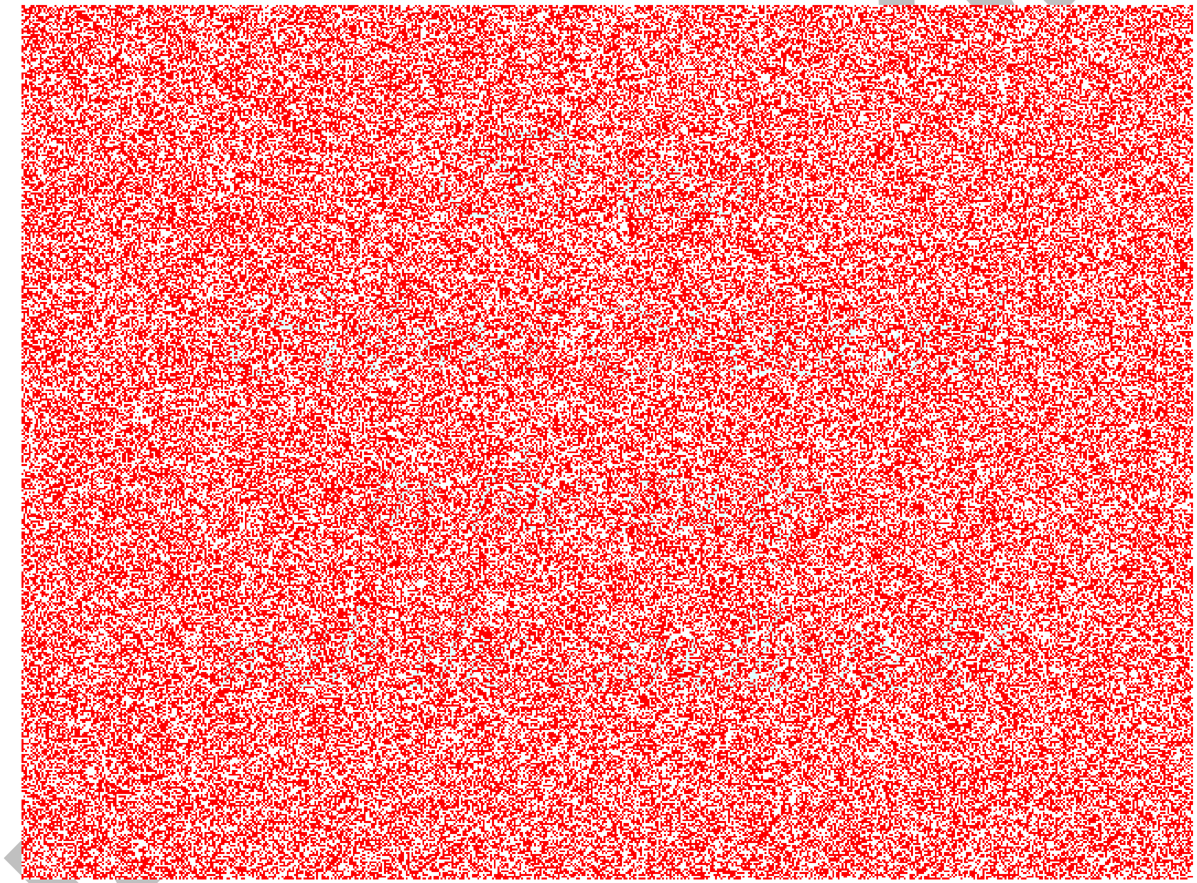
    i. What happens, technically speaking, when a file is deleted on a FAT file system?
    ii. What can someone like you do to ensure (with high probability) that files you delete cannot be recovered?

[1] For this question, you're welcome to consult *How Computers Work*, Google, Wikipedia, a friend, or anyone else, so long as your words are ultimately your own!

Whodunit.

3. Welcome to Tudor Mansion. Your host, Mr. John Boddy, has met an untimely end—he's the victim of foul play. To win this game, you must determine the answer to these three questions: Who done it? Where? And with what weapon?

   Unfortunately for you (though even more unfortunately for Mr. Boddy), the only evidence you have is a 24-bit BMP file called `clue.bmp`, pictured below, that he whipped up for his estranged granddaughter in his final moments.[2] Hidden among this file's red "noise" is a message from Mr. Boddy.



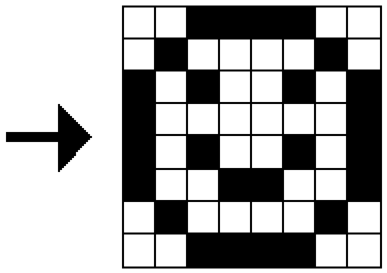
   You long ago threw away that piece of red plastic from childhood that would solve this mystery for you, and so you must attack it as the computer scientist that you almost are.

   But, first, some background.

[2] Realize that this BMP is in color even though you might have printed this document in black and white.

4. Perhaps the simplest way to represent an image is with a grid of pixels (*i.e.*, dots), each of which can be of a different color. For black-and-white images, we thus need 1 bit per pixel, as 0 could represent black and 1 could represent white, as in the below.[3]

```
11000011
10111101
01011010
01111110
01011010
01100110
10111101
11000011
```



In this sense, then, is an image just a bitmap (*i.e.*, a map of bits). For more colorful images, you simply need more bits per pixel. A file format (like GIF) that supports "8-bit color" uses 8 bits per pixel. A file format (like BMP or JPEG) that supports "24-bit color" uses 24 bits per pixel.[4] (We just spoiled #1i!)

A 24-bit BMP like Mr. Boddy's uses 8 bits to signify the amount of red in a pixel's color, 8 bits to signify the amount of green in a pixel's color, and 8 bits to signify the amount of blue in a pixel's color. If you've ever heard of RGB color, well, there you have it: red, green, blue.

If the R, G, and B values of some pixel in a BMP are, say, `0xff`, `0x00`, and `0x00` in hexadecimal, that pixel is purely red, as `0xff` (otherwise known as 255 in decimal) implies "a lot of red," while `0x00` and `0x00` imply "no green" and "no blue," respectively. Given how red Mr. Boddy's BMP is, it clearly has a lot of pixels with those RGB values. But it also has a few with other values.

Incidentally, XHTML and CSS (languages in which webpages can be written) model colors in this same way. In fact, for more RGB "codes," see the URL below.

`http://www.w3schools.com/html/html_colors.asp`

Now let's get more technical. Recall that a file is just a sequence of bits, arranged in some fashion. A 24-bit BMP file, then, is essentially just a sequence of bits, every 24 of which happen to represent some pixel's color. But a BMP file also contains some "metadata," information like an image's height and width. That metadata is stored at the beginning of the file in the form of two data structures generally referred to as "headers" (not to be confused with C's header files). The first of these headers, called `BITMAPFILEHEADER`, is 14 bytes long. (Recall that 1 byte equals 8 bits.) The second of these headers, called `BITMAPINFOHEADER`, is 40 bytes long. Immediately following these headers is the actual bitmap: an array of bytes, triples of which represent a pixel's color. However, BMP stores these triples backwards (*i.e.*, as BGR), with 8 bits for blue, followed by 8 bits for green, followed by 8 bits for red.[5] In

HACKER EDITION

[3] Image adapted from `http://www.brackeen.com/vga/bitmaps.html`.
[4] BMP actually supports 1-, 4-, 8-, 16-, 24-, and 32-bit color.
[5] Some BMPs also store the entire bitmap backwards, with an image's top row at the end of the BMP file. But we've stored this problem set's BMPs as described herein, with each bitmap's top row first and bottom row last.

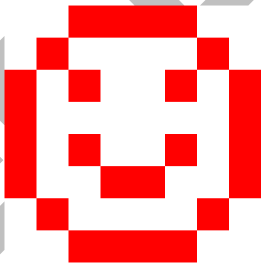
other words, were we to convert the 1-bit smiley above to a 24-bit smiley, substituting red for black, a 24-bit BMP would represent his bitmap as follows, where `0000ff` signifies red and `ffffff` signifies white; we've highlighted in red all instances of `0000ff`.

```
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
```

To be clear, recall that a hexadecimal digit represents 4 bits. Accordingly, `ffffff` in hexadecimal actually signifies `111111111111111111111111` in binary.

Okay, stop! Don't proceed further until you're sure you understand why `0000ff` represents a red pixel in a 24-bit BMP file.

5. Okay, let's transition from theory to practice. Navigate your way to `~/cs50/ps4/whodunit/`. In that directory is a file called `smiley.bmp`. If you feel like SFTPing that file to your hard drive and double-clicking it, you'll see that it resembles the below, albeit much smaller (since it's only 8 pixels by 8 pixels).



Open this file in `xxd`, a "hex editor," by executing the command below.

```
xxd -c 24 -g 3 -s 54 smiley.bmp
```

You should see the below; we've again highlighted in red all instances of `0000ff`.

```
0000036: ffffff ffffff 0000ff 0000ff 0000ff 0000ff ffffff ffffff  ........................
000004e: ffffff 0000ff ffffff ffffff ffffff ffffff 0000ff ffffff  ........................
0000066: 0000ff ffffff 0000ff ffffff ffffff 0000ff ffffff 0000ff  ........................
000007e: 0000ff ffffff ffffff ffffff ffffff ffffff ffffff 0000ff  ........................
0000096: 0000ff ffffff 0000ff ffffff ffffff 0000ff ffffff 0000ff  ........................
00000ae: 0000ff ffffff ffffff 0000ff 0000ff ffffff ffffff 0000ff  ........................
00000c6: ffffff 0000ff ffffff ffffff ffffff ffffff 0000ff ffffff  ........................
00000de: ffffff ffffff 0000ff 0000ff 0000ff 0000ff ffffff ffffff  ........................
00000f6: 0000                                                     ..
```

In the leftmost column above are addresses within the file or, equivalently, offsets from the file's first byte, all of them given in hex. Note that `00000036` in hexadecimal is 54 in decimal. You're thus looking at byte 54 onward of `smiley.gif`, all in hexadecimal. Recall that a 24-bit

HACKER EDITION

BMP's first 14 + 40 = 54 bytes are filled with metadata. If you really want to see that metadata in addition to the bitmap, execute the command below.

```
xxd -c 24 -g 3 smiley.bmp
```

If `smiley.bmp` actually contained ASCII characters, you'd see them in `xxd`'s rightmost column instead of all of those dots.

Okay, back to Mr. Boddy.

6. (30 points.) Open up `whodunit.h` and look over our (well, Microsoft's) definitions of `BITMAPFILEHEADER` and `BITMAPINFOHEADER`. You don't need to understand everything going on in that file, but know that we're going to use those structures in order to open `clue.bmp`. Take a look at the URLs in that file if curious as to what each of the structure's fields means.

   Now open up `whodunit.c` and read it over carefully. Be sure you understand what each function call is doing, particularly those to `fopen`, `feof`, `fread`, `fwrite`, and `fclose`. Turn to those functions' man pages or the URL below for usage information.

   ```
   http://www.cppreference.com/stdio/
   ```

   Note that all this program does at present is copy a BMP file. Confirm as much by building the program with Make and executing the command below.

   ```
   whodunit smiley.bmp new.bmp
   ```

   If you list the contents of your current working directory, you should now see `new.bmp`, the size of which is equal to that of `smiley.bmp`. Moreover, if you open `new.bmp` with `xxd`, you'll find that its contents are identical to `smiley.bmp`'s. And if you SFTP `new.bmp` to your hard drive, you'll even find that it looks the same as `smiley.bmp`! Wow, we've just re-implemented `cp`. And only for BMP files, no less.

   Odds are, though, you're going to thank us. Because you've still got work to do.

   Modify `whodunit.c` so that the program doesn't just copy a BMP file but instead reveals Mr. Boddy's final words.

   OMG, what? How?

   Well, think back to childhood when you held that piece of red plastic over similarly hidden messages.[6] Essentially, the plastic turned everything red. But, certain colors still shone through. Implement that same idea in `whodunit`. You needn't futz with `BITMAPFILEHEADER` or `BITMAPINFOHEADER`. Focus instead on changing what the program does within its `while`

HACKER EDITION

[6] If you remember no such piece of plastic, best to ask a friend or TF about his or her childhood.

loop. Ultimately, if you execute a command like the below, stored in `answer.bmp` should be Mr. Boddy's message.

```
whodunit clue.bmp answer.bmp
```

If what you need do is a mystery to you, 'tis precisely the point! Think about it. Try out some ideas. Find one that works. Do it for Mr. Boddy.

There's nothing hidden in `smiley.bmp`, but feel free to test your program out on its pixels nonetheless, if only because that BMP is small and you can thus compare it and your own program's output with `xxd` during development.

Rest assured that more than one solution is possible. So long as your program's output is readable (by your teaching fellow), Mr. Boddy will rest in peace.

Be sure to check in your code often with RCS!

Free Dinner.

7. (30 points.) You (and your optional partner) have just been hired by the local DA's office, and your first assignment is to analyze a 48MB forensic image of a suspect's USB stick. You suspect that this image contains 0 or more ASCII, Excel, HTML, JPEG, Word, and/or Zip files, but you won't know for sure until you perform your analysis. The image doesn't contain a file system (*i.e.*, data structures specific to FAT), Unfortunately, some of the files may be fragmented (*i.e.*, spread out across the stick), so you might need to conduct some reconstruction.

   Navigate your way to `~/cs50/ps4/case/`. In that directory, you will find `stick.raw` and an empty directory called `files`. Fill that directory with as many files as you can recover from `stick.raw`. Report each of your findings in a file called `affidavit.pdf` in `~/cs50/ps4/case/`, documenting for each file that you recover the below.

   i. The type of file.
   ii. A one-sentence description of the file's contents.
   iii. The blocks (0-indexed) within `stick.raw` between which you found the file (*i.e.*, the start block and end block of the file itself or the start blocks and end blocks of the files' fragments).
   iv. A one-paragraph explanation of how you found and recovered the file.

   How to recover these files? Well, that we leave to you. But we suggest that you read up on the formats (*i.e.*, internal structure) of ASCII, Excel, HTML, JPEG, Word, and Zip files. Google and Wikipedia are your friends. You may also find Linux tools like `file`, `grep`, `strings`, and `xxd` helpful. You are welcome to poke around `stick.raw` using tools (*e.g.*, hex editors with GUIs) for Mac OS and Windows as well. However, you may not use any tools that are expressly designed for forensic investigations or file recovery. If in doubt as to the

HACKER EDITION

acceptability of some tool, contact the staff. You are encouraged to develop your own software, in C or any other language, to facilitate your investigation. Assume a block size of 512 B for the USB stick.

Your score for this challenge will be based mostly on the quality and quantity of your ideas but also partly on the number of files that you recover. We will take into account whether you worked alone or with a partner.

And now the proverbial icing on the cake. The students who recover the most files will be awarded a prize: **free dinner in the Square** with their TF(s) some night. We will choose one winner among those who worked alone and two winners among those who worked in pairs. In the event of a tie, the students who submitted the earliest shall be declared the winners of dinners.

HACKER EDITION

Submitting Your Work.

8. Ensure that your work is in `~/cs50/ps4/`. Submit your work by executing the command below.

```
cs50submit ps4 hacker
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your work's successful submission. You will also receive a "receipt" via email to your FAS account, which you should retain until term's end. You may re-submit as many times as you'd like; each resubmission will overwrite any previous submission. But take care not to re-submit after the problem set's deadline, as only your latest submission's timestamp is retained.