

Programming Assignment 5

Due: 11:59pm, Saturday, February 6

Overview

The goals of this assignment are to:

1. Using file I/O
2. Taking what looks to be a very hard problem, understanding the small steps that are needed, and writing out the code to solve it

Setup

Open a new Linux terminal window. In your home directory, create/make a new directory called `HW5` and change directory into that directory:

```
$ mkdir ~/HW5
$ cd ~/HW5
```

There are no starter files for this project.

Part I: Reading (10 pts)

Read the following articles and answer the following questions in a file named `PART1` (note: not `part1.doc` or `part1.txt`. The bundle script will only accept a file named `PART1`):

1. [hexadecimal](#) (sections 1,2.1)
2. [jpeg](#) (Sections 1,2,5)
3. [FAT](#) (Sections 1,2)
4. [Slack Space](#) (Intro)
5. [File Deletion](#)

Q1: What is the largest decimal value that can be stored using 1 byte? What is the hexadecimal representation of 252?

Q2: How many bytes are used to store an int? A double?

Q3: I have an image that's 102,321 bytes in size and I write to a FAT file system that uses 512 byte blocks. How many blocks are needed to store this file? Explain. Hint: you may want to read the rest of the document before trying to answer.

Q4: What needs to happen in order to securely delete a file (i.e. no file remnants remain)

Q5: The Linux `xxd` command creates a hex dump of a given input. What does the output of

```
xxd /home/linux/ieng6/cs11wb/public/HW5/card.raw | head -1
```

return and explain what this command does.

Q6: How can you convert an integer to a hex string. Hint: use google in this class.

Part II: File recovery (30 pts)

I accidentally deleted all of my photos (JPEG format) from my digital camera's CompactFlash (CF) card. Luckily, in the computer world, "deleted" tends not to mean deleted so much as "forgotten". My computer tells me the flash card is blank, but I'm sure that the image data still exists on the card, just the file names and locations have been deleted. **For this assignment, write a program Recover.java to recover the photos from my flash card.**

Recover.java will need to read over a copy of my CF card, looking for JPEG signatures. The first four bytes of most JPEGs are either:

`0xff 0xd8 0xff 0xe0`

Or

`0xff 0xd8 0xff 0xe1`

Odds are if you find one of these patterns of bytes on a disk known to store photos, they mark the start of a JPEG. Each time you find a signature, open a new file for writing and start filling that file with bytes from my CF card, closing the file once you encounter another signature. File names should be 1.jpg, 2.jpg, ...

I've made a copy of the CF card that you can open directly by your program, i.e. do not copy to your local directory:

`/home/linux/ieng6/cs11wb/public/HW5/card.raw`

To ensure your program works correctly, try opening each of the generated *.jpg(s). If you open them and don't see anything, something went wrong. I won't tell you how many images should be recovered, but it is more than 5.

Additional Information:

Digital cameras tend to store photographs contiguously on CF cards. So, the start of a JPEG usually marks the end of another. Digital cameras generally initialize CF cards with a FAT file system whose "block size" is 512 bytes, meaning they only write to those cards in blocks of 512 bytes. JPEG's can span contiguous blocks. Otherwise, no JPEG could be larger than 512 B. Thus a photo that's 1 MB (i.e. 1,048,576 B) takes up $1048576/512 = 2048$ blocks on a CF card.

The last byte of a JPEG might not fall at the very end of a block. A photo that uses a few bytes less than 1MB, e.g. 1,048,574 B, will still use 2048 blocks. Because the CF card was brand new when I started taking pictures, it was most likely "zeroed" (i.e. filled with 0s) by the manufacturer, in which case any slack space will be filled with 0s. It's ok if those trailing 0s end up in the JPEGs you recover.

Rather than reading the card's bytes one at a time, you can read 512 B at once into a buffer for efficiency's sake. Thanks to FAT, the JPEG's signatures will be "block-aligned". That is, you need only look for those signatures in a block's first four bytes.

Rubric:

This assignment will be graded on a range of:

- Correctness (15pts): program recovers all pictures. 3 pts for reading in data. 3 pts for writing out data, 4 pts for generating at least 1 image, 5 pts for getting all images resorted.
- Efficiency (5pts): Is buffered input/output used.
- Clarity (10 pts): Program flow is logical, clear, and easy to understand (comments and appropriate variable names will help). Your tutor can look through your code and easily understand how your program works because it's well documented, clear variable names, logically broken up. Range from 1-10 pts.

Programmer of the week (optional)

Follow the clues and find the prize. The first student from each section to turn the award into their tutor or to me wins this week's programmer of the week challenge.

Style Requirements (10 pts) – Different from other assignments

You will be graded for the style of programming on this assignment.

- Use reasonable comments to make your code clear and readable.
- All methods must have javadoc comments. We will be testing this by running "javadoc filename.java" and ensuring that the resulting documentation pages appear.
- Use reasonable variable names that are meaningful.
- Use static final constants to make your code as general as possible. No hardcoding constant values inline (no magic numbers).
 - There will be many cases in this homework where you want to compare card 1 to card 2, card 2 to card 3, etc. I'm allowing you to hardcode numbers for this.
- Judicious use of blank spaces around logical chunks of code makes your code much easier to read and debug.
- Keep all lines less than 80 characters. Make sure each level of indentation lines up evenly.
- Every time you open a new block of code (use a '{'), indent farther by 2 spaces. Go back to the previous level of indenting when you close the block (use a '}').
- Always recompile and run your program right before turning it in, just in case you commented out some code by mistake.

Turnin Instructions

Remember the deadline to turn in your assignment is Saturday, February 6 by 11:59pm.

When you are ready to turn in your program in, type in the following command and answer the prompted questions:

```
$ cd ~/
$ bundleP5
Good; all required files are present:
```

HW5

Do you want to go ahead and turnin these files? [y/n]y
OK. Proceeding.

Performing turnin of approx. 6144 bytes (+/- 10%)
Copying to /home/linux/ieng6/cs11wb/turnin.dest/cs11wb.P5
...
Done.
Total bytes written: 31744
Please check to be sure that's reasonable.
Turnin successful.