CS360
Lab 4

The Job

After graduation you find yourself working for a hip software company that does contracting for various outside companies and agencies. You haven't been working there long so you're the lowest vertebrate on the food chain, often picking up scrap jobs. One day your boss comes to you and says, 'Got an easy job for ya – you'll like it.' With a disinterested glance from your screens you respond, 'yeah right'. The memories of the last contract flood your mind – a tedious chore in a decrepit old language – skills wasted, lights going dim . . . She says, 'pays double your rate?' 'Hmmm', you think.

After the usual non-disclosure agreement and an hour of back and forth you have your next job. It seems an unheard-of little company from the Bahamas wants a piece of code written quickly and quietly. Do the job, no questions asked. Information extraction problem. Input is a binary file in a known format. Read it, get the necessary information and move on.

mp3

The input file is a `.mp3`. The job is to read it and first determine if it is in a MPEG Layer 3 format. If it isn't in that format don't bother going on. Provided it is, extract the bit rate for the file, the frequency and the copyright information. If the file is copyrighted you're to signal an alert, but only if the file is a copy and not the original.

'Hmmm, checking music files for copyright. This sounds familiar. I wonder who the client *really* is?' You go straight to the mp3 documentation. Wikipedia is decent: `http://en.wikipedia.org/wiki/MP3`. Great description of the file format in that figure. The format specifies where the bit rate and frequency information are, as well as . . . two bits holding copyright information. Copyright information is specified by single bits? Bit 29 set equals ©material, unset means non-copyrighted. Bit 30 set means it is the original, unset means it is a copy of original media. That's not too bright. You immediately think of the infamous *evil bit* (`http://www.faqs.org/rfcs/rfc3514.html`). 'Oh, well, should be easy.'

So now how to find the mp3 header? The file is known to have a *synchronization* bit sequence somewhere, possibly repeated. The sequence to look for is 0xFFF. This is so hardware can find the music within the bit stream. mp3 players just send bit streams to the decoding hardware and it synchronizes itself when 0xFFF passes by. (The chips themselves are cheap: `http://www.sparkfun.com/commerce/product_info.php?products_id=305`.) 'Aha, find that and we've got the necessary 32 bits to work from. Sure hope it is aligned at byte boundaries and not stuck in the middle of a word!' There might be an ID3v2 metadata tag at the beginning: artist, song and other meta information. We'll just skip over that, read until we find the sync sequence.

Your company maintains a database of old code to reuse. After a quick regular expression search you find some old C code. 'Hey, C. I haven't programmed in C since my University days so long ago (2 years). This will be fun.'

Here's your starting code: `Lab3.c` Pretty ugly. Better work on some things first:

- Try compiling. From the command line: `gcc Lab3.c` Fix that.
- Now that it is compiling, you remember that C doesn't do a lot of type checking. You better compile with all the warnings turned on: `gcc -Wall Lab3.c`. Aha, fix those.
- The code uses `goto`!!! Quaint, but let's fix that.
- Loose constants (lines 29 and 35). Those should be `#define`'d or declared with constants. What in the heck is 10485760?
- Fix that so it prints out the file size to 2 decimal points.
- Memory Leak!! The memory allocated on line 37 isn't ever freed. Fix that.
- All the code is in the main function. Let's modularize and put things in separate functions. That stuff at the top (lines 7 - 19) just gets the file name and opens a file. Put that into a function named `initialize`. Lines 21 - 44 only read in the file into memory, put that into a function called `readFile`. We could make all variables global to do that? No way, that's a hack. We can get by with only one global variable: `File * fp`, everything else needs to be passed in as an argument or returned as the return value. If we have a lot to pass in and out, maybe we could just pass in a pointer to a struct? Yeah, that would be cool. Probably get extra credit for that one.
- Looks good now. Compile and run (`./a.out song.mp3`) and see how it goes. Grab some more mp3's to try it out.

OK, now that it's all cleaned up and running nicely, we'll get the job done. We have access to the contents of the file through the pointer `unsigned char * data` and we know how big it is (wouldn't want to walk off the end!). Search through the data for the sync bits. Once you find the first instance, stop and extract the relevant information. Print out to standard output the Bit Rate and Frequency in human terms, i.e. 128kbps at 44.1kHz. Then print out whether or not it is copyrighted and whether it is the original or a copy. Do these things only if the file is a MPEG Layer 3 file.

Hey, you're done! You submit it and then wonder what they're going to use it for.

Turn In: Your .c file only (this must be your code file, NOT a .txt, .doc, or .pdf!) You do not need to turn in the .mp3s