CS 0449 – Project 1: Blackjack & EXIF Viewer

Due: Sunday, October 2, 2016 at 11:59pm

Your first project is to write two programs in C that provide some experience with a wide range of the topics we have been discussing in class.

Blackjack Implementation (30 points)

Blackjack (also known as 21) is a multiplayer card game, with fairly simple rules. For this assignment, you will be implementing a simplified version where a user can play against the computer who acts as dealer.

Two cards are dealt to each player. The dealer shows one card face up, and the other is face down. The player gets to see both of his or her cards and the total of them is added. Face cards (Kings, Queens, and Jacks) are worth 10 points, Aces are worth 1 or 11 points, and all other cards are worth their face value. The goal of the game is to get as close to 21 ("blackjack") without going over (called "busting").

The human player goes first, making his or her decisions based on the single dealer card showing. The player has two choices: Hit or Stand. Hit means to take another card. Stand means that the player wishes no more cards, and ends the turn, allowing for the dealer to play.

The dealer must hit if their card total is less than 17, and must stand if it is 17 or higher.

Whichever player gets closest to 21 without exceeding it, wins.

```
The dealer:
? + 10

You:
4 + 10 = 14

Would you like to "hit" or "stand"? hit

The dealer:
? + 10

You:
14 + 10 = 24 BUSTED!

You busted. Dealer wins.
```

Requirements and Hints

- Have the program intelligently determine if an Ace should be interpreted as a 1 or an 11 by counting the number of aces dealt and adjusting the total down if over 21 and an ace exists.

- Generating random numbers in C is a two-step part. First, we need to seed the random number generator *once* per program. The idiom to do this is:
  `srand((unsigned int)time(NULL));`

- When we need random numbers, we can use the rand() function. It returns an unsigned integer between 0 and RAND_MAX. We can use modulus to reduce it to the range we need:
  `int value = rand() % (high - low + 1) + low;`

- Remember that getting a card worth 10 is more common because of the face cards, so generate a random card, not a random value.

- You can assume there is an infinite deck of cards (i.e. drawing an Ace will not decrease the probability of drawing an Ace the next time.)

EXIF Viewer (70 points)

An EXIF tag is embedded in many image files taken on a digital camera to add metadata about the camera and exposure. It is a complicated format, but we can simplify it to where we can write a simple viewer that will work with many JPEG files.

A JPEG file begins with the 2 byte sequence 0xFF, 0xD8. After that, a special JPEG marker (0xFF 0xE1) indicates an application specific segment, called APP1. We will assume (but you must verify) that there is no APP0 (0xFF 0xE0) section prior to APP1. This means the first 20 bytes of a JPEG with an EXIF tag will be:

Offset	Length	Value	Description
0	2	`0xFF 0xD8`	JPEG Start of File marker
2	2	`0xFF 0xE1`	JPEG APP1 Marker
4	2		Length of the APP1 block (always big endian)
6	4	`Exif`	Exif string
10	2	`0x00 0x00`	NUL Terminator and zero byte
12	2	`II or MM`	Endianness. `II` means Intel (little endian) *This is the start of the TIFF header*
14	2	`42`	“version number” is always 42
16	4		Offset to start of Exif block from start of TIFF block (byte 12 of the file).

Next, an array of TIFF (Tagged Image File Format) Tags will store the information we are looking for. At offset 20, we find a 2-byte (unsigned short) *count* that tells us how many tags there will be in this section.

A TIFF tag is 12 bytes that are defined as:

Offset	Length	Description
0	2	Tag identifier
2	2	Data type
4	4	Number of data items
8	4	Value or offset of data items.

Loop through the file *count* times, reading a single TIFF tag at a time. We will only be concerned with 3 different tags in this section. Simply ignore any other tag that appears. The three tags we are concerned with have the 2-byte identifiers in the table below:

Tag identifier	Data Type	Description
0x010F	2 (ASCII string)	Manufacturer String
0x0110	2 (ACSII string)	Camera Model String
0x8769	4 (32-bit integer)	Exif sub block address

Let us take the first one as an example. We read in a 12-byte TIFF tag and find that its identifier field is 0x010F. Its data type field will be 2, which means that the data is encoded in an ASCII string. The number of data items field will tell us how many bytes our string has.

The final field in the tag can contain the value of the data itself if it fits in 4 bytes, or it can contain an offset to the data elsewhere in the file. Since an arbitrary string cannot fit in 4 bytes, in our case this value is an offset. It's an offset from the beginning of the TIFF header, which occurred at byte 12 of the file. So we seek to 12 + offset in the file and read each letter from that position in the file, until we have read them all (we'll encounter a NUL terminator at the end).

At this point, we've read the manufacturer string. We must seek back to the location in the file where we were reading tags and continue on to the next one.

To make this concrete, say we encounter our Manufacturer String tag while reading bytes 22-34 of our file. The tag would be:

`0x010F`	2	6	158

The 6 tells us how many bytes the string will be (including the NUL terminator). The 158 tells us to seek to 158 + 12 bytes from the start of the file (the 12 bytes is the offset of the TIFF header from the start of the file).

When we seek to offset 170, we read in "Canon", the manufacturer of the camera.

We must now seek back to offset 34 so that we can read the next tag in this section.

If we encounter the 0x8769 identifier, there is an additional Exif block elsewhere in the file. We can stop reading at this point even if we haven't read all *count* tags because the TIFF format states that all identifiers must be in sorted order.

We will seek to the offset specified in this Exif sub block tag, again +12 bytes. There, we'll repeat the above process one more time to get more specific information about the picture.

First, read in a new *count* as an unsigned short. Next, loop, reading more 12-byte TIFF tags from the file.

This time, we'll be concerned with the following fields:

Tag identifier	**Data Type**	**Description**
0xA002	4 (32-bit integer)	Width in pixels
0xA003	4 (32-bit integer)	Height in pixels
0x8827	3 (16-bit integer)	ISO speed
0x829a	5 (fraction of 2 32-bit unsigned integers)	Exposure speed
0x829d	5 (fraction of 2 32-bit unsigned integers)	F-stop
0x920A	5 (fraction of 2 32-bit unsigned integers)	Lens focal length
0x9003	2 (ASCII String)	Date taken

The good news is that type 4 means that the value is directly encoded in the last 4 bytes of our tag and no seeking needs to be done.

Type 5 requires us to behave like we did with the string, but rather than reading several single-byte characters, we will read 2 unsigned ints. Display the ratio of the two numbers as shown in the example below.

What To Do

For your project you will make a utility that can print the contents of an existing tag, if there.

Make a program called `exifview` and make it so that it runs with the following command line:

```
exifview FILENAME
```

It should print the contents of the EXIF tag to the console if present, or give a message if not present or readable by our program.

Output

```
$ ./exifview img1.jpg
Manufacturer:   Canon
Model:          Canon EOS REBEL SL1
Exposure Time:  1/80 second
F-stop:         f/2.8
ISO:            ISO 2000
```

```
Date Taken:      2013:08:20 22:23:45
Focal Length:    40 mm
Width:           512 pixels
Height:          768 pixels
```

Hints and Requirements

- We need to treat these files as *binary files* rather than *text files*. Make sure to open the file correctly, and to use fread for I/O.
- Please use a structure to represent a JPEG/TIFF/EXIF header and another struct to represent a TIFF tag. Do NOT use a bunch of disjoint variables. Do your fread() with the whole structure at once. (That is, read an entire tag in one file operation.)
- If the header field does not contain the Exif string in the right place, print an error message that the tag was not found. If the TIFF header contains MM instead of II, print an error message that we do not support the endianness.

Environment

Ensure that your program builds and runs on thoth.cs.pitt.edu as that will be where we are testing.

We have provided two sample jpg files with valid EXIF tags according to our limitations. Copy them to your working directory on thoth by using the command:

```
cp ~wahn/public/cs449/*.jpg .
```

The dot at the end is important as it represents the current directory in Linux.

Submission

When you're done, create a gzipped tarball (as we did in the first lab) of your commented source files and compiled executables. Name the gzipped tarball USERNAME_project1.tar.gz, similarly as we did for lab1.

Copy your archive to the directory:

```
~wahn/submit/449/RECITATION_CLASS_NUMBER
```

Make sure you name the file with your ***username****, and that you have your name in the comments of your source file. Please do NOT submit the sample picture files we provide.*

Note that this directory is insert-only, you may not delete or modify your submissions once in the directory. If you've made a mistake before the deadline, resubmit with a number suffix like `USERNAME_project1_1.tar.gz`

The highest numbered file before the deadline will be the one that is graded, however for simplicity, please make sure you’ve done all the work and included all necessary files before you submit