Project 1: Craps & EXIF Viewer

CS/COE 0449 — Introduction to System Software

See CourseWeb for the Due Date

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.

Part I: The Game of Craps (30 points)

For the first part of this project, you will be implementing the game of Craps. For those unfamiliar with the rules, Craps is played with two dice which are rolled together. The sum of these two dice is computed and if the sum is:

- 2, 3, or 12 the player immediately loses.
- 7 or 11 the player immediately wins.
- Any other number is referred to as the point. The goal then becomes to keep rolling the dice until:
  - The player rolls a 7 the player loses
  - The player rolls the "point" the player wins.

Your job on this assignment is to make a simple craps game. In this game the player will enter his or her name, the program will display a welcome message, and ask if the player would like to play or quit. The program will then roll two dice, and display them and their total. If the player has won, display a message congratulating them. If the player has lost, report it to them. Otherwise, store this first roll as the point roll and keep rolling until the player wins or loses. When the game ends, ask if the player would like to play again. Example:

```
Welcome to CS0449s Casino!
Please enter your name: John
John, would you like to play or quit? play

You have rolled 5 + 2 = 7
You Win!

Would you like to play again? yes
```

```
You have rolled 5 + 5 = 10 (point)
You have rolled 1 + 2 = 3
You have rolled 3 + 5 = 8
You have rolled 6 + 4 = 10
You Win!

Would you like to play again? yes

You have rolled 4 + 5 = 9 (point)
You have rolled 1 + 1 = 2
You have rolled 3 + 4 = 7
You Lose!

Would you like to play again? no

Goodbye, John!
```

Hints

- 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 rolling two 6-sided dice is different than rolling one 12-sided die.
- You must do string input, do not make an integer menu or use single characters.

Part II: 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. So, your first task is to verify that the first two bytes is 0xFF follows by 0xD8. If it is not, simply says the file is not supported and quit your program. After that, a special (2 bytes) JPEG marker 0xFF follows by either 0xE0 or 0xE1 which indicates an application specific segment. 0xFF follows by 0xE0 is called APP0 segment. We do not care about this segment. We will focus only on APP1 segment where its marker is 0xFF follows by 0xE1. So, if the marker is 0xFF follows by 0xE1, you are all set. Otherwise, you have to search for 0xFF follows by 0xE1 which will be somewhere after APP0 marker. **Note** that the

location of APP1 marker is important. Location of information we are looking for usually based on the location of the APP1 marker. So, let's call this `app1Location`. Once you fine the APP1 marker, the meaning of 18 bytes starting at the marker are as follows:

Offset from APP1 Marker	**Length**	**Value**	**Description**
0	2	0xFF 0xE1	JPEG APP1 Marker
2	2		Length of the APP1 block (big endian)
4	4	Exif	Exif string
8	2	0x00 0x00	NUL Terminator and zero byte
10	2	II or MM	Endianness: II means Intel (little endian) *This is the start of the TIFF header*
12	2	42	"Version number" is always 42 (little endian)
14	4		Offset to start of Exif block from start of TIFF header (byte 10 from APP1 marker) We will call this value `startOffset`

From the Exif information, it tells you where is the start of Exif block which is `app1Location + 10 + startOffset`. The first two bytes (`unsigned short`) at the start of Exif block tells us how many TIFF (Tagged Image File Format) tags there will be in this section (block). Let's call this number `count`. A TIFF tag store the information we are looking for. Each TIFF tag is a 12-byte block. The table below shows descriptions of each TIFF tag:

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

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 tag identifiers in the table below:

Tag Identifier	**Data Type**	**Description**
0x010F	2 (ASCII string)	Manufacturer String
0x0110	2 (ASCII string)	Camera Model String
0x011A	5 (Rational number)	Number of Pixels per Unit (width)
0x011B	5 (Rational number)	Number of Pixels per Unit (height)
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. Its an offset from the beginning of the TIFF header, which occurred at byte `app1Location + 10` of the file. So we seek to `(app1Location + 10) + offset` in the file and read each letter from that position in the file, until we have read them all (well encounter a NULL terminator at the end). At this point, weve 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 and the tag is as follows:

0x010F	2	6	158

The 0x010F tells us that the information in this tag is manufacturer string. The 2 tells us that the data type in this tag is ASCII string. The 6 tells us how many bytes (the string in this case) will be (including the NULL terminator). The 158 tells us to seek to `app1Location + 10 + 158` bytes from the start of the file (the `app1Location + 10` bytes is the offset of the TIFF header from the start of the file). When we seek to that offset, we read in "Canon", the manufacturer of the camera.

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

Note that a rational number consists of two integers, numerator and denominator. Therefore, it requires 8 bytes to store a rational number which does not fit into the 4-byte value. So, if the data type is 5, the last four bytes of the TIFF tag is an offset. At that location, the first four bytes is an unsigned integer representing the numerator and the last four bytes is an unsigned integer representing the denominator. To display a rational number, display the ratio of the two numbers as shown in the example below.

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 havent 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 `+ (app1Location + 10)` bytes. There, well repeat the above process one more time to get more specific information about the picture.

Once you are at a new location, first, read in a new count (2-byte `unsighed short`). Next, loop, reading more 12-byte TIFF tags from the file.

This time, well be concerned with the following fields:

Tag Identifier	**Data Type**	**Description**
0xA002	4 (32-bit integer)	Width in pixel
0xA003	4 (32-bit integer)	Height in pixel
0x8827	4 (32-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.

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

```
Manufacturer : Canon
Model        : Canon EOS REBEL SL1
X Res/Unit   : 720000/10000
Y Res/Unit   : 720000/10000
Res Unit     : Inch
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 cs445.cs.pitt.edu as that will be where we are testing.

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

```
cp /afs/cs.pitt.edu/usr0/tkosiyat/public/cs0449/*.jpg .
```

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

Submission

When youre done, create a gzipped tarball named `USERNAME_project1.tar.gz` (as we did in the first lab) of your commented source files and compiled executables.

Copy your archive to the directory:

```
/afs/cs.pitt.edu/public/incoming/CS0449/tkosiyat/sec1
```

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 youve made a mistake before the deadline, resubmit with a number suffix like `USERNAME_project1_2.tar.gz`

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