Project 1 - Mastermind and ID3 tags

Due by midnight, Saturday 9/15 (or late on Sunday)

Thanks to Dr. Misurda for project concepts and portions of the writeup.

[30%] `mastermind.c`

Maybe you've played this game before, maybe not. **Here's a link to an online version**. When you put in your guess:

- a **black peg** means you have a piece of the correct color in the correct place
- a **white peg** means you have a piece of the correct color in the *wrong* place
- so, **black-black-white** means you have 3 correct colors, 2 of which are in the right place; and 1 incorrect color.

How your version will work

1. Ask the user how many colors (3 to 6).
    - Keep asking until they type in a valid number.
    - This lets them choose the difficulty of the game.
2. Generate a **random sequence** of 4 colors, where each item of the sequence can be one of:
    - 3 colors: { red, green, blue }
    - 4 colors: { red, green, blue, yellow }
    - 5 colors: { red, green, blue, yellow, purple }
    - 6 colors: { red, green, blue, yellow, purple, orange }
    - **Colors may repeat!** So even red-red-red-red is valid.
3. Give the user **at most ten guesses** as follows:
    - Read a guess from the user as a sequence of 4 letters which represent colors.
        - Keep asking until they type in a valid guess.
    - Check their guess.
    - Tell the user how many of each kind they have:
        - correct color in the correct place

        - correct color in the wrong place
    - If they won:
        - **ask them if they want to play again.** If so, go back to the very beginning.
4. If they don't get it in 10 guesses:
    - tell them the solution
    - **ask them if they want to play again.** If so, go back to the very beginning.

Reading and displaying the colors

The colors will be represented as **lowercase letters.**

- `r` = red
- `g` = green
- `b` = blue
- `y` = yellow
- `o` = orange
- `p` = purple

So if the player enters `rggb`, that means red-green-green-blue.

If the player enters anything other than 4 letters, or if they use any invalid letters (e.g. trying to guess orange when they are playing a 3-color game, or using any other letters), it's an invalid guess, and you should ask them again.

When you display the solution, you should use these letters as well.

Hints

The `atoi()` function from `<stdlib.h>` will parse an integer from a string. Be sure to read its documentation to know how it works.

Feel free to use any example code I have given you. Things like `read_line`, `streq` etc will be helpful.

Do not add extra prompts/questions, and do not be too picky about user input. Accept lower- AND upper-case letters for the colors. User experience is an important part of program design! And it will make it faster and easier to grade your projects ;)

To generate random numbers:

- `#include <stdlib.h>` and `#include <time.h>` at the top of your file

- **ONCE** at the beginning of the program, "seed" the random number generator like so:

```
srand((unsigned int)time(NULL));
```

- When you need an **inclusive** range of random numbers such as `[0, 2]` do this:

```
rand() % (high_value - low_value + 1)
```

- **Write a function** to do this. `random_range(1, 10)` makes more sense and is more readable than `rand() % 10 + 1`.

> *This method will generate slightly biased results and should not be used to generate a range of random numbers in general. For something this simple, it's fine.*

[70%] `id3edit.c`

Who remembers MP3s? No? No one? My god I'm old.

ID3 is a standard way of embedding song information in an MP3 file, so it can store the album, artist, song name etc. That's how iTunes or your MP3 player or your phone knows.

You will make a utility that can:

- Check if an MP3 file has an ID3 tag
- Read and display the contents if it has one
- Add an ID3 tag if it doesn't have one
- Change (almost) any field in an ID3 tag

ID3 tag structure

An ID3 tag is a binary structure which looks like this (offsets and lengths are measured in bytes):

Offset	Length	Description
0	3	"TAG" identifier string
3	30	Song title string
33	30	Artist string
63	30	Album string
93	4	Year string

Offset	Length	Description
97	28	Comment string
125	1	Zero byte separator
126	1	Track number byte
127	1	Genre identifier byte

When translating this to a C structure, come up with appropriate types for each field. The size of the entire ID3 tag struct should be 128. You can use `sizeof()` to find out.

> The string fields *may or may not have zero terminators at the end.* See the Hints section.

Files

I will provide you with some test MP3 files to use. **But we will test your program with more.**

You can copy them to your directory like so:

```
cp ~jfb42/public/cs449/*.mp3 .
```

> Don't forget the space and period in the above command.

If you mess up or corrupt the files somehow, don't worry, just run this command again to get clean copies of them.

The files should have the following fields by default:

- `bensound-happyrock.mp3`
  - Title: `"bensound-happyrock.mp3"`
  - Artist: `"Bensound"`
  - Album: `"Royalty Free Music"`
  - Year: `"2014"`
  - Comment: `""` (empty)
  - Track: `4`
- `bensound-clearday.mp3`
  - Title: `"This is a very long song title"`
  - Artist: `"Bensound"`
  - Album: `"This is a very long album titl"` (the 'e' is cut off)

- Year: `"2024"`
- Comment: `""` (empty)
- Track: `210` (NOT a negative number!)

How it will work

```
$ ./id3edit
```

This should display a help message explaining how to use the program.

```
$ ./id3edit FILENAME
```

This should try to read the given file, and display its ID3 tag contents if it has one, or say that it doesn't have one.

```
$ ./id3edit FILENAME -field value
```

This will set one of the ID3 tag's fields to "value," where "field" can be one of:

- title
- artist
- album
- year
- comment
- track

So an example might be `./id3edit test.mp3 -year 1988`

```
$ ./id3edit FILENAME -field value -field value
$ ./id3edit FILENAME -field value -field value -field value
$ ./id3edit FILENAME -field value -field value -field value -f
etc
```

Your program should also allow **any number** of fields to be set at once.

Program behavior

- If you are setting a field, and the MP3 has no ID3 tag, you should create one and add it to the end of the file.
  - In this case, any fields that are not set on the command line should be set to all 0 bytes.

    - (look up the `memset` function.)
- If you are setting a field, and the MP3 *already* has an ID3 tag, you should *not* add a new one to the end of the file. Change the one that is already there.
- You can ignore the "genre identifier" field. Don't change it on existing files, and make it 0 on new tags.
- If you want to put spaces in a field value, you can put quotes around it on the command line:

```
$ ./id3edit test.mp3 -title "This is a test"
```

    - Your program will see `argc == 4`, and `argv[3]` will be `"This is a test"`

Hints

- The zero byte separator is not used for anything, but it must be included for proper data alignment.
- Unlike Java, C has *versions of integer types which cannot hold negative numbers*. That might be useful when making your struct.
- Remember, this is a *binary* file, and you should open it as one.
- **The strings in the ID3 tag may or may not have a zero terminator.** This will present some challenges as most of the C string functions expect one.
- Look up the documentation for `<string.h>`, and look at the `strn___` functions such as `strncmp` and `strncpy`.
- When printing out the strings, you have a couple options: you can copy the string into a longer buffer which you then zero-terminate, or you can use the `.*` format specifier with `printf`, which works like:

```
// somehow figure out the string length...
printf("this is a string: '%.*s'\n", string_length, stri
```

Grading

We will be compiling your programs with the following options, so be sure to compile with them while you develop as well:

```
$ gcc --std=c99 -Wall -Werror -o mastermind mastermind.c
$ gcc --std=c99 -Wall -Werror -o id3edit id3edit.c
```

- **[30]** `mastermind.c`
  - **[10]** Compiles and runs
  - **[5]** Generates random sequence
  - **[5]** Implements rules correctly
  - **[5]** Proper input and output (PLEASE read the testing section above)
  - **[5]** Style (TRY to split your code into functions.)
- **[70]** `id3edit.c`
  - **[10]** Compiles and runs
  - **[5]** Struct matches ID3 tag structure
  - **[5]** Uses command-line arguments to do its work (`argc/argv`)
  - **[5]** Shows a "usage" message if run without command-line arguments
  - **[5]** Checks that file exists (exit if `fopen` returns `NULL`)
  - **[5]** Properly detects presence or absence of an ID3 tag (by looking for the "TAG" string)
  - **[10]** Reads and displays existing ID3 tags in a nicely-formatted way
  - **[10]** Can update existing ID3 tags on files which already have them
  - **[10]** Can create a new ID3 tag on files which don't have one
  - **[5]** Style (again, TRY to split your code into functions.)

Submission

Follow this checklist:

- Did you make sure to put both `.c` files in the same `proj1` directory?
- Did you **make a copy** of that directory just in case?
- Did you name them `mastermind.c` and `id3edit.c`?
- Did you put your **full name and username** at the top of each file in comments?
- Do they compile **using the command lines I gave** in the testing section above?

> ***Never turn in a program that doesn't compile.***

If you answered yes to all those questions, then you can **submit using the same directions as lab1**.