Assignment 2: Photo Gallery

Due date: *Wednesday, November 20, 2019 at 22:00*

> *This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own.*

In a file called `gallery.c` write a C library that manages a photo gallery. For simplicity, the library only supports one file format: PNG. For each photo, the gallery stores three types of metadata: the filename, width and height (in pixels). The library is used for:

- adding/removing photos to the gallery;
- counting the number of photos in the gallery;
- filtering the photos using a predicate; and
- finding a "best fit" photo for some maximum dimensions.

A "best fit" photo of the gallery is a photo of maximal size that fits within a specified maximum width and height. For example, if a gallery contains three photos with dimensions `100x100`, `100x150` and `100x200` respectively, then the "best fit" for maximum dimensions `150x150` would be the second photo.

The library must define (i.e., implement) all the declarations in the following header file:

gallery.h

```c
#ifndef GALLERY_H__
#define GALLERY_H__

struct gallery;

/* Constructor: allocates memory and returns a pointer to a new gallery.
 */
struct gallery *gallery_new();

/* Destructor: frees all the memory allocated to the gallery.
 */
void gallery_destroy(struct gallery *g);

/* Adds a PNG by filename to the gallery. If the filename is already in the
 * gallery, returns -1; if the file does not exist, or if it is not a valid PNG
 * format, returns -2; otherwise, returns 1 on success.
 */
int gallery_add(struct gallery *g, char *filename);

/* Removes a photo from the gallery. Returns 1 on successful removal, otherwise
 * 0 if the photo is not in the gallery.
 */
int gallery_rm(struct gallery *g, char *filename);

/* Returns the number of photos in the gallery.
 */
int gallery_count(struct gallery *g);
```

```c
/* Removes photos from the gallery that do not match a predicate function. The
 * predicate is passed the filename, width and height of each photo. The
 * predicate should return 1 to keep the photo, and 0 to remove the photo. The
 * function returns the number of photos present after filtering.
 */
int gallery_filter(struct gallery *g,
               int (*f)(char *filename, int width, int height));

/* Returns the filename of the photo that is the best fit for the dimensions
 * width*height. If there are multiple best fits, any may be returned. If there
 * are no photos that fit, returns NULL.
 */
char *gallery_bestfit(struct gallery *g, int max_width, int max_height);

#endif // __GALLERY_H__
```

Reading PNG files. PNG files use a binary format. The file begins with a signature, described here:

http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html

After the signature, there are multiple "chunks". The `IHDR` chunk is the first chunk after the signature:

http://www.libpng.org/pub/png/spec/1.2/PNG-Chunks.html

The only information you need extract from the file is the image dimensions (width and height), which is stored in the `IHDR` chunk. To extract this information, you should use a C struct that corresponds to the format of the PNG headers, specifically the signature and the first chunk, `IHDR`.

Note that the bytes of the width and height fields are stored in "network order" or *big-endian*, so you should convert them to the endianess of your computer, which is probably little-endian. You can find information on endianness here: https://en.wikipedia.org/wiki/Endianness. You can easily compute the width and height independently of the endianness of your computer by accessing the fields as sequences of bytes (`uint8_t`). You may also access the fields as wider integers (`uint32_t`) and then convert their values. One way to do that is to use the POSIX function `ntohl`. You may find documentation on-line. For example, a direct implementation of `ntohl` is here: https://codereview.stackexchange.com/a/149751.

Example

Below is an example of adding, removing, filtering and finding the best-fit photo from a few PNGs stored in the local directory.

test.c

```c
#include "gallery.h"

int main() {
   struct gallery *g = gallery_new();

   assert(gallery_add(g, "480x320.png") == 1);
   assert(gallery_add(g, "640x480.png") == 1);
   assert(gallery_count(g) == 2);

   assert(gallery_add(g, "480x320.png") == -1); // already exists

   assert(gallery_add(g, "test.c") == -2); // invalid PNG format
   assert(gallery_count(g) == 2); // so not added

   assert(strcmp(gallery_bestfit(g, 640, 480), "640x480.png") == 0);
   assert(strcmp(gallery_bestfit(g, 640, 481), "640x480.png") == 0);
   assert(strcmp(gallery_bestfit(g, 640, 479), "480x320.png") == 0);
```

```
    assert(gallery_bestfit(g, 0, 0) == NULL); // no best fit found

    int myfilter(char *fn, int w, int h) { return w < 640; }
    assert(gallery_filter(g, myfilter) == 1);

    assert(gallery_rm(g, "480x320.png") == 1);
    assert(gallery_count(g) == 0);

    gallery_delete(g);

    return 0;
}
```

Note that, although the example uses filenames that include the image dimensions, the library extracts the image dimensions from the *contents* of the file, not the filename.

Requirements

To grade your implementation, we will run automated tests. To ensure you receive a full grade, your program must behave exactly as specified here. Below is a check-list of the most important requirements:

- *clean compilation:* your code must compile without warnings (`-Wall`).
- *interface:* your program must use the exact same interface defined in `gallery.h`.
- *error handling:* you should check that the files exist and are valid (i.e. checking the 8-byte PNG signature).

Submission Instructions

Submit one source file named `gallery.c`. Add comments to your code to explain sections of the code that might not be clear. You may also add comments at the beginning of the source file to refer to any and all external sources of information you may have used, including code, suggestions, and comments from other students. If your implementation has limitations and errors you are aware of and were unable to fix, list those as well in the initial comments.

You may use an integrated development environment (IDE) of your choice. However, *do not submit any IDE-specific file,* such as project description files. Also, *make absolutely sure that the file you submit can be compiled and tested with a simple invocation of the standard C compiler.*

Submit your solution through the iCorsi system.