Graded Homework Assignment n. 2

Due date: *Tuesday, December 6, 2022 at 22:00*

In a source file called `musicstore.c` or `musicstore.cc` (or `musicstore.cpp`) write a C or C++ module that implements the following features of a music player application.

```
struct musicstore;

struct musicstore * ms_create ();
void ms_destroy (struct musicstore *);

int ms_read_from_directory (struct musicstore * s, const char * dirname);

typedef void (*song_callback)(const char * artist, const char * album,
                      unsigned sequence, const char * title);

typedef void (*album_callback)(const char * artist, const char * album,
                       unsigned songs_count);

typedef void (*artist_callback)(const char * artist, unsigned albums_count,
                        unsigned songs_count);

void ms_get_artist (const struct musicstore * s,
               const char * artist, artist_callback cb);

void ms_get_albums (const struct musicstore * s,
               const char * artist, const char * album, album_callback cb);

void ms_get_songs (const struct musicstore * s,
              const char * artist, const char * album, const char * title,
              song_callback cb);
```

Each object of type `struct musicstore` represents a collection of music files, each containing a song from an album by a certain artist. `ms_create` and `ms_destroy` are the constructor and destructor for musicstore objects. A `musicstore` stores information about artists, albums and songs as read from the *name* of music files in the file systems (not their content). In particular, `ms_read_from_directory` reads the names of all the files in the given directory and all its subdirectories. Of all the files, the ones that store songs have a name formatted as follows:

artist-name – *album-title* – *song-number* – *song-title*.*extension*

The name consists of five parts. The first four parts are separated by the string " – " (a space, the minus sign, a space), and indicate the artist name, the title of the album, the number of the track within the album, and the title of the song. The fifth and last part is separated from the first four parts by a single period (".") and indicates the type of media encoding. For example, the file name

`Miles Davis – Kind Of Blue – 02 – Freddie Freeloader.ogg`

represents a song entitled *Freddie Freeloader* that is the second track (02) of an album entitled *Kind Of Blue* by artist *Miles Davis.* The file uses the *OGG* encoding. The `ms_read_from_directory` function must record the information derived from all and only the files that conform to this format with extensions `flac`, `mp3`, `mpc`, `ogg`, and `wma`. Any other file must be ignored.

The `ms_read_from_directory` function can be called multiple times with the same `musicstore` object. Artist names are unique, so two identical names represent the same artist. Album titles are unique for each artist, so the file name "`Miles Davis - Kind Of Blue - 01 - So What.ogg`" represents another track in the same *Kind Of Blue* album by Miles Davis. However, different artists might have albums with the same title. For example, there might be multiple *Greatest Hits* albums in the collection, each by a different artist.

A `musicstore` object can be queried using `ms_get_artist`, `ms_get_albums`, or `ms_get_songs`. All these query functions return their results by means of a "callback" function passed as their last parameter. That parameter is a *function pointer,* that is, a pointer to a function that the query function will invoke for each entry matching the query. For example:

```
void ms_get_artist (const struct musicstore * s, const char * a, artist_callback cb) {
   const char * name;
   unsigned album_count, song_count;
   /* ... */
   cb (name, album_count, song_count);
}
```

The `ms_get_artist` function takes a C string, `const char * artist`, interpreted as a query for artists as follows. If `artist` is not the null pointer, then `ms_get_artist` will match zero or at most one artist. Otherwise, if `artist` is the null pointer, then `ms_get_artist` will match *all* the artists stored in the collection. In this case, the artists are returned in alphabetical order of their name. In any case, the results are returned each through one invocation of the call-back function. The parameters of the call-back function are the artist name, and the number of albums and songs (total) by that artist.

The `ms_get_albums` function takes a `const char * artist` and a `const char * album`, representing the name of an artist and the title of an album, respectively. The matching semantics is again that a non-null argument must match exactly, while null values match anything (any names and titles). If the query matches all artists and album titles, the results are returned ordered by artist name and then album title. The parameters of the call-back function are the artist name, the album title, and the number of songs in the album.

The `ms_get_songs` function takes three C strings: `const char * artist`, `const char * album`, and `const char * title`, representing the artist name, the album title, and the song title, respectively. The matching semantics is again that a non-null argument must match exactly, while a null value matches anything. When the song title is not specified, and therefore the results are all the songs of the matching albums and/or by the matching artist, then the songs are returned in their proper sequence within each album. The parameters of the call-back function are the artist name, the album title, the sequence number of the song, and the song title.

The implementation of the query functions must be efficient. In particular, whenever the selection is done by artist name, album title, or song title, then the complexity should be sublinear.

Submission Instructions

Submit one source file as required through the iCorsi system. Add comments to your code to explain sections of the code that might not be clear. You must also add comments at the beginning of the source file to properly acknowledge 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), then 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/C++ compiler.*