06/27/18 07:37:20 /home/ivan/Desktop/Repositorio-Final/track.c

```
#include <stdio.h>
          #include <stdlib.h>
         #include <string.h>
         #include "types.h"
#include "track.h"
  6
         #include "errors.h'
         #include "main.h'
10
         extern char * errors_dictionary[MAX_ERRORS];
11
12
13
         /*Diccionarios de formato, tipos de ordenes y generos*/
14
15
         status_t (*format_output[MAX_FORMATS]) (void *, const void *, FILE *) =
16
17
                   ADT track export to csv.
18
                   ADT_track_export_to_xml
19
         };
20
         int (*sort_output[MAX_SORTS]) (void *, void *) =
21
22
23
                   ADT_track_compare_by_title,
24
25
                   ADT_track_compare_by_artist,
                   ADT_track_compare_by_genre
26
         }:
27
28
         char * genres_dictionary[MAX_GENRES] =
                 "Blues", "Classic Rock", "Country", "Dance", "Disco", "Funk", "Grunge",
"Hip-Hop", "Jazz", "Metal", "New Age", "Oldies", "Other", "Pop",
"R&B", "Rap", "Reggae", "Rock", "Techno", "Industrial", "Alternative",
"Ska", "Death Metal", "Pranks", "Soundtrack", "Euro-Techno", "Ambient", "Trip-Hop",
"Vocal", "Jazz+Funk", "Fusion", "Trance", "Classical", "Instrumental", "Acid",
"House", "Game", "Sound Clip", "Gospel", "Noise", "AlternRock", "Bass",
"Soul", "Punk", "Space", "Meditative", "Instrumental Pop", "Instrumental Rock", "Ethnic",
"Gothic", "Darkwave", "Techno-Industrial", "Electronic", "Pop-Folk", "Eurodance", "Dream",
"Southern Rock", "Comedy", "Cult", "Gangsta", "Top 40", "Christian Rap", "Pop/Funk",
"Jungle", "Native American", "Cabaret", "New Wave", "Psychadelic", "Rave", "Showtunes",
"Trailer", "Lo-Fi", "Tribal", "Acid Punk", "Acid Jazz", "Polka", "Retro",
"Musical", "Rock & Roll", "Hard Rock", "Folk", "Folk-Rock", "National Folk", "Swing",
"Fast Fusion", "Bebob", "Latin", "Revival", "Celtic", "Bluegrass", "Avantgarde",
"Gothic Rock", "Progressive Rock", "Psychedelic Rock", "Symphonic Rock", "Slow Rock", "Big Band", "Chorus",
"Easy Listening", "Acoustic", "Humour", "Speech", "Chanson", "Opera", "Chamber Music",
"Sonata", "Symphony", "Booty Brass", "Primus", "Porn Groove", "Satire", "Slow Jam",
"Club", "Tango", "Samba", "Folklore", "Ballad", "Poweer Ballad", "Rhytmic Soul",
"Freestyle", "Duet", "Punk Rock", "Drum Solo", "A Capela", "Euro-House", "Dance Hall"
29
30
31
32
33
34
35
37
38
39
40
42
43
44
45
47
48
         }:
49
         /*Esta función crea una nueva pista*/
status_t ADT_track_new (ADT_track_t ** track)
51
52
53
                   if (track == NULL)
54
                             return ERROR_NULL_POINTER;
                   if ((*track =(ADT_track_t*)malloc(sizeof(ADT_track_t))) == NULL)
    return ERROR_OUT_OF_MEMORY;
57
58
59
                   (*track)->tag[0] = '\0';
                   (*track)->title[0] = '\0';
(*track)->artist[0] = '\0';
(*track)->album[0] = '\0';
(*track)->year[0] = '\0';
60
61
62
63
                    (*track)->comment[0] = '\0';
65
                    (*track)->genre = 0;
66
                   return OK:
67
68
        }
70
          /*Esta funcion destruye una pista*/
71
72
          status_t ADT_track_delete (void * t)
73
                   ADT_track_t * track;
74
75
                   track = (ADT_track_t *)t;
76
77
                   if (track == NULL)
78
                             return ERROR_NULL_POINTER;
79
                   track->tag[0] = '\0';
80
                  track->tag[0] = \0;
track->title[0] = '\0';
track->artist[0] = '\0';
track->album[0] = '\0';
track->year[0] = '\0';
track->comment[0] = '\0';
81
82
83
84
85
86
                   track->genre = 0;
88
                   free(track);
```

```
89
           track = NULL:
 90
           return OK;
 93
      /*Esta función establece una pista*/
 94
      status_t ADT_track_set (char header[], ADT_track t * track)
 95
 96
 97
           char aux[2]:
 98
          if (header == NULL || track == NULL)
    return ERROR_NULL_POINTER;
 99
100
           \label{eq:memcpy} $$ \ensuremath{\mathsf{memcpy}}$ (track->tag,header+LEXEM_START_TAG,LEXEM_SPAN_TAG); $$ track->tag[LEXEM_SPAN_TAG] = ' \ensuremath{\backslash} \theta'; $$
102
103
104
105
           memcpy(track->title,header+LEXEM_START_TITLE,LEXEM_SPAN_TITLE);
106
           track->title[LEXEM_SPAN_TITLE] =
107
           memcpy(track->artist,header+LEXEM_START_ARTIST,LEXEM_SPAN_ARTIST);
track->artist[LEXEM_SPAN_ARTIST] = '\0';
108
109
110
111
           memcpy(track->album,header+LEXEM_START_ALBUM,LEXEM_SPAN_ALBUM);
112
           track->album[LEXEM_SPAN_ALBUM] = '\0';
113
114
           memcpy(track->year,header+LEXEM_START_YEAR,LEXEM_SPAN_YEAR);
           track->year[LEXEM_SPAN_YEAR] =
115
116
           memcpy(track->comment,header+LEXEM_START_COMMENT,LEXEM_SPAN_COMMENT);
track->comment[LEXEM_SPAN_COMMENT] = '\0';
117
118
120
            emcpy(aux,header+LEXEM_START_GENRE,LEXEM_SPAN_GENRE);
121
           track->genre = aux[0];
122
123
           return OK:
124
125
126
      /*Esta función exporta una pista a un archivo csv*/
      status_t ADT_track_export_to_csv (void * t, const void * context, FILE * file_out)
127
128
129
130
           ADT_track_t * track;
131
           del = *((char *)context);
132
           track = (ADT_track_t *)t;
133
134
          if(fprintf(file_out, "%s", track->title) < 0)
    return ERROR_WRITING_TO_FILE;</pre>
135
136
137
           if (fputc(del, file_out) == EOF)
139
                return ERROR_WRITING_TO_FILE;
140
           if(fprintf(file_out, "%s", track->artist) < 0)</pre>
141
                return ERROR WRITING TO FILE;
142
143
           if(fputc(del, file_out) == EOF)
    return ERROR_WRITING_TO_FILE;
144
145
146
           if(fprintf(file_out, "%s\n", genres_dictionary[track->genre]) < 0)</pre>
                return ERROR_WRITING_TO_FILE;
148
149
150
           return OK:
151
153
      /*Esta función exporta una pista a un archivo csv*/
      \verb|status_t| ADT_track_export_to_xml| (void * t, const void * context, FILE * file_out)|
154
155
156
            har ** xml_contexts;
157
           ADT_track_t * track;
158
           xml_contexts = (char **)context;
track = (ADT_track_t *)t;
159
160
162
                                     "\t%s%s%s\n", xml_contexts[1], xml_contexts[5], xml_contexts[3]) < 0)
                return ERROR_WRITING_TO_FILE;
163
164
           if(fprintf(file_out, "\t\t%s%s%s"
    return ERROR_WRITING_TO_FILE;
165
                                     '\t\t%s%s%s", xml_contexts[1], xml_contexts[6], xml_contexts[3]) < 0)
166
167
           if(fprintf(file_out, "%s", track->title) < 0)
    return ERROR_WRITING_TO_FILE;</pre>
168
169
171
           return ERROR_WRITING_TO_FILE;
172
173
174
           if(fprintf(file\_out, "\t\s%s%s", xml\_contexts[1], xml\_contexts[7], xml\_contexts[3]) < 0)
                return ERROR_WRITING_TO_FILE;
176
           if(fprintf(file_out, "%s", track->artist) < 0)
    return ERROR_WRITING_TO_FILE;</pre>
177
178
```

```
180
          return ERROR_WRITING_TO_FILE;
181
182
          if(fprintf(file_out, "\t\%s%s%s", xml_contexts[1], xml_contexts[8], xml_contexts[3]) < 0)</pre>
184
               return ERROR_WRITING_TO_FILE;
185
          if(fprintf(file_out, "%s", genres_dictionary[track->genre]) < 0)
    return ERROR_WRITING_TO_FILE;</pre>
186
187
          if(fprintf(file_out, "%s%s%s\n", xml_contexts[2], xml_contexts[8], xml_contexts[3]) < 0)
    return ERROR_WRITING_TO_FILE;</pre>
189
190
191
          if(fprintf(file\_out, "\t%s%s%s\n", xml\_contexts[2], xml\_contexts[5], xml\_contexts[3]) < 0)
              return ERROR WRITING TO FILE;
193
194
195
          return OK:
196
197
     /*Esta función compara dos pistas según el artista*/
int ADT_track_compare_by_artist (void * t1, void * t2)
198
199
200
202
          ADT_track_t *track1, *track2;
203
          track1 = (ADT_track_t *)t1;
track2 = (ADT_track_t *)t2;
204
207
          if (track1 == NULL || track2 == NULL)
208
              return 0;
209
          for(i=0; track1->artist[i] && track2->artist[i]; i++)
211
212
              if (track1->artist[i] != track2->artist[i])
213
              {
214
                   return (track1->artist[i] - track2->artist[i]):
215
216
          if (!track1->artist[i] && track2->artist[i])
217
218
          {
219
              return 1;
220
221
          if (track1->artist[i] && !track2->artist[i])
222
          {
223
              return -1:
224
225
          return 0;
226
     }
227
228
     /*Esta función compara dos pistas segun el nombre*/
     int ADT_track_compare_by_title (void * t1, void * t2)
230
231
          size t i:
          ADT_track_t *track1, *track2;
232
233
234
          if (t1 == NULL || t2 == NULL)
235
236
              return 0;
237
          track1 = (ADT_track_t *)t1;
          track2 = (ADT_track_t *)t2;
238
239
240
241
          for(i=0; track1->title[i] && track2->title[i]; i++)
242
243
              if (track1->title[i] != track2->title[i])
244
245
                   return (track1->title[i] - track2->title[i]);
246
248
          if (!track1->title[i] && track2->title[i])
249
250
          {
              return 1:
251
          if (track1->title[i] && !track2->title[i])
253
          {
254
              return -1;
255
256
          return 0;
257
258
259
     /*Esta función compara dos pistas segun el género*/
260
     int ADT_track_compare_by_genre (void * t1, void * t2)
261
262
          ADT_track_t *track1, *track2;
263
          if (t1 == NULL || t2 == NULL)
264
265
              return 0:
          track1 = (ADT_track_t *)t1;
track2 = (ADT_track_t *)t2;
267
268
269
          return track1->genre - track2->genre;
```

271 }