```c
#define _POSIX_C_SOURCE 200809L
#include "abb.h"
#include "analog.h"
#include "cola.h"
#include "fechautil.h"
#include "hash.h"
#include "heap.h"
#include "pila.h"
#include "strutil.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CSV_SEP ','
#define CLAVE_SEP '-'
#define L_CLAVE 29
enum { NUM, AER, ORI, DES, TAI, PRI, FEC, DEP, TIE, CAN };
// Constantes para agregar_archivo
#define ARGS_AGREGAR_ARCHIVO 1
#define AA_FILE_NAME 0
// Constantes para ver_tablero
#define ARGS_VER_TABLERO 4
enum { VT_CANT_VUELOS, VT_MODO, VT_DESDE, VT_HASTA };
// Constantes para info_vuelo
#define ARGS_INFO_VUELO 1
#define IV_NUM_VUELO 0
// Constantes para prioridad_vuelos
#define ARGS_PRIORIDAD_VUELO 1
#define PV_PRIORIDAD 0
// Constantes para borrar
#define ARGS_BORRAR 2
enum { B_DESDE, B_HASTA };
// Constante para abb_vueloscmp
#define AVC_FECHA 0
#define AVC_COD_VUELO 2
// Constante para heap_comparar
#define VPC_PRIORDAD 0
#define VPC_NUM_VUELO 2
#define ASC "asc"
#define DESC "desc"

struct vuelos {
  hash_t *hash_vuelos;
  abb_t *abb_vuelos;
};

typedef struct vuelo {
  char *numero;
  char *aerolinea;
  char *origen;
  char *destino;
  char *numero_cola;
  char *prioridad;
  fecha_t *fecha;
  char *retraso_salida;
  char *tiempo_vuelo;
  char *cancelado;
} vuelo_t;

int abb_vueloscmp(const char *a, const char *b) {
  int resultado;

  char **v_a = split(a, ' ');
  char **v_b = split(b, ' ');

  fecha_t *f_a = fecha_crear(v_a[AVC_FECHA]);
  fecha_t *f_b = fecha_crear(v_b[AVC_FECHA]);
  int fecha_comp = fechacmp(f_a, f_b);
  free(f_a);
  free(f_b);
  resultado = fecha_comp;
  if (resultado == 0)
    resultado = strcmp(v_a[AVC_COD_VUELO], v_b[AVC_COD_VUELO]);
  free_strv(v_a);
  free_strv(v_b);
  return resultado;
}

void destruir_vuelo(void *dato) {
  vuelo_t *vuelo = dato;
  free(vuelo->fecha);
  free(vuelo->numero);
  free(vuelo->aerolinea);
  free(vuelo->origen);
  free(vuelo->destino);
  free(vuelo->numero_cola);
  free(vuelo->prioridad);
  free(vuelo->retraso_salida);
  free(vuelo->tiempo_vuelo);
  free(vuelo->cancelado);
  free(vuelo);
}

vuelos_t *iniciar_vuelos() {
  vuelos_t *vuelos = malloc(sizeof(vuelos_t));
  if (!vuelos)
    return NULL;
  hash_t *hash_vuelos = hash_crear(destruir_vuelo);
  if (!hash_vuelos) {
    free(vuelos);
    return NULL;
  }
  abb_t *abb_vuelos = abb_crear(abb_vueloscmp, free);
```

```
103:   if (!abb_vuelos) {
104:     free(vuelos);
105:     hash_destruir(hash_vuelos);
106:     return NULL;
107:   }
108:   vuelos->hash_vuelos = hash_vuelos;
109:   vuelos->abb_vuelos = abb_vuelos;
110:
111:   return vuelos;
112: }
113:
114: void finalizar_vuelos(vuelos_t *vuelos) {
115:   hash_destruir(vuelos->hash_vuelos);
116:   abb_destruir(vuelos->abb_vuelos);
117:   free(vuelos);
118: }
119:
120: void inicializar_vuelo(char **datos, vuelo_t *vuelo) {
121:   vuelo->numero = strdup(datos[NUM]);
122:   vuelo->aerolinea = strdup(datos[AER]);
123:   vuelo->origen = strdup(datos[ORI]);
124:   vuelo->destino = strdup(datos[DES]);
125:   vuelo->numero_cola = strdup(datos[TAI]);
126:   vuelo->prioridad = strdup(datos[PRI]);
127:   vuelo->fecha = fecha_crear(datos[FEC]);
128:   vuelo->retraso_salida = strdup(datos[DEP]);
129:   vuelo->tiempo_vuelo = strdup(datos[TIE]);
130:   vuelo->cancelado = strdup(datos[CAN]);
131: }
132:
133: /* Dado una fecha y un numero_vuelo
134:  * devuelve una cadena de caracteres con el formato:
135:  * "fecha - numero_vuelo".
136:  * La memoria de la cadena debe ser borrada manualmente.
137:  */
138: char *generar_clave(fecha_t *fecha, char *numero_vuelo) {
139:   char *clave = malloc(sizeof(char) * (L_CLAVE));
140:   char *fecha_hora = fecha_a_str(fecha);
141:   sprintf(clave, "%s %c %s", fecha_hora, CLAVE_SEP, numero_vuelo);
142:   free(fecha_hora);
143:   return clave;
144: }
145:
146: bool _agregar_archivo(vuelos_t *vuelos, const char *nombre_archivo) {
147:   FILE *archivo_vuelos = fopen(nombre_archivo, "r");
148:   if (!archivo_vuelos)
149:     return false;
150:   hash_t *hash_vuelos = vuelos->hash_vuelos;
151:   abb_t *abb_vuelos = vuelos->abb_vuelos;
152:
153:   char *linea = NULL;
154:   size_t cantidad = 0;
155:   ssize_t leidos = 0;
156:
157:   while ((leidos = getline(&linea, &cantidad, archivo_vuelos)) > 0) {
158:     linea[leidos - 1] = '\0';
159:     char **datos_vuelo = split(linea, CSV_SEP);
160:     char *numero_vuelo = datos_vuelo[NUM];
161:
162:     vuelo_t *vuelo = malloc(sizeof(vuelo_t));
163:     inicializar_vuelo(datos_vuelo, vuelo);
164:     char *clave = generar_clave(vuelo->fecha, numero_vuelo);
165:
166:     if (hash_pertenece(hash_vuelos, numero_vuelo)) {
167:       vuelo_t *vuelo_borrado = hash_borrar(hash_vuelos, numero_vuelo);
168:       char *clave_borrado =
169:           generar_clave(vuelo_borrado->fecha, vuelo_borrado->numero);
170:       free(abb_borrar(abb_vuelos, clave_borrado));
171:       destruir_vuelo(vuelo_borrado);
172:       free(clave_borrado);
173:     }
174:     abb_guardar(abb_vuelos, clave, NULL);
175:     free(clave);
176:     hash_guardar(hash_vuelos, numero_vuelo, vuelo);
177:     free_strv(datos_vuelo);
178:   }
179:   free(linea);
180:   fclose(archivo_vuelos);
181:   return true;
182: }
183: /* Dada una cola ya inicializada, se invierten los loselementos
184:  * de la misma, es decir, el primer elemento pasa a estar Ãºltimo.
185:  */
186: void invertir_cola(cola_t *cola) {
187:   pila_t *pila_aux = pila_crear();
188:   while (!cola_esta_vacia(cola))
189:     pila_apilar(pila_aux, cola_desencolar(cola));
190:   while (!pila_esta_vacia(pila_aux))
191:     cola_encolar(cola, pila_desapilar(pila_aux));
192:   pila_destruir(pila_aux);
193: }
194:
195: bool _ver_tablero(vuelos_t *vuelos, size_t cant_vuelos, const char *modo,
196:                   fecha_t *desde, fecha_t *hasta) {
197:   if (abb_cantidad(vuelos->abb_vuelos) == 0 ||
198:       hash_cantidad(vuelos->hash_vuelos) == 0)
199:     return true;
200:   cola_t *resultado = cola_crear();
201:   if (!resultado)
202:     return false;
203:   // Fecha - 0 (cualquier vuelo)
204:   char *clave = generar_clave(desde, 0);
```

```
205:   // Incremento 1 segundo la fecha lÃmite
206:   fecha_sumar_segundos(hasta, 1);
207:   char *clave_limite = generar_clave(hasta, 0);
208:
209:   abb_iter_t *iter_vuelos = abb_iter_in_crear_desde(vuelos->abb_vuelos, clave);
210:   free(clave);
211:   int i = 0;
212:   while (!abb_iter_in_al_final(iter_vuelos)) {
213:     const char *clave_actual = abb_iter_in_ver_actual(iter_vuelos);
214:     if (abb_vueloscmp(clave_actual, clave_limite) >= 0)
215:       break;
216:     cola_encolar(resultado, strdup(clave_actual));
217:     i++;
218:     if (strcmp(modo, ASC) == 0 && i >= cant_vuelos)
219:       break;
220:     if (i > cant_vuelos)
221:       free(cola_desencolar(resultado));
222:     abb_iter_in_avanzar(iter_vuelos);
223:   }
224:   free(clave_limite);
225:   abb_iter_in_destruir(iter_vuelos);
226:
227:   if (strcmp(modo, DESC) == 0)
228:     invertir_cola(resultado);
229:
230:   while (!cola_esta_vacia(resultado)) {
231:     char *str_res = cola_desencolar(resultado);
232:     printf("%s\n", str_res);
233:     free(str_res);
234:   }
235:   cola_destruir(resultado, NULL);
236:
237:   return true;
238: }
239:
240: /* Dado un vuelo, se muestra en pantalla toda la informaciÃ³n
241:  * de la estructura.
242:  */
243: void mostrar_vuelo(vuelo_t *vuelo) {
244:   char *fecha_hora = fecha_a_str(vuelo->fecha);
245:   printf("%s %s %s %s %s %s %s %s %s %s\n", vuelo->numero, vuelo->aerolinea,
246:          vuelo->origen, vuelo->destino, vuelo->numero_cola, vuelo->prioridad,
247:          fecha_hora, vuelo->retraso_salida, vuelo->tiempo_vuelo,
248:          vuelo->cancelado);
249:   free(fecha_hora);
250: }
251:
252: bool _info_vuelo(vuelos_t *vuelos, const char *cod_vuelo) {
253:   vuelo_t *vuelo = (vuelo_t *)hash_obtener(vuelos->hash_vuelos, cod_vuelo);
254:   if (!vuelo)
255:     return false;
256:   mostrar_vuelo(vuelo);
257:   return true;
258: }
259:
260: /* Devuelve un numero:
261:  *   menor a 0  si  a < b
262:  *       0      si  a == b
263:  *   mayor a 0  si  a > b
264:  * a es mayor a b si tiene mayor prioridad que b.
265:  * Si dos vuelos tienen la misma prioridad, se desempatarÃ¡ por el cÃ³digo de
266:  * vuelo mostrÃ¡ndolos de menor a mayor (tomado como cadena).
267:  */
268: int vuelos_prioridad_cmp(const void *a, const void *b) {
269:   char **datos_vuelo1 = split((char *)a, ' ');
270:   char **datos_vuelo2 = split((char *)b, ' ');
271:   int prioridad1 = atoi(datos_vuelo1[VPC_PRIORDAD]);
272:   int prioridad2 = atoi(datos_vuelo2[VPC_PRIORDAD]);
273:   char *num_vuelo1 = datos_vuelo1[VPC_NUM_VUELO];
274:   char *num_vuelo2 = datos_vuelo2[VPC_NUM_VUELO];
275:   int resultado = prioridad2 - prioridad1;
276:   if (resultado == 0)
277:     resultado = strcmp(num_vuelo1, num_vuelo2);
278:   free_strv(datos_vuelo1);
279:   free_strv(datos_vuelo2);
280:   return resultado;
281: }
282:
283: /* Dado una prioridad y un numero_vuelo (ambos strings)
284:  * devuelve una cadena de caracteres con el formato:
285:  * "prioridad - numero_vuelo".
286:  * La memoria de la cadena debe ser borrada manualmente.
287:  */
288: char *generar_elemento_heap(char *prioridad, char *numero_vuelo) {
289:   size_t lng_clave = strlen(prioridad) + strlen(numero_vuelo) + 10;
290:   char *clave = malloc(lng_clave * sizeof(char));
291:   sprintf(clave, "%s %c %s", prioridad, CLAVE_SEP, numero_vuelo);
292:   return clave;
293: }
294:
295: bool _prioridad_vuelos(vuelos_t *vuelos, int cant_vuelos) {
296:   heap_t *vuelos_mayor_prioridad = heap_crear(vuelos_prioridad_cmp);
297:   if (!vuelos_mayor_prioridad)
298:     return false;
299:   hash_iter_t *iter = hash_iter_crear(vuelos->hash_vuelos);
300:
301:   if (!iter)
302:     return false;
303:
304:   vuelo_t *vuelo = NULL;
305:   char *clave = NULL;
306:
```

```
307:   for (int i = 0; i < cant_vuelos; i++) {
308:     if (hash_iter_al_final(iter))
309:       break;
310:     clave = (char *)hash_iter_ver_actual(iter);
311:     vuelo = (vuelo_t *)hash_obtener(vuelos->hash_vuelos, clave);
312:     char *elemento_heap =
313:         generar_elemento_heap(vuelo->prioridad, vuelo->numero);
314:
315:     heap_encolar(vuelos_mayor_prioridad, elemento_heap);
316:     hash_iter_avanzar(iter);
317:   }
318:
319:   char *vuelo_menor_prioridad = (char *)heap_ver_max(vuelos_mayor_prioridad);
320:   while (!hash_iter_al_final(iter)) {
321:     clave = (char *)hash_iter_ver_actual(iter);
322:     vuelo = (vuelo_t *)hash_obtener(vuelos->hash_vuelos, clave);
323:     char *vuelo_nuevo =
324:         (char *)generar_elemento_heap(vuelo->prioridad, vuelo->numero);
325:
326:     if (vuelos_prioridad_cmp(vuelo_nuevo, vuelo_menor_prioridad) < 0) {
327:       free(heap_desencolar(vuelos_mayor_prioridad));
328:       heap_encolar(vuelos_mayor_prioridad, vuelo_nuevo);
329:       vuelo_menor_prioridad = (char *)heap_ver_max(vuelos_mayor_prioridad);
330:     } else
331:       free(vuelo_nuevo);
332:     hash_iter_avanzar(iter);
333:   }
334:
335:   hash_iter_destruir(iter);
336:
337:   size_t cantidad_vuelos = heap_cantidad(vuelos_mayor_prioridad);
338:   char **vuelos_prioritarios = malloc(cantidad_vuelos * sizeof(char *));
339:
340:   for (int i = (int)cantidad_vuelos - 1; i >= 0; i--) {
341:     vuelos_prioritarios[i] = (char *)heap_desencolar(vuelos_mayor_prioridad);
342:   }
343:
344:   for (int i = 0; i < cantidad_vuelos; i++) {
345:     printf("%s\n", vuelos_prioritarios[i]);
346:     free(vuelos_prioritarios[i]);
347:   }
348:
349:   heap_destruir(vuelos_mayor_prioridad, free);
350:   free(vuelos_prioritarios);
351:   return true;
352: }
353:
354: bool _borrar(vuelos_t *vuelos, fecha_t *desde, fecha_t *hasta) {
355:   cola_t *resultado = cola_crear();
356:   if (!resultado)
357:     return false;
358:
359:   // Fecha - 0 (cualquier vuelo)
360:   char *clave_inicial = generar_clave(desde, 0);
361:   // Incremento 1 segundo la fecha lÃmite
362:   fecha_sumar_segundos(hasta, 1);
363:
364:   char *clave_limite = generar_clave(hasta, 0);
365:   abb_iter_t *iter_vuelos =
366:       abb_iter_in_crear_desde(vuelos->abb_vuelos, clave_inicial);
367:   free(clave_inicial);
368:   while (!abb_iter_in_al_final(iter_vuelos)) {
369:     const char *clave_actual = abb_iter_in_ver_actual(iter_vuelos);
370:     if (abb_vueloscmp(clave_actual, clave_limite) >= 0)
371:       break;
372:     cola_encolar(resultado, strdup(clave_actual));
373:     abb_iter_in_avanzar(iter_vuelos);
374:   }
375:
376:   while (!cola_esta_vacia(resultado)) {
377:     char *actual = cola_desencolar(resultado);
378:     char **actual_v = split(actual, ' ');
379:
380:     abb_borrar(vuelos->abb_vuelos, actual);
381:     vuelo_t *vuelo_borrado =
382:         hash_borrar(vuelos->hash_vuelos, actual_v[AVC_COD_VUELO]);
383:     mostrar_vuelo(vuelo_borrado);
384:     destruir_vuelo(vuelo_borrado);
385:     free(actual);
386:     free_strv(actual_v);
387:   }
388:
389:   cola_destruir(resultado, free);
390:   free(clave_limite);
391:   abb_iter_in_destruir(iter_vuelos);
392:
393:   return true;
394: }
395:
396: /************************************************************
397:  *                        FUNCIONES WRAPER
398:  ************************************************************/
399: /*
400:  * Son las funciones principales del programa.
401:  * Reciben por parametro una estructura de vuelos ya inicializada,
402:  * un vector (args) con los argumentos necesarios para el comando particular,
403:  * y la cantidad de argumentos en ese vector(argc).
404:  * Devuelven true en caso de que los parametros con correctos y ademÃ¡s
405:  * pudo ejecutar correctamente el comando.
406:  */
407: bool agregar_archivo(vuelos_t *vuelos, char **args, size_t argc) {
408:   if (argc != ARGS_AGREGAR_ARCHIVO)
```

```
409:     return false;
410:   return _agregar_archivo(vuelos, args[AA_FILE_NAME]);
411: }
412:
413: bool ver_tablero(vuelos_t *vuelos, char **args, size_t argc) {
414:
415:   if (argc != ARGS_VER_TABLERO)
416:     return false;
417:   size_t cant_vuelos = atoi(args[VT_CANT_VUELOS]);
418:   if (cant_vuelos <= 0)
419:     return false;
420:   char *modo = args[VT_MODO];
421:   if (strcmp(modo, DESC) != 0 && strcmp(modo, ASC) != 0)
422:     return false;
423:   fecha_t *desde = fecha_crear(args[VT_DESDE]);
424:   fecha_t *hasta = fecha_crear(args[VT_HASTA]);
425:
426:   // Si pudo allocar memoria para las fechas y ademÃ¡s son validas
427:   bool resultado =
428:       (desde && hasta) && _ver_tablero(vuelos, cant_vuelos, modo, desde, hasta);
429:   free(desde);
430:   free(hasta);
431:   return resultado;
432: }
433:
434: bool info_vuelo(vuelos_t *vuelos, char **args, size_t argc) {
435:   if (argc != ARGS_INFO_VUELO)
436:     return false;
437:   return _info_vuelo(vuelos, args[IV_NUM_VUELO]);
438: }
439:
440: bool prioridad_vuelos(vuelos_t *vuelos, char **args, size_t argc) {
441:   if (argc != ARGS_PRIORIDAD_VUELO)
442:     return false;
443:   int prioridad = atoi(args[PV_PRIORIDAD]);
444:   return (prioridad > 0) && _prioridad_vuelos(vuelos, prioridad);
445: }
446:
447: bool borrar(vuelos_t *vuelos, char **args, size_t argc) {
448:   if (argc != ARGS_BORRAR)
449:     return false;
450:   bool ok = true;
451:   fecha_t *desde = fecha_crear(args[B_DESDE]);
452:   fecha_t *hasta = fecha_crear(args[B_HASTA]);
453:   if (fechacmp(desde, hasta) > 0)
454:     ok = false;
455:   // Si pudo allocar memoria para las fechas y ademÃ¡s son validas
456:   ok &= (desde && hasta) && _borrar(vuelos, desde, hasta);
457:   free(desde);
458:   free(hasta);
459:   return ok;
460: }
```

```c
#ifndef ANALOG_H
#define ANALOG_H

#include "fechautil.h"
#include "hash.h"
#include <stdbool.h>
#include <stddef.h>

typedef struct vuelos vuelos_t;

/* Inicializa la estructura de tipo vuelo y la devuelve.
 * En caso de que no pueda generarla o allocar memoria,
 * la funciÃ³n devuelve NULL.
 */
vuelos_t *iniciar_vuelos();
/* Dada una estructura de vuelo ya inicializada,
 * libera la memoria correspondiente la estructura
 * de vuelo.
 */
void finalizar_vuelos(vuelos_t *vuelos);

/* Dada una estrucutra de vuelos y un array args donde contiene
 * el nombre del archivo a cargar, la funciÃ³n actualiza la estructura
 * vuelos con los datos del archivo indicado.
 * argc: longitud de args.
 */
bool agregar_archivo(vuelos_t *vuelos, char **args, size_t argc);

/* Muestra en pantalla la informaciÃ³n de vuelo en base a:
 * K cantidad vuelos: cantidad K de vuelos a mostrar
 * modo: asc/desc : cadena con â\200\234ascâ\200\235 o â\200\234descâ\200\235, indican el orden a elegir
 * utilizando el campo fecha de despegue
 * desde: cadena en formato YYYY-MM-DDTHH:MM:SS que indica el tiempo desde que
 * se tienen que mostrar los vuelos, los vuelos con una fecha de despegue
 * anteriores al tiempo ingresado no se tienen que mostrar.
 * hasta: cadena en formato YYYY-MM-DDTHH:MM:SS que indica el tiempo hasta que
 * se tienen que mostrar los vuelos, los vuelos con una fecha de despegue
 * posteriores al tiempo ingresado no se tienen que mostrar.
 *
 * Almacenados en ese orden en args.
 * argc: longitud de args.
 */
bool ver_tablero(vuelos_t *vuelos, char **args, size_t argc);

/* Dado un numero de vuelo (almacenado en args),
 * devuelve toda la informaciÃ³n de ese vuelo almacenado
 * en la estructura de vuelos.
 * argc: longitud de args.
 */
bool info_vuelo(vuelos_t *vuelos, char **args, size_t argc);
/* Muestra en pantalla los K vuelos con
 * mayor prioridad de vuelos (K almacenado en args),
 * argc: longitud de args.
 */
bool prioridad_vuelos(vuelos_t *vuelos, char **args, size_t argc);
/* Dada una estructura de vuelos ya inicializada, borra toda la informaciÃ³n
 * comprendida entre las fechas desde, hasta (almacenadas en el array args)
 * argc: longitud de args.
 */
bool borrar(vuelos_t *vuelos, char **args, size_t argc);

#endif // ANALOG_H
```

```c
#include "fechautil.h"
#include "strutil.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define L_FECHA 11 // Longitud Fecha: 4 + 2 + 2 + 2(separadores) + 1
#define L_HORA 9   // Longitud Hora: 2 + 2 + 2 + 2(separadores) + 1
#define L_FECHA_HORA (L_FECHA + L_HORA + 1)
#define L_FECHA_S 10
#define VALID_FECHA 19
#define FECHA_HORA_SEP 'T'
#define FECHA_SEP '-'
#define HORA_SEP ':'
enum { FECHA, HORA };    // Orden Fecha_Hora
enum { ANIO, MES, DIA }; // Orden de Fecha
enum { HH, MM, SS };     // Orden de Hora
#define SEGUNDOS 60      // Segundos en 1 minuto
#define MINUTOS 60       // Minutos en 1 hora
struct fecha {
  int d; // DÃ-a
  int m; // Mes
  int a; // AÃ±o
  int H; // Hora
  int M; // Min
  int S; // Seg
};

fecha_t *fecha_crear(const char *str) {
  if (!fecha_valida(str))
    return NULL;
  fecha_t *fecha_r = malloc(sizeof(fecha_t));
  if (!fecha_r)
    return NULL;

  char **fecha_hora = split(str, FECHA_HORA_SEP);
  char **fecha = split(fecha_hora[FECHA], FECHA_SEP);

  fecha_r->d = atoi(fecha[DIA]);
  fecha_r->m = atoi(fecha[MES]);
  fecha_r->a = atoi(fecha[ANIO]);

  fecha_r->H = 0;
  fecha_r->M = 0;
  fecha_r->S = 0;
  if (strlen(str) != L_FECHA_S) {
    char **hora = split(fecha_hora[HORA], HORA_SEP);
    fecha_r->H = atoi(hora[HH]);
    fecha_r->M = atoi(hora[MM]);
    fecha_r->S = atoi(hora[SS]);
    free_strv(hora);
  }
  free_strv(fecha);
  free_strv(fecha_hora);

  return fecha_r;
}
int fechacmp(const fecha_t *a, const fecha_t *b) {
  // Me fijo primero la fecha
  int dif_anio = a->a - b->a;
  if (dif_anio != 0)
    return dif_anio;
  // Mismo aÃ±o
  int dif_mes = a->m - b->m;
  if (dif_mes != 0)
    return dif_mes;
  // Mismo aÃ±o y mismo mes
  int dif_dia = a->d - b->d;
  if (dif_dia != 0)
    return dif_dia;
  // Mismo aÃ±o, mes, dia
  // Me fijo la hora->
  int dif_hora = a->H - b->H;
  if (dif_hora != 0)
    return dif_hora;
  // Misma hora
  int dif_min = a->M - b->M;
  if (dif_min != 0)
    return dif_min;
  // Misma hora y minutos
  int dif_seg = a->S - b->S;
  return dif_seg;
}

char *fecha_a_str(fecha_t *fecha) {
  char fecha_s[L_FECHA];
  sprintf(fecha_s, "%04d%c%02d%c%02d", fecha->a, FECHA_SEP, fecha->m, FECHA_SEP,
          fecha->d);
  char hora[L_HORA];
  sprintf(hora, "%02d%c%02d%c%02d", fecha->H, HORA_SEP, fecha->M, HORA_SEP,
          fecha->S);
  char *fecha_hora = malloc(sizeof(char) * (L_FECHA_HORA));
  sprintf(fecha_hora, "%s%c%s", fecha_s, FECHA_HORA_SEP, hora);
  return fecha_hora;
}

bool fecha_valida(const char *str) {
  return strlen(str) == VALID_FECHA || strlen(str) == L_FECHA_S;
}

void fecha_sumar_segundos(fecha_t *fecha, int segundos) {
  fecha->S = fecha->S + segundos;
}
```

```c
#ifndef FECHAUTIL_H
#define FECHAUTIL_H

#include <stdbool.h>
#include <stddef.h>
typedef struct fecha fecha_t;

/* Dada una cadena de caracteres en el formato
 * "YYYY-MM-DDTHH:MM:SS" devuelve una estrucutra
 * de fecha. En caso de no poder allocar memoria
 * devuelve NULL.
 */
fecha_t *fecha_crear(const char *str);

/* Dada una estructura de fecha, devuelve una cadena
 * con el formato "YYYY-MM-DDTHH:MM:SS".
 * Se debe liberar la memoria de la cadena manualmente.
 */
char *fecha_a_str(fecha_t *fecha);

/* Dada dos fechas a, b, devolver:
 *   menor a 0  si  a < b
 *       0      si  a == b
 *   mayor a 0  si  a > b
 */
int fechacmp(const fecha_t *a, const fecha_t *b);

/*
 * Devuelve true si el string cumple con el formato
 * "YYYY-MM-DDTHH:MM:SS". False caso contrario.
 */
bool fecha_valida(const char *str);

/* Sirve para incrementar una fecha.
 * No se asegura que la fecha sea vÃ¡lida.
 */
void fecha_sumar_segundos(fecha_t *fecha, int segundos);

#endif // FECHAUTIL_H
```

```c
#define _POSIX_C_SOURCE 200809L
#include "analog.h"
#include "hash.h"
#include "strutil.h"
#include "testing.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CMD 0
#define CMD_SEP ' '
#define ERROR -1

typedef bool (*comando_t)(vuelos_t *, char **, size_t);

typedef struct func {
  comando_t *comando;
} func_t;
/* ******************************************************************
 *                          PROGRAMA PRINCIPAL
 * *****************************************************************/
/* Dado una cadena de caracteres, la funcion devuelve un puntero
 * a la funciÃ³n correspondiente con ese comando.
 * En caso de que el comando sea invÃ¡lido, devuelve NULL
 */
comando_t obtener_comando(char *comando) {
  const char *NOMBRE_COMANDO[] = {"agregar_archivo", "ver_tablero",
                                  "info_vuelo", "prioridad_vuelos", "borrar"};
  comando_t FUNCION_COMANDO[] = {agregar_archivo, ver_tablero, info_vuelo,
                                 prioridad_vuelos, borrar};
  const size_t CANTIDAD_COMANDOS = 5;
  for (int i = 0; i < CANTIDAD_COMANDOS; i++) {
    if (strcmp(comando, NOMBRE_COMANDO[i]) != 0)
      continue;
    return FUNCION_COMANDO[i];
  }
  return NULL;
}
/* Dado un array de cadenas de caracteres, devuelve
 * un nuevo array que tiene los mismos elementos del
 * array original desde inicio hasta final.
 * El array debe ser liberado manualmente.
 */
char **slice(char *arr[], int inicio, int fin) {
  char **resultado = calloc((fin - inicio + 1), sizeof(char *));
  for (int i = 0; i < (fin - inicio); i++) {
    resultado[i] = strdup(arr[inicio + i]);
  }
  return resultado;
}
/* Dado un vector de cadenas de caracteres, con el Ãºltimo elemento NULL
 * la funciÃ³n devuelve la longitud del vector hasta NULL.
 */
size_t len_entrada(char **entrada) {
  size_t i;
  for (i = 0; entrada[i]; i++)
    if (strlen(entrada[i]) == 0)
      return 0;
  return i++;
}
/* Dado un array de cadenas, libera la memoria de
 * cada una de las cadenas y del array en sÃ.
 */
void free_args(char *args[]) {
  for (int i = 0; args[i]; i++)
    free(args[i]);
  free(args);
}

/* Dado un puntero a la estructura de vuelos,
 * previamente inicializado, y una entrada de
 * usuario (vector de cadena de caracteres cuyo
 * primer elemento es el comando a ejecutar)
 * Ejecuta el comando y devuelve True si pudo
 * ejecutar el comando correctamente.
 */
bool ejecutar(vuelos_t *vuelos, char *entrada[]) {
  char *comando = entrada[CMD];
  size_t len = len_entrada(entrada);
  if (len == 0)
    return false;
  char **args = slice(entrada, 1, (int)len);

  comando_t funcion = obtener_comando(comando);
  bool ok = false;
  if (funcion)
    ok = funcion(vuelos, args, len - 1);
  free_args(args);
  return ok;
}

/* FunciÃ³n principal del programa.
 */
int main(void) {
  vuelos_t *vuelos = iniciar_vuelos();
  if (!vuelos)
    return ERROR;
  char *linea = NULL;
  size_t tam = 0;
  ssize_t len = 0;

  while ((len = getline(&linea, &tam, stdin)) > 0) {
```

```
103:
104:     linea[len - 1] = '\0';
105:     char **entrada = split(linea, CMD_SEP);
106:     if (ejecutar(vuelos, entrada))
107:       printf("OK\n");
108:     else if (*entrada[CMD]) // Evita saltos de linea
109:       fprintf(stderr, "Error en comando %s\n", entrada[CMD]);
110:     free_strv(entrada);
111:   }
112:
113:   free(linea);
114:
115:   finalizar_vuelos(vuelos);
116:
117:   return 0;
118: }
```