```c

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <windows.h>
#include "listaHabitaciones.h"



/** PROTOTIPOS DE FUNCIONES FASE 1*/

/**
    leerTexto: Solicita que se introduzca desde la entrada estándar una cadena de caracteres. La cadena
    se devolverá en dato y será del tamaño máximo indicado en size (sin contabilizar '/0'). En mensaje
    se pasa la frase que aparece en la salida estándar para solicitar la cadena.
    De la cadena leída se elimina el '\n' en caso de que se hubiera guardado. Si se teclea una cadena
    de tamaño mayor al indicado, se guardan los size primeros caracteres.
    No admite como válida una cadena vacía. Si se pulsa enter tras la frase de solicitud, se vuelve
    a pedir al usuario que introduzca la cadena.
    Parámetros de entrada:
        mensaje: Cadena de caracteres con la frase de solicitud.
        size: Entero. Tamaño efectivo (sin contar '\0') máximo de la cadena que se quiere leer.
    Parámetro de salida pasado por referencia:
        dato: Cadena de caracteres. Cadena que se introduce desde la entrada estándar.
*/
void leerTexto(const char mensaje[], char dato[], int size);
/**
    leerBooleano: Solicita que se introduzca desde la entrada estándar un carácter que puede ser 's',
    'S', 'n' o 'N'. Si se introduce cualquier otro carácter solicita de nuevo la respuesta.
    Si el usuario teclea 's o 'S' la función devuelve true y si se teclea 'n' o 'N' devuelve false.
    Parámetro de entrada:
        mensaje: Cadena de caracteres con la frase de solicitud.
    Valor devuelto por la función:
        Booleano: true si se teclea 's' o 'S', false si se teclea 'n' o 'N'.
*/
bool leerBooleano(const char mensaje[]);
/**
    existeCliente: Comprueba si en la estructura ocupacion existe un cliente con el dni que se pasa
    como parámetro.
    Parámetros de entrada:
        dni: Cadena de caracteres con el DNI del cliente a buscar.
        ocupacion: Estructura con la información de la habitación en la que se busca al cliente.
    Precondiciones:
        ocupacion: Tiene que estar inicializado.
    Valor devuelto por la función:
        Booleano: true si el cliente está registrado en la habitación, false si no lo está.
*/
bool existeCliente(const tId dni, tRegistroClientes ocupacion);
/**
    habitacionLlena: Comprueba si la habitación está llena.
    Parámetro de entrada:
        ocupacion: Estructura con la información de la habitación.
    Precondiciones:
        ocupacion tiene que estar inicializado.
    Valor devuelto por la función:
        Booleano: true si la habitación está llena, false si no lo está.
*/
bool habitacionLlena (tRegistroClientes ocupacion);
/**
    vaciarHabitacion: Modifica los datos de la habitación para indicar que está vacía. Para ello pone a 0
    el valor del campo numeroClientes.
    Parámetro de salida pasado por referencia:
        ocupacion: Estructura con la información de la habitación.
*/
void vaciarHabitacion (tRegistroClientes *ocupacion);
/**
    altaCliente: Si la habitación no está llena, y el cliente no está ya registrado en esa habitación,
    se incluyen los datos del cliente en la estructura ocupacion, detrás de los datos del último cliente
    registrado.
    Parámetros de entrada:
        dni: Cadena de caracteres con el DNI del cliente.
```

```
73          nombre: Cadena de caracteres con el nombre y el apellido o apellidos del cliente.
74          allInclusive: Booleano con valor true si el cliente elige la modalidad de "todo incluido",
75          false si elige "alojamiento y desayuno" (son las 2 modalidades de reserva disponibles).
76      Parámetro de salida pasado por referencia:
77          ocupacion: Estructura con la información de la habitación.
78      Valor devuelto por la función para control de errores:
79          Entero: 0 si el cliente se ha registrado correctamente
80                  1 si el cliente ya estaba registrado
81                  2 si no hay sitio en la habitación para registrar nuevos clientes
82  */
83  int altaCliente (tId dni, tNombre nombre, bool allInclusive, tRegistroClientes *ocupacion);
84  /**
85      listarClientes: Lista todos los datos de los clientes registrados en la habitación.
86      Parámetro de entrada:
87          ocupacion: Estructura con la información de la habitación.
88      Precondiciones:
89          ocupacion tiene que estar inicializado.
90  */
91  void listarClientes (tRegistroClientes ocupacion);
92
93  /** PROTOTIPOS DE FUNCIONES FASE 2*/
94
95  /**
96      leerEnteroEnRango: Solicita que se introduzca desde la entrada estándar un entero comprendido en el
97      rango [inferior, superior]. Si se teclea un valor fuera de rango, se vuelve a pedir el número.
98      Parámetros de entrada:
99          mensaje: Cadena de caracteres con la frase de solicitud.
100         inferior: Entero, rango inferior.
101         superior: Entero, rango superior.
102     Valor devuelto por la función:
103         Entero: el número dentro del rango.
104 */
105 int leerEnteroRango (const char mensaje [], int inferior, int superior);
106
107 /**
108     habitacionRegistrada: Comprueba si la habitación indicada en numero está dada de alta
109     en la lista de habitaciones. Si lo está devuelve la posición en la que está en el array habitaciones.
110     Parámetros de entrada:
111         numero: Entero con el número de la habitación.
112         habitaciones: Array con la lista de habitaciones registradas en el hotel.
113     Precondiciones:
114         numero: Tiene que estar en el rango [100,200].
115         habitaciones: Tiene que estar inicializado.
116     Parámetro de salida pasado por referencia:
117         pos: Entero con la posición en la que se encuentra la habitación en el array habitaciones.
118     Valor devuelto por la función:
119         Booleano: true si la habitación está registrada, false si no lo está.
120 */
121 bool habitacionRegistrada (int numero, const tListaHabitaciones habitaciones, int *pos);
122
123 /**
124     hayEspacioEnHotel: Comprueba si hay espacio libre en la lista de habitaciones del hotel.
125     Si lo hay devuelve la posición del primer hueco libre en el array de habitaciones.
126     Parámetro de entrada:
127         habitaciones: Array con la lista de habitaciones registradas en el hotel.
128     Precondiciones:
129         habitaciones: tiene que estar inicializado.
130     Parámetro de salida pasado por referencia:
131         pos: Entero con la posicion en la que se encuentra el primer hueco libre en habitaciones.
132     Valor devuelto por la función:
133         Booleano: true si hay hueco en el array de habitaciones del hotel, false si no lo hay.
134 */
135 bool hayEspacioEnHotel (const tListaHabitaciones habitaciones, int *pos);
136
137 /**
138     abrirHotel: Se abre el hotel poniendo todas las habitaciones libres.
139     Parámetro de salida pasado por referencia:
140         habitaciones: Array con la información de las habitaciones del hotel.
141 */
142 void abrirHotel (tListaHabitaciones *habitaciones);
143
144 /**
145     annadirHabitacion: Se comprueba si  hay sitio en la habitación. Si lo hay, se comprueba
146     si la habitación cuyo número se pasa como parámetro está registrada en el hotal.
147     Si no lo está se registran sus datos en la primera posición libre del array de habitaciones.
```

```
148         Parámetro de entrada:
149             numero: Entero con el número de la habitación.
150             tipo: Cadena de caracteres con la descripción del tipo de habitación, por ejemplo
    "suite",
151             "individual", etc.
152         Precondiciones:
153             numero: Tiene que estar en el rango [100,200].
154         Parámetro de entrada/salida pasado por referencia:
155             habitaciones: Array con la información de las habitaciones del hotel.
156         Precondiciones:
157             habitaciones: Tiene que estar inicializado.
158         Valor devuelto por la función para control de errores:
159             Entero: 0 si la habitación se ha registrado correctamente.
160                     1 si la habitación ya estaba registrada.
161                     2 si no hay sitio en el hotel para registrar nuevas habitaciones.
162     */
163     int annadirHabitacion (int numero, tTextoTipo tipo, tListaHabitaciones *habitaciones);
164
165     /**
166         borrarHabitacion: Se comprueba si la habitación cuyo número se pasa como parámetro
167         está registrada en el hotal. Si lo está se borra poniendo el campo posicionLibre a true.
168         Parámetro de entrada:
169             numero: Entero con el número de la habitación.
170         Precondiciones:
171             numero: Tiene que estar en el rango [100,200].
172         Parámetro de entrada/salida pasado por referencia:
173             habitaciones: Array con la información de las habitaciones del hotel.
174         Precondiciones:
175             habitaciones: Tiene que estar inicializado.
176         Valor devuelto por la función:
177             Booleano: true si la habitación se ha localizado y se ha podido borrar, false si la
178             habitación no estaba registrada en el hotel.
179     */
180     bool borrarHabitacion (int numero, tListaHabitaciones *habitaciones);
181
182     /**
183         listarHabitacionesOcupadas: Escribe en la salida estándar la información de las
184         habitaciones ocupadas en el hotel.
185         Parámetro de entrada:
186             habitaciones: Array con la información de las habitaciones del hotel.
187         Precondiciones:
188             habitaciones: Tiene que estar inicializado.
189     */
190     void listarHabitacionesOcupadas (tListaHabitaciones habitaciones);
191
192     /** PROTOTIPOS DE FUNCIONES FASE 3*/
193
194     /**
195         buscarCliente: Se comprueba si el cliente cuyo dni se pasa como parámetro
196         está registrado en el hotal. Si lo está, se devuelve el número de la primera
    habitación en
197         la que se le ha localizado.
198         Parámetros de entrada:
199             dni: Cadena de caracteres con el dni del cliente.
200             habitaciones: Array con la información de las habitaciones del hotel.
201         Precondiciones:
202             habitaciones: Tiene que estar inicializado.
203         Parámetro de entrada/salida pasado por referencia:
204             habitacion: Número de la primera habitación en la que se ha localicado al cliente.
205         Valor devuelto por la función:
206             Booleano: true si se ha localizado al cliente, false si el cliente no está
    registrado
207             en el hotel.
208     */
209     bool buscarCliente (const tId dni, const tListaHabitaciones habitaciones, int *habitacion);
210     /**
211         totalClientesEnHotel: Calcula el número de clientes que hay registrados en el hotel.
212         Parámetro de entrada:
213             habitaciones: Array con la información de las habitaciones del hotel.
214         Precondiciones:
215             habitaciones: Tiene que estar inicializado.
216         Valor devuelto por la función:
217             Entero: Número de clientes registrados en el hotel.
218     */
219     int totalClientesEnHotel (const tListaHabitaciones habitaciones);
220
221     /** PROTOTIPOS DE FUNCIONES FASE 4 */
222
223     /**
224         menu: Escribe en la pantalla el menu de la aplicación y solicita que se elija una opción.
225         Valor devuelto por la función:
226             Entero: La opción tecleada.
227     */
228
```

```
229 int menu (void);
230
231 /**
232     MenuRegistraHabitacion: Pide que se introduzca desde teclado el número de habitación
    que se desea añadir
233     a la lista de habitaciones y se añade invocando a la correspondiente función.
234     Una vez añadida la habitación se escribe por la salida estándar
235     un mensaje indicando el número de habitación que se ha dado de alta, o el motivo por
    el que no ha dado de alta.
236     Parámetro de entrada/salida pasado por referencia:
237         habitaciones: Array con la lista de habitaciones registradas en el hotel.
238     Precondiciones:
239         habitaciones: Tiene que estar inicializado.
240 */
241 void MenuRegistraHabitacion (tListaHabitaciones *habitaciones);
242
243 /**
244     MenuEliminaHabitacion: Pide que se introduzca desde teclado el número de habitación
    que se desea borrar
245     de la lista de habitaciones y se borra invocando a la correspondiente función.
246     Una vez borrada la habitación se escribe por la salida estándar
247     un mensaje indicando el número de habitación borrada, o que no se ha podido borrar
    porque la habitación
248     no estaba registrada.
249     Parámetro de entrada/salida pasado por referencia:
250         habitaciones: Array con la lista de habitaciones registradas en el hotel.
251     Precondiciones:
252         habitaciones: Tiene que estar inicializado.
253 */
254 void MenuEliminaHabitacion (tListaHabitaciones *habitaciones);
255
256 /**
257     MenuRegistraClientes: Pide que se introduzca desde teclado el número de habitación en
    la que se desea dar de alta
258     a los clientes.
259     Si la habitación no está registrada en habitaciones, se escribe por pantalla el
    correspondiente mensaje.
260     Si la habitación está registrada se van dando de alta clientes mientras el usuario
    quiera introducir datos de
261     nuevos clientes y haya sitio en la habitación. Cuando no se pueda dar de alta un nuevo
    cliente se debe escribir
262     por pantalla un mensaje indicando el motivo.
263     Parámetro de entrada/salida pasado por referencia:
264         habitaciones: Array con la lista de habitaciones registradas en el hotel.
265     Precondiciones:
266         habitaciones: Tiene que estar inicializado.
267 */
268 void MenuRegistraClientes (tListaHabitaciones *habitaciones);
269 /**
270     MenuListaTotal: Lista la información de todos los clientes que están en las todas las
    habitaciones ocupadas.
271     Parámetro de entrada:
272         habitaciones: Array con la lista de habitaciones registradas en el hotel.
273     Precondiciones:
274         habitaciones: Tiene que estar inicializado.
275 */
276 void MenuListaTotal (tListaHabitaciones habitaciones);
277
278 /**
279     MenuListaHabitacion: Pide que se introduzca desde teclado el número de habitación de
    la que se desea escribir
280     la información de los clientes que la ocupan. Si la habitación está registrada se
    escribe la información de todos
281     los clientes que están en esa habitación, y si no está registrada se escribe un
    mensaje indicándolo.
282     Parámetro de entrada:
283         habitaciones: Array con la lista de habitaciones registradas en el hotel.
284     Precondiciones:
285         habitaciones: Tiene que estar inicializado.
286 */
287 void MenuListaHabitacion (tListaHabitaciones habitaciones);
288 /**
289     MenuBuscaCliente: Pide que se introduzca desde teclado el DNI de un cliente, si el
    cliente se localiza
290     se escribe por pantalla la habitación en la que está registrado, y si no se le
    localiza se escribe un
291     mensaje indicándolo.
292     Parámetro de entrada:
293         habitaciones: Array con la lista de habitaciones registradas en el hotel.
294     Precondiciones:
295         habitaciones: Tiene que estar inicializado.
296 */
297 void MenuBuscaCliente (tListaHabitaciones habitaciones);
298 /**
```

```
299         MenuCuentaClientes: Muestra por pantalla el número total de clientes del hotel.
300         Parámetro de entrada:
301             habitaciones: Array con la lista de habitaciones registradas en el hotel.
302         Precondiciones:
303             habitaciones: Tiene que estar inicializado.
304     */
305     void MenuCuentaClientes (tListaHabitaciones habitaciones);
306
307     /** PROTOTIPOS DE FUNCIONES FASE 5 */
308
309     /**
310         leerDatosHotel: Abre el fichero, lee la información contenida en él y la copia en el
        array habitaciones.
311         Una vez finalizada la lectura de la información del fichero, lo cierra.
312         Parámetro de entrada:
313             nomFichero: Cadena de caracteres. Nombre del fichero del que se quiere leer
        información.
314         Parámetro de entrada/salida pasado por referencia:
315             habitaciones: Array con la lista de habitaciones registradas en el hotel, en el
        que se guarda
316             la información leída del fichero.
317         Precondiciones:
318             habitaciones: Tiene que estar inicializado.
319         Valor devuelto por la función:
320             Booleano: true si se ha podido abrir el fichero y no ha habido errores al leer los
        datos del fichero,
321                       false si ha habido algún tipo de error al abrir el fichero o al leer los
        datos.
322     */
323     bool leerDatosHotel (tListaHabitaciones habitaciones, const char *nomFichero);
324
325     /**
326         guardarDatosHotel: Abre el fichero, y escribe en él la información de cada habitación
        contenida en
327         las posiciones ocupadas el array habitaciones.
328         Una vez finalizada la escritura de la información, cierra el fichero.
329         Parámetros de entrada:
330             nomFichero: Cadena de caracteres. Nombre del fichero del que se quiere leer
        información.
331             habitaciones: Array con la lista de habitaciones registradas en el hotel.
332         Precondiciones:
333             habitaciones: Tiene que estar inicializado.
334         Valor devuelto por la función:
335             Booleano: true si se ha podido abrir el fichero y no ha habido errores al escribir
        los datos del fichero,
336                       false si ha habido algún tipo de error al abrir el fichero o al escribir
        los datos.
337     */
338     bool guardarDatosHotel (const tListaHabitaciones habitaciones, const char *nomFichero);
339
340
341     int main(void){
342
343         tRegistroClientes ocupacionHabitacion;
344         tListaHabitaciones registroHabitaciones;
345         char nomFichero[] = "datosHotel.dat";
346
347         vaciarHabitacion(&ocupacionHabitacion);
348         abrirHotel(&registroHabitaciones);
349         leerDatosHotel(registroHabitaciones, nomFichero);
350
351        int opcion;
352
353
354      do {
355
356             opcion = menu();
357
358
359             switch (opcion) {
360
361                 case 1:
362                      printf("\n");
363                      printf("Registrando de habitacion: \n");
364                      printf("----------------------------\n");
365                     MenuRegistraHabitacion(&registroHabitaciones);
366
367                     break;
368
369                 case 2:
370                     printf("\n");
371                     printf("Borrado de una habitacion del registro.\n");
372                     printf("----------------------------------------\n");
373                     MenuEliminaHabitacion(&registroHabitaciones);
```

```
374
375                 break;
376
377             case 3:
378                 printf("\n");
379                 printf("Peticion de datos de clientes para su registro en una habitacion.\n");
380
    printf("-------------------------------------------------------------------\n");
381                 MenuRegistraClientes(&registroHabitaciones);
382
383                 break;
384             case 4:
385                 printf("\n");
386                 printf("Listado de ocupacion del hotel.\n");
387                 printf("-------------------------------\n");
388                 MenuListaTotal(registroHabitaciones);
389
390                 break;
391             case 5:
392                 printf("\n");
393                 printf("Listado de ocupacion de una habitacion.\n");
394                 printf("---------------------------------------\n");
395                 MenuListaHabitacion(registroHabitaciones);
396
397                 break;
398
399             case 6:
400                 printf("\n");
401                 printf("Busqueda de la habitacion de un cliente.\n");
402                 printf("----------------------------------------\n");
403                 MenuBuscaCliente(registroHabitaciones);
404
405                 break;
406             case 7:
407                 printf("\n");
408                 printf("Mostrar el numero total de clientes en el hotel.\n");
409                 printf("------------------------------------------------\n");
410                 MenuCuentaClientes(registroHabitaciones);
411
412                 break;
413             case 8:
414                 guardarDatosHotel(registroHabitaciones, nomFichero);
415
416                 break;
417
418             default:
419                 break;
420
421         }
422     } while (opcion !=8);
423
424     return 0;
425 }
426
427 int menu(){
428
429     int opcion;
430
431     printf("\n************************Opciones*****************************\n");
432     printf("* 1. Registrar habitacion a ser ocupada                       *\n");
433     printf("* 2. Eliminar habitacion del registro de ocupacion            *\n");
434     printf("* 3. Incluir clientes en una habitacion registrada            *\n");
435     printf("* 4. Listar ocupacion total del hotel                         *\n");
436     printf("* 5. Listar ocupacion de una habitacion                       *\n");
437     printf("* 6. Buscar la habitacion de un cliente                       *\n");
438     printf("* 7. Indicar el numero total de clientes en el hotel          *\n");
439     printf("* 8. Salir                                                    *\n");
440     printf("*****************************************************************\n");
441     printf("Elija una opcion: ");
442     scanf("%d", &opcion);
443
444     while(opcion > 8 ){
445         printf("Elija una opcion: ");
446         scanf("%d", &opcion);
447     }
448     return opcion;
449 }
450
451
452 /** CÓDIGO DE FUNCIONES FASE 5: */
453 bool leerDatosHotel (tListaHabitaciones habitaciones, const char *nomFichero){
454
455     FILE *pf;
```

```
456       int contador=0;
457       bool error = NOERROR;
458       pf= fopen (nomFichero, "rb");
459       if (pf != NULL){
460            printf("***Registro de clientes y habitaciones actualizado*** \n");
461           fread (&(habitaciones[contador].habitacion),
      sizeof(habitaciones[contador].habitacion), 1, pf);
462
463           while ((!feof(pf)) && (!ferror(pf)))
464           {
465              habitaciones[contador].posicionLibre = false;
466               contador ++;
467               fread (&(habitaciones[contador].habitacion),
      sizeof(habitaciones[contador].habitacion), 1, pf);
468           }
469           if (ferror(pf))
470               error= ERROR;
471           fclose (pf);
472       }
473       else{
474           printf("***No se han encontrado datos de clientes y habitaciones***\n");
475           printf("***El hotel esta actualmente vacio***\n");
476           error=ERROR;
477       }
478
479       return error;
480   }
481
482   bool guardarDatosHotel (const tListaHabitaciones habitaciones, const char *nomFichero){
483
484       FILE *pf;
485       int contador=0;
486       bool error= NO_ERROR;
487       pf= fopen (nomFichero, "wb");
488       if (pf != NULL)
489       {
490           while ((!ferror(pf)) && (contador < MAX_HABITACIONES))
491           {
492               if (habitaciones[contador].posicionLibre==false)
493               {
494                   fwrite(&(habitaciones[contador].habitacion),
      sizeof(habitaciones[contador].habitacion), 1, pf);
495               }
496               contador ++;
497           }
498           if (ferror(pf))
499           {
500               error= ERROR;
501           }
502           fclose (pf);
503
504       }
505       else
506       {
507           error=ERROR;
508       }
509       return error;
510   }
511
512
```