```c

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

#define MAX_ID 9
#define MAX_NOMBRE_CLIENTE 30
#define MAX_CLIENTES 4

#define MAX_HABITACIONES 3
#define MAX_TIPO 10
#define NUM_HABITACION_INF 100
#define NUM_HABITACION_SUP 200


typedef char tId[MAX_ID+1];
typedef char tNombre[MAX_NOMBRE_CLIENTE+1];

typedef struct {
   tId dni;
   tNombre nombre;
   bool todoIncluido;
} tDatosCliente;

typedef tDatosCliente tListaClientes[MAX_CLIENTES];

typedef struct {
   tListaClientes clientes;
   int numeroClientes;
} tRegistroClientes;


typedef char tTextoTipo [MAX_TIPO+1];

typedef struct{
    int numeroHabitacion;
    tTextoTipo tipo;
    tRegistroClientes ocupacion;
}tDatosHabitacion;

typedef struct{
    tDatosHabitacion habitacion;
    bool posicionLibre;
}tCelda;

typedef tCelda tListaHabitaciones [MAX_HABITACIONES];


/** 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[]);
```

```
78  /**
79      existeCliente: Comprueba si en la estructura ocupacion existe un cliente con el dni
    que se pasa
80      como parámetro.
81      Parámetros de entrada:
82          dni: Cadena de caracteres con el DNI del cliente a buscar.
83          ocupacion: Estructura con la información de la habitación en la que se busca al
    cliente.
84      Precondiciones:
85          ocupacion: Tiene que estar inicializado.
86      Valor devuelto por la función:
87          Booleano: true si el cliente está registrado en la habitación, false si no lo está.
88  */
89  bool existeCliente(const tId dni, tRegistroClientes ocupacion);
90  /**
91      habitacionLlena: Comprueba si la habitación está llena.
92      Parámetro de entrada:
93          ocupacion: Estructura con la información de la habitación.
94      Precondiciones:
95          ocupacion tiene que estar inicializado.
96      Valor devuelto por la función:
97          Booleano: true si la habitación está llena, false si no lo está.
98  */
99  bool habitacionLlena (tRegistroClientes ocupacion);
100 /**
101     vaciarHabitacion: Modifica los datos de la habitación para indicar que está vacía.
    Para ello pone a 0
102     el valor del campo numeroClientes.
103     Parámetro de salida pasado por referencia:
104         ocupacion: Estructura con la información de la habitación.
105 */
106 void vaciarHabitacion (tRegistroClientes *ocupacion);
107 /**
108     altaCliente: Si la habitación no está llena, y el cliente no está ya registrado en esa
    habitación,
109     se incluyen los datos del cliente en la estructura ocupacion, detrás de los datos del
    último cliente
110     registrado.
111     Parámetros de entrada:
112         dni: Cadena de caracteres con el DNI del cliente.
113         nombre: Cadena de carcteres con el nombre y el apellido o apellidos del cliente.
114         allInclusive: Booleano con valor true si el cliente elige la modalidad de "todo
    incluido",
115         false si elige "alojamiento y desayuno" (son las 2 modalidades de reserva
    disponibles).
116     Parámetro de salida pasado por referencia:
117         ocupacion: Estructura con la información de la habitación.
118     Valor devuelto por la función para control de errores:
119         Entero: 0 si el cliente se ha registrado correctamente
120                 1 si el cliente ya estaba registrado
121                 2 si no hay sitio en la habitación para registrar nuevos clientes
122 */
123 int altaCliente (tId dni, tNombre nombre, bool allInclusive, tRegistroClientes *ocupacion);
124 /**
125     listarClientes: Lista todos los datos de los clientes registrados en la habitación.
126     Parámetro de entrada:
127         ocupacion: Estructura con la información de la habitación.
128     Precondiciones:
129         ocupacion tiene que estar inicializado.
130 */
131 void listarClientes (tRegistroClientes ocupacion);
132
133 /** PROTOTIPOS DE FUNCIONES FASE 2*/
134
135 /**
136     leerEnteroEnRango: Solicita que se introduzca desde la entrada estándar un entero
    comprendido en el
137     rango [inferior, superior]. Si se teclea un valor fuera de rango, se vuelve a pedir el
    número.
138     Parámetros de entrada:
139         mensaje: Cadena de caracteres con la frase de solicitud.
140         inferior: Entero, rango inferior.
141         superior: Entero, rango superior.
142     Valor devuelto por la función:
143         Entero: el número dentro del rango.
144 */
145 int leerEnteroRango (const char mensaje [], int inferior, int superior);
146
147 /**
148     habitacionRegistrada: Comprueba si la habitación indicada en numero está dada de alta
149     en la lista de habitaciones. Si lo está devuelve la posición en la que está en el
    array habitaciones.
150     Parámetros de entrada:
151         numero: Entero con el número de la habitación.
```

```
152             habitaciones: Array con la lista de habitaciones registradas en el hotel.
153         Precondiciones:
154             numero: Tiene que estar en el rango [100,200].
155             habitaciones: Tiene que estar inicializado.
156         Parámetro de salida pasado por referencia:
157             pos: Entero con la posición en la que se encuentra la habitación en el array
    habitaciones.
158         Valor devuelto por la función:
159             Booleano: true si la habitación está registrada, false si no lo está.
160     */
161     bool habitacionRegistrada (int numero, const tListaHabitaciones habitaciones, int *pos);
162
163     /**
164         hayEspacioEnHotel: Comprueba si hay espacio libre en la lista de habitaciones del hotel.
165         Si lo hay devuelve la posición del primer hueco libre en el array de habitaciones.
166         Parámetro de entrada:
167             habitaciones: Array con la lista de habitaciones registradas en el hotel.
168         Precondiciones:
169             habitaciones: tiene que estar inicializado.
170         Parámetro de salida pasado por referencia:
171             pos: Entero con la posicion en la que se encuentra el primer hueco libre en
    habitaciones.
172         Valor devuelto por la función:
173             Booleano: true si hay hueco en el array de habitaciones del hotel, false si no lo
    hay.
174     */
175     bool hayEspacioEnHotel (const tListaHabitaciones habitaciones, int *pos);
176
177     /**
178         abrirHotel: Se abre el hotel poniendo todas las habitaciones libres.
179         Parámetro de salida pasado por referencia:
180             habitaciones: Array con la información de las habitaciones del hotel.
181     */
182     void abrirHotel (tListaHabitaciones *habitaciones);
183
184     /**
185         annadirHabitacion: Se comprueba si  hay sitio en la habitación. Si lo hay, se comprueba
186         si la habitación cuyo número se pasa como parámetro está registrada en el hotal.
187         Si no lo está se registran sus datos en la primera posición libre del array de
    habitaciones.
188         Parámetro de entrada:
189             numero: Entero con el número de la habitación.
190             tipo: Cadena de caracteres con la descripción del tipo de habitación, por ejemplo
    "suite",
191                 "individual", etc.
192         Precondiciones:
193             numero: Tiene que estar en el rango [100,200].
194         Parámetro de entrada/salida pasado por referencia:
195             habitaciones: Array con la información de las habitaciones del hotel.
196         Precondiciones:
197             habitaciones: Tiene que estar inicializado.
198         Valor devuelto por la función para control de errores:
199             Entero: 0 si la habitación se ha registrado correctamente.
200                     1 si la habitación ya estaba registrada.
201                     2 si no hay sitio en el hotel para registrar nuevas habitaciones.
202     */
203     int annadirHabitacion (int numero, tTextoTipo tipo, tListaHabitaciones *habitaciones);
204
205     /**
206         borrarHabitacion: Se comprueba si la habitación cuyo número se pasa como parámetro
207         está registrada en el hotal. Si lo está se borra poniendo el campo posicionLibre a true.
208         Parámetro de entrada:
209             numero: Entero con el número de la habitación.
210         Precondiciones:
211             numero: Tiene que estar en el rango [100,200].
212         Parámetro de entrada/salida pasado por referencia:
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             Booleano: true si la habitación se ha localizado y se ha podido borrar, false si la
218             habitación no estaba registrada en el hotel.
219     */
220     bool borrarHabitacion (int numero, tListaHabitaciones *habitaciones);
221
222     /**
223         listarHabitacionesOcupadas: Escribe en la salida estándar la información de las
224         habitaciones ocupadas en el hotel.
225         Parámetro de entrada:
226             habitaciones: Array con la información de las habitaciones del hotel.
227         Precondiciones:
228             habitaciones: Tiene que estar inicializado.
229     */
230     void listarHabitacionesOcupadas (tListaHabitaciones habitaciones);
```

```
231
232 /** PROTOTIPOS DE FUNCIONES FASE 3*/
233
234 /**
235     buscarCliente: Se comprueba si el cliente cuyo dni se pasa como parámetro
236     está registrado en el hotal. Si lo está, se devuelve el número de la primera
    habitación en
237     la que se le ha localizado.
238     Parámetros de entrada:
239         dni: Cadena de caracteres con el dni del cliente.
240         habitaciones: Array con la información de las habitaciones del hotel.
241     Precondiciones:
242         habitaciones: Tiene que estar inicializado.
243     Parámetro de entrada/salida pasado por referencia:
244         habitacion: Número de la primera habitación en la que se ha localicado al cliente.
245     Valor devuelto por la función:
246         Booleano: true si se ha localizado al cliente, false si el cliente no está
    registrado
247         en el hotel.
248 */
249 bool buscarCliente (const tId dni, const tListaHabitaciones habitaciones, int *habitacion);
250 /**
251     totalClientesEnHotel: Calcula el número de clientes que hay registrados en el hotel.
252     Parámetro de entrada:
253         habitaciones: Array con la información de las habitaciones del hotel.
254     Precondiciones:
255         habitaciones: Tiene que estar inicializado.
256     Valor devuelto por la función:
257         Entero: Número de clientes registrados en el hotel.
258 */
259 int totalClientesEnHotel (const tListaHabitaciones habitaciones);
260
261 /** PROTOTIPOS DE FUNCIONES FASE 4 */
262
263 /**
264     menu: Escribe en la pantalla el menu de la aplicación y solicita que se elija una opción.
265     Valor devuelto por la función:
266         Entero: La opción tecleada.
267 */
268
269 int menu (void);
270
271 /**
272     MenuRegistraHabitacion: Pide que se introduzca desde teclado el número de habitación
    que se desea añadir
273     a la lista de habitaciones y se añade invocando a la correspondiente función.
274     Una vez añadida la habitación se escribe por la salida estándar
275     un mensaje indicando el número de habitación que se ha dado de alta, o el motivo por
    el que no ha dado de alta.
276     Parámetro de entrada/salida pasado por referencia:
277         habitaciones: Array con la lista de habitaciones registradas en el hotel.
278     Precondiciones:
279         habitaciones: Tiene que estar inicializado.
280 */
281 void MenuRegistraHabitacion (tListaHabitaciones *habitaciones);
282
283 /**
284     MenuEliminaHabitacion: Pide que se introduzca desde teclado el número de habitación
    que se desea borrar
285     de la lista de habitaciones y se borra invocando a la correspondiente función.
286     Una vez borrada la habitación se escribe por la salida estándar
287     un mensaje indicando el número de habitación borrada, o que no se ha podido borrar
    porque la habitación
288     no estaba registrada.
289     Parámetro de entrada/salida pasado por referencia:
290         habitaciones: Array con la lista de habitaciones registradas en el hotel.
291     Precondiciones:
292         habitaciones: Tiene que estar inicializado.
293 */
294 void MenuEliminaHabitacion (tListaHabitaciones *habitaciones);
295
296 /**
297     MenuRegistraClientes: Pide que se introduzca desde teclado el número de habitación en
    la que se desea dar de alta
298     a los clientes.
299     Si la habitación no está registrada en habitaciones, se escribe por pantalla el
    correspondiente mensaje.
300     Si la habitación está registrada se van dando de alta clientes mientras el usuario
    quiera introducir datos de
301     nuevos clientes y haya sitio en la habitación. Cuando no se pueda dar de alta un nuevo
    cliente se debe escribir
302     por pantalla un mensaje indicando el motivo.
303     Parámetro de entrada/salida pasado por referencia:
304         habitaciones: Array con la lista de habitaciones registradas en el hotel.
```

```
305         Precondiciones:
306             habitaciones: Tiene que estar inicializado.
307     */
308     void MenuRegistraClientes (tListaHabitaciones *habitaciones);
309     /**
310         MenuListaTotal: Lista la información de todos los clientes que están en las todas las
    habitaciones ocupadas.
311         Parámetro de entrada:
312             habitaciones: Array con la lista de habitaciones registradas en el hotel.
313         Precondiciones:
314             habitaciones: Tiene que estar inicializado.
315     */
316     void MenuListaTotal (tListaHabitaciones habitaciones);
317
318     /**
319         MenuListaHabitacion: Pide que se introduzca desde teclado el número de habitación de
    la que se desea escribir
320         la información de los clientes que la ocupan. Si la habitación está registrada se
    escribe la información de todos
321         los clientes que están en esa habitación, y si no está registrada se escribe un
    mensaje indicándolo.
322         Parámetro de entrada:
323             habitaciones: Array con la lista de habitaciones registradas en el hotel.
324         Precondiciones:
325             habitaciones: Tiene que estar inicializado.
326     */
327     void MenuListaHabitacion (tListaHabitaciones habitaciones);
328     /**
329         MenuBuscaCliente: Pide que se introduzca desde teclado el DNI de un cliente, si el
    cliente se localiza
330         se escribe por pantalla la habitación en la que está registrado, y si no se le
    localiza se escribe un
331         mensaje indicándolo.
332         Parámetro de entrada:
333             habitaciones: Array con la lista de habitaciones registradas en el hotel.
334         Precondiciones:
335             habitaciones: Tiene que estar inicializado.
336     */
337     void MenuBuscaCliente (tListaHabitaciones habitaciones);
338     /**
339         MenuCuentaClientes: Muestra por pantalla el número total de clientes del hotel.
340         Parámetro de entrada:
341             habitaciones: Array con la lista de habitaciones registradas en el hotel.
342         Precondiciones:
343             habitaciones: Tiene que estar inicializado.
344     */
345     void MenuCuentaClientes (tListaHabitaciones habitaciones);
346
347     /** PROTOTIPOS DE FUNCIONES FASE 5 */
348
349     /**
350         leerDatosHotel: Abre el fichero, lee la información contenida en él y la copia en el
    array habitaciones.
351         Una vez finalizada la lectura de la información del fichero, lo cierra.
352         Parámetro de entrada:
353             nomFichero: Cadena de caracteres. Nombre del fichero del que se quiere leer
    información.
354         Parámetro de entrada/salida pasado por referencia:
355             habitaciones: Array con la lista de habitaciones registradas en el hotel, en el
    que se guarda
356             la información leída del fichero.
357         Precondiciones:
358             habitaciones: Tiene que estar inicializado.
359         Valor devuelto por la función:
360             Booleano: true si se ha podido abrir el fichero y no ha habido errores al leer los
    datos del fichero,
361                       false si ha habido algún tipo de error al abrir el fichero o al leer los
    datos.
362     */
363     bool leerDatosHotel (tListaHabitaciones habitaciones, const char *nomFichero);
364
365     /**
366         guardarDatosHotel: Abre el fichero, y escribe en él la información de cada habitación
    contenida en
367         las posiciones ocupadas el array habitaciones.
368         Una vez finalizada la escritura de la información, cierra el fichero.
369         Parámetros de entrada:
370             nomFichero: Cadena de caracteres. Nombre del fichero del que se quiere leer
    información.
371             habitaciones: Array con la lista de habitaciones registradas en el hotel.
372         Precondiciones:
373             habitaciones: Tiene que estar inicializado.
374         Valor devuelto por la función:
375             Booleano: true si se ha podido abrir el fichero y no ha habido errores al escribir
```

```
los datos del fichero,
376                  false si ha habido algún tipo de error al abrir el fichero o al escribir
los datos.
377  */
378  bool guardarDatosHotel (const tListaHabitaciones habitaciones, const char *nomFichero);
379
380
381  int main(void){
382
383      tRegistroClientes ocupacionHabitacion;
384      tListaHabitaciones registroHabitaciones;
385      char nomFichero[] = "datosHotel.dat";
386
387      vaciarHabitacion(&ocupacionHabitacion);
388      abrirHotel(&registroHabitaciones);
389      leerDatosHotel(registroHabitaciones, nomFichero);
390
391     int opcion;
392
393
394   do {
395          opcion = menu();
396
397          switch (opcion) {
398
399              case 1:
400                   printf("\n");
401                   printf("Registrando de habitacion: \n");
402                   printf("---------------------------\n");
403                  MenuRegistraHabitacion(&registroHabitaciones);
404
405                  break;
406
407              case 2:
408                  printf("\n");
409                  printf("Borrado de una habitacion del registro.\n");
410                  printf("---------------------------------------\n");
411                  MenuEliminaHabitacion(&registroHabitaciones);
412
413                  break;
414
415              case 3:
416                  printf("\n");
417                  printf("Peticion de datos de clientes para su registro en una
habitacion.\n");
418
printf("-----------------------------------------------------------------\n");
419                  MenuRegistraClientes(&registroHabitaciones);
420
421                  break;
422              case 4:
423                  printf("\n");
424                  printf("Listado de ocupacion del hotel.\n");
425                  printf("-------------------------------\n");
426                  MenuListaTotal(registroHabitaciones);
427
428                  break;
429              case 5:
430                  printf("\n");
431                  printf("Listado de ocupacion de una habitacion.\n");
432                  printf("---------------------------------------\n");
433                  MenuListaHabitacion(registroHabitaciones);
434
435                  break;
436
437              case 6:
438                  printf("\n");
439                  printf("Busqueda de la habitacion de un cliente.\n");
440                  printf("---------------------------------------\n");
441                  MenuBuscaCliente(registroHabitaciones);
442
443                  break;
444              case 7:
445                  printf("\n");
446                  printf("Mostrar el numero total de clientes en el hotel.\n");
447                  printf("-------------------------------------------------\n");
448                  MenuCuentaClientes(registroHabitaciones);
449
450                  break;
451              case 8:
452                  guardarDatosHotel(registroHabitaciones, nomFichero);
453
454                  break;
455
```

```
456                 default:
457                     break;
458 
459             }
460         } while (opcion !=8);
461 
462         return 0;
463     }
464 
465     int menu(){
466 
467         int opcion;
468 
469         printf("\n************************Opciones*****************************\n");
470         printf("* 1. Registrar habitacion a ser ocupada                      *\n");
471         printf("* 2. Eliminar habitacion del registro de ocupacion           *\n");
472         printf("* 3. Incluir clientes en una habitacion registrada           *\n");
473         printf("* 4. Listar ocupacion total del hotel                        *\n");
474         printf("* 5. Listar ocupacion de una habitacion                      *\n");
475         printf("* 6. Buscar la habitacion de un cliente                      *\n");
476         printf("* 7. Indicar el numero total de clientes en el hotel         *\n");
477         printf("* 8. Salir                                                   *\n");
478         printf("*************************************************************\n");
479         printf("Elija una opcion: ");
480         scanf("%d", &opcion);
481 
482         while(opcion > 8 ){
483             printf("Elija una opcion: ");
484             scanf("%d", &opcion);
485         }
486         return opcion;
487     }
488 
489 
490     /** CÓDIGO DE FUNCIONES FASES 1 */
491     void leerTexto(const char mensaje[], char dato[], int size){
492        do{
493           printf ("%s", mensaje);
494           fflush(stdin);
495           fgets (dato, size+1, stdin);
496        } while (dato[0] == '\n' );
497        if(dato[strlen(dato)-1] == '\n'){dato[strlen(dato)-1] = '\0';}
498     }
499 
500     bool leerBooleano(const char mensaje[]){
501        char respuesta;
502        bool devuelve = false;
503 
504        do{
505           printf ("%s", mensaje);
506           fflush(stdin);
507           scanf("%c", &respuesta);
508        } while ((respuesta != 's')&&(respuesta != 'S')&&(respuesta != 'n')&&(respuesta !=
    'N'));
509        if ((respuesta == 's')||(respuesta == 'S')){
510           devuelve = true;
511        }
512 
513        return devuelve;
514     }
515 
516     bool existeCliente(const tId dni, tRegistroClientes ocupacion){
517        bool encontrado = false;
518        int compara;
519        int n = 0;
520 
521        while ((n < ocupacion.numeroClientes)&&(!encontrado)){
522           compara = strcmp(ocupacion.clientes[n].dni, dni);
523           if (compara == 0){
524              encontrado = true;
525           }
526           n++;
527        }
528 
529        return encontrado;
530     }
531 
532     bool habitacionLlena (const tRegistroClientes ocupacion){
533 
534         bool llena = false;
535 
536 
537         if(ocupacion.numeroClientes < MAX_CLIENTES){
538             llena = false;
```

```
539         }
540         else{
541              llena = true;
542         }
543 
544         return llena;
545     }
546 
547     void vaciarHabitacion (tRegistroClientes *ocupacion){
548 
549             ocupacion->numeroClientes = 0;
550     }
551 
552     int altaCliente (tId dni, tNombre nombre, bool allInclusive, tRegistroClientes *ocupacion){
553 
554         int error;
555         bool encontrado = false;
556         leerTexto("Cliente-DNI: ", dni, MAX_ID);
557         leerTexto("Cliente-Nombre: ", nombre, MAX_NOMBRE_CLIENTE);
558         allInclusive = leerBooleano("El cliente tiene todo incluido? (s/n): ");
559 
560 
561         encontrado = existeCliente(dni, *ocupacion);
562 
563          if(encontrado == false){
564              if (!habitacionLlena(*ocupacion)) {
565 
566                  int posicionHueco = ocupacion->numeroClientes;
567                  strncpy(ocupacion->clientes[posicionHueco].dni, dni, MAX_ID);
568                  strncpy(ocupacion->clientes[posicionHueco].nombre, nombre, MAX_NOMBRE_CLIENTE);
569                  ocupacion->clientes[posicionHueco].todoIncluido = allInclusive;
570                  error = 0;
571 
572             }else{
573                 error = 2;
574             }
575         }else{
576             error = 1;
577         }
578 
579         return error;
580     }
581 
582     void listarClientes (const tRegistroClientes ocupacion){
583 
584         tId dniCliente;
585        int i;
586        bool Incluido;
587 
588         for(i = 0; i < MAX_CLIENTES; i++){
589                 if(existeCliente(dniCliente, ocupacion)){
590                     Incluido = ocupacion.clientes[i].todoIncluido;
591 
592                     if (Incluido == true){
593                         printf("***Cliente %d: %s - Todo incluido \n", i+1,
    ocupacion.clientes[i].dni, ocupacion.clientes[i].nombre);
594                     }else{
595                         printf("***Cliente %d: %s - Solo desayuno \n", i+1,
    ocupacion.clientes[i].dni, ocupacion.clientes[i].nombre);
596                     }
597                 }
598         }
599     }
600 
601 
602 
603 
604     /** CÓDIGO DE FUNCIONES FASE 2*/
605 
606     int leerEnteroRango (const char mensaje [], int inferior, int superior){
607 
608          int dato;
609 
610         do{
611             printf ("%s", mensaje);
612             fflush(stdin);
613             scanf("%d", &dato);
614 
615         } while ( dato < inferior || dato > superior);
616 
617         return dato;
618     }
619 
620     bool habitacionRegistrada (int numero, const tListaHabitaciones habitaciones, int *pos){
```

```
621
622         bool encontrado = false;
623
624
625         while ((*pos) < MAX_HABITACIONES  && encontrado == false){
626
    if((!habitaciones[*pos].posicionLibre)&&(habitaciones[*pos].habitacion.numeroHabitacion ==
    numero)){
627                 encontrado = true;
628             }else{
629              *pos =  *pos + 1;
630             }
631         }
632         return encontrado;
633     }
634
635     bool hayEspacioEnHotel (const tListaHabitaciones habitaciones, int *pos){
636
637          bool haySitio = false;
638         (*pos)=0;
639
640         while (*pos < MAX_HABITACIONES && !haySitio) {
641             if(habitaciones[*pos].posicionLibre){
642                 haySitio = true;
643
644             }else{
645               (*pos)++;
646             }
647         }
648     return haySitio;
649     }
650
651     void abrirHotel (tListaHabitaciones *habitaciones){
652
653         int i;
654         printf("***Hotel abierto*** \n");
655         for(i = 0; i < MAX_HABITACIONES; i++){
656
657             (*habitaciones)[i].habitacion.ocupacion.numeroClientes = 0;
658             (*habitaciones)[i].posicionLibre = true;
659
660         }
661     }
662
663     int annadirHabitacion (int numero, tTextoTipo tipo, tListaHabitaciones *habitaciones){
664
665         int error;
666         int posicionExiste=0;
667         int posicionHueco;
668
669
670
671         if (habitacionRegistrada(numero, *habitaciones, &posicionExiste)){
672                 error = 1;
673
674         }else{
675
676             if (!hayEspacioEnHotel(*habitaciones, &posicionHueco)){
677                 error = 2;
678             }else{
679                 (*habitaciones)[posicionHueco].habitacion.numeroHabitacion = numero;
680                 strncpy((*habitaciones)[posicionHueco].habitacion.tipo, tipo, MAX_TIPO);
681                 (*habitaciones)[posicionHueco].posicionLibre = false;
682                error = 0;
683             }
684         }
685
686         return error;
687     }
688
689     bool borrarHabitacion (int numero, tListaHabitaciones *habitaciones){
690
691         bool eliminado = false;
692         int posicion=0;
693
694         if (habitacionRegistrada(numero, *habitaciones, &posicion)) {
695             (*habitaciones)[posicion].posicionLibre = true;
696             (*habitaciones)[posicion].habitacion.numeroHabitacion = 0;
697
698             (*habitaciones)[posicion].habitacion.ocupacion.numeroClientes = 0;
699             eliminado = true;
700         }
701
702         return eliminado;
```

```
703 }
704
705 void listarHabitacionesOcupadas (tListaHabitaciones habitaciones){
706
707         for(int i = 0; i < MAX_HABITACIONES; i++){
708                 if (habitaciones[i].posicionLibre == false){
709                     printf("-Habitacion %d (%s). Numero de ocupantes: %d. Listado:
    \n",habitaciones[i].habitacion.numeroHabitacion, habitaciones[i].habitacion.tipo,
    habitaciones[i].habitacion.ocupacion.numeroClientes);
710
711                 }
712         }
713 }
714
715
716
717
718 /** CÓDIGO DE FUNCIONES FASE 3*/
719
720 bool buscarCliente (const tId dni, const tListaHabitaciones habitaciones, int *habitacion){
721
722     bool encontrado = false;
723      int i;
724     tRegistroClientes ocupacionHabitacion;
725     int pos = 0;
726
727     for( i = 0; i < MAX_HABITACIONES; i++){
728                 if(existeCliente(dni, ocupacionHabitacion) &&
    habitacionRegistrada(*habitacion, habitaciones, &pos)){
729                     encontrado = true;
730                     pos = i;
731                     }
732
733     }
734
735     return encontrado;
736 }
737
738 int totalClientesEnHotel (const tListaHabitaciones habitaciones){
739
740     int numeroClientes = 0;
741
742     for (int i=0; i < MAX_HABITACIONES;i++){
743         numeroClientes = numeroClientes +
    habitaciones[i].habitacion.ocupacion.numeroClientes;
744     }
745
746
747 return numeroClientes;
748 }
749
750
751
752
753 /** CÓDIGO DE FUNCIONES FASE 4: */
754
755 void MenuRegistraHabitacion (tListaHabitaciones *habitaciones){
756
757
758     tTextoTipo tipoHabitacion;
759     int errorRegistro;
760
761     int numHabitacion = leerEnteroRango("Numero de habitacion: ",NUM_HABITACION_INF,
    NUM_HABITACION_SUP);
762     leerTexto("Tipo de habitacion: ",tipoHabitacion, MAX_TIPO);
763
764
765     errorRegistro = annadirHabitacion(numHabitacion, tipoHabitacion, habitaciones);
766
767       if (errorRegistro == 0){
768          printf("***Habitacion %d dada de alta correctamente***\n", numHabitacion);
769       }
770       else{
771          if (errorRegistro == 1)
772             printf("***Error, no se pudo realizar el alta: La habitacion %d ya fue
    registrada***\n", numHabitacion);
773          else
774             printf("***Error, no se pudo realizar el alta: El hotel no tiene permisos para
    abrir mas habitaciones***\n");
775       }
776    }
777
778 void MenuEliminaHabitacion (tListaHabitaciones *habitaciones){
779
```

```
780       int numHabitacion;
781
782      numHabitacion = leerEnteroRango("Numero de habitacion a borrar: ",NUM_HABITACION_INF,
    NUM_HABITACION_SUP);
783
784       if (borrarHabitacion(numHabitacion, &(*habitaciones))){
785
786               printf("***La habitacion %d ha sido borrada se su sitema***\n", numHabitacion);
787
788       }else{
789           printf("***Error, la habitacion %d no consta como registrada en su sistema***\n",
    numHabitacion);
790       }
791   }
792
793   void MenuRegistraClientes (tListaHabitaciones *habitaciones){
794
795       tId dniCliente;
796       tNombre nombreCliente;
797       tRegistroClientes ocupacionHabitacion;
798
799       int errorAltaCliente;
800       bool todoIncluido = false;
801       bool respuesta;
802       int posicion;
803       int pos=0;
804       int error = 0;
805
806
807       int  numHabitacion = leerEnteroRango("Numero de habitacion en la que registrar
    cliente: ",NUM_HABITACION_INF, NUM_HABITACION_SUP);
808       bool registrado = habitacionRegistrada(numHabitacion, *habitaciones, &pos);
809
810       do{
811
812
813       if(registrado == true ){
814
815           for(int i = 0; i < MAX_CLIENTES; i++){
816
    strncpy(ocupacionHabitacion.clientes[i].dni,(*habitaciones)[pos].habitacion.ocupacion.client
    es[i].dni , MAX_ID);
817           }
818
819           ocupacionHabitacion.numeroClientes =
    (*habitaciones)[pos].habitacion.ocupacion.numeroClientes;
820           errorAltaCliente = altaCliente(dniCliente, nombreCliente, todoIncluido,
    &ocupacionHabitacion);
821
822           if (errorAltaCliente == 0){
823
824               posicion = ocupacionHabitacion.numeroClientes;
825
826               strncpy((*habitaciones)[pos].habitacion.ocupacion.clientes[posicion].dni,
    ocupacionHabitacion.clientes[posicion].dni,MAX_ID);
827               strncpy((*habitaciones)[pos].habitacion.ocupacion.clientes[posicion].nombre,
    ocupacionHabitacion.clientes[posicion].nombre, MAX_NOMBRE_CLIENTE);
828                (*habitaciones)[pos].habitacion.ocupacion.clientes[posicion].todoIncluido =
    ocupacionHabitacion.clientes[posicion].todoIncluido;
829
830               (*habitaciones)[pos].habitacion.ocupacion.numeroClientes++;
831               printf("***Cliente dado de alta correctamente***\n");
832
833           }else{
834               if (errorAltaCliente == 1){
835                       printf("***Error, no se pudo realizar el alta: El DNI %s ya fue
    registrado previamente***\n", dniCliente);
836               }else{
837                   printf("***Error, no se pudo realizar el alta: La habitacion %d esta
    llena***\n", numHabitacion);
838                   }
839           }
840       respuesta = leerBooleano("\nDesea introducir mas clientes en la habitacion? (s/n): ");
841
842       }else{
843
844           printf("***Error, la habitacion %d no aparece como registrada en su sistema***\n",
    numHabitacion);
845            error = 1;
846       }
847
848      }while(respuesta == true && error != 1);
849   }
850
```

```
851  void MenuListaTotal (tListaHabitaciones habitaciones){
852
853      int i;
854      bool Incluido;
855      bool posicionLibre = false;
856      bool registrado = false;
857      int numClientes;
858
859          for( i = 0; i < MAX_HABITACIONES; i++){
860                  posicionLibre = habitaciones[i].posicionLibre;
861
862                  if(posicionLibre == false){
863                      printf("-Habitacion %i (%s). Numero de ocupantes: %i. Listado:
     \n",habitaciones[i].habitacion.numeroHabitacion, habitaciones[i].habitacion.tipo,
     habitaciones[i].habitacion.ocupacion.numeroClientes);
864                      numClientes = habitaciones[i].habitacion.ocupacion.numeroClientes;
865                      registrado = true;
866
867                      for(int j = 0; j < numClientes && numClientes !=0 ; j++){
868                          Incluido =
     habitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
869                              if (Incluido == true){
870                                  printf("***Cliente %d: %s - %s - Todo incluido \n", j+1,
     habitaciones[i].habitacion.ocupacion.clientes[j].dni,
     habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
871                              }else{
872                                  printf("***Cliente %d: %s - %s - Solo desayuno \n", j+1,
     habitaciones[i].habitacion.ocupacion.clientes[j].dni,
     habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
873                              }
874                      }
875                      if(numClientes == 0 ){
876                              printf("***No hay clientes en la habitacion \n");
877                      }
878
879              }
880          }
881          if(registrado == false){
882              printf("***No hay ninguna habitacion registrada en su hotel \n");
883          }
884
885  }
886
887  void MenuListaHabitacion (tListaHabitaciones habitaciones){
888
889
890      int i;
891      int j;
892      int numHabitacion;
893      int numClientes;
894      bool registrado = false;
895      bool Incluido = false;
896      printf("Introduzca el numero de la habitacion: ");
897      scanf("%d",&numHabitacion);
898
899      for(i=0; i<MAX_HABITACIONES && !registrado; i++){
900             registrado =  habitacionRegistrada(numHabitacion, habitaciones, &i);
901             numClientes = habitaciones[i].habitacion.ocupacion.numeroClientes;
902
903             if(registrado == true){
904                  for(j=0; j < numClientes && numClientes != 0;  j++){
905                      Incluido =
     habitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
906                      if (Incluido == true){
907                              printf("***Cliente %d: %s - %s - Todo incluido \n", j+1,
     habitaciones[i].habitacion.ocupacion.clientes[j].dni,
     habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
908                              }else{
909                                  printf("***Cliente %d: %s - %s - Solo desayuno \n", j+1,
     habitaciones[i].habitacion.ocupacion.clientes[j].dni,
     habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
910                              }
911                      }
912                  }
913      }
914
915      if(registrado == false ){
916              printf("***Error, la habitacion %d no aparece como registrada en su
     sistema***\n", numHabitacion);
917      }
918
919      if(numClientes == 0){
920          printf("***No hay clientes en la habitacion \n");
921      }
```

```
922  }
923
924  void MenuBuscaCliente (tListaHabitaciones habitaciones){
925
926       bool encontrado = false;
927      int i;
928      int j;
929      tId dniCliente;
930
931      leerTexto("Indiqueme el DNI del cliente a buscar: ", dniCliente, MAX_ID);
932
933      for(i=0; i < MAX_HABITACIONES && !encontrado;  i++){
934              for(j=0; j < ( habitaciones[i].habitacion.ocupacion.numeroClientes);  j++)   {
935
     if(strcmp(dniCliente,habitaciones[i].habitacion.ocupacion.clientes[j].dni) == 0 &&
     habitaciones[i].habitacion.numeroHabitacion !=0){
936                          encontrado = true;
937                  }
938              }
939              if(encontrado){
940                      printf("***El cliente se encuentra en la habitacion numero
     %i***",habitaciones[i].habitacion.numeroHabitacion);
941              }
942      }
943      if(encontrado == false){
944              printf("***El cliente indicado no se encuentra alojado en el hotel***");
945
946      }
947  }
948
949  void MenuCuentaClientes (tListaHabitaciones habitaciones){
950
951      int numeroTotal = totalClientesEnHotel (habitaciones);
952
953      printf("El numero total de clientes alojados actualmente en el hotel es de %d
     personas", numeroTotal);
954
955  }
956
957
958
959
960
961  /** CÓDIGO DE FUNCIONES FASE 5: */
962  bool leerDatosHotel (tListaHabitaciones habitaciones, const char *nomFichero){
963
964      FILE *pf;
965      int contador=0;
966      bool error = NOERROR;
967      pf= fopen (nomFichero, "rb");
968      if (pf != NULL){
969           printf("***Registro de clientes y habitaciones actualizado*** \n");
970          fread (&(habitaciones[contador].habitacion),
     sizeof(habitaciones[contador].habitacion), 1, pf);
971
972          while ((!feof(pf)) && (!ferror(pf)))
973          {
974             habitaciones[contador].posicionLibre = false;
975              contador ++;
976              fread (&(habitaciones[contador].habitacion),
     sizeof(habitaciones[contador].habitacion), 1, pf);
977          }
978          if (ferror(pf))
979              error= ERROR;
980          fclose (pf);
981      }
982      else{
983          printf("***No se han encontrado datos de clientes y habitaciones***\n");
984          printf("***El hotel esta actualmente vacio***\n");
985          error=ERROR;
986      }
987
988      return error;
989  }
990
991  bool guardarDatosHotel (const tListaHabitaciones habitaciones, const char *nomFichero){
992
993      FILE *pf;
994      int contador=0;
995      bool error= NO_ERROR;
996      pf= fopen (nomFichero, "wb");
997      if (pf != NULL)
998      {
999          while ((!ferror(pf)) && (contador < MAX_HABITACIONES))
```

```
1000            {
1001                if (habitaciones[contador].posicionLibre==false)
1002                {
1003                    fwrite(&(habitaciones[contador].habitacion),
      sizeof(habitaciones[contador].habitacion), 1, pf);
1004                }
1005                contador ++;
1006            }
1007            if (ferror(pf))
1008            {
1009                error= ERROR;
1010            }
1011            fclose (pf);
1012
1013        }
1014        else
1015        {
1016            error=ERROR;
1017        }
1018        return error;
1019    }
1020
```