```c


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


#define ERROR 0

#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];


/** PROTOPTIPOS 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 leida se elimina el '\n' en caso de que se hubiera guardado. Si se teclea una cadena
    de tamaño mayor al indicado, se guandan los size primeros caracteres.
    No admite como válida una cadena vacia. Si se pulsa enter tras la frase de solicitud, se vuelve
    a pedir al ususario 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.
```

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

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

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

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

```
374         nomFichero: Cadena de caracteres. Nombre del fichero del que se quiere leer
    información.
375         habitaciones: Array con la lista de habitaciones registradas en el hotel.
376     Precondiciones:
377         habitaciones: Tiene que estar inicializado.
378     Valor devuelto por la función:
379         Booleano: true si se ha podido abrir el fichero y no ha habido errores al escribir
    los datos del fichero,
380                   false si ha habido algún tipo de error al abrir el fichero o al escribir
    los datos.
381 */
382 bool guardarDatosHotel (const tListaHabitaciones registroHabitaciones,const char
    *nomFichero);
383
384
385 int main(void){
386
387     tRegistroClientes ocupacionHabitacion;
388     tListaHabitaciones registroHabitaciones;
389      char nomFichero[] = "dataHotel.dat";
390
391     vaciarHabitacion(&ocupacionHabitacion);
392     abrirHotel(registroHabitaciones);
393
394     if(leerDatosHotel(registroHabitaciones, nomFichero)){
395         printf("***No se han encontrado datos de clientes y habitaciones. El hotel esta
    actualmente vacio***\n");
396     }else{
397        printf("***Registro de clientes y habitaciones actualizado*** \n");
398     }
399
400
401
402    int opcion;
403
404
405  do {
406
407         opcion = menu();
408
409
410         switch (opcion) {
411
412             case 1:
413                  printf("\n");
414                  printf("Registrando de habitacion: \n");
415                  printf("---------------------------\n");
416                 MenuRegistraHabitacion(&registroHabitaciones);
417
418                 break;
419
420             case 2:
421                 printf("\n");
422                 printf("Borrado de una habitacion del registro.\n");
423                 printf("---------------------------------------\n");
424                 MenuEliminaHabitacion(&registroHabitaciones);
425
426                 break;
427
428             case 3:
429                 printf("\n");
430                 printf("Peticion de datos de clientes para su registro en una
    habitacion.\n");
431
    printf("-----------------------------------------------------------------\n");
432                 MenuRegistraClientes(&registroHabitaciones);
433
434                 break;
435             case 4:
436                 printf("\n");
437                 printf("Listado de ocupacion del hotel.\n");
438                 printf("-------------------------------\n");
439                 MenuListaTotal(registroHabitaciones);
440
441                 break;
442             case 5:
443                 printf("\n");
444                 printf("Listado de ocupacion de una habitacion.\n");
445                 printf("---------------------------------------\n");
446                 MenuListaHabitacion(registroHabitaciones);
447
448                 break;
449
450             case 6:
```

```
451                 printf("\n");
452                 printf("Busqueda de la habitacion de un cliente.\n");
453                 printf("----------------------------------------\n");
454                 MenuBuscaCliente(registroHabitaciones);
455 
456                 break;
457             case 7:
458                 printf("\n");
459                 printf("Mostrar el numero total de clientes en el hotel.\n");
460                 printf("-----------------------------------------------\n");
461                 MenuCuentaClientes(registroHabitaciones);
462 
463                 break;
464             case 8:
465                 guardarDatosHotel(registroHabitaciones, nomFichero);
466 
467                 break;
468 
469             default:
470                 break;
471 
472         }
473     } while (opcion !=8);
474 
475     return 0;
476 }
477 
478 int menu(){
479 
480     int opcion;
481 
482     printf("\n************************Opciones*****************************\n");
483     printf("* 1. Registrar habitacion a ser ocupada                      *\n");
484     printf("* 2. Eliminar habitacion del registro de ocupacion           *\n");
485     printf("* 3. Incluir clientes en una habitacion registrada           *\n");
486     printf("* 4. Listar ocupacion total del hotel                        *\n");
487     printf("* 5. Listar ocupacion de una habitacion                      *\n");
488     printf("* 6. Buscar la habitacion de un cliente                      *\n");
489     printf("* 7. Indicar el numero total de clientes en el hotel         *\n");
490     printf("* 8. Salir                                                   *\n");
491     printf("*************************************************************\n");
492     printf("Elija una opcion: ");
493     scanf("%d", &opcion);
494 
495     while(opcion > 8 ){
496         printf("Elija una opcion: ");
497         scanf("%d", &opcion);
498     }
499     return opcion;
500 }
501 
502 
503 /* CÓDIGO DE FUNCIONES FASES 1, 2 y 3 */
504 void leerTexto(const char mensaje[], char dato[], int size){
505    do{
506       printf ("%s", mensaje);
507       fflush(stdin);
508       fgets (dato, size+1, stdin);
509    } while (dato[0] == '\n' );
510    if(dato[strlen(dato)-1] == '\n'){dato[strlen(dato)-1] = '\0';}
511 }
512 
513 bool leerBooleano(const char mensaje[]){
514    char respuesta;
515    bool devuelve = false;
516 
517    do{
518       printf ("%s", mensaje);
519       fflush(stdin);
520       scanf("%c", &respuesta);
521    } while ((respuesta != 's')&&(respuesta != 'S')&&(respuesta != 'n')&&(respuesta !=
    'N'));
522    if ((respuesta == 's')||(respuesta == 'S')){
523       devuelve = true;
524    }
525 
526    return devuelve;
527 }
528 
529 bool existeCliente(const tId dniCliente, tRegistroClientes ocupacionHabitacion){
530    bool encontrado = false;
531    int compara;
532    int n = 0;
533 
```

```
534      while ((n < ocupacionHabitacion.numeroClientes)&&(!encontrado)){
535         compara = strcmp(dniCliente, ocupacionHabitacion.clientes[n].dni);
536         if (compara == 0){
537             encontrado = true;
538         }
539         n++;
540      }
541
542      return encontrado;
543   }
544
545   bool habitacionLlena (const tRegistroClientes ocupacionHabitacion){
546
547       bool llena = false;
548
549
550       if(ocupacionHabitacion.numeroClientes < MAX_CLIENTES){
551           llena = false;
552       }
553       else{
554            llena = true;
555       }
556
557       return llena;
558   }
559
560   void vaciarHabitacion (tRegistroClientes *ocupacionHabitacion){
561
562           ocupacionHabitacion->numeroClientes = 0;
563   }
564
565   int altaCliente (tId dniCliente, tNombre nombreCliente, bool todoIncluido,
      tRegistroClientes *ocupacionHabitacion){
566
567       int error;
568       bool encontrado = false;
569       int i = 0;
570       leerTexto("Cliente-DNI: ", dniCliente, MAX_ID);
571       leerTexto("Cliente-Nombre: ", nombreCliente, MAX_NOMBRE_CLIENTE);
572       todoIncluido = leerBooleano("El cliente tiene todo incluido? (s/n): ");
573
574       while(i < MAX_CLIENTES && !encontrado){
575               if(existeCliente(dniCliente, *ocupacionHabitacion) ){
576                   encontrado = true;
577               }
578               i++;
579       }
580        if(encontrado == false){
581            if (!habitacionLlena(*ocupacionHabitacion)) {
582
583                int posicionHueco = ocupacionHabitacion->numeroClientes;
584                strncpy(ocupacionHabitacion->clientes[posicionHueco].dni, dniCliente, MAX_ID);
585                strncpy(ocupacionHabitacion->clientes[posicionHueco].nombre, nombreCliente,
      MAX_NOMBRE_CLIENTE);
586                ocupacionHabitacion->clientes[posicionHueco].todoIncluido = todoIncluido;
587                error = 0;
588
589           }else{
590               error = 2;
591           }
592       }else{
593           error = 1;
594       }
595
596       return error;
597   }
598
599   void listarClientes (const tRegistroClientes ocupacionHabitacion){
600
601       tId dniCliente;
602      int i;
603      bool Incluido;
604
605       for(i = 0; i < MAX_CLIENTES; i++){
606               if(existeCliente(dniCliente, ocupacionHabitacion)){
607                   Incluido = ocupacionHabitacion.clientes[i].todoIncluido;
608
609                   if (Incluido == true){
610                       printf("***Cliente %d: %s - Todo incluido \n", i+1,
      ocupacionHabitacion.clientes[i].dni, ocupacionHabitacion.clientes[i].nombre);
611                   }else{
612                       printf("***Cliente %d: %s - Solo desayuno \n", i+1,
      ocupacionHabitacion.clientes[i].dni, ocupacionHabitacion.clientes[i].nombre);
613                   }
```

```
614                 }
615         }
616     }
617
618
619
620
621     /** PROTOPTIPOS DE FUNCIONES FASE 2*/
622
623     int leerEnteroRango (const char mensaje [], int inferior, int superior){
624
625          int dato;
626
627         do{
628             printf ("%s", mensaje);
629             fflush(stdin);
630             scanf("%d", &dato);
631
632         } while ( dato < inferior || dato > superior);
633
634         return dato;
635     }
636
637     bool habitacionRegistrada (int numHabitacion, const tListaHabitaciones
    registroHabitaciones, int *posiArray ){
638
639         bool encontrado = false;
640
641
642         while ((*posiArray) < MAX_HABITACIONES  && encontrado == false){
643
    if((!registroHabitaciones[*posiArray].posicionLibre)&&(registroHabitaciones[*posiArray].habi
    tacion.numeroHabitacion == numHabitacion)){
644                 encontrado = true;
645             }else{
646              *posiArray =  *posiArray + 1;
647             }
648         }
649         return encontrado;
650     }
651
652     bool hayEspacioEnHotel (const tListaHabitaciones registroHabitaciones, int *posiArray){
653
654          bool haySitio = false;
655         (*posiArray)=0;
656
657         while (*posiArray < MAX_HABITACIONES && !haySitio)  {
658             if(registroHabitaciones[*posiArray].posicionLibre){
659                 haySitio = true;
660
661             }else{
662               (*posiArray)++;
663             }
664         }
665     return haySitio;
666     }
667
668     void abrirHotel (tListaHabitaciones registroHabitaciones){
669
670            int i;
671
672         for(i = 0; i < MAX_HABITACIONES; i++){
673
674             registroHabitaciones[i].habitacion.ocupacion.numeroClientes = 0;
675
676             registroHabitaciones[i].posicionLibre = true;
677
678         }
679     }
680
681     int annadirHabitacion (int numHabitacion, tTextoTipo tipoHabitacion, tListaHabitaciones
    *registroHabitaciones){
682
683         int error;
684         int posicionExiste=0;
685         int posicionHueco;
686
687
688
689         if (habitacionRegistrada(numHabitacion, *registroHabitaciones, &posicionExiste)){
690                 error = 1;
691
692         }else{
693
```

```
694             if (!hayEspacioEnHotel(*registroHabitaciones, &posicionHueco)){
695                 error = 2;
696             }else{
697                 (*registroHabitaciones)[posicionHueco].habitacion.numeroHabitacion =
    numHabitacion;
698                 strncpy((*registroHabitaciones)[posicionHueco].habitacion.tipo,
    tipoHabitacion, MAX_TIPO);
699                 (*registroHabitaciones)[posicionHueco].posicionLibre = false;
700                error = 0;
701             }
702         }
703 
704         return error;
705     }
706 
707     bool borrarHabitacion (int numHabitacion, tListaHabitaciones *registroHabitaciones){
708 
709         bool eliminado = false;
710         int posicion=0;
711 
712         if (habitacionRegistrada(numHabitacion, *registroHabitaciones, &posicion)) {
713             (*registroHabitaciones)[posicion].posicionLibre = true;
714             (*registroHabitaciones)[posicion].habitacion.numeroHabitacion = 0;
715 
716             (*registroHabitaciones)[posicion].habitacion.ocupacion.numeroClientes = 0;
717             eliminado = true;
718         }
719 
720         return eliminado;
721     }
722 
723     void listarHabitacionesOcupadas (tListaHabitaciones registroHabitaciones){
724 
725             for(int i = 0; i < MAX_HABITACIONES; i++){
726                     if (registroHabitaciones[i].posicionLibre == false){
727                         printf("-Habitacion %d (%s). Numero de ocupantes: %d. Listado:
    \n",registroHabitaciones[i].habitacion.numeroHabitacion,
    registroHabitaciones[i].habitacion.tipo,
    registroHabitaciones[i].habitacion.ocupacion.numeroClientes);
728 
729                     }
730             }
731     }
732 
733 
734 
735 
736     /** PROTOPTIPOS DE FUNCIONES FASE 3*/
737 
738     bool buscarCliente (const tId dniCliente, const tListaHabitaciones *registroHabitaciones){
739 
740         bool encontrado = false;
741          int i;
742          int j;
743 
744         for( i = 0; i < MAX_HABITACIONES; i++){
745 
746                 for(j = 0; i < MAX_CLIENTES; j++){
747 
    if(strcmp(dniCliente,registroHabitaciones[i]->habitacion.ocupacion.clientes[j].dni) == 0){
748                         encontrado = true;
749                         }
750                     }
751         }
752 
753         return encontrado;
754     }
755 
756     int totalClientesEnHotel (const tListaHabitaciones registroHabitaciones){
757 
758         int numeroClientes = 0;
759 
760         for (int i=0; i < MAX_HABITACIONES;i++){
761             numeroClientes = numeroClientes +
    registroHabitaciones[i].habitacion.ocupacion.numeroClientes;
762         }
763 
764 
765     return numeroClientes;
766     }
767 
768 
769 
770 
```

```
771  /** CÓDIGO DE FUNCIONES FASE 4: */
772
773  void MenuRegistraHabitacion (tListaHabitaciones *registroHabitaciones){
774
775
776      tTextoTipo tipoHabitacion;
777      int errorRegistro;
778
779      int numHabitacion = leerEnteroRango("Numero de habitacion: ",NUM_HABITACION_INF,
     NUM_HABITACION_SUP);
780      leerTexto("Tipo de habitacion: ",tipoHabitacion, MAX_TIPO);
781
782
783      errorRegistro = annadirHabitacion(numHabitacion, tipoHabitacion, registroHabitaciones);
784
785        if (errorRegistro == 0){
786           printf("***Habitacion %d dada de alta correctamente***\n", numHabitacion);
787        }
788        else{
789           if (errorRegistro == 1)
790              printf("***Error, no se pudo realizar el alta: La habitacion %d ya fue
     registrada***\n", numHabitacion);
791           else
792              printf("***Error, no se pudo realizar el alta: El hotel no tiene permisos para
     abrir mas habitaciones***\n");
793        }
794     }
795
796  void MenuEliminaHabitacion (tListaHabitaciones *registroHabitaciones){
797
798      int numHabitacion;
799
800     numHabitacion = leerEnteroRango("Numero de habitacion a borrar: ",NUM_HABITACION_INF,
     NUM_HABITACION_SUP);
801
802      if (borrarHabitacion(numHabitacion, &(*registroHabitaciones))){
803
804              printf("***La habitacion %d ha sido borrada se su sitema***\n", numHabitacion);
805
806      }else{
807          printf("***Error, la habitacion %d no consta como registrada en su sistema***\n",
     numHabitacion);
808      }
809  }
810
811  void MenuRegistraClientes (tListaHabitaciones *registroHabitaciones){
812
813      tId dniCliente;
814      tNombre nombreCliente;
815      tRegistroClientes ocupacionHabitacion;
816
817      int errorAltaCliente;
818      bool todoIncluido = false;
819      bool respuesta;
820      int posicion;
821      int pos=0;
822      int error = 0;
823
824
825      int  numHabitacion = leerEnteroRango("Numero de habitacion en la que registrar
     cliente: ",NUM_HABITACION_INF, NUM_HABITACION_SUP);
826      bool registrado = habitacionRegistrada(numHabitacion, *registroHabitaciones, &pos);
827
828      do{
829
830      if(registrado == true ){
831
832          ocupacionHabitacion.numeroClientes =
     (*registroHabitaciones)[pos].habitacion.ocupacion.numeroClientes;
833          errorAltaCliente = altaCliente(dniCliente, nombreCliente, todoIncluido,
     &ocupacionHabitacion);
834
835          if ( errorAltaCliente == 0  ){
836
837              posicion = ocupacionHabitacion.numeroClientes;
838
839
     strncpy((*registroHabitaciones)[pos].habitacion.ocupacion.clientes[posicion].dni,
     ocupacionHabitacion.clientes[posicion].dni,MAX_ID);
840
     strncpy((*registroHabitaciones)[pos].habitacion.ocupacion.clientes[posicion].nombre,
     ocupacionHabitacion.clientes[posicion].nombre, MAX_NOMBRE_CLIENTE);
841
     (*registroHabitaciones)[pos].habitacion.ocupacion.clientes[posicion].todoIncluido =
```

```
    ocupacionHabitacion.clientes[posicion].todoIncluido;
842
843                 (*registroHabitaciones)[pos].habitacion.ocupacion.numeroClientes++;
844                 printf("***Cliente dado de alta correctamente***\n");
845
846             }else{
847                 if (errorAltaCliente == 1){
848                         printf("***Error, no se pudo realizar el alta: El DNI %s ya fue
    registrado previamente***\n", dniCliente);
849                 }else{
850                     printf("***Error, no se pudo realizar el alta: La habitacion %d esta
    llena***\n", numHabitacion);
851                 }
852             }
853         respuesta = leerBooleano("\nDesea introducir mas clientes en la habitacion? (s/n): ");
854
855         }else{
856
857             printf("***Error, la habitacion %d no aparece como registrada en su sistema***\n",
    numHabitacion);
858              error = 1;
859         }
860
861        }while(respuesta == true && error != 1);
862    }
863
864    void MenuListaTotal (tListaHabitaciones registroHabitaciones){
865
866         int i;
867         bool Incluido;
868         bool posicionLibre = false;
869         bool registrado = false;
870         int numClientes;
871
872             for( i = 0; i < MAX_HABITACIONES; i++){
873                     posicionLibre = registroHabitaciones[i].posicionLibre;
874
875                     if(posicionLibre == false){
876                         printf("-Habitacion %i (%s). Numero de ocupantes: %i. Listado:
    \n",registroHabitaciones[i].habitacion.numeroHabitacion,
    registroHabitaciones[i].habitacion.tipo,
    registroHabitaciones[i].habitacion.ocupacion.numeroClientes);
877                         numClientes =
    registroHabitaciones[i].habitacion.ocupacion.numeroClientes;
878                         registrado = true;
879
880                         for(int j = 0; j < numClientes && numClientes !=0 ; j++){
881
882                             Incluido =
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
883                                 if (Incluido == true){
884                                     printf("***Cliente %d: %s - %s - Todo incluido \n", j+1,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].dni,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].nombre);
885                                 }else{
886                                     printf("***Cliente %d: %s - %s - Solo desayuno \n", j+1,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].dni,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].nombre);
887                                 }
888                         }
889                         if(numClientes == 0 ){
890                                 printf("***No hay clientes en la habitacion \n");
891                         }
892
893                 }
894             }
895             if(registrado == false){
896                 printf("***No hay ninguna habitacion registrada en su hotel \n");
897             }
898
899    }
900
901    void MenuListaHabitacion (tListaHabitaciones registroHabitaciones){
902
903
904         int i;
905         int j;
906         int numHabitacion;
907         int numClientes;
908         bool registrado = false;
909         bool Incluido = false;
910         printf("Introduzca el numero de la habitacion: ");
911         scanf("%d",&numHabitacion);
912
```

```
913         for(i=0; i<MAX_HABITACIONES && !registrado; i++){
914                 registrado =  habitacionRegistrada(numHabitacion, registroHabitaciones, &i);
915                 numClientes = registroHabitaciones[i].habitacion.ocupacion.numeroClientes;
916 
917                 if(registrado == true){
918                     for(j=0; j < numClientes && numClientes != 0;  j++){
919                         Incluido =
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
920                         if (Incluido == true){
921                                 printf("***Cliente %d: %s - %s - Todo incluido \n", j+1,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].dni,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].nombre);
922                                 }else{
923                                     printf("***Cliente %d: %s - %s - Solo desayuno \n", j+1,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].dni,
    registroHabitaciones[i].habitacion.ocupacion.clientes[j].nombre);
924                                 }
925                         }
926                     }
927         }
928 
929         if(registrado == false ){
930                 printf("***Error, la habitacion %d no aparece como registrada en su
    sistema***\n", numHabitacion);
931         }
932 
933         if(numClientes == 0){
934             printf("***No hay clientes en la habitacion \n");
935         }
936     }
937 
938     void MenuBuscaCliente (tListaHabitaciones registroHabitaciones){
939 
940         bool encontrado = false;
941         int i;
942         int j;
943         tId dniCliente;
944 
945         leerTexto("Indiqueme el DNI del cliente a buscar: ", dniCliente, MAX_ID);
946 
947         for(i=0; i<MAX_HABITACIONES && !encontrado;  i++){
948 
949                 for(j=0; j < ( registroHabitaciones[i].habitacion.ocupacion.numeroClientes);
    j++)   {
950 
    if(strcmp(dniCliente,registroHabitaciones[i].habitacion.ocupacion.clientes[j].dni) == 0
    && registroHabitaciones[i].habitacion.numeroHabitacion !=0){
951                             encontrado = true;
952                     }
953                 }
954                 if(encontrado){
955                         printf("***El cliente se encuentra en la habitacion numero
    %i***",registroHabitaciones[i].habitacion.numeroHabitacion);
956                 }
957         }
958         if(encontrado == false){
959                 printf("***El cliente indicado no se encuentra alojado en el hotel***");
960 
961         }
962 
963     }
964 
965     void MenuCuentaClientes (tListaHabitaciones registroHabitaciones){
966 
967         int numeroTotal = totalClientesEnHotel (registroHabitaciones);
968 
969         printf("El numero total de clientes alojados actualmente en el hotel es de %d
    personas", numeroTotal);
970 
971     }
972 
973 
974 
975 
976 
977     /** CÓDIGO DE FUNCIONES FASE 5: */
978 
979     bool leerDatosHotel (tListaHabitaciones registroHabitaciones, const char *nomFichero){
980 
981         FILE *pf;       // Referencia al fichero
982         int contador=0;    // Contador de elementos
983         bool error = NOERROR;
984         pf= fopen (nomFichero, "rb");
985         if (pf != NULL)    // Apertura del fichero
```

```
986         {
987
988             fread (&(registroHabitaciones[contador].habitacion),
    sizeof(registroHabitaciones[contador].habitacion), 1, pf);
989
990             while ((!feof(pf)) && (!ferror(pf)))  //feof-> find End Of File
991             {
992                 registroHabitaciones[contador].posicionLibre = false;
993                 contador ++;
994                 fread (&(registroHabitaciones[contador].habitacion),
    sizeof(registroHabitaciones[contador].habitacion), 1, pf);
995             }
996             if (ferror(pf)) // Ha habido error en la lectura del fichero
997                 error= ERROR;
998             fclose (pf);
999         }
1000        else   //Ha habido error en la apertura de fichero
1001        {
1002            error=ERROR;
1003        }
1004
1005        return error;
1006    }
1007
1008    bool guardarDatosHotel (const tListaHabitaciones registroHabitaciones, const char
    *nomFichero)
1009    {
1010
1011        FILE *pf; // Referencia al fichero
1012        int contador=0; // Contador de elementos
1013        bool error= NO_ERROR; // Comprueba si ha habido error
1014        pf= fopen (nomFichero, "wb"); //wb -> escritura/binario
1015        if (pf != NULL)
1016        {
1017            while ((!ferror(pf)) && (contador < MAX_HABITACIONES))
1018            {
1019                if (registroHabitaciones[contador].posicionLibre==false)
1020                {
1021                    fwrite(&(registroHabitaciones[contador].habitacion),
    sizeof(registroHabitaciones[contador].habitacion), 1, pf);
1022                }
1023                contador ++;
1024            }
1025            if (ferror(pf))  // Ha habido error en la escritura del fichero
1026            {
1027                error= ERROR;
1028            }
1029            fclose (pf);
1030
1031        }
1032        else   //Ha habido error en la apertura de fichero
1033        {
1034            error=ERROR;
1035        }
1036        return error;
1037    }
1038
```