```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];


/** 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 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 dni, 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 ocupacion);
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 *ocupacion);
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 dni, tNombre nombre, bool allInclusive, tRegistroClientes *ocupacion);
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 ocupacion);
136
137     /** PROTOTIPOS 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 numero, const tListaHabitaciones habitaciones, int *pos);
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 habitaciones, int *pos);
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 *habitaciones);
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 numero, tTextoTipo tipo, tListaHabitaciones *habitaciones);
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 numero, tListaHabitaciones *habitaciones);
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 habitaciones);
235
236     /** PROTOTIPOS 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 dni, const tListaHabitaciones habitaciones, int *habitacion);
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 habitaciones);
264
265     /** PROTOTIPOS 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 *habitaciones);
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 *habitaciones);
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 *habitaciones);
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 habitaciones);
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 habitaciones);
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 habitaciones);
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 habitaciones);
350
351     /** PROTOTIPOS 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 habitaciones, 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 (tListaHabitaciones habitaciones, char *nomFichero);
383
384
385     int main(void){
386
387         tRegistroClientes ocupacionHabitacion;
388         tListaHabitaciones registroHabitaciones;
389         char nomFichero[] = "datosHotel.dat";
390
391         vaciarHabitacion(&ocupacionHabitacion);
392         abrirHotel(&registroHabitaciones);
393
394         leerDatosHotel(registroHabitaciones, nomFichero);
395
396
397
398        int opcion;
399
400
401      do {
402
403             opcion = menu();
404
405
406             switch (opcion) {
407
408                 case 1:
409                      printf("\n");
410                      printf("Registrando de habitacion: \n");
411                      printf("--------------------------\n");
412                     MenuRegistraHabitacion(&registroHabitaciones);
413
414                     break;
415
416                 case 2:
417                     printf("\n");
418                     printf("Borrado de una habitacion del registro.\n");
419                     printf("---------------------------------------\n");
420                     MenuEliminaHabitacion(&registroHabitaciones);
421
422                     break;
423
424                 case 3:
425                     printf("\n");
426                     printf("Peticion de datos de clientes para su registro en una
    habitacion.\n");
427
    printf("-----------------------------------------------------------------\n");
428                     MenuRegistraClientes(&registroHabitaciones);
429
430                     break;
431                 case 4:
432                     printf("\n");
433                     printf("Listado de ocupacion del hotel.\n");
434                     printf("-------------------------------\n");
435                     MenuListaTotal(registroHabitaciones);
436
437                     break;
438                 case 5:
439                     printf("\n");
440                     printf("Listado de ocupacion de una habitacion.\n");
441                     printf("---------------------------------------\n");
442                     MenuListaHabitacion(registroHabitaciones);
443
444                     break;
445
446                 case 6:
447                     printf("\n");
448                     printf("Busqueda de la habitacion de un cliente.\n");
449                     printf("---------------------------------------\n");
450                     MenuBuscaCliente(registroHabitaciones);
451
452                     break;
453                 case 7:
454                     printf("\n");
455                     printf("Mostrar el numero total de clientes en el hotel.\n");
```

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

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

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

```
703         bool eliminado = false;
704         int posicion=0;
705 
706         if (habitacionRegistrada(numero, *habitaciones, &posicion)) {
707             (*habitaciones)[posicion].posicionLibre = true;
708             (*habitaciones)[posicion].habitacion.numeroHabitacion = 0;
709 
710             (*habitaciones)[posicion].habitacion.ocupacion.numeroClientes = 0;
711             eliminado = true;
712         }
713 
714         return eliminado;
715     }
716 
717     void listarHabitacionesOcupadas (tListaHabitaciones habitaciones){
718 
719             for(int i = 0; i < MAX_HABITACIONES; i++){
720                     if (habitaciones[i].posicionLibre == false){
721                         printf("-Habitacion %d (%s). Numero de ocupantes: %d. Listado:
    \n",habitaciones[i].habitacion.numeroHabitacion, habitaciones[i].habitacion.tipo,
    habitaciones[i].habitacion.ocupacion.numeroClientes);
722 
723                     }
724             }
725     }
726 
727 
728 
729 
730     /** CÓDIGO DE FUNCIONES FASE 3*/
731 
732     bool buscarCliente (const tId dni, const tListaHabitaciones habitaciones, int *habitacion){
733 
734         bool encontrado = false;
735          int i;
736         tRegistroClientes ocupacionHabitacion;
737         int pos = 0;
738 
739         for( i = 0; i < MAX_HABITACIONES; i++){
740                     if(existeCliente(dni, ocupacionHabitacion) &&
    habitacionRegistrada(*habitacion, habitaciones, &pos)){
741                         encontrado = true;
742                         pos = i;
743                         }
744 
745         }
746 
747         return encontrado;
748     }
749 
750     int totalClientesEnHotel (const tListaHabitaciones habitaciones){
751 
752         int numeroClientes = 0;
753 
754         for (int i=0; i < MAX_HABITACIONES;i++){
755             numeroClientes = numeroClientes +
    habitaciones[i].habitacion.ocupacion.numeroClientes;
756         }
757 
758 
759     return numeroClientes;
760     }
761 
762 
763 
764 
765     /** CÓDIGO DE FUNCIONES FASE 4: */
766 
767     void MenuRegistraHabitacion (tListaHabitaciones *habitaciones){
768 
769 
770         tTextoTipo tipoHabitacion;
771         int errorRegistro;
772 
773         int numHabitacion = leerEnteroRango("Numero de habitacion: ",NUM_HABITACION_INF,
    NUM_HABITACION_SUP);
774         leerTexto("Tipo de habitacion: ",tipoHabitacion, MAX_TIPO);
775 
776 
777         errorRegistro = annadirHabitacion(numHabitacion, tipoHabitacion, habitaciones);
778 
779           if (errorRegistro == 0){
780              printf("***Habitacion %d dada de alta correctamente***\n", numHabitacion);
781           }
```

```
782         else{
783            if (errorRegistro == 1)
784               printf("***Error, no se pudo realizar el alta: La habitacion %d ya fue registrada***\n", numHabitacion);
785            else
786               printf("***Error, no se pudo realizar el alta: El hotel no tiene permisos para abrir mas habitaciones***\n");
787         }
788      }
789
790   void MenuEliminaHabitacion (tListaHabitaciones *habitaciones){
791
792       int numHabitacion;
793
794      numHabitacion = leerEnteroRango("Numero de habitacion a borrar: ",NUM_HABITACION_INF, NUM_HABITACION_SUP);
795
796       if (borrarHabitacion(numHabitacion, &(*habitaciones))){
797
798               printf("***La habitacion %d ha sido borrada se su sitema***\n", numHabitacion);
799
800       }else{
801           printf("***Error, la habitacion %d no consta como registrada en su sistema***\n", numHabitacion);
802       }
803   }
804
805   void MenuRegistraClientes (tListaHabitaciones *habitaciones){
806
807       tId dniCliente;
808       tNombre nombreCliente;
809       tRegistroClientes ocupacionHabitacion;
810
811       int errorAltaCliente;
812       bool todoIncluido = false;
813       bool respuesta;
814       int posicion;
815       int pos=0;
816       int error = 0;
817
818
819       int  numHabitacion = leerEnteroRango("Numero de habitacion en la que registrar cliente: ",NUM_HABITACION_INF, NUM_HABITACION_SUP);
820       bool registrado = habitacionRegistrada(numHabitacion, *habitaciones, &pos);
821
822       do{
823
824       if(registrado == true ){
825
826           ocupacionHabitacion.numeroClientes = (*habitaciones)[pos].habitacion.ocupacion.numeroClientes;
827           errorAltaCliente = altaCliente(dniCliente, nombreCliente, todoIncluido, &ocupacionHabitacion);
828
829
830
831           if ( errorAltaCliente == 0  ){
832
833               posicion = ocupacionHabitacion.numeroClientes;
834
835               strncpy((*habitaciones)[pos].habitacion.ocupacion.clientes[posicion].dni, ocupacionHabitacion.clientes[posicion].dni,MAX_ID);
836               strncpy((*habitaciones)[pos].habitacion.ocupacion.clientes[posicion].nombre, ocupacionHabitacion.clientes[posicion].nombre, MAX_NOMBRE_CLIENTE);
837                (*habitaciones)[pos].habitacion.ocupacion.clientes[posicion].todoIncluido = ocupacionHabitacion.clientes[posicion].todoIncluido;
838
839               (*habitaciones)[pos].habitacion.ocupacion.numeroClientes++;
840               printf("***Cliente dado de alta correctamente***\n");
841
842           }else{
843               if (errorAltaCliente == 1){
844                       printf("***Error, no se pudo realizar el alta: El DNI %s ya fue registrado previamente***\n", dniCliente);
845               }else{
846                   printf("***Error, no se pudo realizar el alta: La habitacion %d esta llena***\n", numHabitacion);
847                   }
848           }
849       respuesta = leerBooleano("\nDesea introducir mas clientes en la habitacion? (s/n): ");
850
851       }else{
852
853           printf("***Error, la habitacion %d no aparece como registrada en su sistema***\n",
```

```
    numHabitacion);
854             error = 1;
855         }
856
857        }while(respuesta == true && error != 1);
858    }
859
860    void MenuListaTotal (tListaHabitaciones habitaciones){
861
862        int i;
863        bool Incluido;
864        bool posicionLibre = false;
865        bool registrado = false;
866        int numClientes;
867
868            for( i = 0; i < MAX_HABITACIONES; i++){
869                    posicionLibre = habitaciones[i].posicionLibre;
870
871                    if(posicionLibre == false){
872                        printf("-Habitacion %i (%s). Numero de ocupantes: %i. Listado:
    \n",habitaciones[i].habitacion.numeroHabitacion, habitaciones[i].habitacion.tipo,
    habitaciones[i].habitacion.ocupacion.numeroClientes);
873                        numClientes = habitaciones[i].habitacion.ocupacion.numeroClientes;
874                        registrado = true;
875
876                        for(int j = 0; j < numClientes && numClientes !=0 ; j++){
877                            Incluido =
    habitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
878                                if (Incluido == true){
879                                    printf("***Cliente %d: %s - %s - Todo incluido \n", j+1,
    habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
880                                }else{
881                                    printf("***Cliente %d: %s - %s - Solo desayuno \n", j+1,
    habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
882                                }
883                        }
884                        if(numClientes == 0 ){
885                                printf("***No hay clientes en la habitacion \n");
886                        }
887
888                }
889            }
890            if(registrado == false){
891                printf("***No hay ninguna habitacion registrada en su hotel \n");
892            }
893
894    }
895
896    void MenuListaHabitacion (tListaHabitaciones habitaciones){
897
898
899        int i;
900        int j;
901        int numHabitacion;
902        int numClientes;
903        bool registrado = false;
904        bool Incluido = false;
905        printf("Introduzca el numero de la habitacion: ");
906        scanf("%d",&numHabitacion);
907
908        for(i=0; i<MAX_HABITACIONES && !registrado; i++){
909               registrado =  habitacionRegistrada(numHabitacion, habitaciones, &i);
910               numClientes = habitaciones[i].habitacion.ocupacion.numeroClientes;
911
912               if(registrado == true){
913                    for(j=0; j < numClientes && numClientes != 0;  j++){
914                        Incluido =
    habitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
915                        if (Incluido == true){
916                                printf("***Cliente %d: %s - %s - Todo incluido \n", j+1,
    habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
917                                }else{
918                                    printf("***Cliente %d: %s - %s - Solo desayuno \n", j+1,
    habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
919                                }
920                        }
921                    }
922        }
923
924        if(registrado == false ){
```

```
925             printf("***Error, la habitacion %d no aparece como registrada en su sistema***\n", numHabitacion);
926         }
927
928         if(numClientes == 0){
929             printf("***No hay clientes en la habitacion \n");
930         }
931     }
932
933     void MenuBuscaCliente (tListaHabitaciones habitaciones){
934
935          bool encontrado = false;
936         int i;
937         int j;
938         tId dniCliente;
939
940         leerTexto("Indiqueme el DNI del cliente a buscar: ", dniCliente, MAX_ID);
941
942         for(i=0; i < MAX_HABITACIONES && !encontrado;  i++){
943                 for(j=0; j < ( habitaciones[i].habitacion.ocupacion.numeroClientes);  j++)   {
944
945     if(strcmp(dniCliente,habitaciones[i].habitacion.ocupacion.clientes[j].dni) == 0 && habitaciones[i].habitacion.numeroHabitacion !=0){
946                             encontrado = true;
947                     }
948                 }
949                 if(encontrado){
950                         printf("***El cliente se encuentra en la habitacion numero %i***",habitaciones[i].habitacion.numeroHabitacion);
951                 }
952         }
953         if(encontrado == false){
954                 printf("***El cliente indicado no se encuentra alojado en el hotel***");
955
956         }
957     }
958
959     void MenuCuentaClientes (tListaHabitaciones habitaciones){
960
961         int numeroTotal = totalClientesEnHotel (habitaciones);
962
963         printf("El numero total de clientes alojados actualmente en el hotel es de %d personas", numeroTotal);
964
965     }
966
967
968
969
970     /** CÓDIGO DE FUNCIONES FASE 5: */
971
972     bool leerDatosHotel (tListaHabitaciones habitaciones, char *nomFichero){
973
974         FILE *datosHotel;
975         tId dniCliente;
976         tNombre nombreCliente;
977         tTextoTipo tipo;
978         bool Incluido;
979         bool posicionLibre = false;
980         bool registrado = false;
981         int numClientes;
982
983         bool error = NOERROR;
984
985         datosHotel = fopen (nomFichero, "rb");
986
987         if (datosHotel != NULL ) {
988                 printf("***Registro de clientes y habitaciones actualizado*** \n");
989             for (int i = 0; i < MAX_HABITACIONES; i++) {
990                     posicionLibre = habitaciones[i].posicionLibre;
991
992                     if(posicionLibre == false){
993
994                             fscanf(datosHotel,"-Habitacion %d (%s). Numero de ocupantes: %d. Listado: \n",&habitaciones[i].habitacion.numeroHabitacion, habitaciones[i].habitacion.tipo,  &habitaciones[i].habitacion.ocupacion.numeroClientes);
995
996                             numClientes = habitaciones[i].habitacion.ocupacion.numeroClientes;
997                             registrado = true;
998
999                             for (int j = 0; j < numClientes && numClientes !=0; j++){
1000                                     Incluido = habitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
```

```
1001
1002                                    if (Incluido == true){
1003
1004                                        fscanf(datosHotel,"***Cliente %d: %s - %s - Todo
    incluido \n", &j+1, habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
1005                                    }else{
1006
1007                                        fscanf(datosHotel,"***Cliente %d: %s - %s - Todo
    incluido \n", &j+1, habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
1008                                    }
1009                            }
1010                            if(numClientes == 0 ){
1011
1012                                    fscanf(datosHotel,"***No hay clientes en la habitacion \n");
1013                            }
1014                        }
1015                    }
1016                    if(registrado == false){
1017
1018                                    fscanf(datosHotel,"***No hay ninguna habitacion registrada en
    su hotel \n");
1019                    }
1020
1021            }
1022            if(datosHotel == 0){
1023
1024                printf("***No se han encontrado datos de clientes y habitaciones. El hotel esta
    actualmente vacio***\n");
1025                for(int i = 0; i < MAX_HABITACIONES; i++){
1026
1027
    fread(&(habitaciones[i].habitacion.numeroHabitacion),sizeof(int),1,datosHotel);
1028                            fread((tipo),sizeof(tTextoTipo),1,datosHotel);
1029
    fread(&(habitaciones[i].habitacion.ocupacion.numeroClientes),sizeof(int),1,datosHotel);
1030                            fread((dniCliente),sizeof(tId),1,datosHotel);
1031                            fread((nombreCliente),sizeof(tNombre),1,datosHotel);
1032                        }
1033            }
1034
1035            fclose(datosHotel);
1036
1037
1038            return error;
1039        }
1040
1041        bool guardarDatosHotel (tListaHabitaciones habitaciones,char *nomFichero){
1042
1043            FILE *datosHotel;
1044            bool error = NOERROR;
1045            datosHotel = fopen (nomFichero, "wb");
1046            bool Incluido;
1047            bool posicionLibre = false;
1048            bool registrado = false;
1049            int numClientes;
1050
1051            if (datosHotel != NULL) {
1052                for (int i = 0; i < MAX_HABITACIONES; i++) {
1053                        posicionLibre = habitaciones[i].posicionLibre;
1054
1055                        if(posicionLibre == false){
1056
1057                                fprintf(datosHotel,"-Habitacion %d (%s). Numero de ocupantes: %d.
    Listado: \n",habitaciones[i].habitacion.numeroHabitacion,
    habitaciones[i].habitacion.tipo, habitaciones[i].habitacion.ocupacion.numeroClientes);
1058                                numClientes = habitaciones[i].habitacion.ocupacion.numeroClientes;
1059                                registrado = true;
1060
1061                                for (int j = 0; j < numClientes && numClientes !=0; j++){
1062                                        Incluido =
    habitaciones[i].habitacion.ocupacion.clientes[j].todoIncluido;
1063
1064                                    if (Incluido == true){
1065
1066                                        fprintf(datosHotel,"***Cliente %d: %s - %s - Todo
    incluido \n", j+1, habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
1067                                    }else{
1068
1069                                        fprintf(datosHotel,"***Cliente %d: %s - %s - Todo
    incluido \n", j+1, habitaciones[i].habitacion.ocupacion.clientes[j].dni,
    habitaciones[i].habitacion.ocupacion.clientes[j].nombre);
```

```
1070                                }
1071                        }
1072                        if(numClientes == 0 ){
1073
1074                                fprintf(datosHotel,"***No hay clientes en la habitacion \n");
1075                        }
1076                    }
1077                }
1078                if(registrado == false){
1079
1080                                 fprintf(datosHotel,"***No hay ninguna habitacion registrada en
    su hotel \n");
1081                }
1082
1083            fclose(datosHotel);
1084        }else{
1085            error = ERROR;
1086        }
1087
1088        return error;
1089    }
1090
1091
```