```c
/* PROGRAMACION 1 - CURSO 2020-2021 - PRACTICA 4 FASES 2 y 3 - p4f2f3.c */

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

/* DEFINICION DE CONSTANTES FASE 1*/

#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


/* DEFINICION DE CONSTANTES FASES 2 y 3*/
/* A CODIFICAR POR EL ALUMNO/A */



/* DEFINICION DE TIPOS FASE 1*/

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];



/* DEFINICION DE TIPOS FASES 2 y 3*/
/* A CODIFICAR POR EL ALUMNO/A */


/* 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 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);
```

```
80
81
82  /*
83      leerBooleano: Solicita que se introduzca desde la entrada estándar un carácter que
    puede ser 's',
84      'S', 'n' o 'N'. Si se introduce cualquier otro carácter solicita de nuevo la respuesta.
85      Si el usuario teclea 's' o 'S' la función devuelve true y si se teclea 'n' o 'N'
    devuelve false.
86      Parámetro de entrada:
87          mensaje: Cadena de caracteres con la frase de solicitud.
88      Valor devuelto por la función:
89          Booleano: true si se teclea 's' o 'S', false si se teclea 'n' o 'N'.
90  */
91  bool leerBooleano(const char mensaje[]);
92
93  /*
94      existeCliente: Comprueba si en la estructura ocupacion existe un cliente con el dni
    que se pasa
95      como parámetro.
96      Parámetros de entrada:
97          dni: Cadena de caracteres con el DNI del cliente a buscar.
98          ocupacion: Estructura con la información de la habitación en la que se busca al
    cliente.
99      Precondiciones:
100         ocupacion tiene que estar inicializado.
101     Valor devuelto por la función:
102         Booleano: true si el cliente está registrado en la habitación, false si no lo está.
103 */
104 bool existeCliente(const tId dni, tRegistroClientes ocupacion);
105
106 bool habitacionLlena (tRegistroClientes ocupacionHabitacion);
107 void vaciarHabitacion (tRegistroClientes *ocupacionHabitacion);
108 int altaCliente (tId dniCliente, tNombre nombreCliente, bool todoIncluido,
    tRegistroClientes *ocupacionHabitacion);
109 void listarClientes (tRegistroClientes ocupacionHabitacion);
110
111 int leerEnteroRango (const char mensaje [], int inferior, int superior);
112 bool habitacionRegistrada (int numHabitacion, const tListaHabitaciones
    registroHabitaciones, int *posiArray );
113 bool hayEspacioEnHotel (const tListaHabitaciones registroHabitaciones, int *posiArray);
114 void abrirHotel (tListaHabitaciones registroHabitaciones);
115
116 int annadirHabitacion (int numHabitacion, tTextoTipo tipoHabitacion, tListaHabitaciones
    *registroHabitaciones);
117 bool borrarHabitacion (int numHabitacion, tListaHabitaciones registroHabitaciones);
118 void listarHabitacionesOcupadas (tListaHabitaciones registroHabitaciones);
119
120 bool buscarCliente (const tId dniCliente, const tListaHabitaciones registroHabitaciones);
121 int totalClientesEnHotel (const tListaHabitaciones registroHabitaciones);
122
123
124 int main(void)
125 {
126    tListaHabitaciones registroHabitaciones; /* Array con la lista de posibles habitaciones
    disponibles */
127    tRegistroClientes ocupacionHabitacion;
128
129    int numHabitacion; /* Número de habitación */
130    tTextoTipo tipoHabitacion; /* Tipo de habitación */
131    int posiArray; /* Posición en el array en la que se ha localizado una habitación */
132     int errorRegistro;
133    tId dniCliente;
134    tNombre nombreCliente; /* Nombre y apellidos de un posible cliente */
135    bool todoIncluido;    /* Modalidad de reserva que se solicita */
136    int errorAltaCliente; /* DNI de cliente a buscar en el hotel */
137    int totalClientes; /* Número total de cliente en el hotel */
138
139    /* Abrir el hotel con todas las habitaciones marcadas como libres*/
140    /* A CODIFICAR POR EL ALUMNO/A */
141     vaciarHabitacion(&ocupacionHabitacion);
142     abrirHotel(registroHabitaciones);
143
144    /* DATOS PARA HACER LAS PRUEBAS
145    Se debe probar el programa intentando registrar 6 habitaciones:
146     - 100 ("Suite")
147     - 150 ("Individual")
148     - 100 ("Suite") (Tiene que resultar repetida)
149     - 250 (Tiene que resultar que el número de la habitación está fuera de rango)
150     - 200 ("Doble")
151     - 120 ("Doble") (No debe dejar porque el hotel tiene ya registradas todas las posibles
    habitaciones)
152     */
153
154
```

```
155      printf("\n");
156      printf("Registro de habitaciones:\n");
157
         printf("********************************************************************************\n");
158      for(int i = 1; i <= 5; i++){
159         printf("\n");
160         printf("Habitacion %d:\n", i);
161         printf("-------------\n");
162
163         numHabitacion = leerEnteroRango("Numero de habitacion: ",NUM_HABITACION_INF,
         NUM_HABITACION_SUP);
164
165         /* Pedir número de habitación a registrar */
166         /* A CODIFICAR POR EL ALUMNO/A */
167
168         /* Pedir tipo de habitación a registrar */
169         /* A CODIFICAR POR EL ALUMNO/A */
170         leerTexto("Tipo de habitacion: ",tipoHabitacion, MAX_TIPO);
171
172
173         /* Registrar la habitación en el hotel */
174         /* A CODIFICAR POR EL ALUMNO/A */
175          errorRegistro = annadirHabitacion(numHabitacion, tipoHabitacion,
         &registroHabitaciones);
176
177         if (errorRegistro == 0){
178            printf("***Habitacion %d dada de alta correctamente***\n", numHabitacion);
179         }
180         else{
181            if (errorRegistro == 1)
182               printf("***Error, no se pudo realizar el alta: La habitacion %d ya fue
         registrada***\n", numHabitacion);
183            else
184               printf("***Error, no se pudo realizar el alta: El hotel no tiene permisos para
         abrir mas habitaciones***\n");
185         }
186      }
187
188
189      /* DATOS PARA HACER LAS PRUEBAS
190      Tras registrar las habitaciones, se van a incluir clientes en la 100 y en la 200.
191      En la 100: (04672211P, David Martin, todoIncluido) y (12345678R, Laura Lopez, Solo
         desayuno)
192      En la 200: (22334455A, Maria Perez, todoIncluido)
193      Se dejará la 150 sin ocupantes.
194      No se van a controlar errores de insercion porque con estos valores de prueba no se
         producirán.
195      */
196
197      printf("\n");
198      printf("Peticion de datos de clientes para su registro en habitaciones.\n");
199
         printf("********************************************************************************\n");
200
201      /* Pedir número de habitación en la que se quieren incluir clientes. En la prueba se
         teclará 100 */
202      /* A CODIFICAR POR EL ALUMNO/A */
203      numHabitacion = leerEnteroRango("Numero de habitacion: ",NUM_HABITACION_INF,
         NUM_HABITACION_SUP);
204
205      /* Localizar la habitación cuyo número se acaba de pedir usando habitacionRegistrada() */
206      if(habitacionRegistrada(numHabitacion, registroHabitaciones, &posiArray)){
207         /* Dar de alta al cliente 04672211P, David Martin, todoIncluido */
208         /* A CODIFICAR POR EL ALUMNO/A */
209         printf("\n");
210         printf("-----------\n");
211         leerTexto("DNI: ", dniCliente, MAX_ID);
212         leerTexto("Nombre: ",nombreCliente,MAX_NOMBRE_CLIENTE);
213
214         /* Pedir la modalidad de reserva */
215         todoIncluido = leerBooleano("El cliente tiene todo incluido? (s/n): ");
216
217
218         /* Dar de alta al cliente 12345678R, Laura Lopez, Solo desayuno */
219         /* A CODIFICAR POR EL ALUMNO/A */
220           altaCliente(dniCliente, nombreCliente, todoIncluido, &ocupacionHabitacion);
221
222         printf("***Registro finalizado correctamente***\n");
223      }
224      else{
225         printf("***Error, la habitacion indicada no esta registrada***\n");
226      }
227
228      printf("\n");
```

```
229      /* Pedir número de habitación en la que se quieren incluir clientes. En la prueba se tecleará 200 */
230      /* A CODIFICAR POR EL ALUMNO/A */
231
232      numHabitacion = leerEnteroRango("Numero de habitacion: ",NUM_HABITACION_INF, NUM_HABITACION_SUP);
233
234      /* Localizar la habitación cuyo número se acaba de pedir usando habitacionRegistrada() */
235      if(habitacionRegistrada(numHabitacion, registroHabitaciones, &posiArray)){
236         /* Dar de alta al cliente 22334455A, Maria Perez, todoIncluido */
237         /* A CODIFICAR POR EL ALUMNO/A */
238         altaCliente(dniCliente, nombreCliente, todoIncluido, &ocupacionHabitacion);
239
240         printf("***Registro finalizado correctamente***\n");
241      }
242      else{
243         printf("***Error, la habitacion indicada no esta registrada***\n");
244      }
245
246      /* Tras la toma de datos, se presentará un listado de ocupación por pantalla */
247      printf("\n");
248      printf("Listado de ocupacion del hotel:\n");
249      printf("*********************************************************************************\n");
250      /* A CODIFICAR POR EL ALUMNO/A */
251           listarClientes(ocupacionHabitacion);
252
253   /* FASE 3. Se deben quitar los comentarios generales de esta parte y completar lo indicado */
254
255       // DATOS PARA HACER LAS PRUEBAS
256       // Se deben buscar un par de clientes indicados por el usuario para señalar en qué habitación se encuentran.
257       // El usuario puede indicar un cliente correcto y otro que no exista en el hotel.
258      printf("\n");
259      printf("Operacion de busqueda de clientes:\n");
260      printf("*********************************************************************************\n");
261      for(int i = 1; i <= 2; i++){
262         printf("\n");
263         printf("Preguntando por cliente %d:\n", i);
264         printf("----------------------------\n");
265
266         //Pedir DNI de un cliente
267         //A CODIFICAR POR EL ALUMNO/A
268         leerTexto("DNI: ", dniCliente, MAX_ID);
269
270         // Localizar al cliente con el DNI solicitado
271         if (buscarCliente(dniCliente, registroHabitaciones) ){
272            printf("***El cliente se encuentra ubicado en la habitacion %d***\n", numHabitacion);
273         }
274         else{
275            printf("***El cliente indicado no se encuentra alojado en el hotel***\n");
276         }
277      }
278
279      //Calcular el número total de clientes en el hotel
280      printf("\n");
281      printf("Mostrar el numero total de clientes en el hotel:\n");
282      printf("*********************************************************************************\n");
283
284      //A CODIFICAR POR EL ALUMNO/A
285       totalClientes = totalClientesEnHotel(registroHabitaciones);
286
287      printf("El numero total de clientes alojados actualmente en el hotel es de %d personas\n", totalClientes);
288
289      return 0;
290   }
291
292
293   void leerTexto(const char mensaje[], char dato[], int size){
294      do{
295         printf ("%s", mensaje);
296         fflush(stdin);
297         fgets (dato, size+1, stdin);
298      } while (dato[0] == '\n' );
299      if(dato[strlen(dato)-1] == '\n'){dato[strlen(dato)-1] = '\0';}
300   }
301
302   bool leerBooleano(const char mensaje[]){
303      char respuesta;
304      bool devuelve = false;
305
306      do{
```

```
307         printf ("%s", mensaje);
308         fflush(stdin);
309         scanf("%c", &respuesta);
310      } while ((respuesta != 's')&&(respuesta != 'S')&&(respuesta != 'n')&&(respuesta != 'N'));
311      if ((respuesta == 's')||(respuesta == 'S')){
312         devuelve = true;
313      }
314
315      return devuelve;
316   }
317
318   bool existeCliente(const tId dni, tRegistroClientes ocupacionHabitacion){
319      bool encontrado = false;
320      int compara;
321      int n = 0;
322
323      while ((n < ocupacionHabitacion.numeroClientes)&&(!encontrado)){
324         compara = strcmp(ocupacionHabitacion.clientes[n].dni, dni);
325         if (compara == 0){
326            encontrado = true;
327         }
328         n++;
329      }
330
331      return encontrado;
332   }
333
334   bool habitacionLlena (const tRegistroClientes ocupacionHabitacion){
335
336       return ocupacionHabitacion.numeroClientes < MAX_CLIENTES;
337   }
338
339   void vaciarHabitacion (tRegistroClientes *ocupacionHabitacion){
340
341           ocupacionHabitacion->numeroClientes = 0;
342   }
343
344   int altaCliente (tId dniCliente, tNombre nombreCliente, bool todoIncluido, tRegistroClientes *ocupacionHabitacion){
345
346     int error;
347       int posicionExiste = 0;
348       int posicionHueco = 0;
349       bool habitacionlleno;
350
351
352       leerTexto("Cliente-DNI: ", dniCliente, MAX_ID);
353       leerTexto("Cliente-Nombre: ",nombreCliente,MAX_NOMBRE_CLIENTE);
354       ocupacionHabitacion->clientes[posicionExiste].todoIncluido = leerBooleano("El cliente tiene todo incluido? (s/n): ");
355
356        bool encontrado = existeCliente(dniCliente, *ocupacionHabitacion);
357
358       if (encontrado== false){
359               habitacionlleno = habitacionLlena(*ocupacionHabitacion);
360
361           if (habitacionlleno == true){
362
363               error = 1;
364           }else{
365               strncpy(ocupacionHabitacion->clientes[posicionHueco].dni, dniCliente, MAX_ID);
366               strncpy(ocupacionHabitacion->clientes[posicionHueco].nombre, nombreCliente, MAX_NOMBRE_CLIENTE);
367               ocupacionHabitacion->clientes[posicionHueco].todoIncluido = todoIncluido;
368               error = 0;
369           }
370
371       }else{
372           error = 2;
373       }
374
375       return error;
376   }
377
378   void listarClientes (const tRegistroClientes ocupacionHabitacion){
379
380      int i;
381      bool Incluido;
382
383       for(i = 0; i < MAX_CLIENTES; i++){
384
385           Incluido = ocupacionHabitacion.clientes[i].todoIncluido;
386
```

```
387             if (Incluido == true){
388                     printf("***Cliente %d: %s - Todo incluido \n", i+1,
    ocupacionHabitacion.clientes[i].dni, ocupacionHabitacion.clientes[i].nombre);
389             }else{
390                 printf("***Cliente %d: %s - Solo desayuno \n", i+1,
    ocupacionHabitacion.clientes[i].dni, ocupacionHabitacion.clientes[i].nombre);
391             }
392         }
393     }
394
395     int leerEnteroRango (const char mensaje [], int inferior, int superior){
396
397          int dato;
398
399         do{
400             printf ("%s", mensaje);
401             fflush(stdin);
402             scanf("%d", &dato);
403
404         } while ( dato < inferior || dato > superior);
405
406         return dato;
407     }
408
409
410     bool habitacionRegistrada (int numHabitacion, const tListaHabitaciones
    registroHabitaciones, int *posiArray ){
411
412         bool encontrado = false;
413         int n = 0;
414
415         while ((n < MAX_HABITACIONES ) && (!encontrado)){
416
    if((!registroHabitaciones[n].posicionLibre)&&(registroHabitaciones[n].habitacion.numeroHabit
    acion == numHabitacion)){
417                 encontrado = true;
418                 *posiArray = true;
419             }
420            n++;
421         }
422         return encontrado;
423     }
424
425     bool hayEspacioEnHotel (const tListaHabitaciones registroHabitaciones, int *posiArray){
426
427         bool haySitio = false;
428         int i = 0;
429         bool encontrado = false;
430
431         while (i < MAX_HABITACIONES && encontrado == false) {
432             if(registroHabitaciones[i].posicionLibre == false){
433                 *posiArray = i;
434                 encontrado = true;
435                 haySitio = true;
436
437             }
438             i++;
439
440             if (encontrado == false) {
441                *posiArray = -1;
442             }
443         }
444         return haySitio;
445     }
446
447
448
449
450
451
452     void abrirHotel (tListaHabitaciones registroHabitaciones){
453
454            int i;
455
456         for(i = 0; i < MAX_HABITACIONES; i++){
457
458             registroHabitaciones[i].posicionLibre = false;
459
460         }
461     }
462
463     int annadirHabitacion (int numHabitacion, tTextoTipo tipoHabitacion, tListaHabitaciones
    *registroHabitaciones){
464
```

```
465         int error;
466         int posicionExiste;
467         int posicionHueco;
468 
469 
470         if (!habitacionRegistrada(numHabitacion, *registroHabitaciones, &posicionExiste)){
471 
472 
473             if (hayEspacioEnHotel(*registroHabitaciones, &posicionHueco)){
474                 strncpy(registroHabitaciones[posicionHueco]->habitacion.tipo, tipoHabitacion,
    MAX_TIPO);
475                 registroHabitaciones[posicionHueco]->habitacion.numeroHabitacion =
    numHabitacion;
476                 error = 0;
477 
478             }else{
479                 error = 1;
480             }
481 
482         }else{
483             error = 2;
484         }
485 
486         return error;
487     }
488 
489 
490 
491     bool borrarHabitacion (int numHabitacion, tListaHabitaciones registroHabitaciones){
492 
493         bool eliminado = false;
494         int posicion;
495 
496         if (habitacionRegistrada(numHabitacion, registroHabitaciones, &posicion)) {
497             registroHabitaciones[posicion].posicionLibre = false;
498             eliminado = true;
499         }
500 
501         return eliminado;
502     }
503 
504     void listarHabitacionesOcupadas (tListaHabitaciones registroHabitaciones){
505 
506             for(int i = 0; i < MAX_HABITACIONES; i++){
507                 printf("-Habitacion %d (%s). Numero de ocupantes: %d. Listado:
    \n",registroHabitaciones[i].habitacion.numeroHabitacion,
    registroHabitaciones[i].habitacion.tipo,
    registroHabitaciones[i].habitacion.ocupacion.numeroClientes);
508             }
509     }
510 
511     bool buscarCliente (const tId dniCliente, const tListaHabitaciones registroHabitaciones){
512 
513         bool encontrado = false;
514 
515 
516         for(int i = 0; i < MAX_HABITACIONES; i++){
517                 for(int j = 0; i < MAX_CLIENTES; j++){
518                     if(registroHabitaciones[i].posicionLibre == false &&
    strcmp(registroHabitaciones[i].habitacion.ocupacion.clientes[j].dni, dniCliente) < MAX_ID){
519                         encontrado = true;
520                         }
521                     }
522         }
523 
524         return encontrado;
525     }
526 
527     int totalClientesEnHotel (const tListaHabitaciones registroHabitaciones){
528 
529         int numeroClientes = 0;
530 
531         for (int i=0; i < registroHabitaciones[i].habitacion.numeroHabitacion;i++)
532             numeroClientes = numeroClientes +
    registroHabitaciones[i].habitacion.ocupacion.numeroClientes;
533 
534         return numeroClientes;
535 
536     }
537 
```