```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

/* 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;


/* 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 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.
    Valor devuelto por la función:
        Booleano: true si se teclea 's' o 'S', false si se teclea 'n' o 'N'.
*/
bool leerBooleano(const char mensaje[]);

/*
    existeCliente: Comprueba si en la estructura ocupacion existe un cliente con el dni que se pasa
    como parámetro.
    Parámetros de entrada:
        dni: Cadena de caracteres con el DNI del cliente a buscar.
        ocupacion: Estructura con la información de la habitación en la que se busca al cliente.
    Precondiciones:
        ocupacion: Tiene que estar inicializado.
    Valor devuelto por la función:
        Booleano: true si el cliente está registrado en la habitación, false si no lo está.
*/
bool existeCliente(const tId dni, tRegistroClientes ocupacion);

/*
    habitacionLlena: Comprueba si la habitación está llena.
```

```
76          Parámetro de entrada:
77              ocupacion: Estructura con la información de la habitación.
78          Precondiciones:
79              ocupacion tiene que estar inicializado.
80          Valor devuelto por la función:
81              Booleano: true si la habitación está llena, false si no lo está.
82      */
83      /* A CODIFICAR POR EL ALUMNO/A */
84
85      bool habitacionLlena (tRegistroClientes ocupacionHabitacion);
86
87      /*
88          vaciarHabitacion: Modifica los datos de la habitación para indicar que está vacía. Para ello pone a 0
89          el valor del campo numeroClientes.
90          Parámetro de salida pasado por referencia:
91              ocupacion: Estructura con la información de la habitación.
92      */
93      /* A CODIFICAR POR EL ALUMNO/A */
94      void vaciarHabitacion (tRegistroClientes *ocupacionHabitacion);
95
96      /*
97          altaCliente: Si la habitación no está llena, y el cliente no está ya registrado en esa habitación,
98          se incluyen los datos del cliente en la estructura ocupacion, detrás de los datos del último cliente
99          registrado.
100         Parámetros de entrada:
101             dni: Cadena de caracteres con el DNI del cliente.
102             nombre: Cadena de carcteres con el nombre y el apellido o apellidos del cliente.
103             allInclusive: Booleano con valor true si el cliente elige la modalidad de "todo incluido",
104             false si elige "alojamiento y desayuno" (son las 2 modalidades de reserva disponibles).
105         Parámetro de salida pasado por referencia:
106             ocupacion: Estructura con la información de la habitación.
107         Valor devuelto por la función para control de errores:
108             Entero: 0 si el cliente se ha registrado correctamente
109                     1 si el cliente ya estaba registrado
110                     2 si no hay sitio en la habitación para registrar nuevos clientes
111     */
112     /* A CODIFICAR POR EL ALUMNO/A */
113     int altaCliente (tId dniCliente, tNombre nombreCliente, bool todoIncluido,
        tRegistroClientes *ocupacionHabitacion);
114
115
116     /*
117         listarClientes: Lista todos los datos de los clientes registrados en la habitación.
118         Parámetro de entrada:
119             ocupacion: Estructura con la información de la habitación.
120         Precondiciones:
121             ocupacion tiene que estar inicializado.
122     */
123     /* A CODIFICAR POR EL ALUMNO/A */
124     void listarClientes (tRegistroClientes ocupacionHabitacion);
125
126
127     /* FUNCIÓN PRINCIPAL FASE 1 */
128
129     int main(void)
130     {
131        tRegistroClientes ocupacionHabitacion; /* Estructura con la información del registro de clientes de una habitación */
132        tId dniCliente;  /* DNI de un posible cliente */
133        tNombre nombreCliente; /* Nombre y apellidos de un posible cliente */
134        bool todoIncluido;   /* Modalidad de reserva que se solicita */
135        int errorAltaCliente; /* Posible error al intentar dar de alta a un cliente */
136
137
138         vaciarHabitacion(&ocupacionHabitacion);
139
140
141
142
143
144
145
146
147
148
149
150
151        /* Inicializar la habitacion con numero de clientes a 0, usando la función vaciarHabitacion() */
```

```
152      /* A CODIFICAR POR EL ALUMNO/A */
153
154
155
156
157      /* DATOS PARA HACER LAS PRUEBAS
158      Se debe probar el programa introduciendo los datos de los 6 clientes en la habitacion:
159         - 04672211P, David Martin, todoIncluido.
160         - 12345678R, Laura Lopez, todoIncluido.
161         - 22334455A, Maria Perez, solo desayuno.
162         - 12345678R, Laura Lopez, solo desayuno (Tiene que resultar repetido).
163         - 35888822C, Julio Reillo, solo desayuno.
164         - 11111111S, Mariano Ortega, solo desayuno (No debe dejar porque no hay suficiente
    espacio). */
165
166      printf("\n");
167      printf("Peticion de datos de clientes para su registro:\n");
168      printf("*********************************************************************************\n");
169      for(int i = 1; i <= 6; i++){
170         printf("\n");
171         printf("Cliente %d:\n", i);
172         printf("-----------\n");
173         leerTexto("DNI: ", dniCliente, MAX_ID);
174         leerTexto("Nombre: ",nombreCliente,MAX_NOMBRE_CLIENTE);
175
176         /* Pedir la modalidad de reserva */
177         todoIncluido = leerBooleano("El cliente tiene todo incluido? (s/n): ");
178
179
180         /* Intentar dar de alta al cliente */
181         /* A CODIFICAR POR EL ALUMNO/A */
182
183
184         errorAltaCliente = altaCliente(dniCliente, nombreCliente, todoIncluido,
    &ocupacionHabitacion);
185
186         if ( errorAltaCliente == 0  )
187            printf("***Cliente dado de alta correctamente***\n");
188         else{
189            if (errorAltaCliente == 1)
190               printf("***Error, no se pudo realizar el alta: El DNI ya fue registrado
    previamente***\n");
191            else
192               printf("***Error, no se pudo realizar el alta: La habitacion esta llena***\n");
193         }
194      }
195
196      /* Presentar un listado de ocupación por pantalla */
197      printf("\n");
198      printf("Listado de ocupacion de la habitacion:\n");
199      printf("*********************************************************************************\n");
200      /* A CODIFICAR POR EL ALUMNO/A */
201
202
203
204      return 0;
205   }
206
207
208   /* CÓDIGO DE FUNCIONES FASE 1 */
209
210   void leerTexto(const char mensaje[], char dato[], int size){
211      do{
212         printf ("%s", mensaje);
213         fflush(stdin);
214         fgets (dato, size+1, stdin);
215      } while (dato[0] == '\n' );
216      if(dato[strlen(dato)-1] == '\n'){dato[strlen(dato)-1] = '\0';}
217   }
218
219   bool leerBooleano(const char mensaje[]){
220      char respuesta;
221      bool devuelve = false;
222
223      do{
224         printf ("%s", mensaje);
225         fflush(stdin);
226         scanf("%c", &respuesta);
227      } while ((respuesta != 's')&&(respuesta != 'S')&&(respuesta != 'n')&&(respuesta !=
    'N'));
228      if ((respuesta == 's')||(respuesta == 'S')){
229         devuelve = true;
230      }
231
```

```
232      return devuelve;
233   }
234
235   bool existeCliente(const tId dni, tRegistroClientes ocupacionHabitacion){
236      bool encontrado = false;
237      int compara;
238      int n = 0;
239
240      while ((n < ocupacionHabitacion.numeroClientes)&&(!encontrado)){
241         compara = strcmp(ocupacionHabitacion.clientes[n].dni, dni);
242         if (compara == 0){
243            encontrado = true;
244         }
245         n++;
246      }
247
248      return encontrado;
249   }
250
251
252   /* CÓDIGO DE FUNCIONES FASE 1: habitacionLlena(), vaciarHabitacion(), altaCliente(),
      listarClientes()*/
253   /* A CODIFICAR POR EL ALUMNO/A */
254
255   bool habitacionLlena (tRegistroClientes ocupacionHabitacion);
256   void vaciarHabitacion (tRegistroClientes *ocupacionHabitacion);
257   int altaCliente (tId dniCliente, tNombre nombreCliente, bool todoIncluido,
      tRegistroClientes *ocupacionHabitacion);
258   void listarClientes (tRegistroClientes ocupacionHabitacion);
259
260
261   bool habitacionLlena (const tRegistroClientes ocupacionHabitacion){
262
263       return ocupacionHabitacion.numeroClientes < MAX_CLIENTES;
264   }
265
266   void vaciarHabitacion (tRegistroClientes *ocupacionHabitacion){
267
268           ocupacionHabitacion -> numeroClientes = 0;
269   }
270
271
272
273
274   int altaCliente (tId dniCliente, tNombre nombreCliente, bool todoIncluido,
      tRegistroClientes *ocupacionHabitacion){
275
276       int error;
277       int i = 0;
278       bool habitacionlleno;
279
280
281       leerTexto("Cliente-DNI: ", dniCliente, MAX_ID);
282       leerTexto("Cliente-Nombre: ",nombreCliente,MAX_NOMBRE_CLIENTE);
283       ocupacionHabitacion->clientes[i].todoIncluido = leerBooleano("El cliente tiene todo
      incluido? (s/n): ");
284
285        bool encontrado = existeCliente(dniCliente, *ocupacionHabitacion);
286
287       if (encontrado== false){
288               habitacionlleno = habitacionLlena(*ocupacionHabitacion);
289
290           if (habitacionlleno == true){
291
292               error = 1;
293           }else{
294               strncpy(ocupacionHabitacion->clientes[i].dni, dniCliente, MAX_ID);
295               strncpy(ocupacionHabitacion->clientes[i].nombre, nombreCliente,
      MAX_NOMBRE_CLIENTE);
296               ocupacionHabitacion->clientes[i].todoIncluido = todoIncluido;
297               error = 0;
298           }
299
300       }else{
301           error = 2;
302       }
303
304       return error;
305   }
306
307
308   void listarClientes (const tRegistroClientes ocupacionHabitacion){
309
310      int i;
```

```
311
312         for(i = 0; i < MAX_CLIENTES; i++){
313             printf("***Cliente %d: %s - %s \n",i+1, ocupacionHabitacion.clientes[i].dni,
    ocupacionHabitacion.clientes[i].nombre);
314
315         }
316     }
317
318
319
320
321
```