

Metodología de la Programación

Grado en Ingeniería Informática

Curso 2022/2023

Durán Obregón, Alejandro

Ruíz Miñán, José Antonio

Ruíz del Pino, Alejandro

Vázquez Vera, Luis Enrique



Proyecto Modularidad

ESI-SHARE

Índice general

Índice general	1
1. Introducción	2
2. Documentación de usuario	3
2.1. Descripción funcional	3
2.2. Tecnología	5
2.3. Manual de instalación	5
2.4. Acceso al sistema	6
2.5. Manual de referencia	6
3. Documentación de sistema	8
3.1. Especificación del sistema	8
3.2. Módulos	8
3.2.1. Acceso	8
3.2.2. Colores	8
3.2.3. Crear Viaje	8
3.2.4. Eliminar	8
3.2.5. Encontrar	8
3.2.6. Escribir	8
3.2.7. Fecha	8
3.2.8. Leer	8
3.2.9. Listar	24
3.2.10. Menús	24
3.2.11. Modificar	24
3.2.12. Mostrar Reservar	24
3.2.13. Preguntar	24
3.2.14. Reservar	24
3.3. Plan de prueba	24
3.3.1. Prueba de los módulos	24
3.3.2. Prueba de integración	24
3.3.3. Plan de pruebas de aceptación	24
4. Documentación del código fuente	25

1. Introducción

¿Eres parte de la comunidad universitaria de la Escuela Superior de Ingeniería y buscas una alternativa de transporte sostenible y económica?

¿Te gustaría compartir tu vehículo con otros miembros de la comunidad que recorran la misma ruta?

Si es así, estás de suerte, ya que hemos desarrollado un programa para compartir coche, que es exclusivo para los miembros de la comunidad de la ESI.

¿Cómo funciona? Es fácil, ESI-SHARE cuenta con una interfaz amigable e intuitiva que facilita la gestión de los viajes compartidos. Los usuarios pueden registrar sus viajes y especificar la ruta y horario, así como el número de plazas disponibles en su vehículo, para otros miembros de la comunidad que compartan la misma ruta. También, pueden buscar viajes disponibles que se ajusten a sus necesidades y reservarlos. Todo esto con el objetivo de reducir el número de vehículos en circulación, y disminuir el impacto ambiental del transporte.

En definitiva, esta es una herramienta innovadora que contribuye al fomento de la movilidad sostenible y al cuidado del medio ambiente, a la vez que promueve la colaboración y el uso eficiente de los recursos.

¿Te animas a probarlo?

2. Documentación de usuario

2.1. Descripción funcional

Este programa ha sido creado con la finalidad de brindar un servicio de compartición de coches a los miembros de la comunidad universitaria de la Escuela Superior de Ingeniería. Con esto, se conseguirá una gran reducción de los costes diarios de transportes, al igual que una significativa disminución de gases.

Como hemos comentado anteriormente, queremos dar un servicio de calidad, es por esto que con nuestro programa usted podrá realizar diferentes acciones, como:

- Siendo **Pasajero** se le permitirá:
 - Consultar y variar sus datos personales.

```
ID de usuario: 0002
Nombre completo: Guillermo Gomez
Localidad de residencia: Cadiz
Tipo de perfil: usuario
Usuario: 1

¿Que quiere hacer?
(1)Modificar nombre completo.
(2)Modificar localidad de residencia.
(3)Modificar usuario.
(4)Modificar contraseña.
(5)Volver.
```

- Reservar y cancelar viajes.

```
¿Que quiere hacer?
(1)Reservar viaje.
(2)Cancelar viaje.
(3)Volver.
```

- Siendo **Conductor** se le permitirá:
 - Consultar y cambiar sus datos personales.

```

ID de usuario: 0002
Nombre completo: Guillermo Gomez
Localidad de residencia: Cadiz
Tipo de perfil: usuario
Usuario: 1

¿Que quiere hacer?
(1)Modificar nombre completo.
(2)Modificar localidad de residencia.
(3)Modificar usuario.
(4)Modificar contrasena.
(5)Volver.

```

- Dar de alta, modificar y eliminar vehículos.

```

¿Que quiere hacer?
(1)Alta de vehiculo.
(2)Modificar vehiculo.
(3)Eliminar vehículo.
(4)Volver.

```

- Crear, modificar, anular y finalizar viajes.

```

¿Que quiere hacer?
(1)Crear viaje.
(2)Modificar viaje.
(3)Anular/Finalizar viaje.
(4)Volver.

```

- Siendo **Administrador** se le permitirá:

- Crear, descartar, cambiar y listar usuarios.

```

Hola Juan Perez (Administrador)
¿Que quiere hacer?
(1)Alta de usuario.
(2)Baja de usuario.
(3)Modificar usuario.
(4)Listar usuarios.
(5)Volver.

```

- Dar de alta, suprimir, variar y listar vehículos.

- Establecer, anular, finalizar, cancelar, modificar y listar viajes.

2.2. Tecnología

Este programa lo hemos realizado con el entorno de desarrollo integrado (IDE) Code::Blocks, en su versión 20.03. A la hora de programar, hemos tenido que emplear diferentes bibliotecas, como:

- `stdio.h`: Manejar la entrada y salida de datos a través de archivos, dispositivos de entrada/salida estándar, etc.
- `string.h`: Usar cadenas de caracteres.
- `stdlib.h`: Emplear estructuras, memoria dinámica, conversión de cadenas, etc.
- `wchar.h`: Manipular cadenas de caracteres Unicode.
- `locale.h`: Proporcionar el uso de caracteres especiales de un idioma.
- `windows.h`: Incluir comandos del sistema operativo Microsoft Windows.
- `conio.h`: Utilizar la entrada y salida de caracteres por la consola.
- `time.h`: Trabajar con fechas y tiempos.

Por ejemplo, se han empleado vectores con estructuras dinámicas, o incluso matrices/vector de vectores dinámicos. Esto se puede observar en el módulo `reservar`. También se ha hecho uso de ficheros temporales, para eliminar/modificar, como se puede ver en en los módulos `eliminar` y `modificar`.

2.3. Manual de instalación

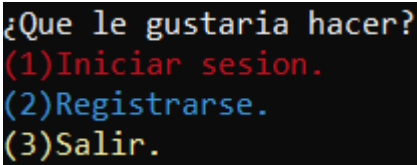
Para usar el programa, sólo tiene que dirigirse a la carpeta ESI-SHARE. Ahí encontrará un archivo ejecutable, llamado `ESI-SHARE.exe`, con el que podrá ejecutar el programa. Otra forma sería instalar Code::Blocks, y ejecutar el archivo de proyecto `ESI-SHARE.cbp`.

2.4. Acceso al sistema

Tras haber iniciado el programa, se encontrará un menú, en el que encontrará dos opciones, una para acceder al sistema, si ya tiene unas credenciales, y otra para registrarse, que le pedirá sus datos personales, para darse de alta. Si quiere testear el acceso al sistema, puede introducir:

- Para acceder como usuario:
 - Usuario: **usua1**
 - Contraseña: **2023**
- Para acceder como administrador:
 - Usuario: **admin**
 - Contraseña: **1234**

A la hora de salir, se ha elaborado un mecanismo intuitivo y sencillo, ya que en todo momento podrá volver al menú anterior, introduciendo el número indicado por pantalla. En el caso de querer salir totalmente del programa, tendrá que cerrar sesión, volviendo al menú principal.



```
¿Que le gustaria hacer?  
(1)Iniciar sesion.  
(2)Registrarse.  
(3)Salir.
```

2.5. Manual de referencia

El programa le puede aportar un gran ahorro de dinero en desplazamientos, debido al óptimo sistema de reservas que desempeña, haciendo posible que cualquier estudiante, profesor o personal de la Escuela Superior de Ingeniería, pueda acceder a dichas ventajas.

Cuando ejecute al programa, podrá acceder con sus credenciales, o registrarse. Si accede con sus credenciales, tendrá las opciones de ser "Pasajero" ó "Conductor", dependiendo de lo que quiera hacer, seleccionará uno u otro, esto se puede cambiar en cualquier momento, volviendo a dicho menú. 2.4

- Si elige la opción de ser **Pasajero**, podrá:
 - Entrar en el menú de **Perfil**, donde puede ver todos sus datos personales, al igual que si selecciona una de las opciones, podrá modificar cada uno de sus datos.

- Acceder al menú de **Viajes**, en el que se le permitirá reservar un viaje ya existente que pase por su localidad, al igual que cancelar cualquier reserva.
- Si elige la opción de ser **Conductor**, podrá:
 - Entrar en el menú de **Perfil**, donde puede ver todos sus datos personales, al igual que si selecciona una de las opciones, podrá modificar cada uno de sus datos.
 - Pasar al menú de **Vehículos**, para introducir nuevos vehículos, modificarlos o incluso eliminarlos. Para esto, la matrícula del vehículo no puede existir en la base de datos.
 - Acceder al menú de **Viajes**, en el que se le permitirá crear un viaje nuevo, siempre y cuando no haya dos abiertos en el mismo día, habría que acabar uno para empezar el siguiente, todo esto con el objetivo de que no se solapen los viajes. Además, se pueden modificar los viajes, si estos están abiertos y sin plazas reservadas, e incluso anular ó finalizar los viajes, esta selección la hará automáticamente el sistema, dependiendo del estado del viaje.

Asimismo, hay un menú especial, si accede con las credenciales del **Administrador**. En este menú, tendrá acceso a todo tipo de información de la base de datos, como:

- Entrar en el menú de **Usuarios**, donde puede crear un usuario desde cero, eliminar cualquier usuario del sistema, modificar los datos personales de cualquiera, e incluso obtener una lista con todos los usuarios que hay en el sistema.
- Pasar al menú de **Vehículos**, para dar de alta nuevos vehículos, eliminar un vehículo, modificar los datos de un vehículo, imprimir una lista de todos los vehículos que hay registrados al igual que obtener el historial de todos los viajes que ha realizado un vehículo, a partir de su matrícula.
- Acceder al menú de **Viajes**, en el que se le permitirá registrar un viaje, anular/finalizar, eliminar y modificar cualquier viaje del sistema, y obtener una lista con todos los viajes que hay registrados en el sistema.

3. Documentación de sistema

3.1. Especificación del sistema

Esta sección debe describir el análisis y la especificación de requisitos. Cómo se descompone el problema en distintos subproblemas y los módulos asociados a cada uno de ellos, acompañados de su especificación. También debe incluir el plan de desarrollo del *software*.

3.2. Módulos

Debe contener una descripción de cada módulo, su funcionalidad y la interfaz. También se podría añadir código fuente.

En este proyecto, hemos empleado una descomposición por tareas, haciendo que el programa se quede dividido en 15 módulos, que son:

3.2.1. Acceso

Sirve para darle acceso a los usuarios al sistema, mediante la autenticación de las credenciales que introduzcan. Para esto se han usado algunos mecanismos interesantes, como, la detección automática del perfil del usuario, si el que inicia sesión es usuario ó administrador. Además, se ha implementado un sistema de seguridad, con el que el usuario solamente tendrá 3 intentos para introducir la contraseña.

3.2.2. Colores

3.2.3. Crear Viaje

3.2.4. Eliminar

3.2.5. Encontrar

3.2.6. Escribir

3.2.7. Fecha

3.2.8. Leer

```

#include "leer.h"

//Prototipo: void leer(Estr_Usuario **, int *,
    Estr_Vehiculo **, int *, Estr_Viaje **, int *,
    Estr_Pasos **, int *, Estr_Reservas **, int *,
    Estr_Rutas ***, int *, int *, Estr_Localidad **, int
    *);
//Precondicion: Tener todas las estructuras
    inicializadas , con sus respectivos contadores.
    Ademas todas tienen que pasarse por puntero , para
    introducir valores en las mismas.
//Postcondicion: Leer todos los ficheros , e introducir
    la informacion en su estructura , con sus contadores.

void leer(Estr_Usuario **usuario , int *numUsuarios ,
    Estr_Vehiculo **vehiculo , int *numVehiculos ,
    Estr_Viaje **viaje , int *numViajes , Estr_Pasos **
    pasos , int *numPasos , Estr_Reservas **reservas , int
    *numReservas , Estr_Rutas ***ruta , int *numRutas , int
    *numRutas2 , Estr_Localidad **localidad , int *
    numLocalidades)
{
    leer_usuario(usuario , numUsuarios); //Llama a todas
        las funciones de leer , cada uno de los ficheros
        .

    leer_vehiculo(vehiculo , numVehiculos);
    leer_viaje(viaje , numViajes);
    leer_pasos(pasos , numPasos);
    leer_localidad(localidad , numLocalidades);
    leer_ruta(ruta , numRutas , numRutas2);
    leer_reservas(reservas , numReservas);
    system("PAUSE");
}

//Prototipo: void leer_usuario(Estr_Usuario **, int *);
//Precondicion: Tener la estructura inicializada , con
    su contador.
//Postcondicion: Leer el fichero "usuarios.txt" e
    introducir la informacion en su estructura ,
    aumentando el contador cada vez que se encuentra uno
    .

```

```

void leer_usuario(Estr_Usuario **usuario , int *i)
{
    FILE *fp;
    char vec[80] , *token;
    *i=0;

    fp=fopen("DATA/usuarios.txt" , "r"); //Abrimos el
        fichero en modo lectura.

    (*usuario)=malloc((*i)*sizeof(Estr_Usuario)); //
        Asignamos un espacio de memoria a la estructura.
    if ((*usuario)==NULL)
    {
        printf("Error al asignar memoria.\n");
        exit(1);
    }

    if(fp==NULL)
    {
        printf("No se ha podido abrir el fichero _
            usuarios.txt.\n");
        return;
    }
    else
    {
        while(fgets(vec , 80 , fp)) //Obtenemos la
            primera linea , y la introducimos en "vec".
        {
            if(strcmp(vec , "\n")!=0) //Cuando haya un
                salto de linea , pasamos a la siguiente.
            {
                *usuario=realloc(*usuario , ((*i)+1)*
                    sizeof(Estr_Usuario)); //Vamos
                    ampliando la memoria dinamica de la
                    estructura.
                if (*usuario==NULL)
                {
                    printf("Error al asignar memoria.\n
                        ");
                    exit(1);
                }
            }
        }
    }
}

```

```

    }
    token=strtok(vec,"-"); //Leemos "vec",
        y lo cortamos cuando tenemos "-".
    strcpy((*usuario)[*i].id_usuario,token)
        ; //Copiamos lo que hemos "cortado",
        en la cadena correspondiente de la
        estructura.
    token=strtok(NULL,"-");
    strcpy((*usuario)[*i].nomb_usuario,
        token);
    token=strtok(NULL,"-");
    strcpy((*usuario)[*i].localidad,token);
    token=strtok(NULL,"-");
    strcpy((*usuario)[*i].perfil,token);
    token=strtok(NULL,"-");
    strcpy((*usuario)[*i].usuario,token);
    token=strtok(NULL,"\n"); //Vamos
        leyendo hasta que haya un salto de
        linea.
    strcpy((*usuario)[*i].contrasena,token)
        ;

    (*i)++; //Vamos aumentando el contador,
        para saber el numero maximo de
        usuarios que hay en la base de datos
        .
    }
}

fclose(fp);

printf("Se_han_cargado_%i_usuarios.\n", *i);
}

//Prototipo: void leer_vehiculo(Estr_Vehiculo **, int
    *);
//Precondicion: Tener la estructura inicializada, con
    su contador.
//Postcondicion: Leer el fichero "vehiculos.txt" e
    introducir la informacion en su estructura,

```

aumentando el contador cada vez que se encuentra uno

.

```
void leer_vehiculo(Estr_Vehiculo **vehiculo , int *i)
{
    FILE *fp;
    char vec[70] , *token;
    *i=0;

    fp=fopen("DATA/vehiculos.txt" , "r"); //Abrimos el
        archivo en modo lectura.

    (*vehiculo)=malloc((*i)*sizeof(Estr_Vehiculo)); //
        Asignamos un espacio de memoria a la estructura.
    if ((*vehiculo)==NULL)
    {
        printf("Error al asignar memoria.\n");
        exit(1);
    }

    if(fp==NULL)
    {
        printf("No se ha podido abrir el archivo _\nvehiculos.txt.\n");
        return;
    }
    else
    {
        while(fgets(vec , 70, fp)) //Obtenemos la
            primera linea , y la introducimos en "vec".
        {
            if(strcmp(vec , "\n")!=0) //Cuando haya un
                salto de linea , pasamos a la siguiente.
            {
                *vehiculo=realloc(*vehiculo , ((*i)+1)*
                    sizeof(Estr_Vehiculo)); //Vamos
                    ampliando la memoria dinamica de la
                    estructura.
                if (*vehiculo==NULL)
                {
```

```

        printf("Error al asignar memoria.\n
        ");
        exit(1);
    }
    token=strtok(vec,"-"); //Leemos "vec",
        y lo cortamos cuando tenemos "-".
    strcpy((*vehiculo)[*i].id_mat,token);
        //Copiamos lo que hemos "cortado",
        en la cadena correspondiente de la
        estructura.
    token=strtok(NULL,"-");
    strcpy((*vehiculo)[*i].id_usuario,token
    );
    token=strtok(NULL,"-");
    strcpy((*vehiculo)[*i].num_plazas,token
    );
    token=strtok(NULL,"\n"); //Vamos
        leyendo hasta que haya un salto de
        linea.
    strcpy((*vehiculo)[*i].desc_veh,token);

    (*i)++; //Vamos aumentando el contador,
        para saber el numero maximo de
        vehiculos que hay en la base de
        datos.
    }
}

fclose(fp);

printf("Se han cargado %i vehiculos.\n", *i);
}

//Prototipo: void leer_viaje(Estr_Viaje **, int *);
//Precondicion: Tener la estructura inicializada, con
    su contador.
//Postcondicion: Leer el fichero "viajes.txt" e
    introducir la informacion en su estructura,
    aumentando el contador cada vez que se encuentra uno
    .

```

```

void leer_viaje(Estr_Viaje **viaje , int *i)
{
    FILE *fp;
    char vec[65] , *token;
    *i=0;

    fp=fopen("DATA/viajes.txt" , "r"); //Abrimos el
        fichero en modo lectura.

    (*viaje)=malloc((*i)*sizeof(Estr_Viaje)); //
        Asignamos un espacio de memoria a la estructura.
    if ((*viaje)==NULL)
    {
        printf("Error al asignar memoria.\n");
        exit(1);
    }

    if(fp==NULL)
    {
        printf("No se ha podido abrir el fichero viajes
            .txt.\n");
        return;
    }
    else
    {
        while(fgets(vec , 65, fp)) //Obtenemos la
            primera linea , y la introducimos en "vec".
        {
            if(strcmp(vec,"\\n")!=0) //Cuando haya un
                salto de linea , pasamos a la siguiente.
            {
                *viaje=realloc(*viaje , ((*i)+1)*sizeof(
                    Estr_Viaje)); //Vamos ampliando la
                    memoria dinamica de la estructura.
                if (*viaje==NULL)
                {
                    printf("Error al asignar memoria.\n
                        ");
                    exit(1);
                }
            }
        }
    }
}

```

```

        token=strtok(vec,"-"); //Leemos "vec",
        y lo cortamos cuando tenemos "-".
        strcpy((*viaje)[*i].id_viaje,token); //
        Copiamos lo que hemos "cortado", en
        la cadena correspondiente de la
        estructura.
        token=strtok(NULL,"-");
        strcpy((*viaje)[*i].id_mat,token);
        token=strtok(NULL,"-");
        strcpy((*viaje)[*i].f_inic,token);
        token=strtok(NULL,"-");
        strcpy((*viaje)[*i].h_inic,token);
        token=strtok(NULL,"-");
        strcpy((*viaje)[*i].h_fin,token);
        token=strtok(NULL,"-");
        strcpy((*viaje)[*i].plazas_libre,token)
        ;
        token=strtok(NULL,"-");
        strcpy((*viaje)[*i].ida_vuelta,token);
        token=strtok(NULL,"-");
        strcpy((*viaje)[*i].precio,token);
        token=strtok(NULL,"\n"); //Vamos
        leyendo hasta que haya un salto de
        linea.
        strcpy((*viaje)[*i].estado,token);

        (*i)++; //Vamos aumentando el contador,
        para saber el numero maximo de
        viajes que hay en la base de datos.
    }
}

fclose(fp);

printf("Se_han_cargado_%i_viajes.\n", *i);
}

//Prototipo: void leer_pasos(Estr_Pasos **, int *);
//Precondicion: Tener la estructura inicializada, con
su contador.

```


//Postcondicion: Leer el fichero "pasos.txt" e introducir la informacion en su estructura, aumentando el contador cada vez que se encuentra uno .

```

void leer_pasos(Estr_Pasos **pasos, int *i)
{
    FILE *fp;
    char vec[35], *token;
    *i=0;

    fp=fopen("DATA/pasos.txt", "r"); //Abrimos el fichero en modo lectura.

    (*pasos)=malloc((*i)*sizeof(Estr_Pasos)); // Asignamos un espacio de memoria a la estructura.
    if ((*pasos)==NULL)
    {
        printf("Error al asignar memoria.\n");
        exit(1);
    }

    if(fp==NULL)
    {
        printf("No se ha podido abrir el fichero pasos.txt.\n");
        return;
    }
    else
    {
        while(fgets(vec, 35, fp)) //Obtenemos la primera linea, y la introducimos en "vec".
        {
            if(strcmp(vec, "\n")!=0) //Cuando haya un salto de linea, pasamos a la siguiente.
            {
                *pasos=realloc(*pasos, ((*i)+1)*sizeof(Estr_Pasos)); //Vamos ampliando la memoria dinamica de la estructura.
                if (*pasos==NULL)
                {

```

```

        printf("Error al asignar memoria.\n
        ");
        exit(1);
    }
    token=strtok(vec,"-"); //Leemos "vec",
    y lo cortamos cuando tenemos "-".
    strcpy((*pasos)[*i].id_viaje,token); //
    Copiamos lo que hemos "cortado", en
    la cadena correspondiente de la
    estructura.
    token=strtok(NULL,"\n"); //Vamos
    leyendo hasta que haya un salto de
    linea.
    strcpy((*pasos)[*i].poblacion,token);

    (*i)++; //Vamos aumentando el contador,
    para saber el numero maximo de
    pasos que hay en la base de datos.
    }
}

fclose(fp);

printf("Se han cargado %i pasos.\n", *i);
}

//Prototipo: void leer_localidad(Estr_Localidad **, int
    *);
//Precondicion: Tener la estructura inicializada, con
    su contador.
//Postcondicion: Leer el fichero "localidades.txt" e
    introducir la informacion en su estructura,
    aumentando el contador cada vez que se encuentra uno
    .

void leer_localidad(Estr_Localidad **localidad, int *i)
{
    FILE *fp;
    char vec[30], *token;
    *i=0;

```

```

fp=fopen("DATA/localidades.txt", "r"); //Abrimos el
    fichero en modo lectura.

(*localidad)=malloc((*i)*sizeof(Estr_Localidad));
    //Asignamos un espacio de memoria a la
    estructura.
if ((*localidad)==NULL)
{
    printf("Error al asignar memoria.\n");
    exit(1);
}

if(fp==NULL)
{
    printf("No se ha podido abrir el fichero
        localidades.txt.\n");
    return;
}
else
{
    while(fgets(vec, 30, fp)) //Obtenemos la
        primera linea, y la introducimos en "vec".
    {
        if(strcmp(vec, "\n")!=0) //Cuando haya un
            salto de linea, pasamos a la siguiente.
        {
            *localidad=realloc(*localidad, ((*i)+1)*
                sizeof(Estr_Localidad)); //Vamos
                ampliando la memoria dinamica de la
                estructura.
            if (*localidad==NULL)
            {
                printf("Error al asignar memoria.\n
                    ");
                exit(1);
            }
            token=strtok(vec, "-"); //Leemos "vec",
                y lo cortamos cuando tenemos "-".
            strcpy((*localidad)[*i].siglas, token);
                //Copiamos lo que hemos "cortado",

```

```

        en la cadena correspondiente de la
        estructura.
token=strtok(NULL, "\n"); //Vamos
    leyendo hasta que haya un salto de
    linea.
strcpy((*localidad)[*i].localidad, token
);

    (*i)++; //Vamos aumentando el contador,
    para saber el numero maximo de
    localidades que hay en la base de
    datos.
    }
}
}

fclose(fp);

printf("Se han cargado %i localidades.\n", *i);
}

//Prototipo: void leer_ruta(Estr_Rutas ***, int *, int
*);
//Precondicion: Tener la estructura inicializada, con
sus contadores.
//Postcondicion: Leer el fichero "rutas.txt" e
introducir la informacion en su estructura,
aumentando el contador
//cada vez que se encuentra una ruta nueva, y otro que
nos indique el numero de localidades maximas que hay
en todas las rutas.

void leer_ruta(Estr_Rutas ***ruta, int *i, int *j)
{
    FILE *fp;
    char vec[60], *token;
    *i=0, *j=0;

    fp=fopen("DATA/rutas.txt", "r"); //Abrimos el
    fichero en modo lectura.

```

```

(*ruta)=malloc(((*i)*sizeof(Estr_Rutas))); //
    Asignamos un espacio de memoria a la estructura.
if ((*ruta)==NULL)
{
    printf("Error al asignar memoria.\n");
    exit(1);
}
(*ruta)[*i]=malloc(((*i)*sizeof(Estr_Rutas))); //
    Asignamos un espacio de memoria a la estructura,
    para hacerla una matriz.
if ((*ruta)[*i]==NULL)
{
    printf("Error al asignar memoria.\n");
    exit(1);
}

if (fp==NULL)
{
    printf("No se ha podido abrir el fichero rutas.
        txt.\n");
    return;
}
else
{
    while (fgets(vec, 60, fp)) //Obtenemos la
        primera linea, y la introducimos en "vec".
    {
        if (strcmp(vec, "\n") != 0) //Cuando haya un
            salto de linea, pasamos a la siguiente.
        {
            *ruta=(Estr_Rutas **)realloc(*ruta, (*i
                +1)*sizeof(Estr_Rutas *)); //Vamos
            ampliando la memoria dinamica de la
            estructura.
            if (*ruta==NULL)
            {
                printf("Error al asignar memoria.\n
                    ");
                exit(1);
            }
            (*ruta)[*i]=NULL;
        }
    }
}

```

```

int k=0;
token=strtok(vec, "-"); //Leemos "vec",
    y lo cortamos cuando tenemos "-".
while(token!=NULL) //Hasta que la
    cadena que recibe el corte, no es
    nula, no saltamos de linea.
{
    (*ruta)[*i]=(Estr_Rutas *)realloc
        ((*ruta)[*i],(k+1)*sizeof(
        Estr_Rutas)); //Vamos ampliando
        la memoria dinamica de la
        estructura, para anadir mas
        elementos en una fila.
    if(((*ruta)[*i])==NULL)
    {
        printf("Error al asignar
            memoria.\n");
        exit(1);
    }
    strcpy((*ruta)[*i][k].localidad,
        token); //Copiamos en la
        posicion de la matriz, la
        localidad que hemos cortado con
        strtok.
    k++;
    token=strtok(NULL, "-"); //Vamos
        leyendo hasta que en token no
        haya nada.
}
if((*j)<k) //Comparamos si el numero de
    localidades en la fila leida, es
    mayor a j.
{
    (*j)=k; //Guardamos la cantidad
        maxima de localidades que hay en
        una misma ruta/linea.
}
(*i)++; //Aumentamos en 1, para agregar
    una fila mas.

```

```

    }
}

fclose(fp);

printf("Se han cargado %i rutas , con %i ciudades _
      como maximo.\n", *i, *j);
}

//Prototipo: void leer_reservas(Estr_Reservas **, int
*);
//Precondicion: Tener la estructura inicializada , con
su contador.
//Postcondicion: Leer el fichero "reservas.txt" e
introducir la informacion en su estructura ,
aumentando el contador cada vez que se encuentra uno
.

void leer_reservas(Estr_Reservas **reservas , int *i)
{
    FILE *fp;
    char vec[30] , *token;
    *i=0;

    fp=fopen("DATA/reservas.txt" , "r"); //Abrimos el
      fichero en modo lectura.

    (*reservas)=malloc((*i)*sizeof(Estr_Reservas)); //
      Asignamos un espacio de memoria a la estructura.
    if ((*reservas)==NULL)
    {
        printf("Error al asignar memoria.\n");
        exit(1);
    }

    if(fp==NULL)
    {
        printf("No se ha podido abrir el fichero _
      reservas.txt.\n");
    }
}

```

```

        return;
    }
    else
    {
        while(fgets(vec, 30, fp)) //Obtenemos la
            primera linea, y la introducimos en "vec".
        {
            if(strcmp(vec, "\n")!=0) //Cuando haya un
                salto de linea, pasamos a la siguiente.
            {
                *reservas=realloc(*reservas, ((*i)+1)*
                    sizeof(Estr_Reservas)); //Vamos
                    ampliando la memoria dinamica de la
                    estructura.
                if (*reservas==NULL)
                {
                    printf("Error al asignar memoria.\n
                        ");
                    exit(1);
                }
                token=strtok(vec, "-"); //Leemos "vec",
                    y lo cortamos cuando tenemos "-".
                strcpy((*reservas)[*i].id_viaje, token);
                    //Copiamos lo que hemos "cortado",
                    en la cadena correspondiente de la
                    estructura.
                token=strtok(NULL, "\n"); //Vamos
                    leyendo hasta que haya un salto de
                    linea.
                strcpy((*reservas)[*i].id_usuario, token
                    );

                (*i)++; //Vamos aumentando el contador,
                    para saber el numero maximo de
                    reservas que hay en la base de datos
                    .
            }
        }
    }

fclose(fp);

```



```
        printf("Se han cargado %i reservas.\n", *i);  
    }
```

3.2.9. Listar

3.2.10. Menús

3.2.11. Modificar

3.2.12. Mostrar Reservar

3.2.13. Preguntar

3.2.14. Reservar

3.3. Plan de prueba

Debe describir todo el plan de prueba que se ha llevado a cabo. Se distinguen las pruebas realizadas a cada módulo y las pruebas de integración de los distintos módulos probándolo como un todo. Por último, también se debe incluir la descripción del plan de pruebas de aceptación.

3.3.1. Prueba de los módulos

Incluir toda la batería de pruebas realizadas. Pruebas de caja blanca y pruebas de caja negra.

3.3.2. Prueba de integración

Incluir toda la batería de pruebas realizadas. Pruebas de caja blanca y pruebas de caja negra.

3.3.3. Plan de pruebas de aceptación

Estas pruebas deben diseñarse con el usuario del sistema. Deben describir las pruebas que son necesarias pasar para que el sistema sea aceptado por el usuario final.

Si en algún punto del documento se quiere hacer referencia a algún documento es necesario incluir la cita donde corresponda, podemos tomar como ejemplo ?.

4. Documentación del código fuente

La documentación obtenida mediante el programa Doxygen, está aquí.

```

void verificar_viaje(Estr_Viaje *viaje, int numViajes, char *id, char *mat, char *fecha, char *hor_i, char *hor_f, int *rp)
{
    int i, brkp=0; (1)

    for(i=0; i<numViajes && brkp==0; i++) (2)
    {
        if(strcmp(mat, viaje[i].id_mat)==0) (3)
        {
            if(strcmp(fecha, viaje[i].f_inic)==0) (4)
            {
                if(strcmp(viaje[i].estado, "finalizado")!=0 && strcmp(viaje[i].estado, "anulado")!=0) (5)
                {
                    (*rp)=1; (6)
                    brkp=1; (7)
                } (8)
            } (9)
        } (10)
    } (11)
}

```

Complejidad ciclomática (cualquiera de las 3):

$$V(G) = NA - NN + 2 =$$

$$14 - 11 + 2 = 3 + 2 = 5$$

$$V(G) = NNP + 1 = 4 + 1 = 5$$

$$V(G) = \text{número de regiones (R1+R2+R3+R4+R5)} = 5$$

Rutas básicas linealmente independientes:

Ruta 1: 1 - 2 - 11

Ruta 2: 1 - 2 - 3 - 10 - 2 - 11

Ruta 3: 1 - 2 - 3 - 4 - 9 - 10 - 2 - 11

Ruta 4: 1 - 2 - 3 - 4 - 5 - 8 - 9 - 10 - 2 - 11

Ruta 5: 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 2 - 11

