



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	3
2. Documentación de usuario	4
2.1. Descripción funcional	4
2.2. Tecnología	8
2.3. Manual de instalación	8
2.4. Acceso al sistema	9
2.5. Manual de referencia	10
3. Documentación de sistema	12
3.1. Especificación del sistema	12
3.1.1. División de los módulos	12
3.1.2. Relaciones de los módulos	13
3.1.3. Plan de desarrollo del Software	14
3.2. Módulos	14
3.2.1. Acceso	14
3.2.2. Actualizar	15
3.2.3. Buscar	15
3.2.4. Colores	15
3.2.5. Eliminar	15
3.2.6. Encontrar	16
3.2.7. Escribir	16
3.2.8. Estructuras	16
3.2.9. Fecha	17
3.2.10. Leer	17
3.2.11. Listar	17
3.2.12. Menús	17
3.2.13. Modificar	18
3.2.14. Preguntar	18
3.3. Plan de prueba	18
3.3.1. Prueba de los módulos	18

3.3.2.	Prueba de integración	23
3.3.3.	Plan de pruebas de aceptación	29
4.	Documentación del código fuente	35
4.1.	Código de los módulos	35
4.1.1.	Acceso	35
4.1.2.	Actualizar	39
4.1.3.	Buscar	41
4.1.4.	Colores	48
4.1.5.	Eliminar	49
4.1.6.	Encontrar	78
4.1.7.	Escribir	91
4.1.8.	Estructuras	118
4.1.9.	Fecha	120
4.1.10.	Leer	132
4.1.11.	Listar	147
4.1.12.	Menús	163
4.1.13.	Modificar	188
4.1.14.	Preguntar	223

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 significante 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 contrasena.
(5)Volver.
```

- Reservar y cancelar viajes.

```
LISTADO DE SUS RESERVAS:  
RESERVA 1:  
ID del viaje: 000013  
Fecha de partida: 06/07/2023  
Hora de partida: 08:00  
Hora de llegada: 09:00  
Tipo: ida  
Precio: 4 euros  
  
¿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.

```
LISTADO DE SUS VEHICULOS:  
VEHICULO 1:  
    Matricula: 1111FER  
    Numero de plazas: 6  
    Descripcion: HOLA  
  
¿Que quiere hacer?  
(1)Alta de vehiculo.  
(2)Modificar vehiculo.  
(3)Eliminar vehiculo.  
(4)Volver.
```

- Crear, modificar, anular y finalizar viajes.

```
LISTADO DE SUS VIAJES:  
VIAJE 1:  
    ID del viaje: 000013  
    Estado: abierto  
    Plazas libres: 4  
    Fecha de partida: 06/07/2023  
    Hora de partida: 08:00  
    Hora de llegada: 09:00  
    Tipo: ida  
    Precio: 4 euros  
  
¿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.

```
Hola Juan Perez (Administrador)
¿Que quiere hacer?
(1)Alta de vehiculo.
(2)Baja de vehiculo.
(3)Modificar vehiculo.
(4>Listar vehiculos.
(5)Mostrar lista de viajes de un vehiculo.
(6)Volver.
```

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

```
Hola Juan Perez (Administrador)
¿Que quiere hacer?
(1)Crear viaje.
(2>Anular/Finalizar viaje.
(3)Eliminar viaje.
(4)Modificar viaje.
(5>Listar viajes.
(6)Volver.
```

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 Buscar 4.1.3, hemos usado memoria dinámica en gran parte del proyecto, para reducir la carga del mismo, y hacer que todo tipo de ordenadores puedan ejecutarlo. Además, hemos utilizado la función atoi, para hacer que no se tenga que introducir la ID del usuario entera, por ejemplo, "0004", con poner "4", ya funciona. También se ha hecho uso de ficheros temporales, para eliminar/modificar, como se puede ver en los módulos Eliminar 4.1.5 y Modificar 4.1.13.

2.3. Manual de instalación

Para usar el programa, sólo tiene que dirigirse a la carpeta SETUP, donde encontrará un acceso directo, llamado ESI-SHARE, que tiene como ruta, el archivo ejecutable del programa. Otra forma sería instalar Code::Blocks, y ejecutar el archivo de proyecto ESI-SHARE.cbp.

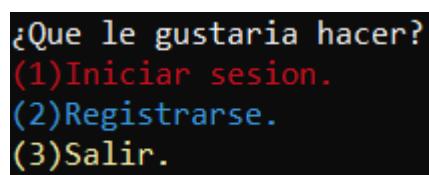
- Requisitos mínimos del sistema:
 - Procesador: Intel Core i3 ó AMD Athlon II
 - Memoria RAM: 2GB
 - Espacio en disco duro: 100MB
- Es recomendable usar cualquier versión de Windows, para que no haya conflicto a la hora de usar algunas funciones.

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:

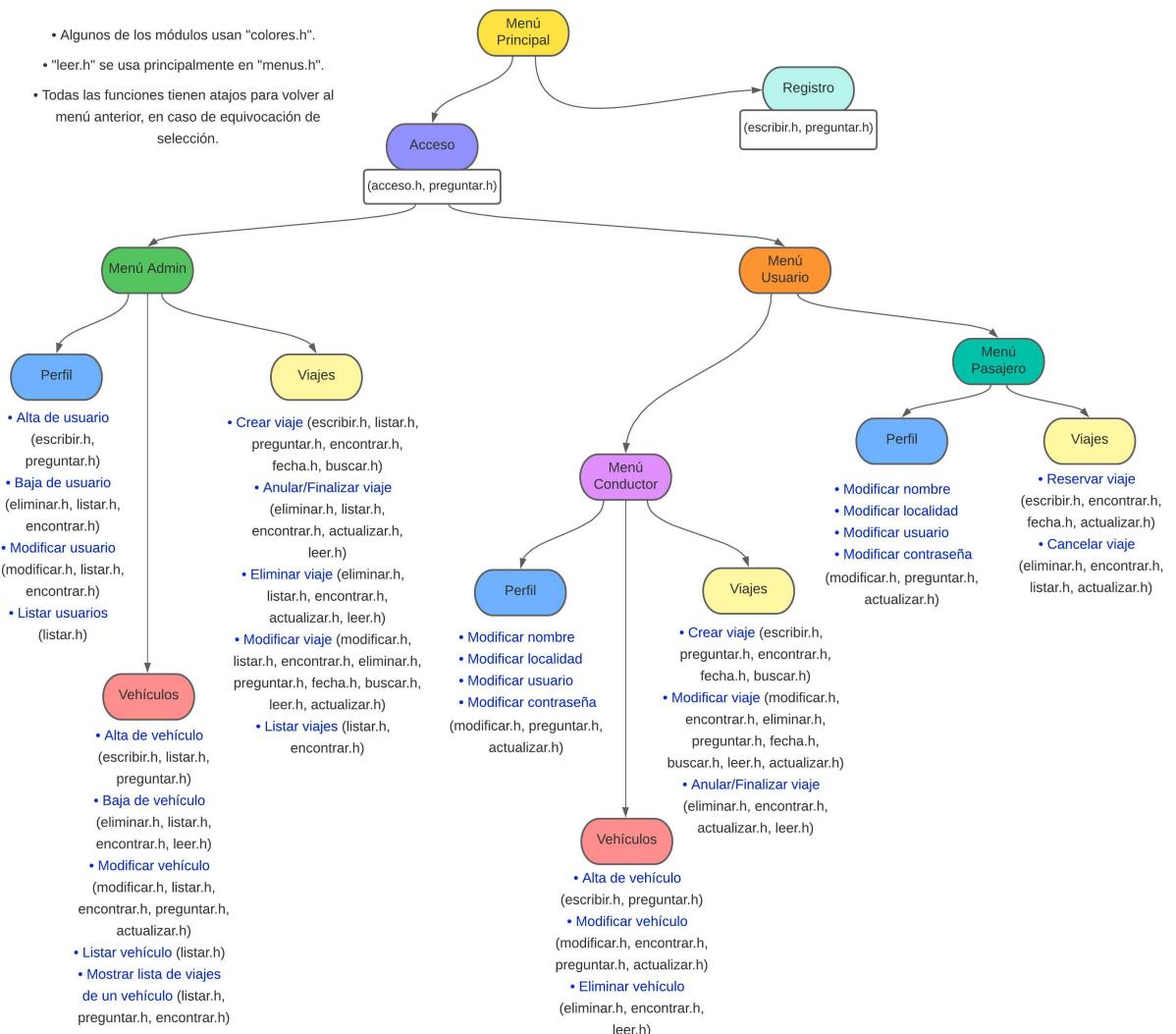
- Entrar en el menú de **Usuarios**, donde puede crear un usuario desde cero, eliminar cualquier usuario del sistema, modificar los datos personales de cualquiera, e incluso obtener una lista con todos los usuarios que hay en el sistema.
- Pasar al menú de **Vehículos**, para dar de alta nuevos vehículos, 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

3.1.1. División de los módulos

Aquí, se puede ver en cuántos subproblemas, hemos dividido el programa, y los módulos que van asociados a cada uno de ellos. Las especificaciones de cada una de las funciones se puede encontrar en Doxygen o en el apartado 4.1.



Puede ver la imagen completa en Organigrama Proyecto.

3.1.2. Relaciones de los módulos

Aquí, puede ver la organización de la descomposición modular del proyecto, es decir, la forma en la que están emparejados los módulos, para hacer que el programa funcione correctamente, y de la forma más óptima posible.



Puede ver la imagen completa en Descomposición Modular.

3.1.3. Plan de desarrollo del Software

Fase	Descripción	Duración
Análisis	Se dividen todos los problemas en subproblemas, para planificar todos los requisitos que necesita cada función para que cumpla los objetivos.	1 semana
Desarrollo	Creamos todas las funciones, para que el programa haga lo que queramos.	3 semanas
Optimización	Tratamos de ver si hay partes del código que se puedan sustituir por funciones ya existentes, para reusar código, y hacer que el programa sea más liviano.	1 semana
Pruebas	Testeamos cada función y módulo por separado, para más tarde probarlo como un todo, y checkear si el proyecto es funcional, si no es así, hay que arreglar los errores.	2 semanas
Documentación	Una vez que se ha realizado todo, se van incluyendo las especificaciones de las funciones, al igual que se agregan comentarios al código.	1 semana

Tras haber pasado todas estas fases, el programa está totalmente preparado para ser distribuido a los usuarios, en este caso, los miembros de la comunidad universitaria de la ESI, es por esto que se pasa a una fase de Mantenimiento y Actualización, en la que se arreglan los errores que pueda tener el programa en las máquinas externas, al igual que se van incluyendo actualizaciones para hacer que el programa tenga más funcionalidades, como por ejemplo, una creación de un sistema de valoraciones, a la hora de acabar un viaje.

3.2. Módulos

En este proyecto, se ha empleado una descomposición por tareas, por lo que el programa se ha dividido en 14 módulos, que son:

3.2.1. Acceso

Usado 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 se dispone 3 intentos para introducir la contraseña.

Para ver el código fuente del módulo puede ir a 4.1.1, o entrar en Doxygen.

3.2.2. Actualizar

Sirve para reescribir todos los datos de la estructura en el fichero, y así sincronizarlos, por si se ha producido alguna modificación.

Para ver el código fuente del módulo puede ir a 4.1.2, o entrar en Doxygen.

3.2.3. Buscar

Empleado para buscar todas las rutas posibles, que hay desde la ciudad que el usuario haya seleccionado hasta la ESI, para hacer que no se repitan todas las rutas, se usa otra matriz, donde se introduce la ruta impresa, para comprobar que no se haya escrito en pantalla anteriormente, y sólo listar las rutas no repetidas, y así ahorrar tiempo al usuario a la hora de crear un viaje. Una vez que el usuario selecciona la ruta, se escriben todos los pasos en el fichero correspondiente.

Para ver el código fuente del módulo puede ir a 4.1.3, o entrar en Doxygen.

3.2.4. Colores

Gracias a este módulo se puede cambiar el fondo y el cuerpo del texto, dándole así algo de formato al proyecto.

Para ver el código fuente del módulo puede ir a 4.1.4, o entrar en Doxygen.

3.2.5. Eliminar

Utilizado para suprimir cualquier vehículo, viaje, paso o reserva. Cuando se le pregunta al usuario, qué vehículo quiere quitar de todos los que tiene, y él selecciona uno, se eliminan todos sus viajes, pasos y reservas. Algo similar

ocurre con los viajes, que se eliminan sus pasos y reservas.

Para ver el código fuente del módulo puede ir a 4.1.5, o entrar en Doxygen.

3.2.6. Encontrar

Se ha usado para buscar todos los vehículos, viajes o reservas que tiene un usuario. Con esto podemos saber cuántos viajes tiene con un cierto estado o si se han ocupado plazas en el viaje, para así darle la posibilidad al usuario de anular, finalizar o modificar un viaje. Todo esto, se devuelve en un vector de enteros dinámico, que indica la posición del vehículo, viaje o reserva en su respectiva estructura.

Para ver el código fuente del módulo puede ir a 4.1.6, o entrar en Doxygen.

3.2.7. Escribir

Sirve para registrar usuarios, vehículos, viajes o reservas. Se han hecho algunas comprobaciones para que cuando se cree un usuario, no se pueda repetir de usuario, o a la hora de registrar un vehículo, que se vea si la matrícula está bien introducida, con sus 4 números y 3 letras, y que no esté registrada en el sistema. Además, en la parte de reservar un viaje, sólo se podrá hacer si pasa algún viaje por la localidad de residencia, en la fecha establecida por el usuario, si hay viajes pero en otra fecha, se le avisará al usuario, si reserva, se eliminará una plaza del viaje.

Se ha creado una característica interesante, que es la reutilización de IDs, cuando se quiere crear un usuario, o un viaje, con esto si se elimina un usuario, se podrá volver a crear otro con esa misma ID, ya que al suprimir un usuario, se quita todo el contenido que haya sobre el mismo en todos los ficheros.

Para ver el código fuente del módulo puede ir a 4.1.7, o entrar en Doxygen.

3.2.8. Estructuras

En este módulo, se han definido las estructuras necesarias para poner en marcha el proyecto.

Para ver el código fuente del módulo puede ir a 4.1.8, o entrar en Doxygen.

3.2.9. Fecha

Gracias a este módulo se puede pedir una fecha, comprobando si la fecha y la hora es posterior a la actual, y si al introducir una hora de partida, esta hora es posterior a la de llegada. Asimismo, se ha creado una función que compruebe todos los viajes que hay, y si alguno ha pasado la hora de inicio, se pone en estado "Iniciado", mientras que si se ha excedido una hora desde la hora de llegada, se establecerá en estado "Finalizado", y se eliminarán sus pasos y reservas.

Para ver el código fuente del módulo puede ir a 4.1.9, o entrar en Doxygen.

3.2.10. Leer

Usado para leer todos los ficheros que hay en la carpeta DATA e introducir la información escaneada en las estructuras creadas. Todo esto se puede hacer gracias a la función strtok, que va rompiendo cada línea del fichero, hasta donde esté el carácter que queramos, en nuestro caso ". También, se ha creado un contador, para usarlo como delimitador en los bucles de otras funciones.

Para ver el código fuente del módulo puede ir a 4.1.10, o entrar en Doxygen.

3.2.11. Listar

Creado con el objetivo de imprimir por pantalla listas de todos los usuarios, vehículos ó viajes que hay en el sistema. También se ha creado para listar todas las localidades que hay en la provincia de Cádiz, al igual que puede escribir todos los vehículos, viajes o reservas que tiene un usuario.

Para ver el código fuente del módulo puede ir a 4.1.11, o entrar en Doxygen.

3.2.12. Menús

En este módulo están todas las funciones que muestran por pantalla los menús, haciendo de puente entre las diferentes funciones de los demás módulos. Se han creado varias interfaces, una para los Pasajeros, otra para los Conductores, y una tercera para el Administrador.

Para ver el código fuente del módulo puede ir a 4.1.12, o entrar en Doxygen.

3.2.13. Modificar

Sirve para cambiar cualquier dato de un vehículo o viaje de un usuario, e incluso para rectificar los datos personales del usuario, o actualizar la contraseña.

Para ver el código fuente del módulo puede ir a 4.1.13, o entrar en Doxygen.

3.2.14. Preguntar

Utilizado para escanear cadenas, cambiando el carácter de salto de línea por el carácter nulo. Además se ha usado para preguntar una contraseña, empleando * para ocultarla, y haciendo que si la contraseña es nula, se vuelva a preguntar. Asimismo, se usa para escanear localidades, o para comprobar si una matrícula, existe en el sistema.

Para ver el código fuente del módulo puede ir a 4.1.14, o entrar en Doxygen.

3.3. Plan de prueba

3.3.1. Prueba de los módulos

Todos los módulos han sido probados a fondo, introduciendo valores aleatorios, poco a poco se han ido eliminando errores, haciendo que el programa sea infalible, y no crashee al entrar en alguna función, como pasaba anteriormente con la aplicación en fase de desarrollo.

- Acceso:

- Usuario: hola - “Usuario no encontrado en nuestra base de datos.”
- Usuario: usua1, Contraseña: - “La contraseña debe tener entre 1 y 8 caracteres.”
- Usuario: usua1, Contraseña: 1111 - “La contraseña introducida es incorrecta.” Vuelve a pedir contraseña, hasta llegar a los 3 intentos.
- Usuario: admin, Contraseña: 1234 - Accede al Menú de Administrador.

- Usuario: usua1, Contraseña: 2023 - Accede al Menú de Usuario.

- altaUsuario:

- Nombre: Manolo, Localidad: JER, Usuario: manol, Contraseña: - “La contraseña debe tener entre 1 y 8 caracteres.”
- Nombre: Manolo, Localidad: JER, Usuario: usual1, Contraseña: 1234 - “El usuario ya está siendo usado.”
- Nombre: Manolo, Localidad: JER, Usuario: u4323, Contraseña: 1234 - “El usuario ha sido agregado correctamente.”

- Acceso:

- Nombre: Federico, - “Su nombre completo se ha actualizado correctamente.”

- Acceso:

- Localidad: SAN - “Has seleccionado Sanlucar. Su localidad de residencia se ha actualizado correctamente.”

- Acceso:

- Usuario: admin - “El nombre de usuario ya está siendo usado.”
- Usuario: u1234 - “El nombre de usuario es válido. Su nombre de usuario se ha actualizado correctamente.”

- Acceso:

- Contraseña antigua: - “La contraseña debe tener entre 1 y 8 caracteres.”
- Contraseña antigua: 1111 - “La contraseña introducida es incorrecta.” Vuelve a pedir contraseña, hasta llegar a los 3 intentos.
- Contraseña antigua: 2023, Contraseña nueva: - “La contraseña debe tener entre 1 y 8 caracteres.”
- Contraseña antigua: 2023, Contraseña nueva: 4567 - “Su contraseña se ha actualizado correctamente.”

- Acceso:

- Día: 10, Mes: 3, Año: 2000 - “El año debe ser posterior a 2022.”
- Día: 10, Mes: 1, Año: 2023 - “La fecha ingresada es anterior a la fecha actual.”

- Día: 20, Mes: 6, Año: 2023 - “No hay viajes en tu localidad para fechas futuras.”
- Día: 20, Mes: 6, Año: 2023 - “Hay viajes en tu localidad, pero para fechas futuras.”
- Día: 20, Mes: 6, Año: 2023 - “Si hay viajes en tu localidad para ese día se mostrarán.”

■ Acceso:

Si tienes reservas, se mostrarán todas, y nos preguntará qué reserva queremos cancelar.

■ Acceso:

- Matrícula: XXX12 - “La matrícula debe tener 7 caracteres.”
- Matrícula: XXX1234 - “Los 4 primeros caracteres de la matrícula deben ser números.”
- Matrícula: XXX1234 - “Los 3 últimos caracteres de la matrícula tienen que ser letras mayúsculas.”
- Matrícula: 1234XXX - “La matrícula ya está registrada.”
- Matrícula: 7865JTR, Nº de plazas: -1 - “Introduzca un número de plazas entre 1 y 9.”
- Matrícula: 7865JTR, Nº de plazas: 5, Descripción: PE - “El vehículo ha sido agregado correctamente.”

■ Acceso:

Si tienes algún vehículo creado, hay que seleccionar el vehículo que queremos cambiar:

- OP: 1, Matrícula: XXX12 - “La matrícula debe tener 7 caracteres.”
- OP: 1, Matrícula: XXX1234 - “Los 4 primeros caracteres de la matrícula deben ser números.”
- OP: 1, Matrícula: XXX1234 - “Los 3 últimos caracteres de la matrícula tienen que ser letras mayúsculas.”
- OP: 1, Matrícula: 1234XXX - “La matrícula ya está registrada.”
- OP: 1, Matrícula: 7841DEW - “La matrícula se ha cambiado correctamente.”
- OP: 2, Plazas: -1 - “Introduzca un número de plazas entre 1 y 9.”

- OP: 2, Plazas: 7 - “El número de plazas del vehículo se ha actualizado correctamente.”

- OP: 3, Descripción: Seat Panda - “La descripción del vehículo se ha actualizado correctamente.”

- Acceso:

Si tienes vehículos, se mostrarán todos, y nos preguntará qué vehículo queremos cancelar.

- Acceso:

Si tienes vehículos registrados:

- Matrícula: 1234XXX, Día: 10, Mes: 3, Año: 2000 - El año debe ser posterior a 2023.
- Matrícula: 1234XXX, Día: 10, Mes: 1, Año: 2023 - “La fecha ingresada es anterior a la fecha actual.”
- Matrícula: 1234XXX, Día: 20, Mes: 6, Año: 2023, Ida/Vuelta: 1, Precio: -1 - “El precio debe estar entre 0 y 10.”
- Matrícula: 1234XXX, Día: 20, Mes: 6, Año: 2023, Ida/Vuelta: 1, Precio: 3 /Ahora se verifica si el vehículo no tiene un viaje creado en ese día, si no tiene ninguno te pide la ruta/, Ciudad de partida/Llegada: SAN, /Hay que seleccionar la ruta que queramos/.

- Acceso:

Si tienes algún viaje creado que no haya sido ocupado, hay que seleccionar el viaje que queremos cambiar:

- OP: 1, Seleccionar una matrícula de la lista de vehículos que tiene el usuario.
- OP: 2, Introducir una fecha en la que el vehículo esté sin ningún viaje, ya que sólo se puede tener un viaje abierto por día.
- OP: 3, El coste debe estar entre 0 y 9.
- OP: 4, Escoger una de las rutas, desde la ciudad que hayamos seleccionado.

- Acceso:

Si tienes un viaje creado sin plazas ocupadas, puedes anularlo, y si tienes un viaje iniciado, puedes finalizarlo.

Seleccionas el número que sale en la lista, del viaje que quiere finalizar/anular.

- Acceso:
Se muestra una lista de todos los usuarios del sistema, y se le pide la ID del usuario que quiere eliminar. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios del sistema, y se le pide la ID del usuario que quiere modificar. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios del sistema.
- Acceso:
Se muestra una lista de todos los usuarios del sistema, y se le pide la ID del usuario con el que quiere crear un vehículo. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios, con sus vehículos, del sistema, y se le pide la ID del usuario al que quiere eliminar el vehículo. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios, con sus vehículos, del sistema, y se le pide la ID del usuario al que quiere modificar el vehículo. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios, con sus vehículos, del sistema.
- Acceso:
Se muestra una lista de todos los usuarios, con sus vehículos, del sistema, y se le pide la matrícula del vehículo, para ver el historial de viajes que tiene.
- Acceso:
Se muestra una lista de todos los usuarios del sistema, y se le pide la ID del usuario del que quiere crear un viaje. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios, con sus viajes, del sistema, y se le pide la ID del usuario del que quiere finalizar un viaje. Se puede escribir 0002 o 2.

- Acceso:
Se muestra una lista de todos los usuarios, con sus viajes, del sistema, y se le pide la ID del usuario del que quiere eliminar un viaje. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios, con sus viajes, del sistema, y se le pide la ID del usuario del que quiere modificar un viaje. Se puede escribir 0002 o 2.
- Acceso:
Se muestra una lista de todos los usuarios, con sus viajes, del sistema.

3.3.2. Prueba de integración

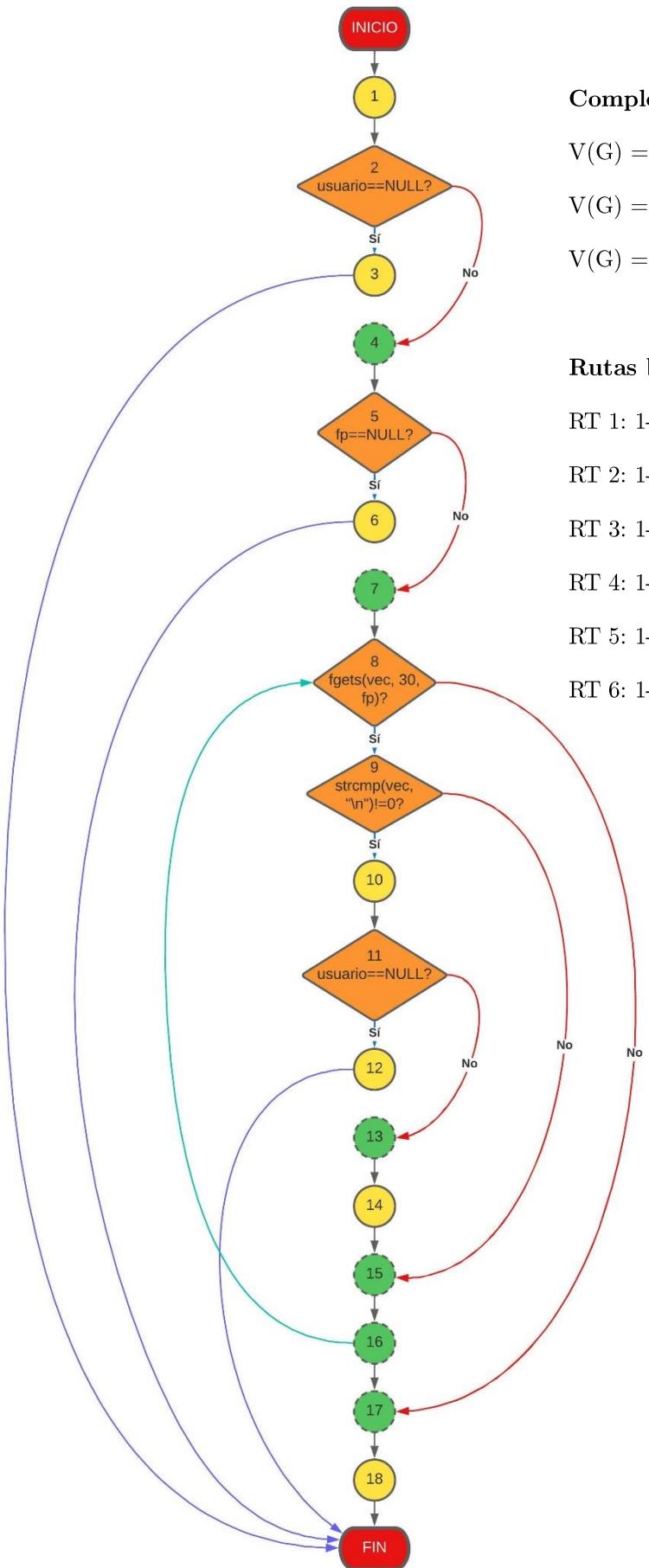
Ahora se exponen las pruebas de caja blanca y negra que hemos realizado cada miembro del equipo, a una de las funciones del programa.

Alejandro Durán Obregón

```
void leer_usuario(Estr_Usuario **usuario, int *i)
{
    FILE *fp;
    char vec[80], *token;
    *i=0;
1   fp=fopen("DATA/usuarios.txt", "r");
    (*usuario)=malloc((*i)*sizeof(Estr_Usuario));
2   if ((*usuario)==NULL)
    {
        3 printf("Error al asignar memoria.\n");
        exit(1);
4   }

5   if(fp==NULL)
    {
        6 printf("No se ha podido abrir el fichero usuarios.txt.\n");
7   }
    else
    {
        8 while(fgets(vec, 80, fp))
        {
            9 if(strcmp(vec, "\n")!=0)
            {
                10 *usuario=realloc(*usuario, ((*i)+1)*sizeof(Estr_Usuario));
                11 if (*usuario==NULL)
                    {
                        12 printf("Error al asignar memoria.\n");
                        exit(1);
13                    }
                token=strtok(vec, "-");
                strcpy((*usuario)[*i].ia_usuario, token);
                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");
                strcpy((*usuario)[*i].contrasena, token);

                (*i)++;
14            }
15        }
16    }
17}
18 fclose(fp);
printf("Se han cargado %i usuarios.\n", *i);
}
```



Complejidad ciclomática

$$V(G) = NA - NN + 2 = 22 - 18 + 2 = 6$$

$$V(G) = NNP + 1 = 5 + 1 = 6$$

$$V(G) = NR = 6$$

Rutas básicas linealmente independientes:

RT 1: 1-2-3-4

RT 2: 1-2-4-5-6-7-18

RT 3: 1-2-4-5-7-8-17-18

RT 4: 1-2-4-5-7-8-9-10-11-12-13

RT 5: 1-2-4-5-7-8-9-15-16-8-17-18

RT 6: 1-2-4-5-7-8-9-10-11-13-14-15-16-8-17-18

Pruebas de caja negra

Cabecera: void leer_usuario(Estr_Usuario **, int *);

Precondición: Tener la estructura inicializada, con su contador.

Postcondición: Leer el fichero "usuarios.txt" e introducir la información en su estructura, aumentando el contador cada vez que se encuentra uno.

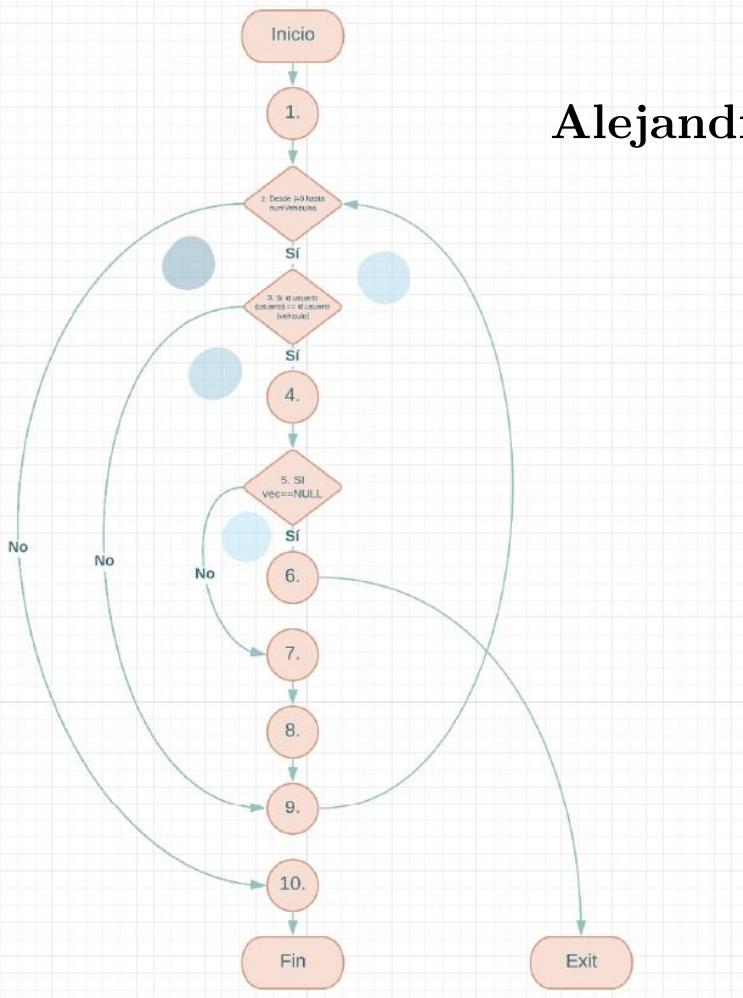
- **Fallo de asignación de la memoria dinámica:** Si la memoria dinámica no se puede asignar correctamente, el programa se cerrará.

- **Fichero no encontrado:** Si el fichero "usuarios.txt" no está en la carpeta designada, es decir, el programa no lo puede abrir, entonces no se podrá completar el proceso de lectura del fichero, para después introducir los datos en la estructura.

- **Apertura de fichero (sin contenido):** Si se ha podido abrir correctamente el fichero, pero no hay ningún dato/línea de contenido en el fichero, pues no se escribe nada en la estructura, al no haber nada en su respectivo fichero. Acaba escribiendo que hay 0 usuarios en el sistema.

- **Apertura de fichero (con contenido):** Si se abre el fichero y hay contenido en el fichero, la función introducirá en cada variable de la estructura, lo que haya detectado de cada línea de información del fichero, acabará cuando no pase la última línea. Finalmente, notificará cuántos usuarios ha leído.

Alejandro Ruiz del Pino



```

void encontrarVehiculos(Estr_Usuario *usuario, Estr_Vehiculo *vehiculo, int numVehiculos, int **vec, int *x, int i)
{
    int j, h;
    *x=0; ①

    for(j=0; j<numVehiculos; j++) ② Nos desplazamos por toda la estructura "vehiculo".
    {
        if(strcmp(usuario[i].id_usuario③, vehiculo[j].id_usuario)==0) //Si la id del usuario que hemos introducido es igual a la id de
        {
            *vec=(int *)realloc(*vec, (*x)+1)*sizeof(int)); ④ Asignamos un espacio de memoria más, para introducir el vehículo detectado
            if ((*vec)==NULL) ⑤
                printf("Error al asignar memoria.\n");
                exit(1);
            (*vec)[*x]=j; //Se introduce la posición del vehículo encontrado en la estructura en el vector de enteros dinámico.
            (*x)++; //Vamos aumentando en 1 la cantidad de vehículos que tiene el usuario, que al mismo tiempo servirá para acceder
        }
    } ⑥
}
    
```

$$V_{cg1} = N_a - N_{a+2} = 13 - 10 + 2 = 5 \quad \bullet V(g) = N_{ad} + 1 = 3 + 1 = 4$$

$$\bullet V(g) = N_{reg} = 4$$

1. 1, 2, 10

2. 1, 2, 3, 9, 2, 10

3. 1, 2, 3, 4, 5, 6, exit

4. 1, 2, 3, 4, 5, 7, 8, 9, 2, 10

Pruebas de caja negra

Cabecera: void encontrarVehiculos(Estr_Usuario *usuario, Estr_Vehiculo *vehiculo, int numVehiculos, int **vec, int *x, int i);

Precondición: usuario, vehiculo, vec y x cargados e inicializados

Postcondición: Encuentra los vehículos que posee un usuario dado.

Este proceso se completa una vez que se han comparado los dueños de cada vehículo registrado con la ID del usuario que se deseé. Si uno de estos coches tiene una ID de vehículo que coincide con la del usuario se tiene en consideración para poder ser nombrado o listado en otra función. Si no encuentra ninguna conexión, el usuario no posee vehículos registrados.

En cuanto a las 4 rutas que se pueden tomar en esta función:

- **1:** Comienza inicializando las variables. Luego entra en el for, el cual repetirá el bucle hasta que j llegue a tener el mismo valor o superior a numVehiculos. En este caso, j si es igual o superior a numVehiculos por lo que salta al final de la función donde finaliza.
- **2:** Volvemos a inicializar las variables, pero entrando dentro del for esta vez. Pasamos por un if que compara si el registro de usuario id.usuario coincide con el registro vehiculo id.usuario. En este caso no son iguales por lo que salta al fin del si, regresando al for inicial y más adelante finalizando el programa pues este for se completa.
- **3:** En este caso, tras inicializar las variables, pasar por el for y llegar al if que compara ambos registros si se da que ambos son iguales, por lo que al entrar dentro de este if y realizar las operaciones que se encontraban dentro llegan a otro if que observa si vec no es nulo. En este caso sí lo es, entrando dentro donde se encuentra un exit(1) que finaliza la ejecución de esta función.
- **4:** En este último caso se repiten los mismos pasos que en el anterior camino. Sin embargo, esta vez en el if donde comprueba si vec es nulo se comprueba que no lo es, saltando al siguiente nodo hasta llegar al noveno donde vuelve al for del principio, del cual permite salir y finalizar la función.

3.3.3. Plan de pruebas de aceptación

En esta sección, se va a definir algunas rutas con las que usted podrá probar todos los puntos del programa. No se ha establecido una serie de datos específicos, para que así pueda probar con todo lo que le guste, para poner a prueba el programa.

Primeramente, se encontrará el menú principal, en el que podrá acceder con las credenciales sugeridas en 2.4, o puede registrar un nuevo usuario.

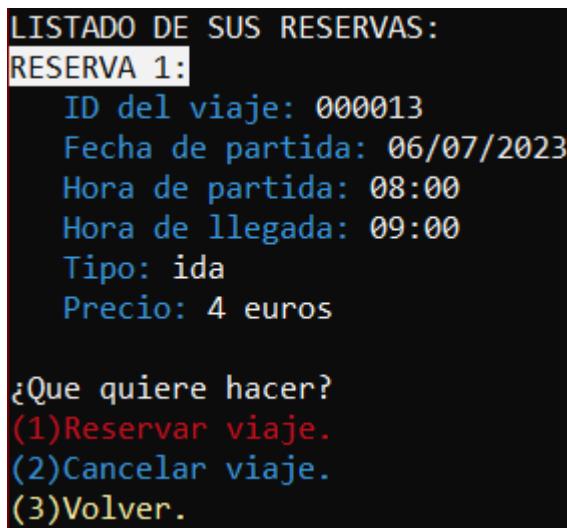
```
¿Que le gustaria hacer?  
(1)Iniciar sesion.  
(2)Registrarse.  
(3)Salir.
```

Empezaremos con el apartado de **Usuario**, introduciendo **usuau1**, como usuario, y **2023**, como contraseña. Tras esto, se dirigirá al menú de **Pasajero**, introduciendo 1, podrá entrar en la parte de **Perfil**, en el que puede ver todos sus datos personales, aquí puede modificar cualquier tipo de dato personal, verá que estos se actualizarán tanto en el fichero como en la estructura. Puede probar si estos se introducen correctamente, etc.

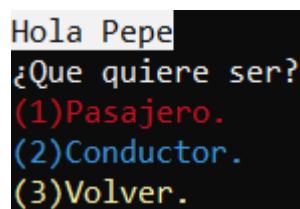
```
Hola Pepe  
¿Que quiere ser?  
(1)Pasajero.  
(2)Conductor.  
(3)Volver.
```

```
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.
```

Después, pulsará 2, para acceder al apartado de **Viajes**, donde podrá ver todos los viajes que tiene reservados, tras esto, puede probar las acciones del menú, para poner a prueba el programa. Puede ver que el usuario sólo puede reservar viajes que pasan por su localidad de residencia, al igual que no puede reservar dos veces en el mismo viaje, o que no le deja introducir una fecha anterior a la actual, etc.



Habiendo probado que todo funciona, puede volver al menú de selección de puesto, para escoger la opción de **Conductor**, introduciendo 1, entrará en el menú de **Perfil**, similar al anterior.



```
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.
```

Al haber probado esto, puede pulsar 2, para dirigirse al menú de **Vehículos**, donde tendrá a su vista un listado con todos los vehículos, que tiene registrados en el sistema, si lo desea, puede explorar los menús y realizar diferentes acciones para probar el sistema, creando nuevos vehículos, o modificando y eliminando los existentes. Aquí puede ver que al eliminar un vehículo, se eliminarán tanto sus viajes, reservas y pasos.

```
LISTADO DE SUS VEHICULOS:
VEHICULO 1:
    Matricula: 1111FER
    Numero de plazas: 6
    Descripcion: HOLA

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

Tras esto, puede dirigirse al menú de **Viajes**, introduciendo un 3, donde verá una lista de todos los viajes abiertos o iniciados que tiene en el momento, puede crear algún viaje, para probar el sistema de creación de los mismos, o incluso modificar o anular o eliminar cualquiera. Sólo podrá anular viajes que estén abiertos, sin ninguna plaza ocupada, o finalizar, viajes que ya estén iniciados.

LISTADO DE SUS VIAJES:

VIAJE 1:

ID del viaje: 000013
Estado: abierto
Plazas libres: 4
Fecha de partida: 06/07/2023
Hora de partida: 08:00
Hora de llegada: 09:00
Tipo: ida
Precio: 4 euros

¿Que quiere hacer?

- (1)Crear viaje.
- (2)Modificar viaje.
- (3)Anular/Finalizar viaje.
- (4)Volver.

En segundo lugar, puede volver al menú principal, e introducir **admin**, como usuario, y **1234**, como contraseña.

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

Tras esto, aparecerá un menú con 3 apartados, en **Usuarios**, podrá realizar diferentes funciones, como la creación de un usuario nuevo, o la eliminación de cualquier usuario existente, al igual que puede modificarlos, y ver todos los usuarios del sistema, estas funciones trabajan de forma similar a las del usuario normal.

```
Hola Juan Perez (Administrador)  
¿Que quiere ver?  
(1)Usuarios.  
(2)Vehiculos.  
(3)Viajes.  
(4)Volver.
```

```
Hola Juan Perez (Administrador)  
¿Que quiere ver?  
(1)Usuarios.  
(2)Vehiculos.  
(3)Viajes.  
(4)Volver.
```

Más tarde, podrá probar todo en el apartado de **Vehículos**, como la creación de nuevos vehículos al perfil de cualquier usuario, o la modificación o eliminación de vehículos existentes, además, podrá ver una lista con todos los vehículos que hay en el sistema, o incluso ver un historial de los viajes que ha hecho un coche.

```
Hola Juan Perez (Administrador)
¿Que quiere hacer?
(1)Alta de vehiculo.
(2)Baja de vehiculo.
(3)Modificar vehiculo.
(4>Listar vehiculos.
(5)Mostrar lista de viajes de un vehiculo.
(6)Volver.
```

Una vez haya terminado de probar todo, puede dirigirse al menú de **Viajes**, para crear un viaje con cualquier usuario del sistema, también podrá eliminar, anular o modificar cualquier viaje que haya en la base de datos, al igual que podrá ver una lista de todos los viajes que han hecho cada usuario del sistema.

```
Hola Juan Perez (Administrador)
¿Que quiere hacer?
(1)Crear viaje.
(2)Anular/Finalizar viaje.
(3)Eliminar viaje.
(4)Modificar viaje.
(5>Listar viajes.
(6)Volver.
```

4. Documentación del código fuente

La documentación obtenida mediante el programa Doxygen está aquí, donde se pueden encontrar todas las definiciones de las estructuras, al igual que el código fuente de cada módulo, junto a su cabecera .h.

4.1. Código de los módulos

En esta parte, se mostrará el código fuente de todos los módulos del proyecto. Se ha puesto al final, para que no moleste a la hora de ver el resto del documento pdf.

4.1.1. Acceso

Para ver la descripción del módulo puede ir a 3.2.1.

```
#include "acceso.h"

//Prototipo: void acceso(Estr_Usuario *, int ,
Estr_Vehiculo *, int , Estr_Viaje *, int , Estr_Pasos
*, int , Estr_Reservas *, int , Estr_Localidad *, int ,
Estr_Rutas **, int , int );
//Precondicion: Tener inicializada la variable "
numUsuarios", con el numero de usuarios maximos del
fichero, y la estructura "usuario", con datos leidos
//desde el fichero "usuarios.txt", y haber introducido
el numero correspondiente en el menu. Tambien se
necesitaran el resto de estructuras, para
introducirlas en el menu.
//Postcondicion: Autenticacion de las credenciales
introducidas, si coinciden con alguna de la base de
datos, pues se accede al programa.
//Si el usuario es "usuario", accedera al menu de
usuario, y si es "administrador", accedera al menu
de admin.

void acceso(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_Localidad *localidad , int
```

```

numLocalidades , Estr_Rutas **ruta , int numRutas , int
numRutas2)
{
    int i , k=0, encontrado=0, encontrado2=0,
        encontrado3=3;
    char usua [6] , contra [9];

    printf("Introduzca sus credenciales de acceso:\n");
    color(0, 2);
    printf("Usuario:\n");
    color(0, 15);
    pregunta(usua , 6);

    for(k=0; k<numUsuarios&&encontrado2==0; k++)
    {
        if(strcmp(usua , usuario [k].usuario)==0) //  

            Comprueba si el usuario esta en la base de  

            datos .
        {
            encontrado2=1;
        }
    }

    if(encontrado2==1) //Si el usuario esta en la base  

        de datos , salta aqui .
    {
        while(encontrado3>0&&encontrado==0) //Sistema  

            de 3 intentos para introducir la contraseña  

            correctamente .
        {
            color(0, 3);
            printf("Contraseña:\n");
            color(0, 15);
            preguntar_contraseña (contra);

            for(i=0; i<numUsuarios || encontrado!=0; i++)
                //Comprueba si el usuario y la  

                contraseña son correctos .
            {
                if(strcmp(usuario [i].usuario , usua)==0)
                {

```

```

if(strcmp( usuario [ i ]. contrasena ,
            contra)==0)
{
    if(strcmp( usuario [ i ]. perfil ,"
                usuario")==0) //Si el
                usuario , tiene perfil de "
                usuario , salta al menu de
                usuario .
{
    encontrado=1;
    menuUsuario( usuario ,
                numUsuarios , vehiculo ,
                numVehiculos , viaje ,
                numViajes , pasos ,
                numPasos , reservas ,
                numReservas , localidad ,
                numLocalidades , ruta ,
                numRutas , numRutas2 , i );
}
else if(strcmp( usuario [ i ].
                perfil , "administrador")==0)
                //Si el usuario , tiene
                perfil de "administrador",
                salta al menu de admin .
{
    encontrado=1;
    menuAdmin( usuario ,
                numUsuarios , vehiculo ,
                numVehiculos , viaje ,
                numViajes , pasos ,
                numPasos , reservas ,
                numReservas , localidad ,
                numLocalidades , ruta ,
                numRutas , numRutas2 , i );
}
}
}

if(encontrado==0)
{

```

```

        color(0, 4);
        printf("\nLa contraseña introducida es "
               "incorrecta.\n"); //Si la contraseña
               "no es correcta, se imprime esta
               frase.
encontrado3--; //Se resta 1 intento,
                 hasta llegar a 0.
if(encontrado3==0)
{
    color(15, 0);
    printf("Intentos agotados! :\(\n");
    color(0, 15);
    system("PAUSE");
}
else if(encontrado3>0)
{
    contra[0]='0';
    color(15, 0);
    printf("Queda(n) %i intentos.\n",
           encontrado3); //Imprime un
                   contador de intentos, para saber
                   cuantos quedan.
    color(0, 15);
    system("PAUSE");
    system("cls");
    printf("Introduzca sus credenciales
           de acceso:\n");
    color(0, 2);
    printf("Usuario:\n");
    color(0, 15);
    printf("%s\n", usua);
}
}
}
else
{
    printf("Usuario no encontrado en nuestra base
           de datos.\n"); //Si el usuario no esta en la
           base de datos.
    system("PAUSE");
}

```

```

    }
}
```

4.1.2. Actualizar

Para ver la descripción del módulo puede ir a 3.2.2.

```
#include "actualizar.h"

//Prototipo: void actualizarUsuario(Estr_Usuario *, int );
//Precondicion: Tener la estructura "usuario"
// inicializada , con su contador .
//Postcondicion: Reescribir todos los datos de la
// estructura en el fichero .

void actualizarUsuario(Estr_Usuario *usuario , int
numUsuarios)
{
    int n=0;
    FILE *fp ;
    fp=fopen ("DATA/ usuarios .txt" , "w+" ); //Abrimos el
// fichero , para actualizarlo desde cero .

    if ( fp==NULL )
    {
        printf ("No se ha podido abrir el fichero -
usuarios .txt" );
    }
    else
    {
        for (n=0;n<numUsuarios ;n++ ) //Escribimos toda la
// estructura usuario en el fichero .
        {
            fprintf (fp , "%s-%s-%s-%s-%s\n" , usuario [
n ]. id_usuario , usuario [n ]. nomb_usuario ,
usuario [n ]. localidad , usuario [n ]. perfil ,
usuario [n ]. usuario , usuario [n ]. contrasena );
        }
    }
}
```

```

        fclose (fp) ;
    }

//Prototipo: void actualizarVehiculo (Estr_Vehiculo *,
int);
//Precondicion: Tener la estructura "vehiculo"
inicializada , con su contador.
//Postcondicion: Reescribir todos los datos de la
estructura en el fichero .

void actualizarVehiculo (Estr_Vehiculo *vehiculo , int
    numVehiculos)
{
    int n=0;
    FILE *fp ;
    fp=fopen ("DATA/ vehiculos .txt" , "w+" ); //Abrimos el
fichero , para actualizarlo desde cero .

    if ( fp==NULL)
    {
        printf ("No_se_ha_podido_abrir_el_fichero_"
            "vehiculos .txt" );
    }
    else
    {
        for (n=0;n<numVehiculos ;n++) //Escribimos toda
la estructura vehiculo en el fichero .
        {
            fprintf (fp , "%s-%s-%s-%s\n" , vehiculo [n] .
                id_mat , vehiculo [n] . id_usuario , vehiculo
                [n] . num_plazas , vehiculo [n] . desc_veh);
        }
    }

    fclose (fp) ;
}

//Prototipo: void actualizarViaje (Estr_Viaje *, int);
//Precondicion: Tener la estructura "viaje"
inicializada , con su contador.

```

```

//Postcondicion: Reescribir todos los datos de la
estructura en el fichero.

void actualizarViaje( Estr_Viaje *viaje , int numViajes)
{
    int n=0;
    FILE *fp ;
    fp=fopen ("DATA/viajes.txt" , "w+" ); //Abrimos el
    fichero , para actualizarlo desde cero .

    if ( fp==NULL)
    {
        printf ("No se ha podido abrir el fichero viajes
                .txt" );
    }
    else
    {
        for (n=0;n<numViajes ;n++) //Escribimos toda la
        estructura viaje en el fichero .
        {
            fprintf (fp , "%s-%s-%s-%s-%s-%s-%s\n" ,
                    viaje [n]. id_viaje , viaje [n]. id_mat , viaje
                    [n]. f_inic , viaje [n]. h_inic , viaje [n].
                    h_fin , viaje [n]. plazas_libre , viaje [n].
                    ida_vuelta , viaje [n]. precio , viaje [n].
                    estado );
        }
    }
    fclose (fp );
}

```

4.1.3. Buscar

Para ver la descripción del módulo puede ir a 3.2.3.

```
#include "buscar.h"
```

```
void imprimirPasos( Estr_Pasos * , int , Estr_Localidad * ,
int , char * , char * );
```

```

//Prototipo: void buscadorRutas(Estr_Rutas **, int , int
, Estr_Localidad *, int , Estr_Pasos *, int , char
[7]);
//Precondicion: Tener las estructuras "ruta", "
localidad" y "pasos", con sus contadores. Ademas,
debemos tener una cadena inicializada con la id de
viaje.
//Postcondicion: Buscar todas las rutas posibles , desde
la ciudad seleccionada hasta la ESI.

void buscadorRutas( Estr_Rutas **ruta , int numRutas , int
numRutas2 , Estr_Localidad *localidad , int
numLocalidades , Estr_Pasos *pasos , int numPasos ,
char id_viaje [7])
{
    FILE *fp ;
    char rutas [numRutas] [50] , partida [numLocalidades] ,
    **rutas_guard , **rutas_impr , *token=NULL, *
    ciudad=NULL;
    int i=0, j=0, encontrado=0, encontrado2=0,
    numGuardados=0, numImpresos=0, repetido=0, opc
    =0;

    fp=fopen ("DATA/rutas.txt" , "r"); //Abrimos el
    fichero rutas.txt
    if (fp==NULL)
    {
        printf ("No se pudo abrir el archivo rutas.txt.\ \
n");
        exit (1);
    }
    while (fgets (rutas [i] , 50 , fp)!=NULL) //Vamos
    introduciendo en la matriz rutas, todas las
    rutas que tiene el fichero, hasta el fin de
    fichero .
    {
        i++;
    }
    fclose (fp);
}

```

```

printf("Ingrese las siglas de la ciudad de partida/
      destino:\n");
pregunta_ruta(localidad, numLocalidades, ruta,
               numRutas, numRutas2, partida); //Pedimos ciudad
               de partida o destino

rutas_guard=(char **)calloc(numRutas, sizeof(char*))
            ; //Asignamos espacio de memoria, para las
            columnas de la matriz.
if (rutas_guard==NULL)
{
    printf("Error al asignar memoria.\n");
    exit(1);
}
for (i=0; i<numRutas; i++) //Vamos introduciendo
                           espacios de memoria de 50 caracteres a cada fila
                           de la matriz.
{
    rutas_guard [i]=(char *)calloc(50, sizeof(char))
                    ;
    if (rutas_guard [i]==NULL)
    {
        printf("Error al asignar memoria.\n");
        exit(1);
    }
}

for (i=0; i<numRutas; i++) //Nos desplazamos por
                           toda la estructura "ruta".
{
    token=strtok(rutas [i], "-");
    encontrado=0;
    while(token!=NULL) //Cuando ya no haya -,
                       significa que ha terminado la ruta.
    {
        ciudad=token;
        if(strcmp(ciudad, partida)==0) //Si al
                                       ciudad de partida esta en la ruta, la
                                       guardamos.
        {
            if(encontrado==0)

```

```

{
    strcat(rutas_guard [numGuardados] ,
           ciudad); //Guardamos la ciudad.
    encontrado=1;
}
token=strtok(NULL, "-");
while( token!=NULL&&encontrado2==0) //
    Hasta que no acabe la ruta, o
    aparezca ESI, no dejamos de
    introducir ciudades, para hacer la
    ruta.
{
    if(strcmp(token, "ESI")==0&&
        encontrado==0) //Si encontramos
        ESI, pasamos a escanear la
        siguiente ruta, con la variable
        bandera.
    {
        strcat(rutas_guard [numGuardados
                           ] , "-");
        strcat(rutas_guard [numGuardados
                           ] , token);
        encontrado=1;
    }
    else
    {
        strcat(rutas_guard [numGuardados
                           ] , "-");
        strcat(rutas_guard [numGuardados
                           ] , token);
    }
    token=strtok(NULL, "-");
}
numGuardados++; //Vamos aumentando la
                variable delimitadora, para luego
                hacer un for.
}
else
{
    token=strtok(NULL, "-");
}

```

```

        }

rutas_impr=(char **)calloc (numRutas, sizeof(char*))
; //Asignamos espacio de memoria, para las
columnas de la matriz.
if (rutas_impr==NULL)
{
    printf("Error al asignar memoria.\n");
    exit(1);
}
for (i=0; i<numRutas2; i++) //Vamos introduciendo
espacios de memoria de 50 caracteres a cada fila
de la matriz.
{
    rutas_impr [ i]=(char *)calloc (50, sizeof(char));
    if (rutas_impr [ i]==NULL)
    {
        printf("Error al asignar memoria.\n");
        exit(1);
    }
}

for (i=0; i<numGuardados; i++) //Nos desplazamos por
toda la matriz "rutas_guard".
{
    repetido=0;
    for (j=0; j<numImpresos&&repetido==0; j++) //Nos
desplazamos por toda la matriz "rutas_impr".
    {
        if (strcmp(rutas_guard [ i ] , rutas_impr [ j ]) ==0) //Si la ruta guardada coincide con
la ruta impresa, se omite.
        {
            repetido=1;
        }
    }

    if (!repetido) //Si la ruta no esta repetida, se
imprime.
}

```

```

{
    rutas_impr=(char **) realloc (rutas_impr ,(
        numImpresos+1)* sizeof(char*)); //  

        Asignamos un espacio de memoria mas.
    rutas_impr [numImpresos]=(char *) calloc (50 ,
        sizeof(char)); //Introducimos un espacio  

        de memoria de 50 caracteres a la fila  

creada.
    strcpy (rutas_impr [numImpresos] , rutas_guard  

        [i]); //Copiamos la ruta impresa en la  

        matriz.
    printf ("%i)%s" , numImpresos+1, rutas_guard  

        [i]); //Imprimimos la ruta .
    numImpresos++;
}
printf ("\nCantidad_de_rutas_encontradas : %i \n" ,
    numImpresos);

do{
    printf ("Seleccione el numero de la ruta que  

        quiere escoger : ");
    scanf ("%li" , &opc);
}while (opc<1||opc>numImpresos);

imprimirPasos (pasos , numPasos , localidad ,
    numLocalidades , id_viaje , rutas_impr [opc-1]); //  

Imprimimos las localidades de la ruta  

seleccionada en el fichero pasos.txt
}

//Prototipo: void imprimirPasos (Estr_Pasos * , int ,
Estr_Localidad * , int , char * , char *);
//Precondicion: Tener las estructuras "pasos" y "
localidad" , con sus contadores. Ademas, debemos
tener una cadena inicializada con la id de viaje , y
otra cadena con la ruta que queremos imprimir.
//Postcondicion: Imprimir los pasos de la ruta
seleccionada , en el fichero .

```

```

void imprimirPasos( Estr_Pasos *pasos , int numPasos ,
    Estr_Localidad *localidad , int numLocalidades , char
    *id_viaje , char *ruta )
{
    FILE *fp ;
    int j=0, i=0, x=0;
    char *ciudad , *ciudades [10] ;

    ciudad=strtok(ruta , "-") ; //Vamos separando la ruta
    , en siglas .
    while(ciudad!=NULL) //Hasta que no haya acabado la
    ruta .
    {
        ciudades [j]=(char *)malloc(4*sizeof(char)) ; //Asignamos un espacio de memoria para las
        siglas de la localidad .
        strcpy(ciudades [j] , ciudad) ; //Copiamos las
        siglas en el vector .
        j++;
        ciudad=strtok(NULL, "-") ; //Vamos haciendo lo
        mismo hasta que acabe la ruta .
    }

    fp=fopen("DATA/pasos.txt" , "a+" ) ; //Abrimos el
    fichero "pasos.txt" en modo edicion .

    if (fp==NULL)
    {
        printf("No se ha podido abrir el fichero pasos .
        txt.\n") ;
        return ;
    }
    else
    {
        for (i=0; i<j ; i++) //Nos desplazamos por el
        vector "ciudades" .
        {
            for (x=0; x<numLocalidades; x++) //Nos
            desplazamos por la estructura "localidad
            ".
            {

```

```

    if( strcmp( ciudades[ i ] , localidad[ x ].  

        siglas )==0) //Si las siglas de la  

        ciudad, coinciden con las de la  

        estructura .  

    {  

        fprintf( fp , "%s-%s\n" , id_viaje ,  

            localidad[ x ].localidad); //  

        Imprimimos la localidad a la que  

        le corresponden las siglas ,  

        junto a la id de viaje .  

        numPasos++;  

    }  

}  

}  

}  

}  

}  

fclose ( fp );  

printf ("La ruta seleccionada se ha asignado  

    correctamente al viaje con ID %s.\n" , id_viaje );  

system ("PAUSE" );
}

```

4.1.4. Colores

Para ver la descripción del módulo puede ir a 3.2.4.

```
#include "colores.h"
```

```

enum colores //Listado de colores
{
    negro=0,
    azul=1,
    verde=2,
    cyan=3,
    rojo=4,
    magenta=5,
    marron=6,
    gris_claro=7,
    gris_oscuro=8,
    azul_claro=9,

```

```

    verde_claro=10,
    cyan_claro=11,
    rojo_claro=12,
    magenta_claro=13,
    amarillo=14,
    blanco=15,
};

//Prototipo: void color(int, int);
//Precondicion: Necesita dos numeros asociado al color
que se quiere colocar, uno para el fondo, y otro
para el cuerpo del texto.
//Postcondicion: Cambia de color, el fondo y el cuerpo
del texto.

void color(int fondo, int texto)
{
    HANDLE consola=GetStdHandle(STD_OUTPUT_HANDLE);
    //Accedemos a la consola.

    //Para cambiar el color, se utilizan numeros
    desde el 0 hasta el 255, pero para convertir
    los colores a un valor adecuado, se realiza
    el siguiente calculo.
    int color_nuevo=texto+(fondo*16);

    SetConsoleTextAttribute(consola, color_nuevo);
    //Guardamos los cambios en la consola.
}

```

4.1.5. Eliminar

Para ver la descripción del módulo puede ir a 3.2.5.

```
#include "eliminar.h"
```

```

//Prototipo: void eliminarVehiculo(Estr_Usuario *, int,
Estr_Vehiculo *, int, Estr_Viaje *, int, Estr_Pasos
*, int, Estr_Reservas *, int, int);
//Precondicion: Tener la variable "i" inicializada, que
representa al usuario en la estructura "Usuarios",

```

```

//tambien necesitaremos las estructuras "Vehiculo", "Viajes", "Pasos" y "Reservas", con sus respectivos contadores.
//Postcondicion: Preguntar al usuario, que vehiculo quiere eliminar de todos los que tiene, para eliminar el vehiculo, con sus viajes, pasos y reservas.

void eliminarVehiculo(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, int i)
{
    int x=0, h=0, m=0, opc=0, opc2=0, *vec=NULL;

    encontrarVehiculos(usuario, vehiculo, numVehiculos,
        &vec, &x); //Encontramos todos los vehiculos del usuario, y los colocamos en un vector de enteros,
    //con las posiciones en la estructura "vehiculo" de sus vehiculos.

    if(x>0){ //Si tiene algun vehiculo, le pregunta el coche que quiere eliminar, si no tiene vehiculo, se imprime un aviso.
        printf("Que_vehiculo_quiere_eliminar?\n");
        for(m=0; m<x; m++) //Imprimimos todos los coches que tiene el usuario.
        {
            color(0,4);
            printf("----Vehiculo");
            color(0,15);
            printf("%i", m+1);
            color(0,4);
            printf(": \t");
            color(0,3);
            printf("----Matricula: ");
            color(0,15);
            printf("%s", vehiculo[vec[m]].id_mat);
        }
    }
}

```

```

        color(0,3);
        printf("|\_Numero_de_plazas:\_");
        color(0,15);
        printf("%s", vehiculo [ vec [m] ]. num_plazas );
        color(0,3);
        printf("|\_Descripcion:\_");
        color(0,15);
        printf("%s\n", vehiculo [ vec [m] ]. desc_veh );
    }
    m++;
    color(0,14);
    printf("(%i) Salir.\n", m);
    color(0,15);
    printf("Ingrese_el_numero_correspondiente_al_
    vehiculo_que_desea_eliminar:\_"); //Seleccion
    del vehiculo que quiere eliminar.
fflush(stdin);
scanf("%d", &opc);
if(opc==m){ //Si introduce el numero mas alto "
    "m", volvera a la funcion anterior, en este
    caso el menu.
    return;
}
if((opc>=1&&opc<m)&&opc!=m) //Si el numero
introducido no esta en el rango que se
representa, es decir,
//es menor que 1 o es mayor que m, pues se
vuelva a entrar a la funcion,
funcionando como un do-while.
{
    h=opc-1;
    do{
        opc2=0;
        system("cls");
        printf("Seguro que quieras eliminar el_
        vehiculo %s , con matricula %s?\n",
        vehiculo [ vec [h] ]. desc_veh , vehiculo [
        vec [h] ]. id_mat );
        //Si el usuario escoge el 1, se
        eliminara el vehiculo, si escribe 2,
        se volvera al menu.

```

```

printf("(1) Si\n");
printf("(2) No\n");
fflush(stdin);
scanf("%d", &opc2);
if(opc2==1){ //Al escoger 1, nos lleva
    a otra funcion, que eliminara los
    viajes, pasos y reservas del
    vehiculo seleccionado.
    system("cls");
    eliminarVehiculoViajes(usuario,
        numUsuarios, vehiculo,
        numVehiculos, viaje, numViajes,
        pasos, numPasos, reservas,
        numReservas, vehiculo[vec[h]].  

        id_mat, x);
    printf("Eliminado con exito.\n");
    system("PAUSE");
}
else ((opc2<1) || (opc2>2));
}
else //Si el numero no esta en el rango,
volvemos a entrar en la funcion.
{
    system("cls");
    eliminarVehiculo(usuario, numUsuarios,
        vehiculo, numVehiculos, viaje, numViajes
        , pasos, numPasos, reservas, numReservas
        , i); //Vuelva a entrar en la funcion.
}
else //Se imprime un aviso, si no tiene vehiculos
asignados .
{
    system("cls");
    printf("No posee vehiculos registrados.\n");
    system("PAUSE");
}
system("cls");
}

```

```

//Prototipo: void eliminarVehiculoViajes(Estr_Usuario
* , int , Estr_Vehiculo *, int , Estr_Viaje *, int ,
Estr_Pasos *, int , Estr_Reservas *, int , char *, int
);
//Precondicion: Tener la cadena "mat", que seria la
matricula del vehiculo que se quiere eliminar, y la
variable "x", que representa el numero
//de vehiculos que tiene el usuario, tambien
necesitaremos las estructuras "Vehiculo", "Viajes",
"Pasos" y "Reservas", con sus respectivos contadores

//Postcondicion: Eliminar el vehiculo, a partir de la
matricula, ademas de todos sus viajes, pasos y
reservas.

void eliminarVehiculoViajes(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 , char *mat , int x)
{
    FILE *fp , *temp;
    int n=0, j=0, k=0, num_v=0, *vec_viaje=NULL,
        counter , encontrado=0;

    fp=fopen ("DATA/ vehiculos .txt" , "r+" );

    if ( fp==NULL )
    {
        printf ("No_se_ha_podido_abrir_el_fichero_"
            "vehiculos .txt .\n");
        return;
    }
    else
    {
        for (n=0; n<numVehiculos&&encontrado==0; n++) // 
            Nos desplazamos por toda la estructura "
            vehiculo", hasta encontrar el vehiculo con
            la matricula deseada.
    {

```

```

if(strcmp( vehiculo [n]. id_mat , mat)==0) //Si
    coincide la matricula de la posicion
    actual en la estructura con la matricula
    deseada .
{
    temp=fopen ( "DATA/vehiculos_Temp .txt" , "w"
        + ); //Se crea un fichero temporal ,
        donde escribir los vehiculos que hay
        , menos el vehiculo que se quiere
        eliminar .
if (temp==NULL)
{
    printf ( "No se ha podido abrir el
        fichero_vehiculos_Temp .txt .\n" );
}
else
{
    for ( counter=0; counter<numVehiculos
        ; counter++)
    {
        if (strcmp( vehiculo [counter ]. id_mat , mat)!=0) //Cuando el
            matricula del vehiculo ,
            coincide con la matricula
            deseada , se salta de linea/
            omite .
        {
            fprintf ( temp , "%s-%s-%s-%s \
                n" , vehiculo [counter ]. id_mat , vehiculo [counter ]. id_usuario , vehiculo [
                    counter ]. num_plazas ,
                    vehiculo [counter ]. desc_veh );
        }
    }
    fclose (temp); //Cerramos los ficheros .
    fclose (fp);
    for (j=0; j<x; j++)
{

```

```

        encontrarViajes( vehiculo ,
                        numVehiculos , viaje , numViajes ,
                        vehiculo [n]. id_mat , &vec_viaje ,
                        &num_v , 0); //Encontramos todos
                        los viajes del vehiculo , para
                        eliminarlos .
    }
    for (k=0; k<num_v; k++) //Eliminamos
        todos los viajes , pasos y reservas .
    {
        eliminarSoloViaje( viaje , numViajes ,
                            pasos , numPasos , reservas ,
                            numReservas , viaje [ vec_viaje [k
                            ] ]. id_viaje );
    }
    encontrado=1;
}
}
remove("DATA/vehiculos.txt"); //Borramos el
fichero original .
rename("DATA/vehiculos_Temp.txt","DATA/
vehiculos.txt"); //Cambiemos de nombre el
fichero temporal por el original .
}
fclose (fp);
}

//Prototipo: void eliminarViaje(Estr_Usuario *, int ,
Estr_Vehiculo *, int , Estr_Viaje *, int , Estr_Pasos
*, int , Estr_Reservas *, int , int );
//Precondicion: Tener la variable "i" inicializada , que
representa al usuario en la estructura "Usuarios",
//Ademas se necesitaran las estructuras "Vehiculo" ,
"Viajes" , "Pasos" y "Reservas" , y sus respectivos
contadores .
//Postcondicion: Preguntar al usuario , que viaje quiere
eliminar de todos los que tiene , para eliminar el
viaje , con sus pasos y reservas .

void eliminarViaje( 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 , int i)
{
    int j=0, x=0, h=0, m=0, num_v=0, opc=0, opc2=0, *
        vec=NULL, *vec_viaje=NULL;

    encontrarVehiculos( usuario , vehiculo , numVehiculos ,
        &vec , &num_v , i); //Busca los vehiculos que
        tiene dicho usuario , para luego encontrar los
        viajes que tiene .

    for (j=0;j<num_v ;j++)
    {
        encontrarViajes( vehiculo , numVehiculos , viaje ,
            numViajes , vehiculo [ vec [ j ] ]. id_mat , &
            vec_viaje , &x , 2);
    }

    if (x>0){
        printf("Que_viaje_quiere_eliminar?\n");
        for (m=0;m<x ;m++) //Imprime una lista de todos
            los viajes que tiene el usuario .
        {
            color(15, 0);
            printf("VIAJE-%i :\n" ,i+1);
            color(0, 3);
            printf("ID del viaje : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [m] ]. id_viaje );
            color(0, 3);
            printf("Estado : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [m] ]. estado );
            color(0, 3);
            printf("Plazas libres : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [m] ]. plazas_libre );
            color(0, 3);
            printf("Fecha de partida : ");
            color(0, 15);
        }
    }
}

```

```

        printf ("%s\n" , viaje [ vec [m] ] . f_inic );
        color (0 , 3);
        printf (" Hora_de_partida : " );
        color (0 , 15);
        printf ("%s\n" , viaje [ vec [m] ] . h_inic );
        color (0 , 3);
        printf (" Hora_de_llegada : " );
        color (0 , 15);
        printf ("%s\n" , viaje [ vec [m] ] . h_fin );
        color (0 , 3);
        printf (" Tipo : " );
        color (0 , 15);
        printf ("%s\n" , viaje [ vec [m] ] . ida_vuelta );
        color (0 , 3);
        printf (" Precio : " );
        color (0 , 15);
        printf ("%s_euros\n\n" , viaje [ vec [m] ] . precio
            );
    }
    x++;
    printf ("(%i) Salir.\n" , x);
    printf (" Ingrese el numero correspondiente al
        viaje que desea eliminar : " ); //Introduce el
        numero del viaje .
fflush (stdin);
scanf ("%d" , &opc);
if (opc==x) //Si el numero introducido coincide
    con el del numero maximo "x" , se vuelve al
    menu.
{
    return ;
}
if ((opc>=1&&opc<x)&&opc!=x) //Si el numero
    introducido no esta en el rango que se
    representa , es decir ,
    //es menor que 1 o es mayor que m, pues se
    vuelve a entrar a la funcion ,
    funcionando como un do-while .
{
    h=opc-1;

```

```

do //Imprime un mensaje de confirmacion
    para eliminar el viaje seleccionado , se
    repite hasta que sea 1 o 2.
{
    opc2=0;
    system("cls");
    printf("Seguro que quieras eliminar el
        viaje %s , con matricula %s?\n",
        viaje [ vec_viaje [ h ] ]. id_viaje , viaje [
        vec_viaje [ h ] ]. id_mat );
    printf("(1) Si\n");
    printf("(2) No\n");
    fflush (stdin);
    scanf("%d" , &opc2);
    if (opc2==1){ //Si seleccionamos 1, se
        va a la funcion , que elimina viajes ,
        pasos y reservas , por la id del
        viaje .
        system("cls");
        eliminarSoloViaje(viaje , numViajes ,
            pasos , numPasos , reservas ,
            numReservas , viaje [ vec_viaje [ h
                ] ]. id_viaje );
        printf("Eliminado con exito.\n");
        system("PAUSE");
    }
    }while((opc2<1) || (opc2>2));
}
else //Si el numero no esta en el rango ,
    volvemos a entrar en la funcion .
{
    system("cls");
    eliminarViaje(usuario , numUsuarios ,
        vehiculo , numVehiculos , viaje , numViajes
        , pasos , numPasos , reservas , numReservas
        , i );
}
else{ //Se imprime un aviso , si no tiene viajes
    creados .
    system("cls");
}

```

```

        printf("No_tiene_viajesRegistrados.\n");
        system("PAUSE");
    }
    system("cls");
}

//Prototipo: void eliminarSoloViaje(Estr_Viaje *, int ,
                 Estr_Pasos *, int , Estr_Reservas *, int , char *);
//Precondicion: Tener la cadena "id" inicializada, que
                 representa la id del viaje, que se quiere eliminar,
//al igual que las estructura "viaje", "pasos" y "
                 reservas", con sus respectivos contadores.
//Postcondicion: Eliminar un viaje, a partir de su id ,
                 junto a todos sus pasos y reservas.

void eliminarSoloViaje(Estr_Viaje *viaje, int numViajes
, Estr_Pasos *pasos, int numPasos, Estr_Reservas *
reservas, int numReservas, char *id)
{
    FILE *fp, *temp;
    int n=0, counter, encontrado=0;

    leer_viaje(&viaje, &numViajes);
    system("cls");

    fp=fopen("DATA/viajes.txt", "r+");
    if (fp==NULL)
    {
        printf("No_se_ha_podido_abrir_el_fichero_viajes
             .txt.\n");
        return;
    }
    else
    {
        for (n=0; n<numViajes&&encontrado==0; n++) //Nos
                desplazamos por toda la estructura "viaje",
                hasta encontrar el viaje con la id deseada.
        {
            if (strcmp(viaje[n].id_viaje, id)==0) //Si
                 la id del viaje de la posicion actual en

```

la estructura coincide con la id deseada .

```

{
    temp=fopen ("DATA/viajes_Temp.txt" , "w+")
        ; //Se crea un fichero temporal,
        donde escribir los viajes que hay,
        menos el viaje que se quiere
        eliminar.
    if (temp==NULL)
    {
        printf ("No se ha podido abrir el
                fichero_viajes_Temp.txt.\n");
    }
    else
    {
        for (counter=0; counter<numViajes ;
            counter++)
        {
            if (counter!=n) //Cuando la
                linea que queremos eliminar ,
                coincide con la linea
                actual , se salta de linea/
                omite.
            {
                fprintf (temp , "%s-%s-%s-%s-
                    -%s-%s-%s-%s-%s\n",
                    viaje [ counter ]. id_viaje ,
                    viaje [ counter ]. id_mat ,
                    viaje [ counter ]. f_inic ,
                    viaje [ counter ]. h_inic ,
                    viaje [ counter ]. h_fin ,
                    viaje [ counter ]. plazas_libre ,
                    viaje [
                    counter ]. ida_vuelta ,
                    viaje [ counter ]. precio ,
                    viaje [ counter ]. estado );
            }
        }
    }
    fclose (temp); //Cerramos los ficheros .
    fclose (fp);
}

```

```

        eliminarPasos(pasos , numPasos , viaje [n]
                      ]. id_viaje); //Eliminamos los pasos
                      y reservas de dicho viaje .
        eliminarReservas(reservas , numReservas ,
                          viaje [n]. id_viaje);
        encontrado=1;
    }
}
remove("DATA/viajes .txt"); //Borramos el
                           fichero original .
rename("DATA/viajes_Temp .txt","DATA/viajes .txt"
      ); //Cambiamos de nombre el fichero temporal
           por el original .
}
fclose(fp );
}

//Prototipo: void eliminarPasos(Estr_Pasos *, int , char *) ;
//Precondicion: Tener la cadena "id" inicializada , que
representa la id del viaje , al igual que la
estructura "pasos" , y su contador "numPasos".
//Postcondicion: Eliminar los pasos de un viaje .

void eliminarPasos(Estr_Pasos *pasos , int numPasos ,
                      char *id)
{
    FILE *fp , *temp ;
    int n=0, counter ;

    leer_pasos(&pasos , &numPasos) ;
    system("cls") ;

    fp=fopen ("DATA/pasos .txt" , "r+" ) ;

    if ( fp==NULL )
    {
        printf("No_se _ha _podido _abrir _el _fichero _pasos .
               txt.\n");
        return ;
    }
}

```

```

else
{
    for(n=0; n<numPasos; n++) //Nos desplazamos por
        toda la estructura "pasos".
    {
        if(strcmp(pasos[n].id_viaje, id)==0) //Si
            la id del viaje de la posicion actual en
            la estructura coincide con la id
            deseada.
        {
            temp=fopen("DATA/pasos_Temp.txt", "w+");
            //Se crea un fichero temporal,
            donde escribir los pasos que hay,
            menos los pasos que se quieren
            eliminar.
            if(temp==NULL)
            {
                printf("No se ha podido abrir el
                    fichero pasos_Temp.txt.\n");
            }
            else
            {
                for(counter=0; counter<numPasos;
                    counter++)
                {
                    if(strcmp(pasos[counter].id_viaje, id)!=0) //Cuando
                        la id del viaje del paso que
                        queremos eliminar,
                        //coincide con la linea
                        actual, se salta de
                        linea/omite.
                    {
                        fprintf(temp, "%s-%s\n",
                            pasos[counter].id_viaje,
                            pasos[counter].
                            poblacion);
                    }
                }
            }
            fclose(temp); //Cerramos los ficheros.
        }
    }
}

```

```

        fclose (fp) ;
    }
}
remove ( "DATA/pasos .txt" ) ; //Borramos el fichero
//Cambiemos de nombre el fichero temporal
por el original .
rename ( "DATA/pasos_Temp .txt" , "DATA/pasos .txt" ) ;
fclose (fp) ;
}

//Prototipo: void eliminarReservas (Estr_Reservas * , int
//Precondicion: Tener la cadena "id" inicializada , que
//representa la id del viaje , al igual que la
//estructura "reservas" , y su contador "numReservas".
//Postcondicion: Eliminar las reservas de un viaje .

void eliminarReservas (Estr_Reservas *reservas , int
numReservas , char *id )
{
    FILE *fp , *temp ;
    int n=0, counter ;

    leer_reservas (&reservas , &numReservas ) ;
    system ( "cls" ) ;

    fp=fopen ( "DATA/reservas .txt" , "r+" ) ;

    if ( fp==NULL )
    {
        printf ( "No se ha podido abrir el fichero "
        "reservas .txt .\n" ) ;
        return ;
    }
    else
    {
        for (n=0; n<numReservas; n++) //Nos desplazamos
        por toda la estructura "reservas" .
    }
}

```

```

if(strcmp( reservas [n]. id_viaje , id )==0) //
    Si la id del viaje de la posicion actual
    en la estructura coincide con la id
    deseada .
{
    temp=fopen ( "DATA/reservas_Temp.txt" , "w+
" ); //Se crea un fichero temporal ,
    donde escribir los reservas que hay ,
    menos las reservas que se quieren
    eliminar .
if (temp==NULL)
{
    printf ( "No se ha podido abrir el
    fichero_reservas_Temp.txt.\n" );
}
else
{
    for ( counter=0; counter<numReservas ;
        counter++)
    {
        if (strcmp( reservas [counter ].
            id_viaje , id)!=0) //Cuando
            la id del viaje de la
            reserva que queremos
            eliminar ,
            //coincide con la linea
            actual , se salta de
            linea/omite .
        {
            fprintf (temp , "%s-%s\n" ,
                reservas [counter ].
                id_viaje , reservas [
                    counter ]. id_usuario );
        }
    }
    fclose (temp); //Cerramos los ficheros .
    fclose (fp );
}
}

```

```

        remove("DATA/reservas.txt"); //Borramos el
                                    fichero original.
        rename("DATA/reservas_Temp.txt","DATA/reservas.
txt"); //Cambiamos de nombre el fichero
temporal por el original.
    }
    fclose(fp);
}

//Prototipo: void eliminarAdminUsuario(Estr_Usuario *,
int , Estr_Vehiculo *, int , Estr_Viaje *, int ,
Estr_Pasos *, int , Estr_Reservas *, int );
//Precondicion: Tener las estructuras "Usuario", "
Vehiculo", "Viajes", "Pasos" y "Reservas",
//al igual que sus contadores "numUsuarios", "
"numVehiculos", "numViajes", "numPasos" y "
"numReservas".
//Postcondicion: Preguntar al admin, la id del usuario
que quiere eliminar, para borrar dicho usuario,
junto a todos sus vehiculos, viajes, pasos y
reservas.

void eliminarAdminUsuario(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)
{
    FILE *fp , *temp ;
    int n=0, o=0, j=0, opc2=0, c=0, *vec=NULL,
        encontrado=0, encontrado2=0, id2 , counter ;
    char vec_id[5] ;

    fp=fopen("DATA/usuarios.txt","r+");
    if( fp==NULL)
    {
        printf("No se ha podido abrir el fichero
                usuarios.txt.\n");
        return ;
    }
}

```

```

else
{
    if ( numUsuarios>1)
    {
        do{ //Se pregunta la id del usuario para
            eliminarlo , hasta que la id sea correcta

                listarUsuarios( usuario , numUsuarios);
                //Obtenemos una lista de todos los
                usuarios del sistema.
                printf("Que_usuario_quiere_eliminar?\n"
                );
                printf("Introduzca_la_ID_del_usuario_
                    que_quiere_eliminar.\n");
                scanf("%i" , &id2);
                if(id2==1) //Si "id2" es igual a 1, se
                    vuelve a preguntar , ya que no se
                    puede eliminar al administrador.
{
                    printf("No_puede_eliminar_al_
                        ADMINISTRADOR, _con_ID_0001.\_
                        nVuelva_a_intentarlo_con_una_ID_
                        valida.\n");
                    system("PAUSE");
}
else
{
    sprintf( vec_id , "%04i" , id2);
    encontrarUsuario( usuario ,
        numUsuarios , vec_id , &j , &
        encontrado); //Si la id esta en
        la base de datos , encontrado=1,
        y sale del bucle.
}
}while(encontrado==0);

do //Repetir lo mismo, hasta que la opcion
    sea 1 o 2.
{
    opc2=0;

```

```

system("cls"); //Imprime mensaje de
               confirmacion.
printf("Seguro que quieras eliminar el
       usuario %s , con ID %s?\n", usuario[ j
       ].nombr_usuario, usuario[ j ].id_usuario);
printf("(1) Si\n");
printf("(2) No\n");
fflush(stdin);
scanf("%d", &opc2);

if(opc2==1){
    system("cls");
    encontrado=0;
    for(n=0; n<numUsuarios&&encontrado2
        ==0; n++) //Nos desplazamos por
                   toda la estructura "usuario",
                   hasta que se encuentra al
                   usuario deseado.
    {
        if(strcmp(usuario[ n ].id_usuario
                   , usuario[ j ].id_usuario)==0)
            //Si la id del usuario de
            la posicion actual
            //en la estructura coincide
            con la id deseada.
        {
            temp=fopen("DATA/
                       usuarios_Temp.txt","w+")
            ; //Se crea un fichero
            temporal, donde escribir
            los usuarios que hay,
            //menos el usuario que se
            quiere eliminar.
        if(temp==NULL)
        {
            printf("No se ha podido
                   abrir el fichero
                   usuarios_Temp.txt.\n
                   ");
        }
    }
}

```

```

else
{
    for (counter=0; counter<
        numUsuarios; counter
++)
{
    if (strcmp( usuario [
        counter ]. 
        id_usuario ,
        vec_id)!=0) // 
        Cuando la id del
        usuario que
        queremos
        eliminar ,
        //coincide con
        la linea
        actual , se
        salta de
        linea / omite .
}
    fprintf(temp, "
    %s-%s-%s-%s
    -%s-%s\n" ,
    usuario [
    counter ]. 
    id_usuario ,
    usuario [
    counter ]. 
    nomb_usuario
    , usuario [
    counter ]. 
    localidad ,
    usuario [
    counter ]. 
    perfil ,
    usuario [
    counter ]. 
    usuario ,
    usuario [
    counter ]. 
    contrasena );
}

```

```

        }
    }
}
fclose (temp); //Cerramos
    los ficheros .
fclose (fp);
encontrarVehiculos( usuario ,
    vehiculo , numVehiculos ,
    &vec , &o , n); //
    Buscamos todos los
    vehiculos del usuario ,
    para eliminarlos .
for(c=0; c<o; c++)
{
    eliminarVehiculoViajes(
        usuario , numUsuarios
        , vehiculo ,
        numVehiculos , viaje ,
        numViajes , pasos ,
        numPasos , reservas ,
        numReservas ,
        vehiculo [ vec [ c ] ].  

        id_mat , o); //
    Eliminamos los
    vehiculos , viajes ,
    pasos y reservas del
    usuario
    seleccionado .
}
encontrado2=1;
}
remove("DATA/usuarios.txt"); //
    Borramos el fichero original.
rename("DATA/usuarios_Temp.txt","  

    DATA/usuarios.txt"); //Cambiemos
    de nombre el fichero temporal
    por el original.
system("cls");
printf("Eliminado con exito\n");
system("PAUSE");

```

```

        }
    }while((opc2<1) || (opc2>2));
}
else //Se imprime un aviso , si no hay usuarios.
{
    system("cls");
    printf("No hay usuarios registrados.\n");
    system("PAUSE");
}
fclose(fp);
system("cls");
}

//Prototipo: void eliminarAdminVehiculo(Estr_Usuario *,
int , Estr_Vehiculo *, int , Estr_Viaje *, int ,
Estr_Pasos *, int , Estr_Reservas *, int , int );
//Precondicion: Tener las estructuras "Usuario", "
Vehiculo", "Viajes", "Pasos" y "Reservas",
//al igual que sus contadores "numUsuarios", "
numVehiculos", "numViajes", "numPasos" y "
numReservas".
//Postcondicion: Preguntar al admin, la id del usuario
para saber sus vehiculos , y eliminar dicho vehiculo ,
y todos sus viajes , pasos y reservas .

void eliminarAdminVehiculo(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 , int x)
{
    char opc2[5];
    int encontrado=0, i=0, counter , id;

    system("cls");
    if(numVehiculos!=0) //Si hay vehiculos en el
    sistema .
    {
        listarVehiculos(usuario , numUsuarios , vehiculo ,
        numVehiculos , x); //Obtenemos una lista de

```

```

    todos los vehiculos del sistema.

printf("Escriba la ID del usuario a la que se
quieras eliminar un vehiculo\n");
scanf("%i", &id); //Se pregunta la id del
usuario al que quiere eliminar el vehiculo.
sprintf(opc2, "%04i", id);

for(counter=0;(counter<numUsuarios)&&
encontrado==0); counter++)
{
    if(strcmp(opc2, usuario [ counter ]. id_usuario)
==0) //Cuando coincida la id del usuario
de la estructura, con la id deseada,
//entra en la funcion eliminarVehiculo ,
y le preguntara la matricula del
vehiculo a eliminar.
{
    encontrado=1;
    i=counter;
    system("cls");
    eliminarVehiculo(usuario , numUsuarios ,
vehiculo , numVehiculos , viaje ,
numViajes , pasos , numPasos , reservas
, numReservas , i );
}
if(encontrado==0) //Si la id del usuario es
incorrecta , se imprime este aviso .
{
    system("cls");
    printf("No se ha encontrado ningun usuario
con la siguiente ID: %s\n",opc2);
    system("PAUSE");
}
}

//Prototipo: void eliminarAdminViaje(Estr_Usuario *,
int , Estr_Vehiculo *, int , Estr_Viaje *, int ,
Estr_Pasos *, int , Estr_Reservas *, int , int , int );

```

```

//Precondicion: Tener la variable "num", y las
estructuras "Usuario", "Vehiculo", "Viajes", "Pasos"
y "Reservas",
//al igual que sus contadores "numUsuarios", "
"numVehiculos", "numViajes", "numPasos" y "
"numReservas".
//Postcondicion: Preguntar al admin, la id del usuario
para saber sus vehiculos, y con todas sus matriculas
, localizar sus viajes.
//Si num=0, elimina un viaje, y si num=1, anula/
finaliza un viaje, depende de su estado. Todo esto
eliminando todos sus pasos y reservas.

void eliminarAdminViaje(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, int x, int num)
{
    char opc2[5];
    int encontrado=0, i=0, counter, n=0, id;

    system("cls");
    listarAdminViajes(usuario, numUsuarios, vehiculo,
        numVehiculos, viaje, numViajes, x, num, &n); // 
    Obtenemos una lista de todos los viajes del
    sistema.
    if(n!=0) //Si hay viajes con las condiciones
        requeridas.
    {
        printf("Escriba la ID del usuario al que quiere
            -eliminar-un-viaje.\n");
        scanf("%i", &id); //Se pregunta la id del
            usuario al que quiere eliminar el viaje.
        sprintf(opc2, "%04i", id);

        for(counter=0;(counter<numUsuarios)&&
            encontrado==0); counter++)
        {
            if(strcmp(opc2, usuario[counter].id_usuario)
                ==0) //Cuando coincida la id del usuario

```

```

de la estructura, con la id deseada,
//entra en la funcion eliminarViaje, y
le preguntara la id del viaje a
eliminar.
{
    encontrado=1;
    i=counter;
    system("cls");
    if(num==0)
    {
        eliminarViaje(usuario , numUsuarios ,
                      vehiculo , numVehiculos , viaje ,
                      numViajes , pasos , numPasos ,
                      reservas , numReservas , i );
    }
    if(num==1)
    {
        finalizar_viaje(usuario , vehiculo ,
                        numVehiculos , viaje , numViajes ,
                        pasos , numPasos , reservas ,
                        numReservas , i );
    }
}
if(encontrado==0) //Si la id del usuario es
                  incorrecta, se imprime este aviso.
{
    system("cls");
    printf("No se ha encontrado ningun usuario
           con la siguiente ID: %s\n",opc2);
    system("PAUSE");
}
}

//Prototipo: void finalizar_viaje(Estr_Usuario *,
          Estr_Vehiculo *, int , Estr_Viaje *, int , Estr_Pasos
          *, int , Estr_Reservas *, int , int );
//Precondicion: se necesitara la variable numuser que
respresenta al usuario en la estructura Usuarios;

```

```

//ademas se necesitaran las estructuras de "vehiculos",
 "viajes", "pasos", "reservas", con sus respectivos
 contadores.
//Cabeza: esta funcion tiene una doble finalidad , que
 seran finalizar y anular los viajes del conductor.
//Postcondicion: el viaje elegido por el usuario sera
 finalizado o anulado , modificandose el estado del
 viaje .

void finalizar_viaje(Estr_Usuario *usuario ,
 Estr_Vehiculo *vehiculo , int numVehiculos ,
 Estr_Viaje *viaje , int numViajes , Estr_Pasos *pasos ,
 int numPasos , Estr_Reservas *reservas , int
 numReservas , int num_user)
{
    int *vec=NULL, x=0, *vec_viaje=NULL, max_viaje=0, i
    =0, j;

    encontrarVehiculos(usuario , vehiculo , numVehiculos ,
        &vec , &x , num_user);

    if( vec!=NULL) //verifica si se han encontrado
        vehiculos
    {
        for(j=0;j<x;j++)//para los vehiculos
            encontrados verifica si tiene asignado
            viajes
        {
            encontrarViajes( vehiculo , numVehiculos ,
                viaje , numViajes , vehiculo [ vec [ j ] ].
                id_mat , &vec_viaje , &max_viaje , 1);
        }
        if( vec_viaje!=NULL)//verifica si se encontraron
            viajes para los coches encontrados
        {
            printf("LISTADO_DE_SUS_VIAJES:\n");
            for(j=0;j<max_viaje;j++)//se muestran en
                pantalla esos viajes
            {
                color(15, 0);
                printf("VIAJE_%i :\n",j+1);
            }
        }
    }
}

```

```

        color(0, 3);
        printf("...ID del viaje: ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ j ]]. id_viaje );
        color(0, 3);
        printf("...Estado: ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ j ]]. estado );
        color(0, 3);
        printf("...Plazas libres: ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ j ]]. plazas_libre );
        color(0, 3);
        printf("...Fecha de partida: ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ j ]]. f_inic );
        color(0, 3);
        printf("...Hora de partida: ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ j ]]. h_inic );
        color(0, 3);
        printf("...Hora de llegada: ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ j ]]. h_fin );
        color(0, 3);
        printf("...Tipo: ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ j ]]. ida_vuelta );
        color(0, 3);
        printf("...Precio: ");
        color(0, 15);
        printf("%s euros\n\n", viaje [ vec [ j ]]. precio );
    }

do{/verifica que se seleccione la opcion correcta
    printf("Seleccione el viaje que desea finalizar/anular:\n");

```

```

        scanf("%i",&i);
    }while(i>max_viaje || i<0);

    system("cls");
    if(strcmp(viaje[vec_viaje[i-1]].estado , "abierto")==0)//verifica si el viaje seleccionado esta abierto
    {
        strcpy(viaje[vec_viaje[i-1]].estado , "anulado");
        printf("El viaje ha sido anulado\n");
    }
    else// caso contrario esta iniciado
    {
        strcpy(viaje[vec_viaje[i-1]].estado , "finalizado");
        printf("El viaje ha sido finalizado\n");
    }
    system("PAUSE");
    actualizarViaje(viaje , numViajes);
    eliminarPasos(pasos , numPasos , viaje[vec_viaje[i-1]].id_viaje);
    eliminarReservas(reservas , numReservas ,
                      viaje[vec_viaje[i-1]].id_viaje);
}
else//caso contrario no hay viajes registrado para los coches del usuario
{
    system("cls");
    printf("No tiene viajes registrados\n");
    system("PAUSE");
}
else//caso contrario el usuario no tiene coches registrados
{
    system("cls");
    printf("No tiene coches registrados\n");
    system("PAUSE");
}

```

```

}

//Prototipo: void cancelarReserva(Estr_Usuario *,
    Estr_Viaje *, int, Estr_Reservas *, int, int):
//Precondicion: esta funcion recibe las estructuras "
    Usuario" y "Viajes" ya inicializadas con sus
    contadores, y la variables num_user que representa
    la posicion del usuario en la estructura viajes.
//Cabecera: esta funcion permite al usuario cancelar sus
    viajes siempre que no esten iniciados.
//Postcondicion: el viaje seleccionado es cancelado
    siendo borrado de la estructura reservas el usuario
    y aumentando en uno las plazas de dicho viaje.

void cancelarReserva( Estr_Usuario *usuario , Estr_Viaje
    *viaje , int numViajes , Estr_Reservas *reservas , int
    numReservas , int num_user)
{
    int *vec=NULL, *vec_viaje=NULL, x, sel , plazas=0;

    encontrarReservas(usuario , viaje , numViajes ,
        reservas , numReservas , num_user , &vec , &
        vec_viaje , &x);//encuentra las reservas del
    usuario que pueden ser canceladas

    if( vec!=NULL){ //verifica que el usuario tiene
        reservas
        do{
            listarReservas(usuario , viaje , numViajes ,
                reservas , numReservas , num_user);
            color(0, 14);
            printf("(%i) Salir.\n", x+1);
            color(0, 15);
            printf(" Seleccione_el_viaje_que_desea_"
                "cancelar:\n");
            scanf("%i",&sel);
            if( sel==x+1)
            {
                return;
            }
            system(" cls");
        }
    }
}

```

```

}while(sel>x || sel<1); //comprueba que el
    usuario haya elegido un viaje que se
    encuentre en el rango de opciones

eliminarReservas(reservas, numReservas,
    reservas[vec[sel-1]].id_viaje);
plazas=atoi(viaje[vec_viaje[sel-1]].plazas_libre);
    plazas++; //aumenta en uno las plazas
plazas++;
sprintf(viaje[vec_viaje[sel-1]].plazas_libre,
    "%0ld", plazas);
actualizarViaje(viaje, numViajes);
}
else
{
    printf("No tiene viajes reservados\n");
    system("PAUSE");
}
}

```

4.1.6. Encontrar

Para ver la descripción del módulo puede ir a 3.2.6.

```
#include "encontrar.h"
```

```

//Prototipo: void encontrarVehiculos(Estr_Usuario *,
    Estr_Vehiculo *, int, int **, int *, int);
//Precondicion: Introducir un vector de enteros, para
    introducir datos en el mismo, al igual que la
    variable "x",
//para saber cuantos vehiculos tiene dicho usuario, y
    la variable "i", para identificar el usuario, es
    decir, la posicion
//en la estructura de "usuario". Ademas, se necesitan
    las estructuras "usuario" y "vehiculo" inicializadas
    , con sus contadores "numVehiculos" y "numUsuarios".
//Postcondicion: Busca todos los vehiculos de un
    usuario mediante su id, e introduce las posiciones
    en la estructura de dichos vehiculos en un vector de
    enteros .

```

```

void encontrarVehiculos( Estr_Usuario *usuario ,
    Estr_Vehiculo *vehiculo , int numVehiculos , int **vec
    , int *x, int i)
{
    int j;
    *x=0;

    for(j=0; j<numVehiculos; j++) //Nos desplazamos por
        toda la estructura "vehiculo".
    {
        if(strcmp( usuario [ i ]. id_usuario , vehiculo [ j ].
            id_usuario)==0) //Si la id del usuario que
            hemos introducido es igual a la id del
            usuario de un vehiculo.
        {
            *vec=(int *) realloc (*vec ,(( *x)+1)*sizeof(int)); //Asignamos un espacio de memoria
                mas, para introducir el vehiculo
                detectado.
            if ((*vec)==NULL)
            {
                printf( " Error al asignar memoria.\n" );
                exit(1);
            }
            (*vec)[*x]=j; //Se introduce la posicion
                del vehiculo encontrado en la estructura
                en el vector de enteros dinamico.
            (*x)++; //Vamos aumentando en 1, la
                cantidad de vehiculos que tiene el
                usuario, que al mismo tiempo servira
                para acceder al vector de enteros.
        }
    }
}

//Prototipo: void encontrarViajes(Estr_Vehiculo *, int ,
    Estr_Viaje *, int , char *, int **, int *, int );
//Precondicion: Introducir la cadena "mat", que sera la
    matricula del vehiculo, al que queremos obtener
    todos sus viajes, un vector de enteros,

```

```

//para introducir datos en el mismo, al igual que la
//variable "x", para saber cuantos viajes tiene dicho
//usuario, y la variable "num", que sera 0, 1, 2 o 3,
//dependiendo de para que queramos usar la funcion.
Ademas, se necesita las estructuras "vehiculos" y "
viaje" inicializadas, con sus contadores.
//Postcondicion: Busca todos los viajes de un vehiculo
mediante su matricula, e introduce las posiciones en
la estructura de dichos viajes en un vector de
enteros.

void encontrarViajes(Estr_Vehiculo *vehiculo, int
numVehiculos, Estr_Viaje *viaje, int numViajes, char
*mat, int **vec, int *x, int num)
{
    int j, i;

    for(j=0; j<numViajes; j++) //Nos desplazamos por
        toda la estructura "viaje".
    {
        if(strcmp(viaje[j].id_mat, mat)==0) //Si la
            matricula del coche que hemos introducido es
            igual a la matricula asignada en un viaje .
        {
            if(num==0) //Si num=0, se encuentran todos
                los viajes de un vehiculo .
            {
                *vec=(int *)realloc(*vec,((*x)+1)*
                    sizeof(int)); //Asignamos un espacio
                    de memoria mas, para introducir el
                    viaje detectado .
                if (*vec==NULL)
                {
                    printf("Error_al_asignar_memoria.\n"
                        );
                    exit(1);
                }
                (*vec)[*x]=j; //Se introduce la
                    posicion del viaje encontrado en la
                    estructura en el vector de enteros
                    dinamico .
            }
        }
    }
}

```

```

(*x)++; //Vamos aumentando en 1, la
cantidad de viajes que tiene el
usuario, que al mismo tiempo servira
para acceder al vector de enteros.
}
if(num==1) //Si num=1, se encuentran los
viajes abiertos, sin plazas ocupadas, e
iniciados (para anularlos/finalizarlos).
{
    for ( i=0;i<numVehiculos ;i++)
    {
        if(( strcmp( viaje [ j ]. estado , "
aberto")==0)&&(strcmp( mat ,
vehiculo [ i ]. id_mat)==0)) //Si el
viaje esta abierto, y la
matricula del coche coincide.
{
            if(strcmp( viaje [ j ]. plazas_libre
, vehiculo [ i ]. num_plazas)
==0) //Si no se han ocupado
plazas del viaje, es decir,
numero plazas libres =
numero plazas vehiculo .
{
                *vec=(int *)realloc (*vec
,(( *x)+1)*sizeof (int));
//Asignamos un espacio
de memoria mas, para
introducir el viaje
detectado .
if (*vec==NULL)
{
                printf ("Error _al _
asignar _memoria .\n" )
;
                exit (1);
}
(*vec)[*x]=j; //Se
introduce la posicion
del viaje encontrado en
la estructura en el

```

```

    vector de enteros
    dinamico.
(*x)++; //Vamos aumentando
en 1, la cantidad de
viajes que tiene el
usuario, que al mismo
tiempo servira para
acceder al vector de
enteros.
}
}
if((strcmp(viaje[j].estado, "
iniciado")==0)&&(strcmp(mat,
vehiculo[i].id_mat)==0)) //Si el
viaje esta iniciado, y la
matricula del coche coincide.
{
*vec=(int *)realloc(*vec,((*x)
+1)*sizeof(int)); //
Asignamos un espacio de
memoria mas, para introducir
el viaje detectado.
if(*vec==NULL)
{
printf("Error al asignar
memoria.\n");
exit(1);
}
(*vec)[*x]=j; //Se introduce la
posicion del viaje
encontrado en la estructura
en el vector de enteros
dinamico.
(*x)++; //Vamos aumentando en
1, la cantidad de viajes que
tiene el usuario, que al
mismo tiempo servira para
acceder al vector de enteros
.
}
}

```

```

}

if(num==2) //Si num=2, se encuentran los
    viajes abiertos sin plazas ocupadas (
        para modificarlos).
{
    if((strcmp(viaje[j].estado, "abierto")
        ==0)) //Si el viaje esta abierto.
    {
        for( i=0;i<numVehiculos ; i++)
        {
            if(strcmp(mat , vehiculo [ i ] .
                id_mat)==0&&(strcmp(mat ,
                    viaje [ j ]. id_mat)==0)) //Si
                    la matricula introducida
                    coincide con la del vehiculo
                    , y la del viaje .
            {
                if(strcmp( viaje[ j ].
                    plazas_libre , vehiculo [ i ]
                    ]. num_plazas)==0) //Si
                    no se han ocupado plazas
                    del viaje , es decir ,
                    num plazas libres = num
                    plazas vehiculo .
            {
                *vec=(int *) realloc(*
                    vec ,(( *x)+1)*sizeof(int)); //Asignamos
                    un espacio de
                    memoria mas , para
                    introducir el viaje
                    detectado .
                if ((* vec )==NULL)
                {
                    printf(" Error _al_
                        asignar _memoria
                        .\n");
                    exit(1);
                }
                (* vec )[ *x]=j ; //Se
                    introduce la

```

*posicion del viaje
encontrado en la
estructura en el
vector de enteros
dinamico.*

```

(*x)++; //Vamos
        aumentando en 1, la
        cantidad de viajes
        que tiene el usuario
        , que al mismo
        tiempo servira para
        acceder al vector de
        enteros .
}
}
}
}
}
if(num==3) //Si num=3, se encuentran los
        viajes abiertos e iniciados (para
        mostrarlos).
{
    if (strcmp( viaje [ j ]. estado , "abierto" )
        ==0) //Si el viaje esta abierto .
{
    *vec=(int *) realloc (*vec ,(( *x)+1)*
        sizeof(int)); //Asignamos un
        espacio de memoria mas, para
        introducir el viaje detectado .
    if (*vec==NULL)
{
    printf(" Error _al _asignar _
            memoria .\n");
    exit(1);
}
(*vec)[*x]=j ; //Se introduce la
        posicion del viaje encontrado en
        la estructura en el vector de
        enteros dinamico .
(*x)++; //Vamos aumentando en 1, la
        cantidad de viajes que tiene el

```

usuario , que al mismo tiempo servira para acceder al vector de enteros .

```

}
if(strcmp( viaje [ j ].estado , " iniciado" )
==0) //Si el viaje esta iniciado .
{
    *vec=(int *)realloc (*vec ,(( *x)+1)*
sizeof(int)); //Asignamos un
espacio de memoria mas, para
introducir el viaje detectado .
if (*vec==NULL)
{
    printf (" Error _al _asignar_
memoria.\n" );
    exit (1);
}
(*vec) [*x]=j ; //Se introduce la
posicion del viaje encontrado en
la estructura en el vector de
enteros dinamico .
(*x)++; //Vamos aumentando en 1, la
cantidad de viajes que tiene el
usuario , que al mismo tiempo
servira para acceder al vector
de enteros .
}
}
}
}

//Prototipo: void encontrarUsuario (Estr_Usuario *, int ,
char *, int *, int *);
//Precondicion: Introducir la cadena "vec_id" , que sera
la id de un usuario que queremos encontrar , junto a
una variable "x" , para obtener la posicion
//del usuario en la estructura "usuario" , y la variable
"encontrado" incializada . Ademas, se necesita la
estructura "usuario" , con su contador "numUsuarios" .

```

```

//Postcondicion: Busca al usuario en la base de datos
mediante su id, e introduce su posicion en la
estructura en una variable "x", y "encontrado" pasa
a ser 1, si se encuentra el usuario.

void encontrarUsuario (Estr_Usuario *usuario , int
    numUsuarios , char vec_id [] , int *x , int *encontrado)
{
    int j;
    *x=0;
    *encontrado=0;

    for (j=0; j<numUsuarios; j++) //Nos desplazamos por
        toda la estructura "usuario".
    {
        if (strcmp (vec_id , usuario [j]. id_usuario)==0) //
            Si la id de usuario introducida esta en la
            estructura "usuario".
        {
            *x=j; //x obtiene la posicion del usuario
                en la estructura .
            *encontrado=1;
        }
    }
}

//Prototipo: void encontrarReservas(Estr_Usuario *,
Estr_Viaje *, int , Estr_Reservas *, int , int ,
int **, int **);
//Precondicion: Introducir la variable "num_user", que
nos dice la posicion del usuario en la estructura "
usuario", junto a una variable "x",
//para obtener el numero maximo de reservas que cumplen
las condiciones, y los vectores "vec" y "vec_viaje"
" , para meter las posiciones en esos vector
dinamicos.
//Ademas, se necesita las estructuras "usuario", "viaje
" y "reservas" inicializadas, con sus contadores.
//Postcondicion: Busca todas las reservas abiertas de
un usuario , mediante su id , e introduce las
posiciones en la estructura de dichas reservas

```

```

//en un vector de enteros, y las posiciones de los
viajes en los que estan esas reservas, en otro
vector de enteros.

void encontrarReservas( Estr_Usuario *usuario ,
    Estr_Viaje *viaje , int numViajes , Estr_Reservas *
    reservas , int numReservas , int num_user , int **vec ,
    int **vec_viaje , int *x)
{
    int i , j ;
    (*x)=0;

    for (i=0;i<numReservas ;i++) //Nos desplazamos por
        toda la estructura "reservas".
    {
        if(strcmp( reservas [ i ]. id_usuario , usuario [
            num_user ]. id_usuario )==0) //Si la id del
            usuario que queremos buscar, coincide con la
            id del usuario en la reserva.
        {
            for (j=0;j<numViajes ;j++) //Nos desplazamos
                por toda la estructura "viajes".
            {
                if(strcmp( viaje [ j ]. id_viaje , reservas [ i ]
                    ]. id_viaje )==0) //Si la id de viaje,
                        coincide con la id de viaje de la
                        reserva.
                {
                    if(strcmp( viaje [ j ]. estado , "abierto"
                        )==0) //Si el viaje esta abierto
                    .
                    {
                        *vec=(int *) realloc(*vec ,(( *x)
                            +1)*sizeof(int)); //
                            Asignamos un espacio de
                            memoria mas, para introducir
                            la reserva detectada.
                    if ((*vec)==NULL)
                    {
                        printf(" Error _al _asignar _\n"
                            " memoria.\n");
                    }
                }
            }
        }
    }
}

```

```

        exit(1);
    }

    *vec_viaje=(int*)realloc(*
        vec_viaje,((*x)+1)*sizeof(
        int)); //Asignamos un
        espacio de memoria mas, para
        introducir el viaje
        detectado.
    if ((*vec_viaje)==NULL)
    {
        printf("Error al asignar la
            memoria.\n");
        exit(1);
    }

    (*vec_viaje)[*x]=j; //Se
        introduce la posicion del
        viaje encontrado en la
        estructura en el vector de
        enteros dinamico.
    (*vec)[*x]=i; //Introducimos la
        posicion de la reserva
        encontrada en la estructura
        en el vector de enteros
        dinamico.
    (*x)++;
        //Vamos aumentando en
        1, la cantidad de viajes y
        reservas que tiene el
        usuario, que al mismo tiempo
        servira para acceder a los
        dos vectores de enteros.
}
}
}
}

//Prototipo: void encontrarViajesReservas(Estr_Usuario
*, Estr_Viaje *, int, Estr_Pasos *, int, int, char

```

```

*, int **, int *, int *);

//Precondicion: Introducir la variable "num_user", que
nos dice la posicion del usuario en la estructura "usuario", para saber su localidad, junto a una
variable "x",
//para obtener el numero maximo de viajes que cumplen
las condiciones, y la variable "loc", para saber si
hay viajes por su localidad, pero en fechas
distintas,
//junto a un cadena "fecha", para saber en que fecha
quiere buscar el viaje, y el vector "vec", para
introducir las posiciones de los viajes
//que pasan por la ciudad del usuario. Ademas, se
necesita las estructuras "usuario", "viaje" y "pasos"
inicializadas, con sus contadores.
//Postcondicion: Busca todas los viajes abiertos o
iniciados, que pasen por la localidad de un usuario,
mediante su id, e introduce las posiciones en la
//estructura de dichos viajes en un vector de enteros.
Si no hay viajes en la fecha indicada, entonces "loc"
sera distinto de 0.

void encontrarViajesReservas( Estr_Usuario *usuario ,
Estr_Viaje *viaje , int numViajes , Estr_Pasos *pasos ,
int numPasos , int num_user , char *fecha , int **vec ,
int **x, int *loc)
{
    int i , j ;
    *x=0;
    *loc=0;

    for( i=0; i<numPasos; i++) //Nos desplazamos por
        toda la estructura "pasos".
    {
        if(strcmp( usuario [ num_user ].localidad , pasos [ i ].
            poblacion)==0) //Si hay algun viaje que pase
            por la localidad del usuario .
        {
            (*loc)++; //Se aumenta en 1, entonces
            sabemos que hay viajes en alguna fecha
            futura .
        }
    }
}

```

```

for ( j=0; j<numViajes ; j++) //Nos desplazamos
    por toda la estructura "viajes".
{
    if (strcmp( pasos [ i ]. id_viaje , viaje [ j ].
        id_viaje)==0) //Si la id de viaje en
            los pasos coincide con la id de
            viaje .
{
    if (strcmp( fecha , viaje [ j ]. f_inic )
        ==0) //Comprobamos que la fecha
            introducida es igual a la fecha
            del viaje .
{
    if (strcmp( viaje [ j ]. estado ,"
        abierto")==0||strcmp( viaje [ j ]. estado ,"
        iniciado")==0) //
            Si el viaje esta abierto o
            iniciado .
{
    *vec=(int *) realloc (*vec
        ,(( *x)+1)*sizeof(int));
        //Asignamos un espacio
        de memoria mas, para
        introducir el viaje
        detectado .
if ((* vec)==NULL)
{
    printf (" Error al
        asignar memoria.\n")
    ;
    exit (1);
}
(* vec)[* x]=j; //Se
    introduce la posicion
    del viaje encontrado en
    la estructura en el
    vector de enteros
    dinamico.
(*x)++; //Vamos aumentando
    en 1, la cantidad de
    viajes que tiene

```

disponible el usuario para la fecha introducida, //que al mismo tiempo servira para acceder a los dos vectores de enteros.

}

}

}

}

}

}

}

4.1.7. Escribir

Para ver la descripción del módulo puede ir a 3.2.7.

```
#include "escribir.h"
```

```

void elegir_coche(Estr_Usuario *, int, Estr_Vehiculo *,
                   int, Estr_Viaje *, int, Estr_Pasos *, int,
                   Estr_Reservas *, int, Estr_Localidad *, int,
                   Estr_Rutas **, int, int, int, char *, int);
void asignar_plazas(Estr_Vehiculo *, int, char *, char
                     *);
void ida_vuelta(char *);
void verificar_viaje(Estr_Viaje *, int, char *, char
                      *, char *, char *, char *, int *);
void mostrar_poblaciones(Estr_Viaje *, int, Estr_Pasos
                          *, int, int *, int, int *);
void guardarPasajero(Estr_Usuario *, Estr_Viaje *,
                      Estr_Reservas *, int, int *, int, int);
void verificar_reserva(Estr_Usuario *, Estr_Vehiculo *,
                       int, Estr_Viaje *, Estr_Reservas *, int, int *,
                       int *, int);

//Prototipo: void altaUsuario(Estr_Usuario *, int,
                           Estr_Localidad *, int);

```

```

//Precondicion: Tener las estructuras "usuario" y "
localidad" inicializada , con sus contadores "
numUsuarios" y "numLocalidades".
//Postcondicion: Dar de alta/Registrar un usuario ,
comprobando que el nombre del usuario no esta
repetido .

void altaUsuario( Estr_Usuario *usuario , int numUsuarios
    , Estr_Localidad *localidad , int numLocalidades)
{
    FILE *fp ;
    int i=0, n=1, idmax=0, k=0, encontrado=0,
        encontrado2=0, encontrado3=0;
    char id [5] , nombre[21] , loc [21] , perfil [14] ,
        usuario2 [6] , contrasena [9];

    fp=fopen ("DATA/ usuarios . txt" , "a+" );

    if ( fp==NULL)
    {
        printf ("No se ha podido abrir el fichero viajes
            .txt.\n");
        return ;
    }
    else
    {
        strcpy (perfil , "usuario"); //Copia la cadena "usuario" en el vector perfil , ya que solo habra 1 administrador.
        printf ("Introduzca sus datos para completar el
            registro:\n");
        printf ("NombreCompleto (Maximo de 20
            caracteres):\n");
        pregunta (nombre , 21); //Funcion similar a un scanf o un fgets , pero con la eliminacion del vector nulo incorporada.
        printf ("Localidad de residencia (Siglas de la
            lista):\n");
        pregunta_localidad (localidad , numLocalidades ,
            loc); //Pregunta de la localidad , imprimiendo una lista con los acronimos de las ciudades .
    }
}

```

```

printf("Nombre_de_usuario_(Maximo_de_5_
caracteres):\n");
pregunta(usuario2, 6);

for(k=0; k<numUsuarios&&encontrado==0; k++) //  

    Se desplaza por toda la estructura "usuario"  

    , hasta que encuentra un usuario con mismo  

    nick introducido.
{
    if(strcmp(usuario2, usuario[k].usuario)==0)
        //Si encuentra un usuario en la base de  

        datos con el mismo nick que hemos  

        introducido.
    {
        encontrando=1;
    }
}

if(encontrado==0) //No seria necesario, pero  

prefiero hacerlo asi, para luego poner un  

aviso.
{
    printf("El_nombre_de_usuario_es_valido.\n")
    ;
    printf("Contrasena_(Maximo_de_8_caracteres)
        :\n");
    pregunta(contrasena, 9);

    if(fp==NULL)
    {
        printf("No_se_ha_podido_abrir_el_
fichero_usuarios.txt.\n");
        return;
    }
    else
    {
        while(encontrado2==0) //Bucle para  

encontrar la id mas baja, y ocuparla
        .
    }
}

```

```

for ( i=0; i<numUsuarios&&encontrado3
      ==0; i++) //Nos desplazamos por
      la estructura "usuario", hasta
      que se encuentra la id mas baja .
{
    idmax=atoi( usuario[ i ].  

                id_usuario); //Pasa la id  

                actual a un variable de  

                entero.
    if(n==idmax) //Si n es igual  

        que la id actual, se pone  

        encontrado3 a 1.
        encontrado3=1;
}
if(encontrado3==1) //Si encontrado3  

=1, se aumenta 1 a la cantidad  

de n, y encontrado3 vuelve a 0,  

para volver al bucle .
{
    n++;
    encontrado3=0;
}
else //Si encontrado3=0, se rompe  

el bucle principal .
{
    encontrado2=1;
}
}

snprintf(id, sizeof(id), "%04d", n); //  

Pasa la nueva id a una cadena de 4  

caracteres .

fprintf(fp, "%s-%s-%s-%s-%s\n", id ,
        nombre, loc, perfil, usuario2 ,
        contrasena); //Se imprimen todos los  

        datos del registro en el fichero "  

        usuarios.txt".
printf("El usuario ha sido agregado -  

correctamente.\n");

```

```

        system("PAUSE");
    }
}
else //Aparece aviso de que el usuario esta
      siendo usado.
{
    printf("El nombre de usuario ya esta siendo
           usado.\n");
    system("PAUSE");
}
fclose(fp);
}

//Prototipo: void altaVehiculo(Estr_Usuario *,
Estr_Vehiculo *, int , int );
//Precondicion: Tener la variable "i", con la posicion
del usuario en la estructura "usuario", y las
estructuras "vehiculo" y "usuario" inicializadas ,
con su contador "numVehiculos".
//Postcondicion: Dar de alta/Registrar un vehiculo ,
comprobando que la matricula es valida , y no esta en
uso .

void altaVehiculo(Estr_Usuario *usuario , Estr_Vehiculo
*vehiculo , int numVehiculos , int i)
{
FILE *fp;
int error_mat , counter;
char mat[8] , plazas[2] , descrip[51];

fp=fopen("DATA/vehiculos.txt","a");

do{ //Se repite hasta que la matricula sea valida .
error_mat=0;
system("cls");
printf("Introduzca los datos de su vehiculo
       para completar su registro:\nMatricula del
       vehiculo (Maximo de 7 caracteres):\n");
fflush(stdin);
scanf("%7s" , mat);
if(strcmp(mat,"")!=0)
{
    if(strlen(mat)>7)
    {
        printf("La matricula debe tener un maximo de 7
               caracteres.\n");
        error_mat++;
    }
    else
    {
        if(error_mat==0)
        {
            strcpy(usuario->matricula,mat);
            strcpy(vehiculo->matricula,mat);
            strcpy(vehiculo->plazas,&plazas);
            strcpy(vehiculo->descrip,&descrip);
            system("cls");
            printf("Vehiculo registrado con exito.\n");
            system("PAUSE");
        }
    }
}
}
while(error_mat!=0);
}

```

```

for( counter=0;( counter<numVehiculos)&&( error_mat==0); counter++) //Nos desplazamos por toda la estructura "vehiculo" para ver si es valida .
{
    if(strcmp(mat , vehiculo [ counter ]. id_mat)==0)
        //Si la matricula introducida , coincide con la de un vehiculo de la estructura , pues la matricula esta en uso .
    {
        error_mat=1;
        system("cls");
        printf("La matricula %s esta actualmente registrada.\n", mat);
        system("PAUSE");
    }
}

if( strlen (mat)<7) //Comprueba que la longitud de la matricula es de 7 caracteres .
{
    system("cls");
    printf("La matricula %s debe poseer una longitud total de 7 caracteres.\n", mat);
    ;
    system("PAUSE");
}

for( counter=0;( counter<4)&&(error_mat==0);
counter++) //Nos desplazamos por los 4 primeros caracteres de la matricula , para saber si son numeros , mediante sus valores ASCII .
{
    if(( mat [ counter]<48) || ( mat [ counter]>57))
    {
        error_mat=1;
        printf("Los 4 primeros caracteres de la matricula tienen que ser numeros.\n");
        system("PAUSE");
    }
}

```

```

        }
    }
for( counter=4;( counter<7)&&(error_mat==0);
    counter++) //Nos desplazamos por los 3
    ultimos caracteres de la matricula, para
    saber si son letras, mediante sus valores
    ASCII.
{
    if(( mat[ counter]<65) || ( mat[ counter]>90))
    {
        error_mat=1;
        printf("Los 3 ultimos caracteres de la
            matricula tienen que ser letras
            mayusulas.\n");
        system("PAUSE");
    }
}
while(( error_mat==1) || ( strlen ( mat)<7));

printf("Numero de plazas libres (sin contar el
    conductor):\n");
pregunta( plazas , 2);
printf("Descripcion del vehiculo (Marca , modelo ,
    color , etc) (Maximo de 50 caracteres):\n");
pregunta( descrip , 51);

if( fp==NULL)
{
    printf("No se ha podido abrir el fichero
        vehiculos.txt.\n");
    return;
}
else
{
    fprintf( fp , "%s-%s-%s-%s\n" , mat , usuario[ i ].
        id_usuario , plazas , descrip); //Se imprimen
        todos los datos del registro en el fichero "
        vehiculos.txt".
}
fclose( fp );

```

```

}

//Prototipo: void altaAdmin(Estr_Usuario *, int,
 Estr_Vehiculo *, int, Estr_Viaje *, int, Estr_Pasos
* , int, Estr_Reservas *, int, Estr_Localidad *, int,
Estr_Rutas **, int, int, int);
//Precondicion: Necesario la introduccion de un entero,
 para saber si queremos registrar un vehiculo o un
 viaje. Tener las estructuras inicializadas, con sus
 contadores.
//Postcondicion: Si n=0, dar de alta/regarstrar un
 vehiculo, comprobando que la matricula es valida, y
 no esta en uso,
//y si n=1, dar de alta/regarstrar un viaje. Ambos,
 introduciendo la id del usuario al que se quiere
 crear.

void altaAdmin(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_Localidad *localidad, int
numLocalidades, Estr_Rutas **ruta, int numRutas, int
numRutas2, int n)
{
    char opc2[5];
    int encontrado=0, i=0, counter;

    system("cls");
    listarUsuarios(usuario, numUsuarios);
    printf("Escriba la ID del usuario al que quiera dar
        de alta el vehiculo.\n");
    scanf("%4s",opc2);
    for(counter=0;(counter<numUsuarios)&&(encontrado
    ==0);counter++) //Nos desplazamos por la
        estructura "usuario", hasta saber que la id
        introducida es valida.
    {
        if(strcmp(opc2,usuario[counter].id_usuario)==0)
            //Si la id introducida es igual que la id
            actual.

```

```

{
    encontrado=1;
    i=counter;
    if (n==0)
    {
        altaVehiculo(usuario , vehiculo ,
                     numVehiculos , i); //Usamos la
                     funcion altaVehiculo , para reusar
                     codigo .
        printf("El_vehiculo_se_ha_agregado_
               correctamente_al_usuario_%s , _con_ID_
               %s.\n" , usuario [ counter ] .
               nombr_usuario , usuario [ counter ] .
               id_usuario );
        system("PAUSE" );
    }
    if (n==1)
    {
        altaViaje(usuario , numUsuarios ,
                   vehiculo , numVehiculos , viaje ,
                   numViajes , pasos , numPasos , reservas
                   , numReservas , localidad ,
                   numLocalidades , ruta , numRutas ,
                   numRutas2 , i , 1);
        //Usamos la funcion altaViaje , para
        reusar codigo .
        printf("El_viaje_se_ha_agregado_
               correctamente_al_usuario_%s , _con_ID_
               %s.\n" , usuario [ counter ] .
               nombr_usuario , usuario [ counter ] .
               id_usuario );
        system("PAUSE" );
    }
}

if (encontrado==0) //Si no se encuentra el usuario ,
                   se imprime un aviso .
{
    system("cls");
}

```

```

        printf("No se ha encontrado ningun usuario con-
            la siguiente ID: %s\n", opc2);
        system("PAUSE");
    }
}

//Prototipo: void altaViaje(Estr_Usuario *, int,
Estr_Vehiculo *, int, Estr_Viaje *, int, Estr_Pasos
*, int, Estr_Reservas *, int, Estr_Localidad *, int,
Estr_Rutas **, int, int, int, int);
//Precondicion: se necesitara la variable numUsuarios
que respresenta al usuario en la estructura Usuarios

//Ademas se necesitaran las estructuras de "vehiculos",
"viajes", "pasos", "reservas", "localidad", "rutas
", con sus respectivos contadores.
//Cabecera: esta funcion se usa para que el conductor
cree viajes ingresando valores como la matricula del
coche la fecha, las horas de inicio y fin o el
coste,
//y asignadose otros valores automaticamente como el
esatdo del viaje, la id del viaje o las plazas de
este.
//Postcondicion: se crea una nueva estructura "Viajes"
que contendra los datos correspondientes al nuevo
viaje creado por el conductor y se habra anadido al
fichero el nuevo viaje

void altaViaje(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_Localidad *localidad, int
numLocalidades, Estr_Rutas **ruta, int numRutas, int
numRutas2, int num_usuario, int num)
{
    FILE *fv;
    int n=1, i, idmax=0, breakp=0, enc=0, precio=0, rp,
        encontrado=0;
    char viaje_id[7], id_vehiculo[8], fecha[11],
        hora_inic[6], hora_fin[6], plazas[2], idavuelta

```

```

[7] , coste [5] , estado [8];

fv=fopen ("DATA/viajes.txt" , "a+");
do //en este bucle se ingresan todos los datos del
    viaje y al final se verifica si rp , si es uno
    deben ingresarse los datos de nuevo
{
    rp=0;
    elegir_coche( usuario , numUsuarios , vehiculo ,
        numVehiculos , viaje , numViajes , pasos ,
        numPasos , reservas , numReservas , localidad ,
        numLocalidades , ruta , numRutas , numRutas2 ,
        num_usuario , id_vehiculo , num); //el usuario
        aqui elige el coche con el que va a realizar
        el viaje
    asignar_plazas( vehiculo , numVehiculos ,
        id_vehiculo , plazas); //se asignan las
        plazas del viaje automaticamente

    leerFecha( fecha , hora_inic , hora_fin);

    ida_vuelta( idavuelta); //se establece si el
        viaje es de ida o vuelta

while( encontrado==0) //aqui se establece el
    coste del viaje hasta que se encuentre en el
    rango establecido
{
    system("cls");
    printf("Estableza el coste de su viaje:\n"
        );
    fflush(stdin);
    scanf("%i" , &precio);
    if( precio>0&&precio<10)
    {
        encontrado=1;
    }
}

sprintf(coste , "%i" , precio);

```

```

strcpy(estado , "abierto");

if( fv==NULL)
{
    printf("No se ha podido abrir el fichero
           viajes.txt.\n");
    return;
}
else
{
    while( breakp==0) //se establece de manera
               automatica la id del viaje
    {
        for( i=0;i<numViajes && enc==0;i++) //
               busca a traves de la estructura la
               siguiente id que se debe asignar
        {
            idmax=atoi( viaje [ i ]. id_viaje );
            if( n==idmax )
                enc=1;
            }
            if( enc==1)
            {
                n++;
                enc=0;
            }
            else
            {
                breakp=1;
            }
    }

    snprintf(viaje_id , sizeof( viaje_id ) , "%06d"
             , n); //pasa la id nueva a un vector
             limitado por 7 espacios.

    verificar_viaje( viaje , numViajes , viaje_id ,
                     id_vehiculo , fecha , hora_inic , hora_fin
                     , &rp ); //se verifica si los datos
                     corresponden con los de algun viaje
                     creado antes
}

```

```

if (rp==0)
{
    fprintf(fv, "%s-%s-%s-%s-%s-%s-%s-%s
\n", viaje_id, id_vehiculo, fecha,
hora_inic, hora_fin, plazas,
idavuelta, coste, estado);
fclose(fv);
system("cls");
buscadorRutas(ruta, numRutas, numRutas2
, localidad, numLocalidades, pasos,
numPasos, viaje_id);
}
else
{
    printf("ERROR/ No se pueden asignar dos
viajes con el mismo vehiculo en la
misma fecha, a no ser que el viaje
este finalizado o anulado.\n");
system("PAUSE");
}
}

}while(rp==1);
fclose(fv);
}

//Prototipo: void elegir_coche(Estr_Usuario *, int,
Estr_Vehiculo *, int, Estr_Viaje *, int, Estr_Pasos
*, int, Estr_Reservas *, int, Estr_Localidad *, int,
Estr_Rutas **, int, int, int, char *, int)
//Precondicion: se necesitara la variable numUsuarios
que respresenta al usuario en la estructura Usuarios
;
//ademas se necesitaran las estructuras de "vehiculos",
"viajes", "pasos", "reservas", "localidad", "rutas
",>, con sus respectivos contadores.
//Cabeza: Esta funcion le muestra al usuario todos
los coches que tiene registrados, con los cuales
podra realizar el viaje.
//Postcondicion: devuelve un dato de tipo caracter, que
es la matricula del coche que ha elegido el
condutor para realizar el viaje

```

```

void elegir_coche( 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_Localidad *localidad
    , int numLocalidades , Estr_Rutas **ruta , int
    numRutas , int numRutas2 , int num_usuario , char *
    id_vehiculo , int num) //0 para usuario , y 1 para
    admin
{
    int i,* vec_vehiculo=NULL, coches , encontrado=0, opc
    =0, breakp=0;

    encontrarVehiculos(usuario , vehiculo , numVehiculos ,
    &vec_vehiculo , &coches , num_usuario);

    if(coches==0) //verifica si se ha encontrado algun
        coche , en caso de que no se hayan encontrado
    {
        if(num==0) //si num=0, es para usuario
        {
            do
            {
                system("cls");
                printf("No tiene ningun vehiculo_
                    registrado.\n\n");
                printf("Que quiere hacer?\n");
                printf("(1) Registrar un vehiculo.\n");
                printf("(2) Volver al menu.\n");
                scanf("%d",&opc);
                switch(opc)
                {
                    case 1:
                        breakp=1;
                        altaVehiculo(usuario , vehiculo ,
                            numVehiculos , num_usuario);
                        menuConductorViajes(usuario ,
                            numUsuarios , vehiculo ,
                            numVehiculos , viaje ,
                            numViajes , pasos , numPasos,

```

```

        reservas , numReservas ,
        localidad , numLocalidades ,
        ruta , numRutas , numRutas2 ,
        num_usuario );
break;
case 2:
    breakp=1;
    menuConductorViajes( usuario ,
        numUsuarios , vehiculo ,
        numVehiculos , viaje ,
        numViajes , pasos , numPasos ,
        reservas , numReservas ,
        localidad , numLocalidades ,
        ruta , numRutas , numRutas2 ,
        num_usuario );
    break;
}
} while( breakp==0);
}
if (num==1) //si num=1, es para admin
{
    do
    {
        system("cls");
        printf("El_usuario_%s , _con_ID_%s_no_"
            "tiene_ningun_vehiculo_registrado.\n\
            n" , usuario [ num_usuario ].
            nombr_usuario , usuario [ num_usuario ].
            id_usuario );
        printf("Que_quiere_hacer?\n");
        printf("(1) Registrar_un_vehiculo.\n");
        printf("(2) Volver_al_menu.\n");
        scanf("%d",&opc);
        switch(opc)
        {
            case 1:
                breakp=1;
                altaAdmin( usuario , numUsuarios ,
                    vehiculo , numVehiculos ,
                    viaje , numViajes , pasos ,
                    numPasos , reservas ,

```

```

        numReservas , localidad ,
        numLocalidades , ruta ,
        numRutas , numRutas , 0) ;
    menuAdminViajes( usuario ,
        numUsuarios , vehiculo ,
        numVehiculos , viaje ,
        numViajes , pasos , numPasos ,
        reservas , numReservas ,
        localidad , numLocalidades ,
        ruta , numRutas , numRutas2 ,
        0) ;
    break;
case 2:
    breakp=1;
    menuAdminViajes( usuario ,
        numUsuarios , vehiculo ,
        numVehiculos , viaje ,
        numViajes , pasos , numPasos ,
        reservas , numReservas ,
        localidad , numLocalidades ,
        ruta , numRutas , numRutas2 ,
        0) ;
    break;
}
}while( breakp==0);
}
else //caso contrario se encontraron coches
{
    do //el usuario elige uno de sus coches
    {
        system(" cls");
        printf(" Introduzca sus datos para completar
            el registro del viaje:\n");
        printf(" Elija el vehiculo que desea usar:\n
            \n");
        for ( i=0;i<coches ; i++)
        {
            color(0 , 3);
            printf(" Matricula: ");

```

```

        color(0, 15);
        printf("%s\n", vehiculo [ vec_vehiculo [ i
            ]]. id_mat);
        color(0, 2);
        printf("Numero_de_plazas: ");
        color(0, 15);
        printf("%s\n", vehiculo [ vec_vehiculo [ i
            ]]. num_plazas);
        color(0, 4);
        printf("Descripcion: ");
        color(0, 15);
        printf("%s\n\n", vehiculo [ vec_vehiculo [ i
            ]]. desc_veh);
    }

    printf("Introduzca la matricula del
        vehiculo que desea usar para el viaje:\n"
    );
    preguntar_veh(vehiculo, numVehiculos,
        id_vehiculo, &encontrado);
}while(encontrado==0); //se repite hasta que
encontrado sea distinto de 0
}

}

//Prototipo: void asignar_plazas(Estr_Vehiculo *, int ,
char *, char *);
//Precondicion: esta funcion debe recibir la
estructura "vehiculo" y su respectivo contador,
ademas debe recibir un dato de tipo entero que
representa la matricula del coche elegido por el
usuario anteriormente.
//Cabecera: esta funcion devuelve el numero de plazas
del coche elegido para que sean asignadas al coche.
//Postcondicion: esta funcion devuelve a traves del
puntero de tipo entero plazas el numero de plazas
del coche elegido para que luego sean asignadas al
viaje.

void asignar_plazas(Estr_Vehiculo *vehiculo, int
numVehiculos, char *id_vehiculo, char *plazas)

```

```

{
    int i , breakp=0;

    for ( i=0; i<numVehiculos&&breakp==0; i++) //busca en
        la estructura vehiculo el que coincide con el
        coche elegido anteriormente
    {
        if (strcmp( vehiculo [ i ] . id_mat , id_vehiculo )==0)
        {
            breakp=1;
            strcpy ( plazas , vehiculo [ i ] . num_plazas );
        }
    }
}

//Prototipo: void ida_vuelta (char *);
//Precondicion: recibe un char
//Postcondicion: devuelve un char que indica si el viaje
    es de ida-vuelta

void ida_vuelta (char *idavuelta)
{
    int opc=0, breakp=0;

    do //no se sale del bucle hasta que el usuario no
        haya elegido la opcion correcta
    {
        system ("cls");
        printf ("Seleccione el tipo de viaje que quiere
            :\n");
        printf (" (1) Ida.\n");
        printf (" (2) Vuelta.\n");
        fflush (stdin);
        scanf ("%1i", &opc);
        switch (opc)
        {
            case 1:
                breakp=1;
                strcpy ( idavuelta , "ida" );
                break;
            case 2:

```

```

        breakp=1;
        strcpy( idavuelta , "vuelta" );
        break;
    }
} while( breakp==0);
}

//Prototipo: void verificar_viaje(Estr_Viaje *, int ,
char *, char *, char *, char *, int *)
//Precondicion: esta funcion recibe la estructura viajes
inicializada y su contador, junto con datos de tipo
caracter como la id del viaje que se ha creado, la
matricula
//la fecha, hora de inicio y fin
//Cabeza: el objetivo de esta funcion es el de
verificar si los datos del viaje que se esta creando
no coinciden con los de un viaje creado previamente
por el usuario
//Postcondicion: devuelve un puntero de tipo entero "rp"
" que indica si el viaje creado ya se habia creado
previamente.

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

    for ( i=0; i<numViajes&&brkp==0; i++ ) //recorre la
        estructura viajes hasta que brkp es distinto de
        0
    {
        if ( strcmp( mat , viaje [ i ]. id_mat )==0 ) //verifica si
            el coche seleccionado es el mismo que el de
            otro viaje
        {
            if ( strcmp( fecha , viaje [ i ]. f_inic )==0 ) //
                verifica si las fechas son las mismas
            {
                if ( strcmp( viaje [ i ]. estado , "finalizado" )
                    !=0 && strcmp( viaje [ i ]. estado , "
```

```

        anulado")!=0)//verifica si el viaje
esta activo
{
    (*rp)=1;
    brkp=1;
}
}
}

}

//Prototipo: void altaReserva(Estr_Usuario *, int ,
Estr_Vehiculo *, int , Estr_Viaje *, int , Estr_Pasos
*, int , Estr_Reservas *, int , int );
//Precondicion://Precondicion:se necesitara la variable
numUsuarios que respresenta al usuario en la
estructura Usuarios ,
//Ademas se necesitaran las estructuras de "vehiculos",
"viajes", "pasos", "reservas", "localidad", "rutas"
", con sus respectivos contadores .
//Cabecera: esta funcion en dependencia de la fecha
elegida por el usuario y su localidad le seran
mostrados todos lo viajes disponibles impidiendole
elegir los viajes ya reservados previamente o
creados por el , y en caso de que
//no haya ningun viaje que pase por su localidad no se
le mostrara ningun viaje y volvera al menu.
//Postcondicion:en el fichero reserva se guardara la id
del viaje que el usuario haya elegido y su id y se
restara una plaza al viaje , dato que tambine se
modificara en el fichero

void altaReserva(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 , int num_user )
{
    int *vec=NULL, x , loc , i , seleccion=0,conf1=0,conf
    =0,plazas ;
    char fecha [11] , num_plazas [2] ;

```

```

do //se repite el bucle hasta que los datos
    ingresados esten correctos
{
    leer_dia (fecha);
    encontrarViajesReservas (usuario , viaje ,
        numViajes , pasos , numPasos , num_user , fecha ,
        &vec , &x , &loc);

    if( loc==0) //verifica si no existen viajes en
        tu localidad
    {
        printf ("No_hay_viajes_disponibles_en_tu_
            localidad_(%s),_para_fechas_futuras.\n" ,
            usuario [ num_user ]. localidad );
        system ("PAUSE" );
        conf=1;
    } //caso contrario existen viajes en tu
        localidad
    else if( vec==NULL) //verifica si existen viajes
        para la fecha que especifico el usuario
    {
        printf ("No_hay_viajes_disponibles_para_esa_
            fecha_(%s),_prueba_con_otra_fecha.\n" ,
            fecha );
        do
        {
            printf ("Desea_reservar_un_viaje_para_
                otra_fecha:\n(1)SI\n(2)NO\n" );
            scanf ("%i",&i );
            switch( i )
            {
                case 1:
                    conf=0;
                    break;
                case 2:
                    conf=1;
                    break;
            }
        }while( i !=1 && i !=2 );
    }
}

```

```

else //caso contrario existen viajes para la
      fecha
{
    do
    {
        system("cls");
        printf("Viajes disponibles para el %s\n"
               ,fecha);
        for(i=0;i<x;i++) //se muestran los
                           datos de los viajes
        {
            color(15, 0);
            printf("VIAJE-%i :\n" , i+1);
            color(0, 3);
            printf("ID del viaje : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [ i ] ].
                   id_viaje);
            color(0, 3);
            printf("Hora de partida : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [ i ] ]. h_inic
                   );
            color(0, 3);
            printf("Hora de llegada : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [ i ] ]. h_fin)
                   ;
            color(0, 3);
            printf("Tipo : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [ i ] ].
                   ida_vuelta);
            color(0, 3);
            printf("Estado : ");
            color(0, 15);
            printf("%s\n" , viaje [ vec [ i ] ]. estado
                   );
            color(0, 3);
            printf("Numero de plazas : ");
            color(0, 15);

```

```

        printf("%s\n", viaje [ vec [ i ] ] .
            plazas_libre);
        color(0, 3);
        printf("Precio:\n");
        color(0, 15);
        printf("%s-euros\n\n", viaje [ vec [ i
            ] ]. precio);
    }
    printf("Elija-un-viaje:\n");
    scanf("%i",&seleccion);

}while(seleccion>x || seleccion<1); //  

comprueba que el usuario haya elegido un  

viaje que se encuentre en el rango de  

opciones

verificar_reserva(usuario , vehiculo ,
numVehiculos , viaje , reservas ,
numReservas , vec , seleccion -1, &conf1 ,
num_user);

if(conf1==0) //verifica que el viaje no  

haya sido reservado previamente por el  

usuario o creado por este  

{
    mostrar_poblaciones(viaje , numViajes ,
pasos , numPasos , vec , seleccion -1, &
conf);

if(conf==2) //verifica que el usuario  

eligio la opcion de reservar este  

viaje  

{
    plazas=atoi(viaje [ vec [ seleccion
-1]]. plazas_libre);
    plazas--;
    sprintf(num_plazas,"%01d",plazas);
    strcpy(viaje [ vec [ seleccion -1]].
plazas_libre , num_plazas);
    if(plazas==0)//verifica si las  

plazas del viaje seleccionado

```

```

    son cero para cambiar su estado
    a cerrado
{
    strcpy( viaje [ vec [ seleccion -1] ]. 
        estado , " cerrado " );
}
actualizarViaje( viaje , numViajes );
guardarPasajero( usuario , viaje ,
    reservas , numReservas , vec ,
    seleccion -1, num_user );
}
}
}
}while( conf==0);
}

//Prototipo: void mostrar_poblaciones(Estr_Viaje *, int
//, Estr_Pasos *, int , int *, int , int *);
//Precondicion: la funcion recibira la estructura viajes
//con su contador, lo mismo con la estructura pasos,
//ademas de un vector de tipo entero con un dato (
//reserva) que es una posicion del vector
//Cabeza: esta funcion muestra las poblaciones del
//viaje elegido por el usuario
//Postcondicion: devuelve un entero conf que indica si
//el usuario desea elegir un nuevo viaje

void mostrar_poblaciones( Estr_Viaje *viaje , int
    numViajes , Estr_Pasos *pasos , int numPasos , int *vec
    , int reserva , int *conf)
{
    int i , opc ;

    system(" cls ");

    printf(" Viaje _seleccionado _para _el _%s \n " , viaje [ vec
        [ reserva ] ]. f_inic );
    color(0, 3);
    printf(" ID _del _viaje : _" );
    color(0, 15);
    printf("%s \n " , viaje [ vec [ reserva ] ]. id_viaje );
}

```

```

color(0, 3);
printf("Hora_de_partida: ");
color(0, 15);
printf("%s\n", viaje [ vec [ reserva ] ]. h_inic );
color(0, 3);
printf("Hora_de_llegada: ");
color(0, 15);
printf("%s\n", viaje [ vec [ reserva ] ]. h_fin );
color(0, 3);
printf("Tipo: ");
color(0, 15);
printf("%s\n", viaje [ vec [ reserva ] ]. ida_vuelta );
color(0, 3);
printf("Estado: ");
color(0, 15);
printf("%s\n", viaje [ vec [ reserva ] ]. estado );
color(0, 3);
printf("Numero_de_plazas: ");
color(0, 15);
printf("%s\n", viaje [ vec [ reserva ] ]. plazas_libre );
color(0, 3);
printf("Precio: ");
color(0, 15);
printf("%s_euros\n\n", viaje [ vec [ reserva ] ]. precio );

printf("Los_pasos_del_viaje_son:\n");
for( i=0; i<numPasos; i++) //muestra todos los pasos
    del viaje
{
    if(strcmp( viaje [ vec [ reserva ] ]. id_viaje , pasos [ i
        ]. id_viaje )==0)
    {
        printf("%s ", pasos [ i ]. poblacion );
    }
}
printf("\n\n");

do{
    printf("Desea_reservar este_viaje :\n(1) SI\n(2)
        NO\n");
    scanf("%i",&opc );

```

```

switch (opc)
{
    case 1:
        (*conf)=2; //el usuario desea reservar
                    el viaje
        break;
    case 2:
        (*conf)=0; //el usuario desea reservar
                    otro viaje
        break;
    default: printf("Introduzca la opcion correcta.\n");
}
}while(opc!=1 && opc!=2); //comprueba que el
                            usuario haya elegido un viaje que se encuentre
                            en el rango de opciones
}

//Prototipo: void guardarPasajero(Estr_Usuario *,
Estr_Viaje *, Estr_Reservas *, int , int *, int , int )
;

//Precondicion: esta funcion recibe la estructura "Usuarios" y "Reservas" con sus contadores, un vector de tipo entero, un entero que es una posicion del vector y un entero que es el usuario en la estructura.

//Cabecera: esta funcion guarda en el fichero reserva la id del viaje reservado y su id.

//Postcondicion: se guarda en el fichero reserva la id del viaje reservado y el del usuario

void guardarPasajero(Estr_Usuario *usuario , Estr_Viaje
*viaje , Estr_Reservas *reservas , int numReservas ,
int *vec , int reserva , int num_user)
{
    FILE *fp ;

    fp=fopen ("DATA/reservas.txt" , "a+" );
    if (fp==NULL)
    {

```

```

        printf("No se ha podido abrir el fichero "
               "reservas.txt.\n");
    return;
}
else
{
    fprintf(fp,"%s-%s\n", viaje [ vec [ reserva ] ]. 
            id_viaje , usuario [ num_user ]. id_usuario );
}
fclose (fp);
}

//Prototipo: void verificar_reserva(Estr_Usuario *,
          Estr_Vehiculo *, int , Estr_Viaje *, Estr_Reservas *,
          int , int *, int , int *, int );
//Precondicion: recibe las estructuras "Usuario", "
          Vehiculo","Reserva","Viaje" y sus respectivos
          contadores, ademas de un vector de tipo entero
//ya inicializado y una variable que indica las
          posiciones en este vector, y una variable que indica
          la posicion del usuario en la estructura usuario.
//Cabecera: esta funcion verifica que el usuario no
          haya reservado el viaje previamente o lo haya creado

//Postcondicion: devuelve un puntero n que indica si el
          usuario ha reservado anteriormente el viaje o lo ha
          creado .

void verificar_reserva(Estr_Usuario *usuario ,
                         Estr_Vehiculo *vehiculo , int numVehiculos ,
                         Estr_Viaje *viaje , Estr_Reservas *reservas , int
                         numReservas , int *vec , int reserva , int *n, int
                         num_user)
{
    int i;
    for(i=0;i<numVehiculos && *n==0;i++) //recorre la
        estructura vehiculos
    {
        if(strcmp(viaje [ vec [ reserva ] ]. id_mat , vehiculo [ i
                ]. id_mat)==0) //verifica que la matricula
            del viaje a reservar es la misma que la de

```

```

    algun coche del usuario
{
    if(strcmp( vehiculo [ i ]. id_usuario , usuario [
        num_user ]. id_usuario )==0)
    {
        printf("ERROR/No_puede_reservar_un_
                viaje_creado_porusted\n");
        system("PAUSE");
        *n=1;
    }
}
if((*n)!=1)
{
    for ( i=0;i<numReservas ; i++)
    {
        if(strcmp( reservas [ i ]. id_usuario , usuario [
            num_user ]. id_usuario )==0) //verifica que
            el usuario no haya reservado el mismo
            viaje
        {
            printf("ERROR/No_puede_reservar_un_
                    viaje_ya_reservado_previamente\n");
            system("PAUSE");
            *n=1;
        }
    }
}
}

```

4.1.8. Estructuras

Para ver la descripción del módulo puede ir a 3.2.8.

```
#ifndef ESTRUCTURAS_H_INCLUDED
#define ESTRUCTURAS_H_INCLUDED
```

```
typedef struct{
    char id_usuario [ 5 ];
    char nomb_usuario [ 21 ];
    char localidad [ 21 ];
```

```

    char perfil [14];
    char usuario [6];
    char contrasena [9];
} Estr_Usuario;

typedef struct{
    char id_mat [8];
    char id_usuario [5];
    char num_plazas [2];
    char desc_veh [51];
} Estr_Vehiculo;

typedef struct{
    char id_viaje [7];
    char id_mat [8];
    char f_inic [11];
    char h_inic [6];
    char h_fin [6];
    char plazas_libre [2];
    char ida_vuelta [6];
    char precio [5];
    char estado [11];
} Estr_Viaje;

typedef struct{
    char id_viaje [7];
    char poblacion [21];
} Estr_Pasos;

typedef struct{
    char siglas [4];
    char localidad [21];
} Estr_Localidad;

typedef struct{
    char localidad [21];
} Estr_Rutas;

typedef struct{
    char id_viaje [7];
    char id_usuario [5];
}

```

```

} Estr_Reservas;

#endif // ESTRUCTURAS_H INCLUDED

```

4.1.9. Fecha

Para ver la descripción del módulo puede ir a 3.2.9.

```

#include "fecha.h"

int dias_meses(int , int);

//Prototipo: void actualizarViajesEstado(Estr_Viaje *,
   int , Estr_Reservas *, int , Estr_Pasos *, int);
//Precondicion: Tener las estructuras "viaje" y "
reservas", con sus contadores.
//Postcondicion: Actualiza el estado de todos los
viajes, si se ha pasado la hora de inicio, pues se
pone en estado "Iniciado",
//mientras que si se ha excedido una hora desde la hora
de llegada, pues se establece en estado "Finalizado"
", y se eliminan sus pasos y reservas.

void actualizarViajesEstado(Estr_Viaje *viaje, int
    numViajes, Estr_Reservas *reservas, int numReservas,
    Estr_Pasos *pasos, int numPasos)
{
    FILE *fp;
    char *token=NULL, vec_fecha[11], vec_h_inic[6],
        vec_h_fin[6];
    int n=0, i=0, min_inic=0, hora_inic=0, min_fin=0,
        hora_fin=0, dia=0, mes=0, ano=0, encontrado=0;

    fp=fopen("DATA/viajes.txt","r+");

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

```

```

else
{
    for( i=0; i<numViajes; i++) //Nos desplazamos
        por todal la estructura "Viajse"
    {
        strcpy( vec_fecha , viaje[ i ].f_inic ); //
            Obtenemos la fecha de inicio , y la
            separamos en 3 enteros .
        token=strtok( vec_fecha , "/" );
        dia=atoi(token);
        token=strtok(NULL, "/");
        mes=atoi(token);
        token=strtok(NULL, "\n");
        ano=atoi(token);

        strcpy( vec_h_inic , viaje[ i ].h_inic ); //
            Obtenemos la hora de inicio , y la
            separamos en 2 enteros .
        token=strtok( vec_h_inic , ":" );
        hora_inic=atoi(token);
        token=strtok(NULL, "\n");
        min_inic=atoi(token);

        time_t ahora=time(NULL); //Esta seria la
            hora actual , en segundos .

    struct tm tiempo_introducido={0}; //
        Definimos estructura para poner anos
        desde 2023, meses y dias .
    tiempo_introducido.tm_year=ano-1900; //
        Metemos todos los enteros de la fecha y
        la hora de inicio , en la estructura .
    tiempo_introducido.tm_mon=mes-1;
    tiempo_introducido.tm_mday=dia;
    tiempo_introducido.tm_hour=hora_inic -1;
    tiempo_introducido.tm_min=min_inic;

        time_t tiempo_introducido_segundos=mktime(&
            tiempo_introducido); //Transforma la
            estructura de (dia/mes/ano y hora:
            segundos) a segundos .
}

```

```

if( tiempo_introducido_segundos<ahora) //Si
    la fecha actual es posterior a la fecha
    y hora de partida del viaje .
{
    if(strcmp( viaje [ i ]. estado , " abierto" )
        ==0||strcmp( viaje [ i ]. estado , "
        cerrado")==0) //Si el viaje esta en
        " abierto " o " cerrado " .
    {
        strcpy( viaje [ i ]. estado , " iniciado" )
            ; //Se le asigna como estado "
            iniciado " .
        encontrado=1;
    }

    if(strcmp( viaje [ i ]. estado , " iniciado" )
        ==0) //Si el viaje esta en "iniciado "
        .
    {
        strcpy( vec_h_fin , viaje [ i ]. h_fin );
            //Obtenemos la hora de llegada ,
            y la separamos en 2 enteros .
        token=strtok( vec_h_fin , ":" );
        hora_fin=atoi( token );
        token=strtok( NULL, "\n" );
        hora_inic=atoi( token );

        struct tm tiempo_introducido={0};
            //Definimos estructura para
            poner anos desde 2023, meses y
            dias .
        tiempo_introducido.tm_year=ano
            -1900; //Metemos todos los
            enteros de la fecha y la hora de
            llegada con 1 hora mas, en la
            estructura .
        tiempo_introducido.tm_mon=mes-1;
        tiempo_introducido.tm_mday=dia ;
        tiempo_introducido.tm_hour=hora_fin
            ;
    }
}

```

```

tiempo_introducido.tm_min=min_fin;

time_t tiempo_introducido_segundos=
    mktime(&tiempo_introducido); //  

Transforma la estructura de (dia  

/mes/ano y hora:segundos) a  

segundos.

if(tiempo_introducido_segundos<
    ahora) //Si la fecha actual es  

posterior a la fecha y hora de  

llegada + 1 hora, del viaje.
{
    strcpy(viaje[i].estado, "  

        finalizado"); //Se le asigna  

como estado "finalizado".
    encontrado=1;
    eliminarPasos(pasos, numPasos,  

        viaje[i].id_viaje); //  

Eliminamos los pasos del  

viaje.
    eliminarReservas(reservas,  

        numReservas, viaje[i].  

        id_viaje); //Eliminamos las  

reservas del viaje.
}
}

if(encontrado==1) //Si ha habido algun cambio,  

se imprime en el fichero.
{
    do{
        fprintf(fp, "%s-%s-%s-%s-%s-%s-%s\n",
            viaje[n].id_viaje, viaje[n].id_mat,  

            viaje[n].f_inic, viaje[n].h_inic, viaje[  

                n].h_fin, viaje[n].plazas_libre, viaje[n].  

                ida_vuelta, viaje[n].precio, viaje[n].  

                estado);
    n++;
}

```

```

}while(n<numViajes-1);
printf(fp, "%s-%s-%s-%s-%s-%s-%s-%s",
      viaje[n].id_viaje, viaje[n].id_mat,
      viaje[n].f_inic, viaje[n].h_inic, viaje[n].
      h_fin, viaje[n].plazas_libre, viaje[n].
      ida_vuelta, viaje[n].precio, viaje[n].
      estado);
}
}

fclose(fp);
}

//Prototipo: void leerFecha(char *, char *, char *);
//Precondicion: Tener los cadenas "fecha", "h_inic" y "
h_fin" inicializados.
//Postcondicion: Leer la fecha, hora de inicio y hora
de llegada, y comprobar si son validas, y
posteriores a las actuales. Despues, se introducen
en su respectiva cadena.

void leerFecha(char fecha[11], char h_inic[6], char
               h_fin[6]) //DD/MM/AAAA
{
    int min_inic=0, hora_inic=0, min_fin=0, hora_fin=0,
         dia=0, mes=0, ano=0, encontrado=0, encontrado3
         =0;
    char *token=NULL;

    leer_dia(fecha); //Usamos la funcion "leer_dia",
                      para leer la fecha, y ver si es valida.

    token=strtok(fecha, "/"); //Desglosamos la fecha en
                               3 enteros, para introducirlos en la estructura,
                               al pedir la hora.
    dia=atoi(token);
    token=strtok(NULL, "/");
    mes=atoi(token);
    token=strtok(NULL, "\n");
    ano=atoi(token);
}

```

```

while(encontrado3==0)
{
    sprintf( fecha , " %02i/%02i/%04i" , dia , mes , ano )
        ; //Se introduce los enteros en formato DD/
        MM/AAAA en la cadena "fecha".
    system("cls");
    printf("La_fecha_introducida_es: %s\n" , fecha );

    printf("Introduzca_la_hora_de_inicio_(HH:MM)_y_
        la_hora_de_llegada_(HH:MM)_de_su_viaje:\n" );
    printf("Hora_de_inicio:\n");

    encontrado=0;
    while(encontrado==0) //Lee el entero "hora_inic"
        ", hasta que sea correcto .
    {
        printf("Hora:");
        fflush(stdin);
        scanf("%2d" , &hora_inic);

        if( hora_inic <0|| hora_inic >23)
        {
            while( getchar ()!= '\n' );
        }
        else
        {
            encontrado=1;
        }
    }

    encontrado=0;
    while(encontrado==0) //Lee el entero "min_inic"
        ", hasta que sea correcto .
    {
        printf("Minutos:");
        fflush(stdin);
        scanf("%2d" , &min_inic);

        if( min_inic <0|| min_inic >59)
        {
            while( getchar ()!= '\n' );
        }
    }
}

```

```

    }
else
{
    encontrado=1;
}
}

sprintf(h_inic , "%02i:%02i" , hora_inic ,
min_inic); //Se introduce los enteros en
formato MM:SS en la cadena "h_inic"

printf("Hora_de_llegada:\n");

encontrado=0;
while(encontrado==0) //Lee el entero "hora_fin"
", hasta que sea correcto.
{
    printf("Hora:");
    fflush(stdin);
    scanf("%2d" , &hora_fin);

    if( hora_fin <0|| hora_fin >23)
    {
        while( getchar ()!= '\n' );
    }
    else
    {
        encontrado=1;
    }
}

encontrado=0;
while(encontrado==0) //Lee el entero "min_fin",
hasta que sea correcto .
{
    printf("Minutos:");
    fflush(stdin);
    scanf("%2d" , &min_fin);

    if( min_fin <0|| min_fin >59)
    {

```

```

        while( getchar ()!= '\n' );
    }
else
{
    break;
}
}

sprintf(h_fin , "%02i:%02i" , hora_fin , min_fin);
//Se introduce los enteros en formato MM:SS
en la cadena "h_fin"

if( hora_fin>hora_inic || min_fin>min_inic ) //Se
compara si la hora y min fin son mayores a
las de inicio .
{
    time_t ahora=time(NULL); //Esta seria la
    hora actual , en segundos .

struct tm tiempo_introducido={0}; //
definimos estructura para poner anos
desde 2023, meses y dias .
tiempo_introducido.tm_year=ano-1900; //
Metemos todos los enteros de la hora de
inicio , en la estructura .
tiempo_introducido.tm_mon=mes-1;
tiempo_introducido.tm_mday=dia ;
tiempo_introducido.tm_hour=hora_inic -1;
tiempo_introducido.tm_min=min_inic ;

time_t tiempo_introducido_segundos=mktime(&
tiempo_introducido); //Transforma la
estructura de (dia/mes/ano y min/seg) a
segundos .

if(tiempo_introducido_segundos<ahora) //
Comparamos si la fecha y hora
introducidas son anteriores a la fecha y
hora actual .
{

```

```

        printf("La_fecha_y_hora_de_inicio_
            ingresada_es_anterior_a_la_fecha_y_
            hora_actual.\n");
        system("PAUSE");
        system("cls");
    }
    else //Si son posteriores, se cumple todo.
    {
        system("cls");
        printf("La_fecha_introducida_es: %s\n",
            fecha);
        printf("La_hora_de_inicio_introducida_
            es: %s\n", h_inic);
        printf("La_hora_de_llegada_introducida_
            es: %s\n", h_fin);
        system("PAUSE");
        encontrado3=1;
    }
}
else //Imprime aviso, si la hora de llegada no
    es posterior.
{
    printf("La_hora_de_llegada_es_anterior_a_la_
        _hora_de_inicio.\n");
    system("PAUSE");
    system("cls");
}
}

//Prototipo: void leer_dia(char *);
//Precondicion: Tener la cadena "fecha" inicializada.
//Postcondicion: Leer la fecha, y comprobar si es
    valida, y posterior a las actual. Si esto se cumple,
    se guarda en la cadena.

void leer_dia(char fecha[11]) //DD/MM/AAAA
{
    int dia=0, mes=0, ano=0, maxdia=0, encontrado=0,
    encontrado2=0;
}

```

```

while(encontrado==0) //Hacer esto hasta que se
    encuentre una fecha valida.
{
    system("cls");
    printf("Introduzca la fecha (DD/MM/AAAA) en la
        que quiere realizar su viaje :\n");

    encontrado=0;
    while(encontrado==0) //Lee el entero "dia",
        hasta que sea correcto.
    {
        printf("Dia:");
        fflush(stdin);
        scanf("%d", &dia);

        if(dia<1||dia>31)
        {
            while(getchar()!='\n');
        }
        else
        {
            encontrado=1;
        }
    }

    encontrado=0;
    while(encontrado==0) //Lee el entero "mes",
        hasta que sea correcto.
    {
        printf("Mes:");
        fflush(stdin);
        scanf("%d", &mes);

        if(mes<1||mes>12)
        {
            while(getchar()!='\n');
        }
        else
        {
            encontrado=1;
        }
    }
}

```

```

}

encontrado=0;
while(encontrado==0) //Lee el entero "ano",
    hasta que sea correcto.
{
    printf("Ano:");
    fflush(stdin);
    scanf("%d",&ano);

    if(ano<2023)
    {
        while(getchar()!='\n');

    }
    else
    {
        encontrado=1;
    }
}
maxdia=dias_meses(mes,ano); //Llama a funcion
    maxdia, para saber si el ano es bisiesto, o
    si el mes tiene 28, 29, 30 o 31 dias.

if(dia<1||dia>maxdia||mes<1||mes>12||ano<1)
{
    printf("La_fecha_no_es_valida.\n");
    system("PAUSE");
    system("cls");
}
else
{
    time_t ahora=time(NULL); //Esta seria la
        hora actual, en segundos.

struct tm tiempo_introducido={0}; //
    definimos estructura para poner anos
    desde 2023, meses y dias.
tiempo_introducido.tm_year=ano-1900; //
    Metemos todos los enteros de la fecha,
    en la estructura.
tiempo_introducido.tm_mon=mes-1;

```

```

tiempo_introducido.tm_mday=dia+1;

time_t tiempo_introducido_segundos=mktime(&
    tiempo_introducido); //Transforma la
    estructura de (dia/mes/ano) a segundos.

if(tiempo_introducido_segundos<ahora) //Si
    la fecha actual es posterior a la fecha
    introducida.
{
    printf("La_fecha_inglesada_es_anterior_
        a_la_fecha_actual.\n");
    system("PAUSE");
    system("cls");
}
else //Si la fecha no es correcta, vuelve a
    pedir la fecha.
{
    encontrado2=1;
}
}
sprintf(fecha, "%02i/%02i/%04i", dia, mes, ano); // 
    Se introduce los enteros en formato DD/MM/AAAA
    en la cadena "fecha".
system("cls");
printf("La_fecha_introducida_es: %s\n", fecha);
system("PAUSE");
}

//Prototipo: int dias_meses(int, int);
//Precondicion: Tener los enteros "mes" y "ano"
    inicializados.
//Postcondicion: Devuelve el numero de dias que hay en
    cada mes del año. Tambien se comtemplan los años
    bisiestos.

int dias_meses(int mes, int ano)
{
    int dias=31; //suponemos que hay 31 dias.

```

```

if (mes==4||mes==6||mes==9||mes==11)
{
    dias=30; //abril , junio , septiembre y noviembre
              tienen 30 dias
}
else if (mes==2)
{
    if ((ano%4==0&&ano%100!=0) || ano%400==0)
    {
        dias=29; //febrero tiene 29 dias en anos
                  bisestos
    }
    else
    {
        dias=28; //febrero tiene 28 dias en anos no
                  bisestos
    }
}

return dias;
}

```

4.1.10. Leer

Para ver la descripción del módulo puede ir a 3.2.10.

```

#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)); //A
    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
fichero 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 fichero "
           "vehiculos.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)); //A
        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);
}

```

4.1.11. Listar

Para ver la descripción del módulo puede ir a 3.2.11.

```
#include "listar.h"
```

```
//Prototipo: void listarUsuarios(Estr_Usuario *, int);
//Precondicion: Tener la estructura "usuario"
inicializada, junto a su contador.
```

```

//Postcondicion: Dar una lista de todos los usuarios,
que hay en la base de datos.

void listarUsuarios( Estr_Usuario *usuario , int
    numUsuarios)
{
    int counter;

    if(numUsuarios!=0) //Si hay usuarios en el sistema.
    {
        printf("LISTADO_DE_USUARIOS:\n");
        for(counter=0;counter<numUsuarios;counter++) //
            Pasa por todos los usuarios, usando el
            contador "numUsuarios", como delimitador.
        {
            color(0,14);
            printf("    Usuario %i :\n", counter+1); //
                Imprime todos los datos de cada usuario.
            color(0,3);
            printf("    ID: ");
            color(0,15);
            printf("%s", usuario[counter].id_usuario);
            color(0,3);
            printf("    Nombre_de_usuario: ");
            color(0,15);
            printf("%s", usuario[counter].nomb_usuario);
            ;
            color(0,3);
            printf("    Localidad: ");
            color(0,15);
            printf("%s", usuario[counter].localidad);
            color(0,3);
            printf("    Perfil: ");
            color(0,15);
            printf("%s", usuario[counter].perfil);
            color(0,3);
            printf("    Usuario: ");
            color(0,15);
            printf("%s\n", usuario[counter].usuario);
        }
    }
}

```

```

else
{ //Si el delimitador "numUsuarios" es igual a 0,
  decimos que no hay usuarios registrados.
  color(0,3);
  printf("\tNo hay usuarios registrados.\n");
}

color(0,15);
system("PAUSE");
}

//Prototipo: void listarVehiculos(Estr_Usuario *, int ,
Estr_Vehiculo *, int , int);
//Precondicion: Tener la estructura "usuario" y "
vehiculo" inicializadas , junto a sus contadores.
//Postcondicion: Dar una lista de todos los vehiculos ,
de cada usuario , que hay en la base de datos.

void listarVehiculos(Estr_Usuario *usuario , int
  numUsuarios , Estr_Vehiculo *vehiculo , int
  numVehiculos , int i)
{
  int contador_veh_usuario , counter , counter2;

  if(numVehiculos!=0) //Si hay vehiculos en el
    sistema .
  {
    printf("LISTADO_DE_VEHICULOS:\n");
    for(counter=0;counter<numUsuarios ;counter++) //"
      Pasa por todos los usuarios , usando el
      contador "numUsuarios" , como delimitador .
    {
      contador_veh_usuario=0; //Resetea el numero
        de vehiculos al empezar un usuario .
      color(0,14);
      printf(">_Vehiculos_de_%s_|_ID:_%s\n",
        usuario [counter ].nomb_usuario , usuario [
        counter ].id_usuario );
      for(counter2=0;counter2<numVehiculos ;
        counter2++) //Pasa por todos los
        vehiculos , usando el contador "

```

```

    numVehiculos ", como delimitador .
{
    if (strcmp (usuario [counter] . id_usuario ,
                vehiculo [counter2] . id_usuario )==0)
        //Si la id del usuario propietario del
        //vehiculo coincide con la id del
        //usuario actual , se imprime los datos
        //del vehiculo .
    {
        color (0,4) ;
        printf ("-----Vehiculo-----") ; //Imprime
        // todos los datos de cada vehiculo

        color (0,15) ;
        printf ("%i" , (contador_veh_usuario)
            +1) ;
        color (0,4) ;
        printf (" :\t") ;
        color (0,3) ;
        printf ("-----Matricula :-----") ;
        color (0,15) ;
        printf ("%s" , vehiculo [counter2] .
            id_mat) ;
        color (0,3) ;
        printf (" | Numero_de_plazas :-----") ;
        color (0,15) ;
        printf ("%s" , vehiculo [counter2] .
            num_plazas) ;
        color (0,3) ;
        printf (" | Descripcion :-----") ;
        color (0,15) ;
        printf ("%s\n" , vehiculo [counter2] .
            desc_veh) ;
        contador_veh_usuario++ ; //Se va
        //sumando 1 por vehiculo
        //encontrado .
    }
}
if (contador_veh_usuario==0) //Si no se ha
//encontrado ningun vehiculo para el
//usuario .

```

```

        {
            color(0,3);
            printf("No posee vehiculos registrados.\n");
        }
    }
else
{
    color(0, 15);
    printf("\tNo hay vehiculos registrados.\n");
}
color(0,15);
system("PAUSE");
}

//Prototipo: void listarAdminViajes(Estr_Usuario *, int
, Estr_Vehiculo *, int, Estr_Viaje *, int, int, int)
;
//Precondicion: Tener la estructura "usuario", "
"vehiculo" y "viaje" inicializadas, junto a sus
contadores.
//Postcondicion: Si n=0, da una lista de todos los
viajes, de cada usuario, que hay en la base de datos
, y si n=1, da una lista de los viajes abiertos que
hay de cada usuario.

void listarAdminViajes(Estr_Usuario *usuario, int
    numUsuarios, Estr_Vehiculo *vehiculo, int
    numVehiculos, Estr_Viaje *viaje, int numViajes, int
    i, int n, int *num)
{
    int contador_viaje, *vec=NULL, *vec_viaje=NULL,
        num_v, m, j, id, counter;

    if(numViajes!=0) //Si hay viajes en el sistema.
    {
        printf("LISTADO DE VIAJES:\n");
        for(counter=0;counter<numUsuarios;counter++) //
            Pasa por todos los usuarios, usando el
            contador "numUsuarios", como delimitador.
    }
}

```

```

{
    color(0,14);
    printf(">_Viajes_de_%s_|_ID:_%s\n", usuario
        [counter].nom_b_usuario, usuario[counter].
        id_usuario);
    id=atoi(usuario[counter].id_usuario); //  

        Pasamos la id del usuario a entero, para  

        que pueda entrar en la funcion.
    id--;
    encontrarVehiculos(usuario, vehiculo,
        numVehiculos, &vec, &num_v, id); //  

        Encontramos todos los vehiculos del  

        usuario.
if(num_v!=0) //Si el usuario tiene algun  

vehiculo.
{
    contador_viaje=0;
    for(j=0;j<num_v; j++) //Se encuentran  

todos los viajes que tiene con todos  

sus vehiculos.
{
    encontrarViajes(vehiculo,
        numVehiculos, viaje, numViajes,
        vehiculo[vec[j]].id_mat, &
        vec_viaje, &contador_viaje, n);
}
for(m=0; m<contador_viaje; m++) //  

Imprimos todos los viajes que tiene.
{
    color(0,4);
    printf("____Viaje_"); //Imprime  

todos los datos de cada vehiculo  

    .
    color(0,15);
    printf("%i", m+1);
    color(0,4);
    printf(": \t");
    color(0,3);
    printf("ID: ");
    color(0,15);
}

```

```

printf("%s", viaje [ vec_viaje [m] ] .
      id_viaje );
color(0,3);
printf(" | Matricula : ");
color(0,15);
printf("%s", viaje [ vec_viaje [m] ] .
      id_mat );
color(0,3);
printf(" | Fecha de partida : ");
color(0,15);
printf("%s", viaje [ vec_viaje [m] ] .
      f_inic );
color(0,3);
printf(" | Hora de partida : ");
color(0,15);
printf("%s", viaje [ vec_viaje [m] ] .
      h_inic );
color(0,3);
printf(" |\n----- Hora de
      llegada : ");
color(0,15);
printf("%s", viaje [ vec_viaje [m] ] .
      h_fin );
color(0,3);
printf(" | Plazas libres : ");
color(0,15);
printf("%s", viaje [ vec_viaje [m] ] .
      plazas_libre );
color(0,3);
printf(" | Ida/Vuelta : ");
color(0,15);
printf("%s", viaje [ vec_viaje [m] ] .
      ida_vuelta );
color(0,3);
printf(" | Precio : ");
color(0,15);
printf("%s", viaje [ vec_viaje [m] ] .
      precio );
color(0,3);
printf(" | Estado : ");
color(0,15);

```

```

        printf("%s\n\n", viaje [ vec_viaje [m
            ] ].estado);
        *num=1;
    }
    if(contador_viaje==0) //Si no tiene
        viajes .
    {
        color(0,3);
        printf("....No posee viajes -
            registrados.\n");
    }
}
else //Si no tiene coches .
{
    color(0,3);
    printf("....No posee vehiculos ni -
        viajes registrados.\n");
}
}
else //Si no hay viajes abiertos en el sistema .
{
    color(0, 15);
    printf("Ningun usuario del sistema tiene un-
        viaje.\n");
}
color(0,15);
system("PAUSE");
}

//Prototipo: void listarAdminVehiculoViajes(
    Estr_Usuario *, int , Estr_Vehiculo *, int ,
    Estr_Viaje *, int , int );
//Precondicion: Tener la estructura "usuario" , " "
    "vehiculo" y "viaje" inicializadas , con sus
    contadores .
//Postcondicion: Dar una lista de todos los viajes que
    ha realizado un vehiculo .

void listarAdminVehiculoViajes(Estr_Usuario *usuario ,
    int numUsuarios , Estr_Vehiculo *vehiculo , int

```

```

    numVehiculos , Estr_Viaje *viaje , int numViajes , int
    i )
{
    int x=0, m=0, z=0, encontrado=0, *vec=NULL;
    char mat[8];

    if( numVehiculos!=0)
    {
        while(encontrado==0) //Hasta que no se
            encuentre un vehiculo existente en el
            sistema .
        {
            listarVehiculos( usuario , numUsuarios ,
                vehiculo , numVehiculos , i); //Listamos
                todos los vehiculos de cada usuario .
            printf("Introduzca la matricula de un
                vehiculo para obtener los viajes que ha
                realizado.\n");
            pregunta(mat , 8);

            for(z=0;z<numVehiculos;z++) //Nos
                desplazamos por toda la estructura "
                vehiculo ".
            {
                if(strcmp( vehiculo [z] . id_mat , mat)==0)
                    //Si la matricula introducida existe
                    en la base de datos .
                {
                    encontrado=1;
                }
            }
            if(encontrado==0) //Si el vehiculo no
                existe en el sistema .
            {
                printf("El vehiculo con matricula %s no
                    existe.\n" , mat);
                system("PAUSE");
            }
            system("cls");
        }
    }
}

```

```

encontrarViajes( vehiculo , numVehiculos , viaje ,
    numViajes , mat , &vec , &x , 0); //Encuentra
    todos los viajes que tiene el vehiculo.

if(x==0) //Si el vehiculo no tiene viajes .
{
    printf("El_vehiculo_con_matricula_%s , _no_ha
        _realizado_ningun_viaje.\n" , mat);
}
else
{
    printf("Los_viajes_realizados_por_el_
        _vehiculo_con_matricula_%s_son:\n" , mat);
    for(m=0; m<x; m++) //Imprimimos todos los
        viajes realizados con ese vehiculo .
    {
        color(15 , 0);
        printf("VIAJE_%i :\n" ,m+1);
        color(0 , 3);
        printf("___ID_del_viaje : " );
        color(0 , 15);
        printf("%s\n" , viaje [ vec [m] ]. id_viaje );
        color(0 , 3);
        printf("___Fecha_de_partida : " );
        color(0 , 15);
        printf("%s\n" , viaje [ vec [m] ]. f_inic );
        color(0 , 3);
        printf("___Hora_de_partida : " );
        color(0 , 15);
        printf("%s\n" , viaje [ vec [m] ]. h_inic );
        color(0 , 3);
        printf("___Hora_de_llegada : " );
        color(0 , 15);
        printf("%s\n" , viaje [ vec [m] ]. h_fin );
        color(0 , 3);
        printf("___Tipo : " );
        color(0 , 15);
        printf("%s\n" , viaje [ vec [m] ]. ida_vuelta
            );
        color(0 , 3);
        printf("___Precio : " );

```

```

        color(0, 15);
        printf("%s-euros\n", viaje [ vec [m] ]. 
            precio);
        color(0, 3);
        printf("...Estado: ...");
        color(0, 15);
        printf("%s\n\n", viaje [ vec [m] ]. estado);
    }
}
else
{
    printf("No hay ni vehiculos ni viajes 
        registrados.\n");
}

system("PAUSE");
}

//Prototipo: void listarLocalidades(Estr_Localidad *, int);
//Precondicion: Tener la estructura "localidad"
    inicializada, con su contador.
//Postcondicion: Imprimir una lista de 4 filas con
    todas las localidades.

void listarLocalidades(Estr_Localidad *localidad, int
    numLocalidades)
{
    int i=0, j=0, k=0;

    numLocalidades--; //Queremos quitarle la ultima
    linea que contiene a la Escuela Superior de
    Ingenieria (ESI), para que no se imprima por
    pantalla, junto a las localidades.

    k=numLocalidades/4; //Calcular filas necesarias.

    if(numLocalidades%4!=0) //Si al dividir las
    localidades entre 4, queda un resto, pues se
    aumenta una fila.

```

```

{
    k++;
}

for( i=0; i<k; i++) //Imprimimos la lista de
    ciudades en 4 columnas homogeneas.
{
    for( j=i; j<numLocalidades; j=j+k)
    {
        color(0, 3);
        printf("%s", localidad[ j ].siglas);
        color(0, 15);
        printf("-%-20s\t", localidad[ j ].localidad);
    }
    printf("\n");
}

//Prototipo: void listarReservas(Estr_Usuario *,
    Estr_Viaje *, int , Estr_Reservas *, int , int );
//Precondicion: Tener las estructuras inicializadas ,
    con sus contadores. Ademas, se necesita la variable
    "num_user", para saber a que usuario nos referimos.
//Postcondicion: Imprimir una lista con todos las
    reservas activas que tiene un usuario.

void listarReservas(Estr_Usuario *usuario , Estr_Viaje *
    viaje , int numViajes , Estr_Reservas *reservas , int
    numReservas , int num_user)
{
    int *vec=NULL, *vec_viaje=NULL, x=0, i;

    encontrarReservas(usuario , viaje , numViajes ,
        reservas , numReservas , num_user , &vec_viaje , &
        vec , &x); //encuentra las reservas del usuario
    que estan abiertas

    if( vec!=NULL)
    { //verifica que el usuario tiene reservas
        printf("LISTADO_DE_SUS_RESERVAS:\n");
        for( i=0;i<x ; i++)

```

```

{
    color(15, 0);
    printf("RESERVA %i :\n", i+1);
    color(0, 3);
    printf(" ---ID del viaje : ");
    color(0, 15);
    printf("%s\n", viaje[vec[i]].id_viaje);
    color(0, 3);
    printf(" ---Fecha de partida : ");
    color(0, 15);
    printf("%s\n", viaje[vec[i]].f_inic);
    color(0, 3);
    printf(" ---Hora de partida : ");
    color(0, 15);
    printf("%s\n", viaje[vec[i]].h_inic);
    color(0, 3);
    printf(" ---Hora de llegada : ");
    color(0, 15);
    printf("%s\n", viaje[vec[i]].h_fin);
    color(0, 3);
    printf(" ---Tipo : ");
    color(0, 15);
    printf("%s\n", viaje[vec[i]].ida_vuelta);
    color(0, 3);
    printf(" ---Precio : ");
    color(0, 15);
    printf("%s euros\n\n", viaje[vec[i]].precio
    );
}
}

else
{
    printf("No tiene viajes reservados.\n\n");
}
}

//Prototipo: void listarVehiculosUsuario(Estr_Usuario
*, Estr_Vehiculo *, int, int);
//Precondicion: Tener las estructuras inicializadas,
con sus contadores. Ademas, se necesita la variable
"num_user", para saber a que usuario nos referimos.

```

```

//Postcondicion: Imprimir una lista con todos los
vehiculos que tiene un usuario.

void listarVehiculosUsuario( Estr_Usuario *usuario ,
    Estr_Vehiculo *vehiculo , int numVehiculos , int
    num_user)
{
    int *vec=NULL, x=0, i ;

    encontrarVehiculos(usuario , vehiculo , numVehiculos ,
        &vec , &x , num_user); //encuentra los vehiculos
del usuario

    if( vec!=NULL)
    { //verifica que el usuario tiene vehiculos
        printf("LISTADO_DE_SUS_VEHICULOS:\n");
        for( i=0;i<x ; i++)
        {
            color(15 , 0);
            printf("VEHICULO-%i :\n" , i+1);
            color(0 , 3);
            printf("...Matricula: ");
            color(0 , 15);
            printf("%s\n" , vehiculo [ vec [ i ] ]. id_mat);
            color(0 , 3);
            printf("...Numero_de_plazas: ");
            color(0 , 15);
            printf("%s\n" , vehiculo [ vec [ i ] ]. num_plazas)
                ;
            color(0 , 3);
            printf("...Descripcion: ");
            color(0 , 15);
            printf("%s\n\n" , vehiculo [ vec [ i ] ]. desc_veh)
                ;
        }
    }
    else
    {
        printf("No_tiene_vehiculosRegistrados.\n\n");
    }
}

```

```

//Prototipo: void listarViajes(Estr_Usuario *,
    Estr_Vehiculo *, int, Estr_Viaje *, int, int);
//Precondicion: Tener las estructuras inicializadas,
    con sus contadores. Ademas, se necesita la variable
    "num_user", para saber a que usuario nos referimos.
//Postcondicion: Imprimir una lista con todos los
    viajes abiertos e iniciados que tiene un usuario.

void listarViajes(Estr_Usuario *usuario, Estr_Vehiculo
    *vehiculo, int numVehiculos, Estr_Viaje *viaje, int
    numViajes, int num_user)
{
    int *vec=NULL, *vec_coche=NULL, x=0, j, i, y=0;

    encontrarVehiculos(usuario, vehiculo, numVehiculos,
        &vec_coche, &x, num_user); //Encuentra todos
        los vehiculos del usuario.
    if( vec_coche!=NULL)
    {
        for(j=0;j<x;j++)
        {
            encontrarViajes(vehiculo, numVehiculos,
                viaje, numViajes, vehiculo[vec_coche[j]
                ].id_mat, &vec, &y, 3); //Encuentra
                todos los viajes abiertos, que no tienen
                plazas ocupadas, del usuario.
        }

        if( vec!=NULL)
        { //verifica que el usuario tiene reservas
            printf("LISTADO DE SUS VIAJES:\n");
            for(i=0;i<y;i++)
            {
                color(15, 0);
                printf("VIAJE-%i :\n", i+1);
                color(0, 3);
                printf(" --ID del viaje: ");
                color(0, 15);
                printf("%s\n", viaje[vec[i]].id_viaje);
                color(0, 3);
            }
        }
    }
}

```

```

        printf("...Estado : ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ i ] ]. estado );
        color(0, 3);
        printf("...Plazas-libres : ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ i ] ]. 
            plazas_libre );
        color(0, 3);
        printf("...Fecha_de_partida : ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ i ] ]. f_inic );
        color(0, 3);
        printf("...Hora_de_partida : ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ i ] ]. h_inic );
        color(0, 3);
        printf("...Hora_de_llegada : ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ i ] ]. h_fin );
        color(0, 3);
        printf("...Tipo : ");
        color(0, 15);
        printf("%s\n", viaje [ vec [ i ] ]. ida_vuelta
            );
        color(0, 3);
        printf("...Precio : ");
        color(0, 15);
        printf("%s_euros\n\n", viaje [ vec [ i ] ]. 
            precio );
    }
}
else
{
    printf("No tiene viajes registrados.\n\n");
}
else
{
    printf("No tiene ni vehiculos ni viajes 
        registrados.\n\n");
}

```

```

    }
}
```

4.1.12. Menús

Para ver la descripción del módulo puede ir a 3.2.12.

```
#include "menus.h"

//Prototipo: void menuPrincipal(Estr_Usuario *, int,
 Estr_Vehiculo *, int, Estr_Viaje *, int, Estr_Pasos
 *, int, Estr_Reservas *, int, Estr_Localidad *, int,
 Estr_Rutas **, int, int);
//Precondicion: Tener las estructuras inicializados,
 con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
 partir de la opcion que se escriba.

void menuPrincipal(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_Localidad *localidad
, int numLocalidades, Estr_Rutas **ruta, int
numRutas, int numRutas2)
{
    int opc;

    while(opc!=3)
    {
        leer_usuario(&usuario, &numUsuarios);
        system("cls");
        color(0, 6);
        printf("-----\n");
        printf(" _| _----| _| _----| _| _----| _| _----| _|\n");
        printf(" _----| _| _| _| _| _| _| _| _| _| _| _|\n");
        printf(" _| _----| _| ( _----| _| _----| _| _----| _|\n");
        printf(" ( _----| _| _| _| _| _| _| _| _| _| _| _|\n");
    }
}
```



```

                break;
            }
        }
    }

//Prototipo: void menuUsuario(Estr_Usuario *, int,
    Estr_Vehiculo *, int, Estr_Viaje *, int, Estr_Pasos
    *, int, Estr_Reservas *, int, Estr_Localidad *, int,
    Estr_Rutas **, int, int, int);
//Precondicion: Tener el entero "i", que nos indica la
    posicion del usuario que ha iniciado sesion en el
    sistema, en la estructura "usuario".
//Tambien necesitamos las estructuras inicializados,
    con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
    partir de la opcion que se escriba.

void menuUsuario(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_Localidad *localidad, int
    numLocalidades, Estr_Rutas **ruta, int numRutas, int
    numRutas2, int i)
{
    int opc;

    while(opc!=3)
    {
        system("cls");
        color(15, 0);
        printf("Hola-%s\n", usuario[i].nomb_usuario);
        color(0,15);
        printf("Que-quiere-ser?\n");
        color(0, 4);
        printf("(1) Pasajero.\n");
        color(0, 3);
        printf("(2) Conductor.\n");
        color(0,14);
        printf("(3) Volver.\n");
        color(0,15);
    }
}

```

```

fflush(stdin);
scanf("%i", &opc);
system("cls");
switch(opc)
{
    case 1:
        menuPasajero( usuario , numUsuarios ,
                      vehiculo , numVehiculos , viaje ,
                      numViajes , pasos , numPasos , reservas ,
                      numReservas , localidad ,
                      numLocalidades , ruta , numRutas ,
                      numRutas2 , i );
        break;
    case 2:
        menuConductor( usuario , numUsuarios ,
                        vehiculo , numVehiculos , viaje ,
                        numViajes , pasos , numPasos , reservas ,
                        numReservas , localidad ,
                        numLocalidades , ruta , numRutas ,
                        numRutas2 , i );
        break;
    case 3:
        menuPrincipal( usuario , numUsuarios ,
                        vehiculo , numVehiculos , viaje ,
                        numViajes , pasos , numPasos , reservas ,
                        numReservas , localidad ,
                        numLocalidades , ruta , numRutas ,
                        numRutas2 );
        break;
}
}
}

//Prototipo: void menuPasajero(Estr_Usuario *, int ,
//Estr_Vehiculo *, int , Estr_Viaje *, int , Estr_Pasos
//*, int , Estr_Reservas *, int , Estr_Localidad *, int ,
//Estr_Rutas **, int , int , int );
//Precondicion: Tener el entero "i", que nos indica la
//posicion del usuario que ha iniciado sesion en el
//sistema, en la estructura "usuario".

```

```

//Tambien necesitamos las estructuras inicializadas ,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones , a
partir de la opcion que se escriba .

void menuPasajero( 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_Localidad *localidad
    , int numLocalidades , Estr_Rutas **ruta , int
    numRutas , int numRutas2 , int i )
{
    int opc;

    while( opc!=3)
    {
        system("cls");
        color(15, 0);
        printf("Hola-%s,\n", usuario[ i ].nomb_usuario);
        color(0,15);
        printf("Que_quiere_ver?\n");
        color(0, 4);
        printf("(1) Perfil.\n");
        color(0, 3);
        printf("(2) Viajes.\n");
        color(0,14);
        printf("(3) Volver.\n");
        color(0,15);
        fflush(stdin);
        scanf("%i", &opc);
        system("cls");
        switch(opc)
        {
            case 1:
                menuPasajeroPerfil(usuario , numUsuarios
                    , localidad , numLocalidades , i );
                break;
            case 2:
                menuPasajeroViajes(usuario , numUsuarios
                    , vehiculo , numVehiculos , viaje ,

```

```

        numViajes , pasos , numPasos , reservas
        , numReservas , localidad ,
        numLocalidades , ruta , numRutas ,
        numRutas2 , i );
    break;
case 3:
    menuUsuario( usuario , numUsuarios ,
        vehiculo , numVehiculos , viaje ,
        numViajes , pasos , numPasos , reservas
        , numReservas , localidad ,
        numLocalidades , ruta , numRutas ,
        numRutas2 , i );
    break;
}
}
}

//Prototipo: void menuPasajeroPerfil(Estr_Usuario *,
int , Estr_Localidad *, int , int);
//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema, en la estructura "usuario".
//Tambien necesitamos las estructuras inicializados,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
partir de la opcion que se escriba.

void menuPasajeroPerfil( Estr_Usuario *usuario , int
    numUsuarios , Estr_Localidad *localidad , int
    numLocalidades , int i )
{
    int opc;

    while( opc!=5)
    {
        leer_usuario(&usuario , &numUsuarios);
        system("cls");
        color(0, 3);
        printf("ID_de_usuario: ");
        color(0, 15);
        printf("%s\n", usuario[ i ].id_usuario);
    }
}

```

```

color(0, 3);
printf("Nombre_completo: ");
color(0, 15);
printf("%s\n", usuario[ i ].nomb_usuario);
color(0, 3);
printf("Localidad_de_residencia: ");
color(0, 15);
printf("%s\n", usuario[ i ].localidad);
color(0, 3);
printf("Tipo_de_perfil: ");
color(0, 15);
printf("%s\n", usuario[ i ].perfil);
color(0, 3);
printf("Usuario: ");
color(0, 15);
printf("%s\n\n", usuario[ i ].usuario);
color(0, 15);
printf("Que_quiere_hacer?\n");
color(0, 2);
printf("(1) Modificar_nombre_completo.\n");
color(0, 4);
printf("(2) Modificar_localidad_de_residencia.\n");
color(0, 11);
printf("(3) Modificar_usuario.\n");
color(0, 6);
printf("(4) Modificar_contrasena.\n");
color(0, 14);
printf("(5) Volver.\n");
color(0, 15);
fflush(stdin);
scanf("%i", &opc);
system("cls");
switch(opc)
{
    case 1:
        modificarPerfilNombre(usuario,
                               numUsuarios, i);
        break;
    case 2:

```

```

        modificarPerfilLocalidad( usuario ,
            numUsuarios , localidad ,
            numLocalidades , i );
        break;
    case 3:
        modificarPerfilUsuario( usuario ,
            numUsuarios , i );
        break;
    case 4:
        modificarPerfilContrasena( usuario ,
            numUsuarios , i );
        break;
    case 5:
        return ;
        break;
    }
}
}

//Prototipo: void menuPasajeroViajes(Estr_Usuario *
usuario, int numUsuarios, Estr_Vehiculo *, int,
Estr_Viaje *, int, Estr_Pasos *, int, Estr_Reservas
*, int, Estr_Localidad *, int, Estr_Rutas **, int,
int, int);
//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema, en la estructura "usuario".
//Tambien necesitamos las estructuras inicializados,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
partir de la opcion que se escriba.

void menuPasajeroViajes( 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_Localidad *localidad
    , int numLocalidades , Estr_Rutas **ruta , int
    numRutas , int numRutas2 , int i )
{
    int opc ;

```

```

while( opc!=3)
{
    leer_vehiculo(&vehiculo , &numVehiculos );
    leer_viaje(&viaje , &numViajes );
    leer_pasos(&pasos , &numPasos );
    leer_reservas(&reservas , &numReservas );
    system(" cls");
    listarReservas( usuario , viaje , numViajes ,
                    reservas , numReservas , i );
    color(0,15);
    printf("Que_quiere_hacer?\n");
    color(0, 4);
    printf("(1) Reservar_viaje.\n");
    color(0, 3);
    printf("(2) Cancelar_viaje.\n");
    color(0,14);
    printf("(3) Volver.\n");
    color(0,15);
    fflush(stdin);
    scanf("%i" , &opc );
    system(" cls");
    switch(opc )
    {
        case 1:
            altaReserva( usuario , numUsuarios ,
                         vehiculo , numVehiculos , viaje ,
                         numViajes , pasos , numPasos , reservas
                         , numReservas , i );
            break;
        case 2:
            cancelarReserva( usuario , viaje ,
                            numViajes , reservas , numReservas , i )
                            ;
            break;
        case 3:
            menuPasajero( usuario , numUsuarios ,
                           vehiculo , numVehiculos , viaje ,
                           numViajes , pasos , numPasos , reservas
                           , numReservas , localidad ,
                           numLocalidades , ruta , numRutas ,

```

```

                numRutas2, i);
        break;
    }
}
//Prototipo: void menuConductor(Estr_Usuario *usuario,
int numUsuarios, Estr_Vehiculo *, int, Estr_Viaje *,
int, Estr_Pasos *, int, Estr_Reservas *, int,
Estr_Localidad *, int, Estr_Rutas **, int, int, int)
;
//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema, en la estructura "usuario".
//Tambien necesitamos las estructuras inicializados,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
partir de la opcion que se escriba.

void menuConductor(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_Localidad *localidad
, int numLocalidades, Estr_Rutas **ruta, int
numRutas, int numRutas2, int i)
{
    int opc;

    while(opc!=4)
    {
        system("cls");
        color(15, 0);
        printf("Hola_%s\n", usuario [ i ].nomb_usuario);
        color(0, 15);
        printf("Que_quiere_ver?\n");
        color(0, 4);
        printf("(1) Perfil.\n");
        color(0, 3);
        printf("(2) Vehiculos.\n");
        color(0, 2);
    }
}
```



```

//Prototipo: void menuConductorPerfil(Estr_Usuario *,
int , Estr_Localidad *, int , int);
//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema, en la estructura "usuario".
//Tambien necesitamos las estructuras inicializados,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
partir de la opcion que se escriba.

void menuConductorPerfil( Estr_Usuario *usuario , int
    numUsuarios , Estr_Localidad *localidad , int
    numLocalidades , int i )
{
    int opc;

    while( opc!=5)
    {
        leer_usuario(&usuario , &numUsuarios);
        system("cls");
        color(0, 3);
        printf("ID_de_usuario: ");
        color(0, 15);
        printf("%s\n", usuario[ i ]. id_usuario);
        color(0, 3);
        printf("Nombre_completo: ");
        color(0, 15);
        printf("%s\n", usuario[ i ]. nomb_usuario);
        color(0, 3);
        printf("Localidad_de_residencia: ");
        color(0, 15);
        printf("%s\n", usuario[ i ]. localidad);
        color(0, 3);
        printf("Tipo_de_perfil: ");
        color(0, 15);
        printf("%s\n", usuario[ i ]. perfil);
        color(0, 3);
        printf("Usuario: ");
        color(0, 15);
        printf("%s\n\n", usuario[ i ]. usuario);
    }
}

```

```

color(0,15);
printf("Que_quiere_hacer?\n");
color(0, 2);
printf("(1) Modificar_nombre_completo.\n");
color(0, 4);
printf("(2) Modificar_localidad_de_residencia.\n
");
color(0, 11);
printf("(3) Modificar_usuario.\n");
color(0, 6);
printf("(4) Modificar_contraseña.\n");
color(0,14);
printf("(5) Volver.\n");
color(0,15);
fflush(stdin);
scanf("%i", &opc);
system("cls");
switch(opc)
{
    case 1:
        modificarPerfilNombre(usuario,
                               numUsuarios, i);
        break;
    case 2:
        modificarPerfilLocalidad(usuario,
                               numUsuarios, localidad,
                               numLocalidades, i);
        break;
    case 3:
        modificarPerfilUsuario(usuario,
                               numUsuarios, i);
        break;
    case 4:
        modificarPerfilContraseña(usuario,
                               numUsuarios, i);
        break;
    case 5:
        return;
        break;
}
}

```

```

}

//Prototipo: void menuConductorVehiculo(Estr_Usuario *
usuario, int numUsuarios, Estr_Vehiculo *, int,
Estr_Viaje *, int, Estr_Pasos *, int, Estr_Reservas
*, int, Estr_Localidad *, int, Estr_Rutas **, int,
int, int);
//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema, en la estructura "usuario".
//Tambien necesitamos las estructuras inicializados,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
partir de la opcion que se escriba.

void menuConductorVehiculo(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_Localidad *localidad
    , int numLocalidades, Estr_Rutas **ruta, int
    numRutas, int numRutas2, int i)
{
    int opc;

    while(opc!=4)
    {
        leer_vehiculo(&vehiculo, &numVehiculos);
        system("cls");
        listarVehiculosUsuario(usuario, vehiculo,
            numVehiculos, i);
        color(0, 15);
        printf("Que_quiere_hacer?\n");
        color(0, 4);
        printf("(1) Alta_de_vehiculo.\n");
        color(0, 3);
        printf("(2) Modificar_vehiculo.\n");
        color(0, 2);
        printf("(3) Eliminar_vehiculo.\n");
        color(0, 14);
        printf("(4) Volver.\n");
    }
}

```

```

color(0, 15);
fflush(stdin);
scanf("%i", &opc);
system("cls");
switch(opc)
{
    case 1:
        altaVehiculo(usuario, vehiculo,
                      numVehiculos, i);
        break;
    case 2:
        modificarVehiculo(usuario, vehiculo,
                           numVehiculos, i);
        break;
    case 3:
        eliminarVehiculo(usuario, numUsuarios,
                           vehiculo, numVehiculos, viaje,
                           numViajes, pasos, numPasos, reservas
                           , numReservas, i);
        break;
    case 4:
        menuConductor(usuario, numUsuarios,
                       vehiculo, numVehiculos, viaje,
                       numViajes, pasos, numPasos, reservas
                       , numReservas, localidad,
                       numLocalidades, ruta, numRutas,
                       numRutas2, i);
        break;
}
}

//Prototipo: void menuConductorViajes(Estr_Usuario *
usuario, int numUsuarios, Estr_Vehiculo *, int,
Estr_Viaje *, int, Estr_Pasos *, int, Estr_Reservas
*, int, Estr_Localidad *, int, Estr_Rutas **, int,
int, int);

//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema, en la estructura "usuario".

```

```

//Tambien necesitamos las estructuras inicializadas ,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones , a
partir de la opcion que se escriba .

void menuConductorViajes( 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_Localidad *localidad
    , int numLocalidades , Estr_Rutas **ruta , int
    numRutas , int numRutas2 , int i )
{
    int opc;

    while( opc!=3)
    {
        leer_vehiculo(&vehiculo , &numVehiculos);
        leer_viaje(&viaje , &numViajes);
        leer_pasos(&pasos , &numPasos);
        leer_reservas(&reservas , &numReservas);
        system(" cls");
        listarViajes(usuario , vehiculo , numVehiculos ,
            viaje , numViajes , i );
        color(0, 15);
        printf(" Que_quiere_hacer?\n");
        color(0, 4);
        printf("(1) Crear_viaje.\n");
        color(0, 3);
        printf("(2) Modificar_viaje.\n");
        color(0, 2);
        printf("(3) Anular/Finalizar_viaje.\n");
        color(0, 14);
        printf("(4) Volver.\n");
        color(0, 15);
        fflush(stdin);
        scanf("%i" , &opc);
        system(" cls");
        switch(opc)
        {
            case 1:

```

```

        altaViaje( usuario , numUsuarios ,
            vehiculo , numVehiculos , viaje ,
            numViajes , pasos , numPasos , reservas
            , numReservas , localidad ,
            numLocalidades , ruta , numRutas ,
            numRutas2 , i , 0 );
        break;
    case 2:
        modificarViaje( usuario , vehiculo ,
            numVehiculos , viaje , numViajes ,
            pasos , numPasos , reservas ,
            numReservas , localidad ,
            numLocalidades , ruta , numRutas ,
            numRutas2 , i );
        break;
    case 3:
        finalizar_viaje( usuario , vehiculo ,
            numVehiculos , viaje , numViajes ,
            pasos , numPasos , reservas ,
            numReservas , i );
        break;
    case 4:
        menuConductor( usuario , numUsuarios ,
            vehiculo , numVehiculos , viaje ,
            numViajes , pasos , numPasos , reservas
            , numReservas , localidad ,
            numLocalidades , ruta , numRutas ,
            numRutas2 , i );
        break;
    }
}
}

//Prototipo: void menuAdmin(Estr_Usuario *usuario , int
//numUsuarios , Estr_Vehiculo *, int , Estr_Viaje *, int
//, Estr_Pasos *, int , Estr_Reservas *, int ,
//Estr_Localidad *, int , Estr_Rutas **, int , int , int )
//;
//Precondicion: Tener el entero "i", que nos indica la
//posicion del usuario que ha iniciado sesion en el
//sistema , en la estructura "usuario".

```

```

//Tambien necesitamos las estructuras inicializadas ,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones , a
partir de la opcion que se escriba .

void menuAdmin( 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_Localidad *localidad , int
numLocalidades , Estr_Rutas **ruta , int numRutas , int
numRutas2 , int i )
{
    int opc;

    while( opc!=4)
    {
        system("cls");
        color(15, 0);
        printf("Hola %s (% Administrador)\n" , usuario[ i ].  

            nombr_usuario);
        color(0, 15);
        printf("Que quiere ver?\n");
        color(0, 4);
        printf("(1) Usuarios.\n");
        color(0, 3);
        printf("(2) Vehiculos.\n");
        color(0, 2);
        printf("(3) Viajes.\n");
        color(0, 14);
        printf("(4) Volver.\n");
        color(0, 15);
        fflush(stdin);
        scanf("%i" , &opc);
        system("cls");
        switch(opc)
        {
            case 1:
                menuAdminUsuarios( usuario , numUsuarios ,
                    vehiculo , numVehiculos , viaje ,
                    numViajes , pasos , numPasos , reservas

```

```

        , numReservas , localidad ,
        numLocalidades , ruta , numRutas ,
        numRutas2 , i );
    break;
case 2:
    menuAdminVehiculos( usuario , numUsuarios
        , vehiculo , numVehiculos , viaje ,
        numViajes , pasos , numPasos , reservas
        , numReservas , localidad ,
        numLocalidades , ruta , numRutas ,
        numRutas2 , i );
    break;
case 3:
    menuAdminViajes( usuario , numUsuarios ,
        vehiculo , numVehiculos , viaje ,
        numViajes , pasos , numPasos , reservas
        , numReservas , localidad ,
        numLocalidades , ruta , numRutas ,
        numRutas2 , i );
    break;
case 4:
    menuPrincipal( usuario , numUsuarios ,
        vehiculo , numVehiculos , viaje ,
        numViajes , pasos , numPasos , reservas
        , numReservas , localidad ,
        numLocalidades , ruta , numRutas ,
        numRutas2 );
    break;
}
}
}

//Prototipo: void menuAdminUsuarios(Estr_Usuario *
//usuario , int numUsuarios , Estr_Vehiculo *, int ,
//Estr_Viaje *, int , Estr_Pasos *, int , Estr_Reservas
//*, int , Estr_Localidad *, int , Estr_Rutas **, int ,
//int , int );

//Precondicion: Tener el entero "i", que nos indica la
//posicion del usuario que ha iniciado sesion en el
//sistema , en la estructura "usuario".
```

```

//Tambien necesitamos las estructuras inicializadas ,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones , a
partir de la opcion que se escriba .

void menuAdminUsuarios( 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_Localidad *localidad
    , int numLocalidades , Estr_Rutas **ruta , int
    numRutas , int numRutas2 , int i )
{
    int opc;

    while( opc!=5)
    {
        leer_usuario(&usuario , &numUsuarios);
        leer_vehiculo(&vehiculo , &numVehiculos);
        leer_viaje(&viaje , &numViajes);
        leer_pasos(&pasos , &numPasos);
        leer_reservas(&reservas , &numReservas);
        system(" cls");
        color(15 , 0);
        printf(" Hola _%s _ ( Administrador ) \n" , usuario[ i ].  

            nomb_usuario);
        color(0 , 15);
        printf(" Que _ quiere _ hacer ? \n" );
        color(0 , 2);
        printf(" (1) Alta _de _usuario . \n" );
        color(0 , 4);
        printf(" (2) Baja _de _usuario . \n" );
        color(0 , 11);
        printf(" (3) Modificar _usuario . \n" );
        color(0 , 6);
        printf(" (4) Listar _usuarios . \n" );
        color(0 , 14);
        printf(" (5) Volver . \n" );
        color(0 , 15);
        fflush(stdin);
        scanf("%i" , &opc);
    }
}

```

```

        system("cls");
        switch(opc)
        {
            case 1:
                altaUsuario( usuario , numUsuarios ,
                            localidad , numLocalidades );
                break;
            case 2:
                eliminarAdminUsuario( usuario ,
                                      numUsuarios , vehiculo , numVehiculos ,
                                      viaje , numViajes , pasos , numPasos ,
                                      reservas , numReservas );
                break;
            case 3:
                modificarAdminUsuario( usuario ,
                                       numUsuarios , localidad ,
                                       numLocalidades );
                break;
            case 4:
                listarUsuarios( usuario , numUsuarios );
                break;
            case 5:
                menuAdmin( usuario , numUsuarios ,
                           vehiculo , numVehiculos , viaje ,
                           numViajes , pasos , numPasos , reservas ,
                           numReservas , localidad ,
                           numLocalidades , ruta , numRutas ,
                           numRutas2 , i );
                break;
        }
    }

}

//Prototipo: void menuAdminVehiculos(Estr_Usuario *
usuario , int numUsuarios , Estr_Vehiculo *, int ,
Estr_Viaje *, int , Estr_Pasos *, int , Estr_Reservas
*, int , Estr_Localidad *, int , Estr_Rutas **, int ,
int , int );
//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema , en la estructura "usuario".

```

```

//Tambien necesitamos las estructuras inicializadas ,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones , a
partir de la opcion que se escriba .

void menuAdminVehiculos( 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_Localidad *localidad
    , int numLocalidades , Estr_Rutas **ruta , int
    numRutas , int numRutas2 , int i )
{
    int opc;

    while( opc!=6)
    {
        leer_vehiculo(&vehiculo , &numVehiculos);
        leer_viaje(&viaje , &numViajes);
        leer_pasos(&pasos , &numPasos);
        leer_reservas(&reservas , &numReservas);
        system("cls");
        color(15, 0);
        printf("Hola %s (Administrador)\n" , usuario[ i ].nombr_usuario);
        color(0, 15);
        printf("Que quiere hacer?\n");
        color(0, 2);
        printf("(1) Alta_de_vehiculo.\n");
        color(0, 4);
        printf("(2) Baja_de_vehiculo.\n");
        color(0, 11);
        printf("(3) Modificar_vehiculo.\n");
        color(0, 6);
        printf("(4) Listar_vehiculos.\n");
        color(0, 9);
        printf("(5) Mostrar_lista_de_viajes_de_un_
            vehiculo.\n");
        color(0,14);
        printf("(6) Volver.\n");
        color(0,15);
    }
}

```

```

fflush(stdin);
scanf("%i", &opc);
system("cls");
switch(opc)
{
    case 1:
        altaAdmin(usuario, numUsuarios,
                  vehiculo, numVehiculos, viaje,
                  numViajes, pasos, numPasos, reservas,
                  numReservas, localidad,
                  numLocalidades, ruta, numRutas,
                  numRutas, 0);
        break;
    case 2:
        eliminarAdminVehiculo(usuario,
                               numUsuarios, vehiculo, numVehiculos,
                               viaje, numViajes, pasos, numPasos,
                               reservas, numReservas, i);
        break;
    case 3:
        modificarAdminVehiculo(usuario,
                               numUsuarios, vehiculo, numVehiculos,
                               i);
        break;
    case 4:
        listarVehiculos(usuario, numUsuarios,
                         vehiculo, numVehiculos, i);
        break;
    case 5:
        listarAdminVehiculoViajes(usuario,
                                 numUsuarios, vehiculo, numVehiculos,
                                 viaje, numViajes, i);
        break;
    case 6:
        menuAdmin(usuario, numUsuarios,
                  vehiculo, numVehiculos, viaje,
                  numViajes, pasos, numPasos, reservas,
                  numReservas, localidad,
                  numLocalidades, ruta, numRutas,
                  numRutas2, i);
        break;
}

```

```

        }
    }
}

//Prototipo: void menuAdminViajes(Estr_Usuario *usuario
, int numUsuarios, Estr_Vehiculo *, int, Estr_Viaje
*, int, Estr_Pasos *, int, Estr_Reservas *, int,
Estr_Localidad *, int, Estr_Rutas **, int, int, int)
;
//Precondicion: Tener el entero "i", que nos indica la
posicion del usuario que ha iniciado sesion en el
sistema, en la estructura "usuario".
//Tambien necesitamos las estructuras inicializados,
con sus contadores.
//Postcondicion: Llevarte a las diferentes funciones, a
partir de la opcion que se escriba.

void menuAdminViajes( 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_Localidad *localidad
    , int numLocalidades , Estr_Rutas **ruta , int
    numRutas , int numRutas2 , int i )
{
    int opc , n=0;

    while(opc!=6)
    {
        leer_vehiculo(&vehiculo , &numVehiculos);
        leer_viaje(&viaje , &numViajes);
        leer_pasos(&pasos , &numPasos);
        leer_reservas(&reservas , &numReservas);
        system("cls");
        color(15, 0);
        printf("Hola %s (Administrador)\n" , usuario[ i ].nombr_usuario);
        color(0, 15);
        printf("Que quiere hacer?\n");
        color(0, 2);
        printf("(1) Crear viaje.\n");
    }
}

```

```

color(0, 4);
printf("(2) Anular/Finalizar_viaje.\n");
color(0, 11);
printf("(3) Eliminar_viaje.\n");
color(0, 6);
printf("(4) Modificar_viaje.\n");
color(0, 9);
printf("(5) Listar_viajes.\n");
color(0,14);
printf("(6) Volver.\n");
color(0,15);
fflush(stdin);
scanf("%i", &opc);
system("cls");
switch(opc)
{
    case 1:
        altaAdmin(usuario, numUsuarios,
                  vehiculo, numVehiculos, viaje,
                  numViajes, pasos, numPasos, reservas
                  , numReservas, localidad,
                  numLocalidades, ruta, numRutas,
                  numRutas, 1);
        break;
    case 2:
        eliminarAdminViaje(usuario, numUsuarios
                           , vehiculo, numVehiculos, viaje,
                           numViajes, pasos, numPasos, reservas
                           , numReservas, i, 1);
        break;
    case 3:
        eliminarAdminViaje(usuario, numUsuarios
                           , vehiculo, numVehiculos, viaje,
                           numViajes, pasos, numPasos, reservas
                           , numReservas, i, 0);
        break;
    case 4:
        modificarAdminViaje(usuario,
                            numUsuarios, vehiculo, numVehiculos,
                            viaje, numViajes, pasos, numPasos,
                            reservas, numReservas, localidad,

```

```

        numLocalidades , ruta , numRutas ,
        numRutas2 , i );
    break;
case 5:
    listarAdminViajes( usuario , numUsuarios ,
        vehiculo , numVehiculos , viaje ,
        numViajes , i , 0 , &n );
    break;
case 6:
    menuAdmin( usuario , numUsuarios ,
        vehiculo , numVehiculos , viaje ,
        numViajes , pasos , numPasos , reservas
        , numReservas , localidad ,
        numLocalidades , ruta , numRutas ,
        numRutas2 , i );
    break;
}
}
}
```

4.1.13. Modificar

Para ver la descripción del módulo puede ir a 3.2.13.

```
#include "modificar.h"
```

```

//Prototipo: void modificarVehiculo(Estr_Usuario *,
    Estr_Vehiculo *, int , int );
//Precondicion: Tener el entero "i", para saber la
    posicion del usuario en la estructura "usuario",
//y las estructuras "usuario" y "vehiculo"
    inicializadas , con sus contadores .
//Postcondicion: Modificar cualquier dato de un
    vehiculo que tenga el usuario .

void modificarVehiculo(Estr_Usuario *usuario ,
    Estr_Vehiculo *vehiculo , int numVehiculos , int i )
{
    FILE *fp ;
    int x=0, h=0, m=0, aux=0, y=0, opc=0, opc2=0, *vec=
        NULL, encontrado=0, error_mat , counter ;
```

```

char mat[8], plazas[2], descrip[51];

if(numVehiculos!=0) //Si hay vehiculos en el sistema .
{
    fp=fopen ("DATA/vehiculos.txt", "r+");
    if(fp==NULL)
    {
        printf("No se ha podido abrir el fichero vehiculos.txt.\n");
        return;
    }
    else
    {
        encontrarVehiculos( usuario , vehiculo ,
            numVehiculos , &vec , &x , i ); //Encuentra todos los vehiculos del usuario .
        aux=x;
        if(vec!=NULL) //Si el usuario tiene vehiculos , el vector "vec" no sera nulo
        {
            while(encontrado==0) //Hasta que no se introduzca un vehiculo que este en el vector de enteros , es decir , un vehiculo que el tenga .
            {
                x=aux;
                y=aux;
                printf("Que vehiculo quiere modificar?\n");
                for(m=0; m<x; m++)
                {
                    color(0,4);
                    printf("----Vehiculo");
                    Imprime todos los datos de los vehiculos que tiene el usuario .
                    color(0,15);
                    printf("%i", m+1);
                }
            }
        }
    }
}

```

```

        color(0,4);
        printf(": \t");
        color(0,3);
        printf("Matricula : ");
        color(0,15);
        printf("%s", vehiculo [ vec [m] ]. id_mat);
        color(0,3);
        printf(" | Numero_de_plazas : ")
        ;
        color(0,15);
        printf("%s", vehiculo [ vec [m] ]. num_plazas);
        color(0,3);
        printf(" | Descripcion : ");
        color(0,15);
        printf("%s\n", vehiculo [ vec [m] ]. desc_veh);
    }
    x++;
    color(0,14);
    printf("(%i) Salir.\n", x);
    color(0,15);
    printf("Ingrese el numero "
        correspondiente al vehiculo que "
        desea modificar : ");
    fflush(stdin);
    scanf("%d", &opc);
    system("cls");
    if(opc==x) //Si introduce lo mismo
        que vale la variable maxima "x".
    {
        return; //Vuelve al menu.
    }
    for(m=1; m<y; m++) //Vamos
        recorriendo el vector de enteros
        , con los vehiculos del usuario.
    {
        if(opc==vec[m-1]) //Si el
            vehiculo seleccionado esta
            en este vector.

```

```

{
    encontrado=1;
}
}
if (( opc>=1&&opc<=x)&&opc!=x)
{
    h=opc-1;
    do
    {
        printf("Que_desea_modificar
            ?\n");
        printf("(1)Matricula.\n");
        printf("(2)Numero_de_plazas
            .\n");
        printf("(3)Descripcion.\n")
            ;
        printf("(4)TODO.\n");
        printf("(5)Salir.\n");
        scanf("%d", &opc2);
        system("cls");
        switch(opc2)
        {
            case 1:
                do{ //Hasta que la
                    matricula no sea
                    correcta.
                    error_mat=0;
                    printf(
                        Matricula_
                        del_vehiculo
                        -(Maximo_de_
                        7_caracteres
                        ):\n);
                    printf(
                        Matricula_
                        actual:_%s\n
                        " , vehiculo [
                        vec[h]].
                        id_mat);
                    pregunta(mat,
                        8); //

```

*Pedimos la
nueva
matricula.*

```
for( counter=0;  
      counter<  
      numVehiculos  
    )&&(  
      error_mat  
    ==0); counter  
    ++) //  
    Comprobamos  
    que la  
    matricula no  
    esta  
    registrada.  
{  
  if( strcmp( mat,  
    vehiculo  
    [ counter  
    ]. id_mat  
  )==0){  
    //Si se  
    encuentra  
    la  
    matricula  
    en las  
    que hay  
    ahora,  
    pues  
    esta  
    usada.  
    error_mat  
    =1;  
    printf( "La_  
    matricula  
    %s_  
    esta  
    _
```

```

actualmente
-
registrada
.\n"
,mat
);
system(
"
PAUSE
");
}
}
if( strlen(mat)
<7) //Si la
nueva
matricula no
tiene 7
caracteres .
{
printf("La_
matricula
%s debe
poseer u
una u
longitud
total u
de 7 u
caracteres
.\n", mat);
system("
PAUSE");
}
for( counter=0;( counter<4)
&&(error_mat
==0); counter
++) //
Comprobamos
que los 4
primeros

```

*caracteres ,
 son dígitos .*
 {
if((mat[
 counter
]<48)||(
 mat[
 counter
]>57))
 {
 error_mat
 =1;
 printf(
 "Los
 4
 primeros
 _
 caracteres
 de
 la_
 matricula
 _
 tienen
 _que
 _ser
 _
 numeros
 .\n"
);
 system(
 "
 PAUSE
 ");
 }
}

for(counter=4;
 counter<7)
 &&(error_mat
 ==0);counter
 ++) //
Comprobamos

que los 3 ultimos caracteres, son letras mayusculas.
 {
if((mat[
 counter
]<65)||(
 mat[
 counter
]>90))
 {
 error_mat
 =1;
 printf(
 "Los
 3
 ultimos
 _-
 caracteres
 de
 la_
 matricula
 _-
 tienen
 _-que
 _-ser
 _-
 letras
 _-
 mayusculas
 .\n"
);
 system(
 "
 PAUSE
 ");
 }
}
system("cls");

```
}while(( error_mat  
==1) ||( strlen(  
mat)<7));
```

```
strcpy( vehiculo [ vec  
[h] ]. id_mat , mat  
); //Copiamos la  
nueva matricula  
en la posicion  
de la estructura  
correspondiente
```

```
printf ("La_  
matricula _del_  
vehiculo _se _ha_  
actualizado_  
correctamente.\n  
");
```

```
system ("PAUSE");  
break;
```

```
case 2:
```

```
printf ("Numero _de_  
plazas _libres _(  
sin _contar _el_  
conductor ) :\n");
```

```
printf ("Plazas_  
actuales : %s\n",  
vehiculo [ vec [ h  
] ]. num_plazas );  
pregunta ( plazas , 2)  
; //Pedimos el  
nuevo numero de  
plazas .
```

```
strcpy ( vehiculo [ vec  
[h] ]. num_plazas ,  
plazas ); //  
Copiamos el  
nuevo numero de  
plazas en la
```

*posicion de la
estructura
correspondiente.*

```
printf("El numero  
de plazas del  
vehiculo se ha  
actualizado  
correctamente.\n") ;  
system("PAUSE");  
break;  
case 3:  
    printf("Descripcion  
del vehiculo (   
Marca , modelo ,  
color , etc ) (   
Maximo de 50  
caracteres ) :\n") ;  
    printf("Descripcion  
actual : %s\n" ,  
vehiculo [ vec [ h  
] ]. desc_veh );  
pregunta ( descrip ,  
51); //Pedimos  
la nueva  
descripcion .  
  
strcpy ( vehiculo [ vec  
[ h ] ]. desc_veh ,  
descrip ); //  
Copiamos la  
nueva  
descripcion en  
la posicion de  
la estructura  
correspondiente .  
  
printf("La  
descripcion del
```

```

    vehiculo_se_ha_
    actualizado_
    correctamente.\n
    " );
    system ("PAUSE" );
    break;
case 4: //Volvemos a
    hacer todo lo
    anterior.
do{
    error_mat=0;
    printf("
        Matricula_
        del_vehiculo
        -(Maximo_de_
        7_caracteres
        ):\n");
    printf("
        Matricula_
        actual: %s\n"
        , vehiculo [
        vec[h]].
        id_mat);
    pregunta(mat,
        8);

for ( counter=0;(

        counter<
        numVehiculos
        )&&(
        error_mat
        ==0);counter
        ++){
        if (strcmp (
        mat,
        vehiculo
        [counter
        ].id_mat
        )==0){
        error_mat
        =1;

```

```

printf(
    "La_
    matricula
    %s_
    esta
    -
    actualmente
    -
    registrada
    .\n"
    ,mat
    );
system(
    "
PAUSE
    ");
}
}
if( strlen( mat )
<7){
    printf("La_
    matricula
    %s _debe
    _poseer_
    una_
    longitud
    _total_
    de_7_
    caracteres
    .\n" ,mat
    );
    system("
    PAUSE");
}
for( counter=0;( counter<4)
&&(error_mat
==0); counter
++){
    if(( mat [
    counter

```

```

] < 48) || (
mat [
counter
] > 57)) {
error_mat
= 1;
printf(
" Los
\u2014 4 \u2014
primeros
\u2014
caracteres
\u2014 de \u2014
la \u2014
matricula
\u2014
tienen
\u2014 que
\u2014 ser
\u2014
numeros
.\n"
);
system(
"
PAUSE
");
}
}
for ( counter = 4; (
counter < 7)
&&(error_mat
== 0); counter
++) {
if (( mat [
counter
] < 65) || (
mat [
counter
] > 90)) {

```

```

error_mat
=1;
printf(
"Los
3
ultimos
 
caracteres
de
la
matricula
 
tienen
que
ser
 
letras
 
mayusculas
.u\n"
);
system(
"PAUSE
");
}
}
system("cls");
}while((error_mat
==1)|| (strlen(
mat)<7));

printf("Matricula
del_vehiculo(
Maximo_de_7_
caracteres):\n")
;
printf("Matricula
actual: %s\n",
vehiculo [vec [h
]]. id_mat);

```

```

printf ("%s\n" , mat)
;

strcpy ( vehiculo [ vec
[ h ] ]. id_mat , mat
);

printf ("Numero_de_
plazas_libres_((
sin_contar_el_
conductor):\n");
printf ("Plazas_
actuales : %s\n" ,
vehiculo [ vec [ h
] ]. num_plazas );
fflush ( stdin );
pregunta ( plazas , 2 )
;

printf ("Descripcion
_del_vehiculo_((
Marca , _modelo , _
color , _etc )_((
Maximo_de_50_
caracteres):\n")
;
printf ("Descripcion
_actual: %s\n" ,
vehiculo [ vec [ h
] ]. desc_veh );
pregunta ( descrip ,
51 );

strcpy ( vehiculo [ vec
[ h ] ]. id_mat , mat
);
strcpy ( vehiculo [ vec
[ h ] ]. num_plazas ,
plazas );
strcpy ( vehiculo [ vec
[ h ] ]. desc_veh ,

```

```

        descrip);

printf("Se han
actualizado
todos los datos
del vehiculo.\n"
);
system("PAUSE");
break;
case 5: //Salimos
break;
}
encontrado=1;
}while((opc2>1&&opc2<5)&&
encontrado==0);
if(opc2!=5)
{
    actualizarVehiculo(vehiculo
    , numVehiculos);
}
}
else
{
    printf("No tiene vehiculos registrados
    .\n");
    system("PAUSE");
}
}
else
{
    printf("No hay vehiculos registrados en el
    sistema.\n");
    system("PAUSE");
}
fclose(fp);
}

```

```

//Prototipo: void modificarViaje(Estr_Usuario *,
    Estr_Vehiculo *, int, Estr_Viaje *, int, Estr_Pasos
    *, int, Estr_Reservas *, int, Estr_Localidad *, int,
    Estr_Rutas **, int, int, int);
//Precondicion: Tener el entero "i", para saber la
    posicion del usuario en la estructura "usuario", y
    las estructuras inicializadas, con sus contadores.
//Postcondicion: Modificar cualquier dato de un viaje,
    que este abierto y sin plazas reservadas, que tenga
    el usuario.

void modificarViaje(Estr_Usuario *usuario ,
    Estr_Vehiculo *vehiculo , int numVehiculos ,
    Estr_Viaje *viaje , int numViajes , Estr_Pasos *pasos ,
    int numPasos , Estr_Reservas *reservas , int
    numReservas , Estr_Localidad *localidad , int
    numLocalidades , Estr_Rutas **ruta , int numRutas , int
    numRutas2 , int i)
{
    FILE *fp ;
    int h=0, x=0, m=0, j=0, opc=0, opc2=0, *vec=NULL, *
        vec_viaje=NULL, cont=0, prec=0, encontrado=0,
        encontrado2=0;
    char mat[8] , fecha[11] , hora_in[6] , hora_fin[6] ,
        coste[2];

    if( numViajes!=0) //Si hay viajes en el sistema .
    {
        fp=fopen ("DATA/ viajes .txt" , "r+" );

        if( fp==NULL)
        {
            printf ("No se ha podido abrir el fichero
                    viajes .txt.\n");
            return;
        }
        else
        {
            encontrarVehiculos( usuario , vehiculo ,
                numVehiculos , &vec , &x , i); //Encuentra
                todos los vehiculos del usuario .

```

```

for ( j=0;j<x ; j++)
{
    encontrarViajes( vehiculo , numVehiculos ,
                      viaje , numViajes , vehiculo [ vec [ j ] ] .
                      id_mat , &vec_viaje , &cont , 2); //  

Encuentra todos los viajes abiertos,  

que no tienen plazas ocupadas, del  

usuario.
}
if( vec_viaje==NULL) //Si el vector esta  

vacio, significa que no hay viajes.
{
    printf("No_possee_viajes_abiertos , que  

           no_hayan_sido_reservados.\n");
    system("PAUSE");
}
else
{
    while( vec_viaje!=NULL&&encontrado==0)
    {
        printf("Que_viaje_quiere_modificar  

               ?\n");

        for (m=0; m<cont ; m++)
        {
            color(0,4);
            printf("Viaje "); //Imprime  

todos los datos de cada  

viaje.
            color(0,15);
            printf("%i", m+1);
            color(0,4);
            printf(": t");
            color(0,3);
            printf("ID: ");
            color(0,15);
            printf("%s", viaje [ vec_viaje [m
                ] ]. id_viaje );
            color(0,3);
            printf(" | Matricula: ");
            color(0,15);
        }
    }
}

```

```

        printf ("%s" , viaje [ vec_viaje [m
            ]]. id_mat );
        color (0,3);
        printf ("| Fecha de partida : " )
            ;
        color (0,15);
        printf ("%s" , viaje [ vec_viaje [m
            ]]. f_inic );
        color (0,3);
        printf ("| Hora de partida : " );
        color (0,15);
        printf ("%s" , viaje [ vec_viaje [m
            ]]. h_inic );
        color (0,3);
        printf ("\n|||||||||||||||"
            Hora de llegada : " );
        color (0,15);
        printf ("%s" , viaje [ vec_viaje [m
            ]]. h_fin );
        color (0,3);
        printf ("| Plazas libres : " );
        color (0,15);
        printf ("%s" , viaje [ vec_viaje [m
            ]]. plazas_libre );
        color (0,3);
        printf ("| Ida/Vuelta : " );
        color (0,15);
        printf ("%s\n\n" , viaje [
            vec_viaje [m]]. precio );
    }
    m++;
    color (0,14);
    printf ("(%i) Salir.\n" , m);
    color (0,15);
    printf ("Ingrese el numero "
        correspondiente al viaje que

```

```

        desea_modificar : " ) ;
fflush ( stdin );
scanf ( "%d" , &opc );
system ( " cls" );
if (opc>=1&&opc<=cont) //Si la
    opcion esta entre 1 y la
    variable maxima.
{
    encontrado=1;
}
if (opc==m) //Si introduce lo mismo
    que vale la variable maxima "x".
{
    return; //Vuelve al menu.
}
encontrado=0;
h=opc-1;
do
{
    printf ( " Que_desea_modificar ? \n" );
    printf ( " (1) Vehiculo . \n" );
    printf ( " (2) Fecha . \n" );
    printf ( " (3) Coste . \n" );
    printf ( " (4) Rutas . \n" );
    printf ( " (5) TODO. \n" );
    printf ( " (6) Salir . \n" );
    scanf ( "%d" , &opc2 );
    system ( " cls" );
    switch (opc2)
    {
        case 1:
            do
            {
                system ( " cls" );
                listarVehiculosUsuario
                    ( usuario ,
                      vehiculo ,
                      numVehiculos , i )
                ;
            }
    }
}
```

```

printf("Matricula_
       del_vehiculo_(_
       Maximo_de_7_
       caracteres)_para_
       asignarle_al_
       viaje_%s:\n",
       viaje[vec_viaje[_
h]].id_viaje);
printf("Matricula_
       actual:_%s\n",
       viaje[vec_viaje[_
h]].id_mat);
pregunta(mat, 8);
//Pedimos la
nueva matricula
que va a llevar
el viaje.
for(j=0;j<x;j++)
{
    if(strcmp(
        vehiculo[vec
        [j]].id_mat,
        mat)==0) ///
    Si la
    matricula
    introducida
    esta entre
    sus
    vehiculos.
    {
        encontrado2
        =1;
    }
}
} while(encontrado2==0);

strcpy(viaje[vec_viaje[_
h]].id_mat, mat); ///
Copiamos la nueva
matricula en la
posicion de la

```

*estructura
correspondiente.*

```
printf("El_vehiculo_del_
_viaje_se_ha_
actualizado_
correctamente.\n");
system("PAUSE");
break;
case 2:
    leerFecha( fecha ,
                hora_in , hora_fin );
    //Leemos la fecha ,
    hora inicio y hora
    llegada .

strcpy(viaje[ vec_viaje [
h]]. f_inic , fecha);
//Introducimos los
datos pedidos al
viaje .
strcpy(viaje[ vec_viaje [
h]]. h_inic , hora_in )
;
strcpy(viaje[ vec_viaje [
h]]. h_fin , hora_fin )
;

printf("La_fecha_del_
viaje_se_ha_
actualizado_
correctamente.\n");
system("PAUSE");
break;
case 3:
    while( prec<=0) //  

        Mientras que el
        precio sea menor o
        igual a 0.
    {
        system("cls");
    }
```

```

        printf("Coste del
                viaje-(Maximo_de
                _1_digito):\n");
        printf("Coste_
                actual: %s\n",
                viaje[vec_viaje [
                h]].precio);
        scanf("%i", &prec)
        ; //Pedimos el
        nuevo precio.
    }

    sprintf(coste, "%i",
            prec); //Pasamos de
            entero a cadena.

    strcpy(viaje[vec_viaje [
    h]].precio, coste);
    //Copiamos el nuevo
    precio.

    printf("El coste del
            viaje se ha_
            actualizado_
            correctamente.\n");
    system("PAUSE");
    break;
case 4:
    eliminarPasos(pasos,
        numPasos, viaje [
        vec_viaje[h]].id_viaje); //Elimina
        los pasos del viaje

    buscadorRutas(ruta,
        numRutas, numRutas2,
        localidad,
        numLocalidades,
        pasos, numPasos,
        viaje[vec[h]].id_viaje); //Pide la

```

*nueva ruta por
 donde va a ir, e
 imprime los pasos.*
break;
case 5: //Volvemos a hacer
 todo lo anterior.
do
 {
 system("cls");
 listarVehiculosUsuario
 (usuario,
 vehiculo,
 numVehiculos, i)
 ;
 printf("Matricula
 del_vehiculo_(
 Maximo_de_7_
 caracteres)_para
 _asignarle_al_
 viaje_%s:\n",
 viaje[vec_viaje[
 h]].id_viaje);
 printf("Matricula
 actual:_%s\n",
 viaje[vec_viaje[
 h]].id_mat);
 pregunta(mat, 8);
for(j=0;j<x;j++)
 {
if(strcmp(
 vehiculo[vec
 [j]].id_mat,
 mat)==0)
 {
 encontrado2
 =1;
 }
 }
} **while**(encontrado2==0);

```

strcpy( viaje [ vec_viaje [
    h ] ]. id_mat , mat ) ;

if ( encontrado2==1)
{
    leerFecha ( fecha ,
        hora_in ,
        hora_fin ) ;

    strcpy ( viaje [
        vec_viaje [ h ] ]. 
        f_inic , fecha ) ;
    strcpy ( viaje [
        vec_viaje [ h ] ]. 
        h_inic , hora_in )
        ;
    strcpy ( viaje [
        vec_viaje [ h ] ]. 
        h_fin , hora_fin )
        ;
}

while ( prec <=0)
{
    system ( " cls " ) ;
    printf ( " Coste_ "
        del_viaje_ (
        Maximo_de_1_
        digito ): \n ) ;
    ;
    printf ( " Coste_ "
        actual : %s \n
        " , viaje [
        vec_viaje [ h
        ] ]. precio ) ;
    scanf ( "%1i" , &
        prec ) ;
}

printf ( coste , "%1i
" , prec ) ;

```

```

strcpy( viaje [
    vec_viaje [h]] .
    precio , coste);

eliminarPasos( pasos
    , numPasos ,
    viaje [ vec_viaje [
        h]] . id_viaje );
//Elimina los
pasos del viaje.
buscadorRutas( ruta ,
    numRutas ,
    numRutas2 ,
    localidad ,
    numLocalidades ,
    pasos , numPasos ,
    viaje [ vec [h]] .
    id_viaje ); //
Pide la nueva
ruta por donde
va a ir , e
imprime los
pasos .
printf("Se han_
actualizado_
todos los datos_
ingresados del_
viaje.\n");
system("PAUSE");
}

break;
case 6: //Salimos
break;
}
encontrado=1;
} while ((opc2>1&&opc2<6)&&encontrado
==0);
if (opc2!=6)
{
    actualizarViaje( viaje ,
        numViajes );

```

```

        }
    }
}
else
{
    printf("No hay viajes registrados.\n");
    system("PAUSE");
}

fclose(fp);
}

//Prototipo: void modificarPerfilNombre(Estr_Usuario *,
int , int);
//Precondicion: Tener el entero "i", para saber la
posicion del usuario en la estructura "usuario", y
la estructura "usuario" inicializada, con su
contador.
//Postcondicion: Modificar el nombre del usuario.

void modificarPerfilNombre(Estr_Usuario *usuario , int
    numUsuarios , int i)
{
    char nomb[21];

    printf("Introduzca su nuevo nombre completo (Maximo
        de 20 caracteres):\n");
    pregunta(nomb, 21); //Pedimos el nuevo nombre.

    strcpy(usuario[i].nomb_usuario , nomb); //Copiamos
        el nuevo nombre en la posicion de la estructura
        correspondiente.
    actualizarUsuario(usuario , numUsuarios); //
        Actualizamos el fichero usuarios.txt

    printf("Su nombre completo se ha actualizado -
        correctamente.\n");
    system("PAUSE");
}

```

```

//Prototipo: void modificarPerfilLocalidad(Estr_Usuario
* , int , Estr_Localidad *, int , int );
//Precondicion: Tener el entero "i", para saber la
posicion del usuario en la estructura "usuario", y
la estructura "usuario" inicializada, con su
contador.
//Postcondicion: Modificar la localidad del usuario.

void modificarPerfilLocalidad (Estr_Usuario *usuario ,
    int numUsuarios , Estr_Localidad *localidad , int
    numLocalidades , int i)
{
    char loc [21];

    printf("Introduzca su nueva localidad de residencia
        (3 siglas):\n");
    fflush(stdin);
    pregunta_localidad(localidad , numLocalidades , loc );
        //Pedimos la nueva localidad.

    strcpy(usuario [i].localidad , loc); //Copiamos la
        nueva localidad en la posicion de la estructura
        correspondiente.
    actualizarUsuario(usuario , numUsuarios); //
        Actualizamos el fichero usuarios.txt

    printf("Su localidad de residencia se ha
        actualizado correctamente.\n");
    system("PAUSE");
}

//Prototipo: void modificarPerfilUsuario(Estr_Usuario
* , int , int );
//Precondicion: Tener el entero "i", para saber la
posicion del usuario en la estructura "usuario", y
la estructura "usuario" inicializada, con su
contador.
//Postcondicion: Modificar el username del usuario.

void modificarPerfilUsuario (Estr_Usuario *usuario , int
    numUsuarios , int i)

```

```

{
    int k=0, encontrado=0;
    char usua [ 6 ];

    printf (" Introduzca su nuevo nombre de usuario -\n "
        " Maximo de 5 caracteres ) :\n" );
    pregunta (usua , 6 ); //Pedimos el nuevo username.

    for (k=0; k<numUsuarios&&encontrado==0; k++) //-
        Comprobamos que no esta usado .
    {
        if (strcmp (usua ,  usuario [ k ]. usuario )==0)
        {
            encontrado=1;
        }
    }

    if (encontrado==0) //Si no esta usado
    {
        printf (" El nombre de usuario es valido .\n" );
        strcpy (usuario [ i ]. usuario ,  usua); //Copiamos el
            nuevo username en la posicion de la
            estructura correspondiente .
        actualizarUsuario (usuario ,  numUsuarios); //-
            Actualizamos el fichero usuarios . txt

        printf (" Su nombre de usuario se ha actualizado -
            correctamente .\n" );
        system ("PAUSE" );
    }
    else
    {
        printf (" El nombre de usuario ya esta siendo -
            usado .\n" );
        system ("PAUSE" );
    }
}

//Prototipo : void modificarPerfilContrasena (
    Estr_Usuario *, int , int );

```

```

//Precondicion: Tener el entero "i", para saber la
posicion del usuario en la estructura "usuario", y
la estructura "usuario" inicializada, con su
contador.
//Postcondicion: Modificar la contraseña del usuario.

void modificarPerfilContrasena( Estr_Usuario *usuario ,
    int numUsuarios , int i )
{
    int encontrado=3, x=0;
    char contra [9] , contra2 [9];

    while( encontrado>0&&x==0)
    {
        printf("Introduzca su antigua contraseña (
            Maximo de 8 caracteres):\n");
        preguntar_contrasena(contra);

        if(strcmp( usuario [ i ]. contrasena , contra)==0) //  

            Si la contraseña introducida es igual a la
            que tiene.
        {
            system("cls");
            x=1;
            printf("Introduzca su nueva contraseña (
                Maximo de 8 caracteres):\n");
            preguntar_contrasena(contra2);

            strcpy( usuario [ i ]. contrasena , contra2); //  

                Copiamos la nueva contraseña en la
                posicion de la estructura
                correspondiente.
            actualizarUsuario( usuario , numUsuarios); //  

                Actualizamos el fichero usuarios.txt

            printf("\nSu contraseña se ha actualizado -
                correctamente.\n");
            system("PAUSE");
        }
        else //Si no es correcta
        {

```

```

encontrado--;
if( encontrado==0) //Cuando no tiene
    intentos
{
    printf("\nIntentos agotados! :\(\n");
    system("PAUSE");
}
else //Si tiene intentos, se van restando .
{
    contra[0]='\0';
    printf("\nQueda(n)%i intentos.\n",
        encontrado);
    system("PAUSE");
    system("cls");
}
}

}

//Prototipo: void modificarAdminUsuario(Estr_Usuario *,
int , Estr_Localidad *, int );
//Precondicion: Tener el entero "i", para saber la
posicion del usuario en la estructura "usuario", y
las estructuras inicializadas, con sus contadores.
//Postcondicion: Modificar los datos de cualquier
usuario .

void modificarAdminUsuario(Estr_Usuario *usuario , int
    numUsuarios , Estr_Localidad *localidad , int
    numLocalidades)
{
    int j , opc=0, encontrado=0, id=0;
    char vec_id [5];

    listarUsuarios(usuario , numUsuarios); //Imprimir
        una lista de todos los usuarios del sistema .

    if(numUsuarios!=0) //Si hay usuarios en el sistema .
    {
        while(encontrado==0)
        {

```

```

printf(" Introduzca la ID del usuario al que
       quiere modificarle el usuario.\n");
scanf("%i",&id); //Pedimos la id del
                  usuario que queremos modificar.
sprintf(vec_id , "%04i" , id); //Introducimos
                                el entero introducido en la cadena con
                                formato XXXX

encontrarUsuario( usuario , numUsuarios ,
                  vec_id , &j , &encontrado); //Buscamos si
                                el usuario existe en el sistema.

if( encontrado==1)
{
    while( opc<1||opc>6) //Elegimos una de
                            las opciones.
    {
        system("cls");
        printf("Que desea modificar?\n");
        printf("(1)Nombre_completo.\n");
        printf("(2)Localidad.\n");
        printf("(3)Nombre_de_usuario.\n");
        printf("(4)Contrasena.\n");
        printf("(5)TODO.\n");
        printf("(6)Salir.\n");
        scanf("%d" , &opc);
        system("cls");
        switch(opc)
        {
            case 1:
                modificarPerfilNombre(
                    usuario , numUsuarios
                    , j);
                break;
            case 2:
                modificarPerfilLocalidad
                    ( usuario ,
                      numUsuarios ,
                      localidad ,
                      numLocalidades , j);
                break;
        }
    }
}

```

```

        case 3:
            modificarPerfilUsuario(
                usuario , numUsuarios
                , j );
            break;
        case 4:
            modificarPerfilContrasena
                ( usuario ,
                  numUsuarios , j );
            break;
        case 5:
            modificarPerfilNombre(
                usuario , numUsuarios
                , j );
            modificarPerfilLocalidad
                ( usuario ,
                  numUsuarios ,
                  localidad ,
                  numLocalidades , j );
            modificarPerfilUsuario(
                usuario , numUsuarios
                , j );
            modificarPerfilContrasena
                ( usuario ,
                  numUsuarios , j );
            break;
        case 6:
            break;
    }
}
}
}
}

//Prototipo: void modificarAdminVehiculo(Estr_Usuario
//*, int , Estr_Vehiculo *, int , int );
//Precondicion: Tener el entero "i", para saber la
//posicion del usuario en la estructura "usuario", y
//las estructuras inicializadas , con sus contadores .

```

```

//Postcondicion: Modificar los datos de un vehiculo de
cualquier usuario.

void modificarAdminVehiculo( Estr_Usuario *usuario , int
    numUsuarios , Estr_Vehiculo *vehiculo , int
    numVehiculos , int x)
{
    char opc2[5];
    int encontrado=0, counter, id;

    system("cls");
    if(numVehiculos!=0) //Si hay vehiculos en el
        sistema.
    {
        listarVehiculos( usuario , numUsuarios , vehiculo ,
            numVehiculos , x); //Listamos todos los
            vehiculos.
        printf("Escriba la ID del usuario a la que se
            quiera modificar el vehiculo\n");
        scanf("%i",&id); //Pedimos la id del usuario ,
            del vehiculo que queremos modificar.
        sprintf(opc2, "%04i", id); //Introducimos el
            entero introducido en la cadena con formato
            XXXX.

        for( counter=0;( counter<numUsuarios)&&
            encontrado==0); counter++)
        {
            if(strcmp(opc2, usuario [counter].id_usuario
                )==0) //Si el usuario existe .
            {
                encontrado=1;
                system("cls");
                modificarVehiculo( usuario , vehiculo ,
                    numVehiculos , counter); //Vamos a la
                    funcion "modificarVehiculo".
            }
        }
        if(encontrado==0) //Si no existe el usuario
            introducido .
    {

```

```

        system("cls");
        printf("No se ha encontrado ningun usuario
               con la siguiente ID: %s\n", opc2);
        system("PAUSE");
    }
}

//Prototipo: void modificarAdminViaje(Estr_Usuario *,
int, Estr_Vehiculo *, int, Estr_Viaje *, int,
Estr_Pasos *, int, Estr_Reservas *, int,
Estr_Localidad *, int, Estr_Rutas **, int, int, int)
;
//Precondicion: Tener el entero "i", para saber la
posicion del usuario en la estructura "usuario", y
las estructuras inicializadas, con sus contadores.
//Postcondicion: Modificar los datos de un viaje de
cualquier usuario.

void modificarAdminViaje(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_Localidad *localidad
, int numLocalidades, Estr_Rutas **ruta, int
numRutas, int numRutas2, int x)
{
    char opc2[5];
    int encontrado=0, i=0, counter, n=0, id=0;

    system("cls");
    listarAdminViajes(usuario, numUsuarios, vehiculo,
    numVehiculos, viaje, numViajes, x, 2, &n); // 
    Listamos todos los viajes, abiertos y sin plazas
    reservadas.
    if(n!=0) //Si tiene viajes abiertos y sin plazas
    reservadas.
    {
        printf("Escriba la ID del usuario a la que se
               quiera modificar el viaje\n");

```

```

scanf("%i", &id); //Pedimos la id del usuario ,
                  del vehiculo que queremos modificar.
sprintf(opc2, "%04i", id); //Introducimos el
                           entero introducido en la cadena con formato
                           XXXX.

for( counter=0;( counter<numUsuarios )&&( 
      encontrado==0); counter++)
{
    if(strcmp(opc2, usuario [ counter ]. id_usuario) 
        ==0) //Si el usuario existe .
    {
        encontrado=1;
        i=counter ;
        system("cls"); //Vamos a la funcion "
                         modificarViaje".
        modificarViaje( usuario , vehiculo ,
                       numVehiculos , viaje , numViajes ,
                       pasos , numPasos , reservas ,
                       numReservas , localidad ,
                       numLocalidades , ruta , numRutas ,
                       numRutas2 , i );
    }
}
if(encontrado==0) //Si no existe el usuario
                    introducido .
{
    system("cls");
    printf("No se ha encontrado ningun usuario
           con la siguiente ID: %s\n", opc2);
    system("PAUSE");
}
}

```

4.1.14. Preguntar

Para ver la descripción del módulo puede ir a 3.2.14.

```
#include "preguntar.h"
```

```

//Prototipo: void pregunta(char *, int);
//Precondicion: Tener una cadena y un entero
    inicializados, que nos indique el num maximo de
    caracteres de la misma.
//Postcondicion: Introducir en una cadena, la
    informacion escaneada, pero cambiando el salto de
    linea \n por el caracter nulo \0.

void pregunta(char *x, int i)
{
    int encontrado=0;

    while (!encontrado)
    {
        fflush(stdin);
        fgets(x, i, stdin);

        char *lin=strchr(x, '\n');
        if (strchr(x, '\n')!=NULL)
        {
            *lin='\0';
        }
        if (strlen(x)>0)
        {
            encontrado=1;
        }
    }
}

//Prototipo: void preguntar_contrasena(char *);
//Precondicion: Tener una cadena inicializada, donde
    introducir la contraseña escaneada.
//Postcondicion: Introducir en una cadena, la
    contraseña escaneada, pero cambiando el salto de
    linea \n por el caracter nulo \0. La contraseña
    introducida debe tener al menos 1 caracter.
//Se hace asi, para poder escribir * cuando se escriba
    una contraseña.

void preguntar_contrasena(char *contra)
{

```

```

char c;
int encontrado=0, j=0;

while (!encontrado) //Bucle para que se compruebe si
    la contraseña tiene al menos 1 carácter.
{
    fflush(stdin);
    while((c=getch())!='r'&&j<8) //Lee el carácter
        , hasta 8.
    {
        if (c=='\b'&&j>0) //Si se borra algo, se
            borra un *, y se resta una posición del
            vector.
        {
            j--;
            printf("\b\b");
        }
        else if (c != '\b') //Si se escribe algo,
            se imprime *, y se introduce en vector
            contra.
        {
            contra[j++] = c;
            printf("*");
        }
    }
    if(j>0) //Si la cadena tiene mas de 1 carácter,
        salimos del bucle.
    {
        encontrado=1;
    }
    else
    {
        printf("\nLa contraseña debe tener entre 1-
            y 8 caracteres.\n");
        printf("Vuelva a introducir una contraseña-
            valida.\n");
        j=0;
    }
    contra[j] = '\0';
}
}

```

```

//Prototipo: void pregunta_localidad(Estr_Localidad *,
int, char *);
//Precondicion: Tener la estructura "localidad"
inicializada, con su contador. Ademas, necesitamos
una cadena, donde introducir la localidad
introducida.
//Postcondicion: Introducir en una cadena, la localidad
seleccionada de la lista.

void pregunta_localidad(Estr_Localidad *localidad, int
    numLocalidades, char *local)
{
    char vec_loc[4];
    int i, encontrado=0;

    listarLocalidades(localidad, numLocalidades); //  

        Imprime una lista con todas las localidades.

    while(encontrado==0) //Hasta que no se introduzca  

        una cadena correcta.
    {
        pregunta(vec_loc, 4);

        for(i=0; i<numLocalidades; i++) //Recorremos  

            toda la estructura "localidad".
        {
            if(strcmp(vec_loc, localidad[i].siglas)==0)
                //Si alguna respuesta coincide
                exactamente con las siglas, salimos del
                bucle.
            {
                encontrado=1;
                printf("Has seleccionado %s.\n",
                    localidad[i].localidad);

                strcpy(local, localidad[i].localidad);
                //Copiamos el nombre de la localidad
                en la cadena "local", para
                devolverla a otras funciones.
            }
        }
    }
}

```

```

        }
    }
}

//Prototipo: void pregunta_ruta(Estr_Localidad *, int ,
          Estr_Rutas **, int , int , char *);
//Precondicion: Tener la estructura "localidad" y "ruta
" inicializadas , con sus contadores. Ademas,
necesitamos una cadena , donde introducir la
localidad introducida .
//Postcondicion: Introducir en una cadena , la localidad
seleccionada de la lista , que este en una ruta .

void pregunta_ruta( Estr_Localidad *localidad , int
    numLocalidades , Estr_Rutas **ruta , int numRutas , int
    numRutas2 , char *rut2 )
{
    int i , j , x , encontrado=0;
    char rut [4];

    listarLocalidades (localidad , numLocalidades); //  

Imprime una lista con todas las localidades .

    while(encontrado==0) //Hasta que no se introduzca  

una cadena correcta .
    {
        pregunta(rut , 4);

        for(x=0; x<numLocalidades; x++) //Recorremos  

toda la estructura "localidad".
        {
            if(strcmp(rut , localidad [x]. siglas)==0) //  

Si alguna respuesta coincide exactamente  

con las siglas .
            {
                encontrado=1;
                printf("Has seleccionado %s.\n" ,
                    localidad [x]. localidad );

                for( i=0; i<numRutas; i++) //Recorremos  

todas las filas de la estructura "

```

```

        ruta”.
    {
        for( j=0; j<numRutas2; j++) //  

            Recorremos todas las columnas de  

            la estructura ”ruta”.
    {
        if(strcmp(rut, ruta[ i ][ j ].  

            localidad)==0) //Si alguna  

            localidad coincide con las  

            localidades de una ruta, se  

            introduce en la cadena.  

        {
            strcpy(rut2, ruta[ i ][ j ].  

                localidad);
        }
    }
    j=0; //Vamos reseteando las  

        columnas al saltar de fila .
}
}
}

//Prototipo: void preguntar_veh(Estr_Vehiculo *, int ,  

char *, int *);
//Precondicion: Tener la estructura ”vehiculo”  

inicializada, con su contador. Ademas, necesitamos  

una cadena, donde introducir la localidad  

introducida, y una variable bandera con puntero.  

//Postcondicion: Preguntar una matricula, y comprobar  

si existe en el sistema.

void preguntar_veh( Estr_Vehiculo *vehiculo , int  

    numVehiculos , char *opc , int *encontrado)
{
    int counter;

    pregunta(opc, 8); //Pedimos la matricula .
    for(counter=0;(counter<numVehiculos)&&((*encontrado  

    )==0);counter++) //Nos desplazamos por toda la

```

estructura "vehiculo", hasta encontrar la matricula.

```

{
    if(strcmp(opc, vehiculo [ counter ]. id_mat)==0) //  

        Si la matricula introducida esta en la  

        estructura.  

    {  

        (*encontrado)=1;  

        printf("Has_seleccionado_el_vehiculo_con_  

            matricula_%s.\n", opc);  

        system("PAUSE");
    }
}
if((*encontrado)==0) //Si no existe la matricula.
{
    system("cls");
    printf("No_se_ha_encontrado_ningun_vehiculo_con_  

        _la_siguiente_matricula:_%s.\n", opc);
    system("PAUSE");
}
}

```