

Manual de Programador

Proyecto: Rutas de Transporte en San Luis Potosí



programación II

Encargado:

Ledezma Ramos Guadalupe

Desarrolladores:

Rivera Carreon Brian Issai - 178481

Rodríguez Torres Sebastián - 179287

TABLA DE CONTENIDO

Tabla de contenido	1
Tabla de ilustraciones	3
Introducción.....	6
Código	6
Librerías.....	6
Estructura: Estacion.....	7
Estructura: Ruta.....	8
Estructura: rutasUsuario	8
Estructura: recorridoUsuario.....	9
Estructura: Usuario	10
Variable usuarioActivo.....	10
Principal.....	11
Logueo	13
Función inicio	13
Función login	14
Archivos.....	16
leerEstaciones:	20
leerRutasGlobales:	20
leerRutasUsuario:	22
leerRecorridoUsuario:.....	23
escribirUsuarios():.....	25
escribirEstaciones:.....	26
escribirRutasGlobales:	26
escribirRutasUsuario:.....	27
escribirRecorridoUsuario:	28
Menús.....	28
menuPrincipal	28
menuAdmin	29
menuPasajero	31
menuAutobus.....	32
Registros	34
registrarUser	34
registrarEstacion	36

registrarRuta	37
registrarRutasUsuario.....	38
Impresiones	40
imprimirUsuarios.....	40
imprimirPasajeros e imprimirAdmin	42
imprimirDatosUsuario e imprimirRutaPorId	42
imprimirEstaciones e imprimirRutas.....	43
ImprimirRutasUsuario	44
historial	45
Modificaciones	45
modificarEstacion:.....	46
modificarRuta:.....	47
modificarRutaUsuario:.....	47
Validaciones	50
Presentación.....	50
Conclusión	52

TABLA DE ILUSTRACIONES

Ilustración 1.Librerías	7
Ilustración 2.Constantes.....	7
Ilustración 3.Estacion	7
Ilustración 4.Ruta	8
Ilustración 5.rutaUsuario	9
Ilustración 6.recorridoUsuario.....	9
Ilustración 7.Usuario.....	10
Ilustración 8.UsuarioActivo	10
Ilustración 9.Librerías.....	11
Ilustración 10.Inicializacion.....	12
Ilustración 11.Lecturas	12
Ilustración 12.UsuarioDios.....	12
Ilustración 13.Escrituras	13
Ilustración 14.Inicio	13
Ilustración 15.opInicio.....	13
Ilustración 16.login o registrar.....	14
Ilustración 17.Login	14
Ilustración 18.Validacion de usuario	15
Ilustración 19.Inicializacion	17
Ilustración 20.LeerUsuarios.....	18
Ilustración 21.LeerEstaciones.....	20
Ilustración 22.LeerRutasGlobales.....	21
Ilustración 23.LeerRutasUsuario.....	22
Ilustración 24.LeerRecorridoUsuario	23
Ilustración 25.LeerRecorridoUsuario	24
Ilustración 26.LlamadoEscribir	24
Ilustración 27.EscribirUsuarios	25
Ilustración 28.EscribirEstaciones	26
Ilustración 29.EscribirRutasGlobales	26
Ilustración 30.EscribirRutasUsuario	27
Ilustración 31.EscribirRecorridoUsuario.....	28
Ilustración 32.menuPrincipal	29
Ilustración 33.menuAdmin	29

Ilustración 34.Registros	29
Ilustración 35.Consultas	30
Ilustración 36.Memoria	30
Ilustración 37.VerPorUsuario.....	30
Ilustración 38.Modificaciones	31
Ilustración 39.menuPasajero	31
Ilustración 40.MisRutas	32
Ilustración 41.EstadoDelAutobus	32
Ilustración 42.Subir	33
Ilustración 43.Bajar	33
Ilustración 44.Horas	33
Ilustración 45.registrarUser.....	34
Ilustración 46.idUser	34
Ilustración 47.Registro	35
Ilustración 48.ValidacionPassword	35
Ilustración 49.ListaSimpleUsuarios	36
Ilustración 50.GenerarId.....	36
Ilustración 51.ListaCircularEstaciones	37
Ilustración 52.IdRuta	37
Ilustración 53.LlenadoDatos.....	38
Ilustración 54.ListaSimpleRutas	38
Ilustración 55.RegistrarRutasUsuario	39
Ilustración 56.ListaSimpleRutasUsuario	39
Ilustración 57.Impresiones.....	40
Ilustración 58.ImpresionUsuarios	40
Ilustración 59.ImpresionUsuarios2	41
Ilustración 60.ImpresionEstados	42
Ilustración 61.ImpresionMemoria.....	42
Ilustración 62.ImprimirDatosUsuario	43
Ilustración 63.ImprimirRutas	44
Ilustración 64.ImprimirRutas	44
Ilustración 65.ImprimirRutasUsuario	45
Ilustración 66.ImprimirHistorial.....	45
Ilustración 67.ModificarEstacion	46

Ilustración 68.ModificarRuta	47
Ilustración 69.RutaNoEncontrada	47
Ilustración 70.LlenadoHorario	48
Ilustración 71.ModificarUsuario	48
Ilustración 72.ModificarNombre	49
Ilustración 73.ModificarPassword	49
Ilustración 74.ValidarEnteros	50
Ilustración 75.Presentacion	50
Ilustración 76.ImpresionPresentación.....	51

INTRODUCCIÓN

Dentro de este documento se encuentra explicada la lógica de los archivos y las funciones que integran el proyecto **Rutas de transporte en San Luis Potosí**, desarrollado durante el semestre Otoño 2022 de la Universidad Politécnica de San Luis potosí

CÓDIGO

Dentro de esta sección se encuentra la descripción de cada uno de los archivos que componen el proyecto y las Funciones que se hacen que funcione correctamente. En total se compone de 11 archivos:

- Librerías
- Presentación
- Principal
- Logueo
- Archivos
- Menús
- Registros
- Impresiones
- Modificaciones

Cada uno de estos archivos contiene diferentes Funciones las cuales son necesarias para el correcto funcionamiento del proyecto

LIBRERÍAS

El archivo librerías contiene la declaración De todas las librerías propias del Compilador

Ilustración 1. Librerías

```
1 //Librerías
2 //#include <conio.h>
3 #include <iostream> //cout, cin, fixed, endl
4 #include <stdlib.h> //malloc
5 #include <iomanip> //setw
6 #include <locale.h> //setlocale
7 #include <windows.h> //SetConsole
8 #include <string> //string
9 #include <fstream>
10 #include <sstream>
```

Además de la definición de algunas palabras clave Útiles para mejorar el Entendimiento del código

Ilustración 2. Constantes

```
12 #define hombre 1
13 #define mujer 0
14 #define activo 1
15 #define inactivo 0
16 #define admin 1
17 #define pasajero 0
18 #define siSubio 1
19 #define noSubio 0
```

También es en este archivo donde se definen las estructuras Con las cuales se generan las listas con las que funciona el proyecto

ESTRUCTURA: ESTACION

Ilustración 3. Estacion

```
23 //Lista circular
24 struct Estacion{
25     int id_lugar; //Auto incremental
26     string nombre;
27     //Variable que guarda un tiempo que hay entre la anterior parada y esta
28     int tiempoAestacion;
29     int estado;
30     Estacion *sig;
31 }*primEstacion,*ultEstacion;
```


La estructura estación contiene un entero el cual guarda el id de la estación, Contiene una variable Llamada nombre de tipo string Que almacenará el nombre de la estación, Cuenta además con una variable tipo int llamada tiempoAestación La cual está declarada para almacenar el tiempo que hay de trayecto entre esta y su anterior estación. una variable int llamada estado que almacenará 0 o 1 con la cual es posible identificar si está activa o inactiva, Además de un apuntador de nombre siguiente y tipo estación que apuntará a la Siguiente estación en el Recorrido

para hacer más fácil la programación Fueron declaradas 2 variables De tipo apuntador las cuales son globales, son primEstacion y ultEstacion.

ESTRUCTURA: RUTA

Ilustración 4. Ruta

```
33 //Lista simple de listas circulares
34 struct Ruta{
35     int id_ruta; //Auto incremental
36     int estado;
37     int horaInicio;
38     int horaFin;
39     //Lista Simple
40     Estacion *primero, *ultimo; //Destino y origen
41     Ruta *sig; //Recorer las rutas
42 }*primRuta,*ultRuta; //Rutas existentes
```

La estructura ruta contiene una variable de una variable de tipo entero La cual mencionara el ID de la ruta en específico además de otra llamada estado con la cual es posible conocer si la ruta está activa o inactiva.

La hora de inicio y hora fin son variables de tipo entero que almacenan las horas a cuáles empiezan a circular y terminan la actividad de esa ruta. Dentro de la estructura rota se encuentran 2 variables apuntadores de tipo estación las cuales apuntarán a La primera y última Estación que conforman esta ruta; Dentro de rutas se encuentra también un apuntador de tipo ruta llamado sig.

para facilitar la programación de las Funciones que afectarán las rutas se declararán 2 variables apuntadores globales de tipo ruta Las cuales son primRuta y ultRuta.

ESTRUCTURA: RUTASUSUARIO

Ilustración 5.rutaUsuario

```
44 struct rutasUsuario{
45     int id_ruta;
46     int dia;
47     int hora;
48     rutasUsuario *sig, *prev;
49 };
```

La estructura rutasUsuario es de apoyo porque con eso se creará una lista la cual almacenara las rutas que el usuario guardo y las cuales va a usar más adelante

Esta estructura contiene 3 enteros los cuales almacenan el id de la ruta, el día y la hora a la cual El usuario piensa utilizarla.

Esta estructura va a conformar una lista doble es por esto por lo que contiene 2 apuntadores de tipo rutasUsuario De nombre sig y prev

ESTRUCTURA: RECORRIDOUSUARIO

Ilustración 6.recorridoUsuario

```
51 //Lista simple
52 struct recorridoUsuario{
53     int idRecorrido; //Auto incremental
54     int idRuta;
55     //Con esta variable evaluamos si el usuario tomo la ruta o no subio
56     int estado;
57     int estacionSubida;
58     int estacionBajada;
59     /*horaSubida y horaBajada son arreglos
60     bidimensionales para guardar horas y minutos*/
61     int horaSubida[2];
62     int horaBajada[2];
63     recorridoUsuario *sig;
64 };
```

Estructura del recorrido usuario almacena el ID de recorrido, el ID de la ruta, el estado, la estación de subida y bajada en variables de tipo entero. Esa estructura cuenta con 2 arreglos bidimensionales de tipo entero los cuales se almacenarán la hora de subida y de bajada en horas y minutos.

recorridoUsuario conformará una lista simple por lo que cuenta con un apuntador siguiente de tipo recorridoUsuario

ESTRUCTURA: USUARIO

Ilustración 7.Usuario

```

66 //Lista de listas simple
67 struct Usuario{
68     int id_usuario; //Auto incremental
69     string nombre;
70     string apellido;
71     int TipoUsuario;
72     string password;
73     int sexo;
74     int estado;
75     //Rutas registradas por usuario
76     rutasUsuario *primero=NULL, *ultimo=NULL; //lista de rutas
77     recorridoUsuario *primRecorrido=NULL, *ultRecorrido=NULL; //Lista simple
78     Usuario *sig;//para recorrer todos los usuarios
79 }*primUser,*ultUser;

```

La estructura Usuario es la estructura principal de este proyecto, Dentro de ellas se almacena en una variable tipo entero el id del usuario, en variables tipo string se almacena en nombre y apellido del usuario además del password, la variable tipo entero llamada "TipoUsuario" Almacenará 1 o 0 Dependiendo de si el usuario es administrador o pasajero, También se guarda el sexo y el estado en variables de tipo entero.

Dentro de cada nodo usuario se encuentran 2 apuntdores llamados primero y último los cuales son de tipo estructura rutasUsuario, También almacena 2 apuntdores llamados primRecorrido y ultRecorrido los cuales son de tipo estructura recorridoUsuario.

Los usuarios conforman una lista de listas simple por lo que también contienen un apuntdor siguiente de tipo usuario y para facilitar la programación del proyecto se declararon 2 variables globales llamadas primUser y ultUser de tipo apuntdor usuario.

VARIABLE USUARIOACTIVO

Se declaró una variable de tipo usuario llamada usuario activar la cual es global y es utilizada en la mayor parte del proyecto, está variable está destinada a Almacenar el nodo del usuario activo que está manipulando en tiempo real el proyecto

Ilustración 8.UsuarioActivo

```

81 Usuario *usuarioActivo=NULL;

```

Después de realizar la declaración de las estructuras se incluyeron los demás archivos De los cuales se compone el proyecto y De los cuales se habla más adelante en este documento.

Ilustración 9. Librerías

```
83  #include "validaciones.h"
84  #include "presentacion.h"
85  #include "impresiones.h"
86  #include "eliminaciones.h"
87  #include "modificaciones.h"
88
89  #include "registros.h"
90
91  #include "archivos.h"
92  #include "menus.h"
93  #include "loggeo.h"
```

PRINCIPAL

El archivo de nombre principal.cpp es el archivo que contiene el hilo conductor del programa.

Al inicio se incluye El archivo que contiene las librerías ya antes declaradas y se inicia la función del main, Antes de Iniciar con la lectura de los archivos se define el usuario root el cual Es como el usuario Dios que nos sirve en caso de que el programa sea iniciado sin que cuente aún con administradores Y se declaran todas las variables de tipo estructura globales en nulo para evitar errores futuros

Ilustración 10. Inicialización

```
1  #include "librerias.h"
2  int main(){
3      //Usuario Dios
4      Usuario *root = new Usuario;
5      root->id_usuario = 0;
6      root->estado = activo;
7      root->password = "admin";
8      root->TipoUsuario = admin;
9
10     primEstacion = NULL;
11     ultEstacion = NULL;
12     primRuta = NULL;
13     ultRuta = NULL;
14     primUser = NULL;
15     ultUser = NULL;
16
```

Inmediatamente después de esto se procede a leer los archivos que contienen los datos de los usuarios las estaciones las rutas y se muestra la presentación del proyecto.

Ilustración 11. Lecturas

```
17  //Debemos leer los archivos de rutas, usuarios y estaciones
18  leerUsuarios();
19  leerEstaciones();
20  leerRutasGlobales();
21  presentacion();
```


Para inicializar el sistema principal se cuenta con un if el cuál evalúa el resultado de la función inicial mandar el usuario Dios y en caso de que esta sea negativa nos sacará del programa indicándole que hubo un error en el inicio de sesión, en caso contrario Continuamos al menú principal.

Ilustración 12. UsuarioDios

```
25  if(!inicio(root)){
26      gotoxy(35, 13);
27      cout<<"Saliendo...";
28      return 0;
29  }
```

Después de que el usuario decida terminar el programase evaluará si el usuario activo es un pasajero, en caso afirmativo se escribirán las rutas y el recorrido que realizó este usuario y después se procederá a escribir los datos de los usuarios, estaciones y las rutas globales en los archivos.

Ilustración 13.Escrituras

```
36  if(usuarioActivo->TipoUsuario==pasajero){
37     escribirRutasUsuario(usuarioActivo);
38     escribirRecorridoUsuario(usuarioActivo);
39 }
40
41 escribirUsuarios();
42 escribirEstaciones();
43 escribirRutasGlobales();
44 return 0;
45 }
```



LOGUEO

Dentro de este archivo podemos encontrarlas Funciones inicio y login.

FUNCIÓN INICIO


Esta función es de tipo void y recibe al usuario como parámetro, se evalúa en caso de que el primer usuario sea igual a nulo significa que aún no hay usuarios registrados por lo que se envía directamente a registrar usuario y después se inicia sesión hasta que éste sea exitoso.

Ilustración 14.Inicio

```
54  if(primUser == NULL){
55     registrarUser(root);
56  do{
57     usuarioActivo = login();
58 }while(usuarioActivo==NULL);
```

En caso de ya existir el primer usuario se presenta directamente el menú de inicio de sesión donde el usuario puede elegir entre salir, iniciar sesión o registrarse

Ilustración 15.opInicio

```
65  do{
66     cout<<"0.-Salir, 1.- Iniciar sesion, 2.-Registrarse: ";
67     opc = validaEntero();
68 }while(opc<0 || opc>2);
```

Según la opción que El usuario elija se le sacará coma se le redirigirá al inicio de sesión donde deberá iniciar sesión correctamente para continuar o de otra manera se le redirigir al registro En caso de haber elegido la opción Número 2

Ilustración 16.login o registrar

```

69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |

    if(opc==0){
        return false;
    }
    if(opc==1){
        //Aqui debemos sustituir los datos del superUsuario a los del Login
        usuarioActivo = login();
        if(usuarioActivo!=NULL){
            break;
        }
    }
    if(opc==2){
        root->TipoUsuario = pasajero;
        registrarUser(root);
        usuarioActivo = login();
        if(usuarioActivo!=NULL){
            break;
        }
    }
}

```

FUNCIÓN LOGIN

La función login es de tipo usuario, dentro de ésta se inicializa una variable usuario auxiliar que Es igual al primer usuario, además de otra variable tipo string password y 2 variables tipo entero De nombre id y nuevo, Se le presenta al usuario el menú para que ingrese el id del usuario con el que quiera iniciar sesión ingrese el número cero en caso de querer regresar.

Ilustración 17.Login

```

8 |
9 |
10 |
11 |
12 |
13 |
14 |
15 |
16 |
17 |
18 |

do{
    gotoxy(35, 13);
    cout<<"Id Usuario: ";
    gotoxy(35, 14);
    cout<<"0.- Regresar";
    gotoxy(47, 13);
    id = validaEntero();
    if(id==0){
        return NULL;
    }
}while(id>ultUser->id_usuario || id<primUser->id_usuario );

```

Después de que el usuario ingresa un id de usuario válido se le solicita la contraseña y procede la evaluación de los datos ingresados

Dentro de un ciclo se recorre la lista de usuarios Existentes Y se evalúa Que el ID ingresado por el usuario coincida con algún id de la lista, Cuando encuentra una coincidencia se compara si la contraseña ingresada es igual a la que contiene este usuario en la lista, En caso de que esto sea incorrecto Se le notificará al usuario de que la contraseña no coincide Iniciaré nuevamente el proceso de inicio de sesión.

En caso de que la contraseña coincida con la contraseña del nodo se evaluará si el usuario está activo y de ser esto cierto se le permitirá proceder, en caso contrario se notificará al usuario y se le pedirá volver a iniciar sesión.

En caso de que el id del usuario no exista en la lista se le avisara al usuario de que no existe y lo regresara al inicio de sesión nuevamente

Ilustración 18. Validación de usuario

```
23 do{
24     if(id==aux->id_usuario){ //Si encuentra al usuario
25         if(password==aux->password){ //Si la contraseña coincide
26             if(aux->estado == activo){ //En caso de que el usuario este activo
27                 gotoxy(35, 16);
28                 cout<<endl<<"--HAS INICIADO SESIÓN EXITOSAMENTE--";
29                 Sleep(2000);
30                 return aux;
31             }else{ //En caso de que el usuario este inactivo
32                 gotoxy(35, 16);
33                 cout<<"EL USUARIO SE ENCUENTRA DESHABILITADO";
34                 return NULL;
35             }
36         }else{ //Si la contraseña no coincide
37             gotoxy(35, 17);
38             cout<<"¡LA CONTRASEÑA NO COINCIDE!";
39             return NULL;
40         }
41     }else{
42         aux=aux->sig;
43     }
44 }while(aux!=NULL);
45 cout<<endl<<"¡EL USUARIO NO EXISTE!"<<endl;
46 return NULL;
47 }
```


ARCHIVOS

Las funciones usadas para trabajar con archivos se encuentran en la librería `archivos.h`, las cuales se dividen en 2 partes, las funciones usadas para escribir en los archivos y las funciones usadas para leer de los archivos.

Las funciones de escritura son:

- `escribirUsuarios`, la cual escribe en el archivo de destino la estructura de usuarios, tanto administradores como pasajeros.
- `escribiEstaciones`, la cual escribe en el archivo de destino la estructura de estaciones.
- `escribirRutasGlobales`, la cual escribe en el archivo de destino la estructura Ruta
- `escribirRutasUsuario`, la cual escribe en el archivo de destino las rutas que el pasajero, el cual inició sesión, guardo en su historial (función `rutasUsuario`).
- `escribirRecorridoUsuario`, la cual escribe en el archivo de destino el historial de cuando el pasajero, el cual inició sesión, toma alguna ruta en específico.

Las funciones de lectura son:

- `leerUsuarios`, la cual lee del archivo de destino los datos guardados y los almacena en la estructura `usuarios`
- `leerEstaciones`, la cual lee del archivo de destino los datos guardados y los almacena en la estructura de estaciones.
- `leerRutasGlobales`, la cual lee del archivo de destino los datos guardados y los almacena en la estructura de rutas.
- `leerRutasUsuario`, la cual lee del archivo de destino los datos guardados del usuario que inició sesión y los guarda en la estructura `rutasUsuario`.
- `leerRecorridoUsuario`, la cual lee del archivo de destino los datos guardados del usuario que inició sesión y los guarda en la estructura `recorridoUsuario`

Ilustración 19.Inicializacion

```
1  #include "librerias.h"
2  int main(){
3      //Usuario Dios
4      Usuario *root = new Usuario;
5      root->id_usuario = 0;
6      root->estado = activo;
7      root->password = "admin";
8      root->TipoUsuario = admin;
9
10     primEstacion = NULL;
11     ultEstacion = NULL;
12     primRuta = NULL;
13     ultRuta = NULL;
14     primUser = NULL;
15     ultUser = NULL;
16
17     //Debemos leer los archivos de rutas, usuarios y estaciones
18     leerUsuarios();
19     leerEstaciones();
20     leerRutasGlobales();
```

Una vez iniciado el programa llamamos a las funciones leerUsuarios, leerEstaciones, leerRutasGlobales:

Ilustración 20. LeerUsuarios

```

216 void leerUsuarios(){
217     Usuario *nuevo = NULL;
218     ifstream arch("Usuarios.xls");
219     string linea, c;
220     if(arch.fail()) cerr<<"No se encontro ningun archivo"<<endl;
221     else{
222         while (getline(arch, linea)){
223             nuevo = new Usuario;
224             nuevo->sig = NULL;
225             stringstream lee(linea);
226             lee>>nuevo->id_usuario;
227             getline(lee, c, '\t');
228             getline(lee, nuevo->nombre, '\t');
229             getline(lee, nuevo->apellido, '\t');
230             lee>>nuevo->sexo;
231             getline(lee, c, '\t');
232             lee>>nuevo->TipoUsuario;
233             getline(lee, c, '\t');
234             lee>>nuevo->estado;
235             getline(lee, c, '\t');
236             getline(lee, nuevo->password, '\n');
237
238             if(primUser==NULL){
239                 primUser = nuevo;
240                 ultUser = nuevo;
241             }else{
242                 ultUser->sig = nuevo;
243                 ultUser=nuevo;
244             }
245         }
246     }
247     arch.close();
248 }
249

```

Dentro de la función generamos un apuntador el cual nos ayudará a llenar la estructura. Abrimos el archivo de destino, en este caso será el archivo llamado "Usuarios.xls".

Posteriormente creamos 2 variables del tipo string las cuales nos ayudaran a leer del archivo. En un comienzo verificamos si el archivo existe, en caso de que marque error no leemos nada.

Si el archivo es encontrado comenzamos a leer, con la función `getline(arch, línea)` vamos avanzando entre cada línea del archivo .xls.

Creamos un nuevo nodo del tipo usuario y leemos la primera línea con stringstream lee(linea). Para leer un valor del tipo entero usamos la siguiente sintaxis: lee>>[variable]; y getline(lee, c, '\t'); en caso de requerir leer una cadena usamos la siguiente syntaxis: getline(lee, [variable], '\t');

Y esto lo repetimos para cada variable que queramos leer, hay que tomar en cuenta el orden en el que se guardó, ya que será el mismo orden en el que tendremos que leer el archivo.

Una vez llenado el nodo lo acomodamos en la lista con las funciones de ordenamiento que se encuentran en el if: else:. Una vez terminamos de leer el archivo cerramos el mismo.

Para las funciones leerEstaciones y leerRutasGlobales usamos la misma syntaxis, únicamente cambiando las variables a guardar y el ordenamiento del final por el correspondiente a la estructura.

LEERESTACIONES:

Ilustración 21. LeerEstaciones

```

250 void leerEstaciones(){
251     Estacion *nuevo = NULL;
252     ifstream arch("Estaciones.xls");
253     string linea, c;
254     if(arch.fail()) cerr<<"No se encontro ningun archivo"<<endl;
255     else{
256         while(getline(arch, linea)){
257             nuevo = new Estacion;
258             nuevo->sig=NULL;
259             stringstream lee(linea);
260             lee>>nuevo->id_lugar;
261             getline(lee, c, '\t');
262             getline(lee, nuevo->nombre, '\t');
263             lee>>nuevo->estado;
264             getline(lee, c, '\t');
265             lee>>nuevo->tiempoAestacion;
266             getline(lee, c, '\t');
267             if(primEstacion==NULL){
268                 primEstacion = nuevo;
269                 ultEstacion = nuevo;
270                 ultEstacion->sig=primEstacion;
271             }else{
272                 ultEstacion->sig=nuevo;
273                 ultEstacion=nuevo;
274                 ultEstacion->sig=primEstacion;
275             }
276         }
277     }
278     arch.close();
279 }
280

```

LEERRUTASGLOBALES:

Ilustración 22. LeerRutasGlobales

```

281 void leerRutasGlobales(){
282     Ruta *nuevo = NULL;
283     Estacion *aux = primEstacion;
284     ifstream arch("RutasGlobales.xls");
285     string linea, c;
286     int primero;
287     int ultimo;
288
289     if(arch.fail()) cerr<<"No se encontro ningun archivo"<<endl;
290     else{
291
292         while(getline(arch, linea)){
293             nuevo = new Ruta;
294             nuevo->sig = NULL;
295             stringstream lee(linea);
296             lee>>nuevo->id_ruta;
297             getline(lee, c, '\t');
298             lee>>primero;
299             getline(lee, c, '\t');
300             while(primero!=aux->id_lugar){
301                 aux=aux->sig;
302             }
303
304             nuevo->primero=aux;
305             lee>>ultimo;
306             getline(lee, c, '\t');
307             while(ultimo!=aux->id_lugar){
308                 aux=aux->sig;
309             }
310             nuevo->ultimo=aux;
311             lee>>nuevo->horaInicio;
312             getline(lee, c, '\t');
313             lee>>nuevo->horaFin;
314             getline(lee, c, '\t');
315             lee>>nuevo->estado;
316             getline(lee, c, '\t');
317
318             if(primRuta == NULL){
319                 primRuta = nuevo;
320                 ultRuta = nuevo;
321             }else{
322                 ultRuta->sig = nuevo;
323                 ultRuta = nuevo;
324             }
325         }
326     }
327     arch.close();
328 }

```

LEERRUTASUSUARIO:

Ilustración 23. LeerRutasUsuario

```

175 void leerRutasUsuario(Usuario *usuarioActual){ //Funcion para leer las rutas
176     rutasUsuario *aux = usuarioActual->primero;
177     rutasUsuario *nuevo=NULL;
178     int num = usuarioActual->id_usuario;
179     char numero[8];
180     enteroAcad(num, numero);
181     char nombre[35] = "RutasUsuarios/rutaUsuario";
182     char final[5]=".xls";
183     strcat(nombre, numero);
184     strcat(nombre, final);
185
186     ifstream arch(nombre);
187     string linea, c;
188     gotoxy(35, 7);
189     if(arch.fail()) cerr<<"No se encontro ningun archivo"<<endl;
190     else{
191         while (getline(arch, linea)){
192             nuevo = new rutasUsuario;
193             nuevo->sig = NULL;
194             stringstream lee(linea);
195             lee>>nuevo->id_ruta;
196             getline(lee, c, '\t');
197             lee>>nuevo->dia;
198             getline(lee, c, '\t');
199             lee>>nuevo->hora;
200             getline(lee, c, '\t');
201
202             if(usuarioActual->primero==NULL){
203                 usuarioActual->primero=nuevo;
204                 usuarioActual->ultimo=nuevo;
205             }else{
206                 nuevo->prev=usuarioActual->ultimo;
207                 usuarioActual->ultimo->sig = nuevo;
208                 usuarioActual->ultimo = nuevo;
209                 usuarioActual->ultimo->sig = NULL;
210             }
211         }
212     }
213     arch.close();
214 }

```

Para esta función le mandamos el parámetro de usuarioActual, el cual es el usuario que inicio sesión. Para esta función generamos las variables aux y nuevo. Almacenamos el id del usuario, y usamos la función enteroAcad mandándole el id y la cadena en la cual almacenaremos el número, la función enteroAcad nos ayuda a pasar una variable del tipo entero al tipo char.

Después generamos la variable de la ruta donde estará el archivo destino, en este caso es en la carpeta RutasUsuario y el nombre del archivo, al cual le concatenamos el id del usuario y la extensión .xls

Abrimos el archivo y verificamos que el archivo exista. De igual manera que con las funciones de leerUsuarios y leerEstaciones, leeremos el archivo, guardaremos en su ubicación y lo enlazaremos a la lista.

Para la función leerRecorridoUsuario usamos la misma lógica que con la función de leerRutasUsuario

LEERRECORRIDOUSUARIO:

Ilustración 24. LeerRecorridoUsuario

```
115 void leerRecorridoUsuario(Usuario *usuarioActual){
116     recorridoUsuario *aux = usuarioActual->primRecorrido;
117     recorridoUsuario *nuevo=NULL;
118     int num = usuarioActual->id_usuario;
119     char numero[8];
120     enteroAcad(num, numero);
121     char nombre[35] = "RecorridosUsuario/recorridoUsuario";
122     char final[5]=".xls";
123     strcat(nombre, numero);
124     strcat(nombre, final);
125     ifstream arch(nombre);
126     string linea, c;
127     int ayuda1;
128     int ayuda2;
129     int ayuda3;
130     int ayuda4;
131     gotoxy(35, 7);
132     if(arch.fail()) cerr<<"No se encontro ningun archivo"<<endl;
133     else{
134         while(getline(arch, linea)){
135             nuevo = new recorridoUsuario;
136             nuevo->sig=NULL;
137             stringstream lee(linea);
138             lee>>nuevo->idRecorrido;
139             getline(lee, c, '\t');
140             lee>>nuevo->idRuta;
141             getline(lee, c, '\t');
142             lee>>nuevo->estado;
143             getline(lee, c, '\t');
144             lee>>nuevo->estacionSubida;
145             getline(lee, c, '\t');
146             lee>>nuevo->estacionBajada;
147             getline(lee, c, '\t');
```


Ilustración 25. LeerRecorridoUsuario

```

148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173

```

```

lee>>ayuda1;
getline(lee, c, '\t');
lee>>ayuda2;
getline(lee, c, '\t');
lee>>ayuda3;
getline(lee, c, '\t');
lee>>ayuda4;
getline(lee, c, '\t');

nuevo->horaSubida[0]=ayuda1;
nuevo->horaSubida[1]=ayuda2;
nuevo->horaBajada[0]=ayuda3;
nuevo->horaBajada[1]=ayuda4;
if(usuarioActual->primRecorrido == NULL){
    usuarioActual->primRecorrido = nuevo;
    usuarioActual->ultRecorrido = nuevo;
}else{
    usuarioActual->ultRecorrido->sig = nuevo;
    usuarioActual->ultRecorrido = nuevo;
    nuevo->sig=NULL;
}
}
}
arch.close();

```

Ilustración 26. LlamadoEscribir

```

35
36
37
38
39
40
41
42
43

```

```

if(usuarioActivo->TipoUsuario==pasajero){
    escribirRutasUsuario(usuarioActivo);
    escribirRecorridoUsuario(usuarioActivo);
}

escribirUsuarios();
escribirEstaciones();
escribirRutasGlobales();

```

Una vez terminado el programa, escribimos todas las estructuras en los archivos, para las funciones escribirRutasUsuario y escribirRecorridoUsuario verificamos antes si el usuarioActivo es del tipo pasajero. En caso de serlo llamamos a las funciones, de lo contrario no guardaremos estas.

ESCRIBIRUSUARIOS():

Ilustración 27.EscribirUsuarios

```
1 void escribirUsuarios(){
2     Usuario *aux = primUser;
3     ofstream archivo("Usuarios.xls");
4     while(aux!=NULL){
5         archivo<<aux->id_usuario<<"\t";
6         archivo<<aux->nombre<<"\t";
7         archivo<<aux->apellido<<"\t";
8         archivo<<aux->sexo<<"\t";
9         archivo<<aux->TipoUsuario<<"\t";
10        archivo<<aux->estado<<"\t";
11        archivo<<aux->password<<"\n";
12        aux = aux->sig;
13    }
14    archivo.close();
15 }
```

Para esta función generamos un auxiliar el cual lo inicializamos en el primUser. Abrimos el archivo de Usuarios.xls y mientras el auxiliar sea diferente de NULL guardamos en el archivo.

la syntaxis del guardado en el archivo es la siguiente: archivo<<[variable]<<"\t"; en caso de ser la última variable a guardar cambiamos el \t por un \n, movemos el auxiliar a su siguiente y una vez se guarde todo cerramos el archivo.

Para las funciones escribirEstaciones y escribirRutasGlobales usaremos la misma syntaxis que antes.

ESCRIBIRESTACIONES:

Ilustración 28.EscribirEstaciones

```
18 void escribirEstaciones(){
19     Estacion *aux = primEstacion;
20     ofstream archivo("Estaciones.xls");
21     do{
22         archivo<<aux->id_lugar<<"\t";
23         archivo<<aux->nombre<<"\t";
24         archivo<<aux->estado<<"\t";
25         archivo<<aux->tiempoAestacion<<"\n";
26         aux=aux->sig;
27     }while(aux!=primEstacion);
28     archivo.close();
29 }
```

ESCRIBIRRUTASGLOBALES:

Ilustración 29.EscribirRutasGlobales

```
31 void escribirRutasGlobales(){
32     Ruta *aux = primRuta;
33     ofstream archivo("RutasGlobales.xls");
34     while(aux != NULL){
35         archivo<<aux->id_ruta<<"\t";
36         archivo<<aux->primero->id_lugar<<"\t";
37         archivo<<aux->ultimo->id_lugar<<"\t";
38         archivo<<aux->horaInicio<<"\t";
39         archivo<<aux->horaFin<<"\t";
40         archivo<<aux->estado<<"\n";
41         aux=aux->sig;
42     };
43     archivo.close();
44 }
```

ESCRIBIRRUTASUSUARIO:

Ilustración 30. EscribirRutasUsuario

```

64 void escribirRutasUsuario(Usuario *usuarioActivo){
65     rutasUsuario *aux = usuarioActivo->primero;
66     int num = usuarioActivo->id_usuario;
67     char numero[8];
68     enteroAcad(num, numero);
69     char nombre[35] = "RutasUsuarios/rutaUsuario";
70     char final[5] = ".xls";
71     strcat(nombre, numero);
72     strcat(nombre, final);
73     ofstream archivo(nombre);
74     while(aux != NULL){
75         archivo<<aux->id_ruta<<"\t";
76         archivo<<aux->dia<<"\t";
77         archivo<<aux->hora<<"\n";
78         aux=aux->sig;
79     }
80     archivo.close();
81 }

```

Para escribir las rutas del usuario recibimos el parámetro del usuarioActivo, el cual será el usuario que inició sesión y realizamos lo mismo que en las funciones de leerRutasUsuario y leerRecorridoUsuario para abrir el archivo de destino. Una vez encontrado y abierto ese archivo entonces comenzamos a escribir en el mismo de la misma manera que con las funciones anteriores.

Para la función escribirRecorridoUsuario usamos el mismo algoritmo que con la función de escribirRutasUsuario

ESCRIBIRRECORRIDOUSUARIO:

Ilustración 31.EscribirRecorridoUsuario

```

02
83 void escribirRecorridoUsuario(Usuario *usuarioActivo){
84     recorridoUsuario *aux = usuarioActivo->primRecorrido;
85     int num = usuarioActivo->id_usuario;
86     char numero[8];
87     enteroAcad(num, numero);
88     char nombre[55] = "RecorridosUsuario/recorridoUsuario";
89     char final[5] = ".xls";
90     strcat(nombre, numero);
91     strcat(nombre, final);
92     ofstream archivo(nombre);
93     int ayuda1, ayuda2, ayuda3, ayuda4;
94     while(aux!=NULL){
95         ayuda1=aux->horaSubida[0];
96         ayuda2=aux->horaSubida[1];
97         ayuda3=aux->horaBajada[0];
98         ayuda4=aux->horaBajada[1];
99         archivo<<aux->idRecorrido<<"\t";
100        archivo<<aux->idRuta<<"\t";
101        archivo<<aux->estado<<"\t";
102        archivo<<aux->estacionSubida<<"\t";
103        archivo<<aux->estacionBajada<<"\t";
104        archivo<<ayuda1<<"\t";
105        archivo<<ayuda2<<"\t";
106        archivo<<ayuda3<<"\t";
107        archivo<<ayuda4<<"\n";
108        aux=aux->sig;
109    }
110    archivo.close();
111 }
112

```

MENÚS

Para la navegación dentro del proyecto se crearon diferentes Funciones de menú, esto servirán para llevar el control de cada proceso durante la ejecución del programa dependiendo del tipo de usuario que se el usuario que esté manipulando.

MENUPRINCIPAL

Menú principal es la función que controla hacia cuál submenú se debe ir el programa, si el tipo de usuario del usuario activo es administrador se le redirigirá hacia el menú de administrador, en caso de no ser administrador, Se le redirigirá al menú de pasajero.

Ilustración 32.menuPrincipal

```

602 void menuPrincipal(){
603     cout<<endl;
604     if(usuarioActivo->TipoUsuario == admin){
605         menuAdmin();
606     }else{
607         menuPasajero();
608     }
609 }
    
```

MENUADMIN

El menú administrador es el menú al que se le dirige el usuario en caso de ser un administrador, dentro de este menú podrá ver opciones que sólo son disponibles para los administradores como lo son el registrar, ver y modificar Todas las listas de usuarios, ruta, estaciones y el historial de los usuarios.

Ilustración 33.menuAdmin

```

16 cout<<"Bienvenid@ administrador/a "<<usuarioActivo->nombre;
17 gotoxy(35, 7);
18 cout<<"1.-Registrar";
19 gotoxy(35, 8);
20 cout<<"2.-Ver";
21 gotoxy(35, 9);
22 cout<<"3.-Modificar";
23 gotoxy(35, 10);
24 cout<<"0.-Salir";
    
```

El administrador ingresa la opción que desea y se le redirige mediante un ciclo switch A la sección de la opción deseada.

En caso de haber seleccionado registrar, se le redirige a un submenú de registro donde se le muestran las siguientes opciones

Ilustración 34.Registros

```

42 do{
43     system("cls");
44     caja();
45     SetConsoleTextAttribute(hCon, 11);
46     gotoxy(35, 5);
47     cout<<"REGISTROS";
48     SetConsoleTextAttribute(hCon, 14);
49     gotoxy(35, 6);
50     cout<<"1.-Registrar Estacion";
51     gotoxy(35, 7);
52     cout<<"2.-Registrar Ruta";
53     gotoxy(35, 8);
54     cout<<"3.-Registrar Usuario/Administrador";
55     gotoxy(35, 9);
56     cout<<"0.-Salir";
57     gotoxy(35, 10);
58     opc = validaEntero();
59 }while(opc>3 || opc<0);
    
```

Después de este submenú se evalúa la opción en otro switch y se invoca a la función necesaria. Ese patrón se repite en cada submenú

En caso de haber seleccionado la opción 2les llegó un inicio se le redirigir al siguiente submenú de consultas:

Ilustración 35.Consultas

```
82 | do{
83 |     system("cls");
84 |     caja();
85 |     SetConsoleTextAttribute(hCon, 11);
86 |     gotoxy(35, 5);
87 |     cout<<"CONSULTAS";
88 |     SetConsoleTextAttribute(hCon, 14);
89 |     gotoxy(35, 6);
90 |     cout<<"1.-Ver todos los Pasajeros";
91 |     gotoxy(35, 7);
92 |     cout<<"2.-Ver todos los Administradores";
93 |     gotoxy(35, 8);
94 |     cout<<"3.-Ver todas las Estaciones";
95 |     gotoxy(35, 9);
96 |     cout<<"4.-Ver todas las Rutas";
97 |     gotoxy(35, 10);
98 |     cout<<"0.-Salir";
99 |     gotoxy(35, 11);
100 |     opc = validaEntero();
101 | }while(opc>4 || opc<0);
```

En esta parte se le solicita al usuario administrador si quiere ver las direcciones de memoria de los nodos que conforman cada lista.

Ilustración 36.Memoria

```
105 | do{
106 |     gotoxy(35, 12);
107 |     cout<<"Ver direcciones de memoria? 1: si, 0:No ";
108 |     memoria=validaEntero();
109 | }while(memoria>1 || memoria<0 );
```

Una vez presentados en pantalla los datos de los nodos de los Pasajeros, se le pregunta al ministro por si quiere verlas rutas de usuario, el historial del usuario O regresar al menú anterior

Ilustración 37.VerPorUsuario

```
125 | cout<<"1.-Ver rutas de usuario";
126 | gotoxy(35, 21);
127 | cout<<"2.-Ver historial de usuario";
128 | gotoxy(35, 22);
129 | cout<<"0.-Regresar";
130 | do{
131 |     gotoxy(35, 23);
132 |     opc = validaEntero();
133 | }while(opc>2 || opc<0);
```

sí al administrador quiere consultar las rutas o el historial del usuario, Se les solicita el id del usuario a consultar, luego se muestran Las rutas del usuario en el historial de recorridos del usuario.

Si en el menú principal del menú usuario El usuario eligió la opción 3, se le mostrará el submenú de modificaciones donde podrá decidir si modificar estaciones, rutas, usuarios, administradores o regresar al menú anterior

Ilustración 38.Modificaciones

```
245 
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
```

```
do{
    system("cls");
    caja();
    SetConsoleTextAttribute(hCon, 11);
    gotoxy(35, 5);
    cout<<"MODIFICACIONES";
    SetConsoleTextAttribute(hCon, 14);
    gotoxy(35, 6);
    cout<<"1.-Estacion";
    gotoxy(35, 7);
    cout<<"2.-Ruta";
    gotoxy(35, 8);
    cout<<"3.-Usuario";
    gotoxy(35, 9);
    cout<<"4.-Administrador";
    gotoxy(35, 10);
    cout<<"0.-Salir";
    gotoxy(35, 11);
    opc = validaEntero();
}while(opc>4 || opc<0);
```

El menú estará en repetición todo el tiempo hasta que el usuario seleccione la opción salir qué cendra mediante una indicación break el ciclo while Que funciona indefinidamente

MENUPASAJERO

El menú administrador es el menú al que se le dirige al usuario en caso de ser pasajero, dentro de este menú encontraremos las opciones de "Mis Rutas" y de "Estado de autobús."

Ilustración 39.menuPasajero

```
//Mostramos el menu del usuario
gotoxy(40, 5);
cout<<".- MENU DE PASAJERO -.";
SetConsoleTextAttribute(hCon, 14);
gotoxy(40, 6);
cout<<"Bienvenid@ pasajero@ "<<usuarioActivo->nombre;
gotoxy(40, 7);
cout<<"1.-Mis Rutas";
gotoxy(40, 8);
cout<<"2.-Estado del autobus";
gotoxy(40, 9);
cout<<"0.-Salir";
//Validamos la opcion que elija el usuario
..
```


Dentro del apartado Mis Rutas el usuario puede ver sus rutas, registrar una ruta nueva, modificar una ruta o eliminar una ruta.

Ilustración 40. MisRutas

```
cout<<".- MIS RUTAS -.";
SetConsoleTextAttribute(hCon, 14);
gotoxy(40, 6);
//Aquí mostramos otro menu de las rutas del us
cout<<"1.-Ver mis rutas";
gotoxy(40, 7);
cout<<"2.-Registrar una ruta";
gotoxy(40, 8);
cout<<"3.-Eliminar ruta";
gotoxy(40, 9);
cout<<"4.-Modificar una ruta";
gotoxy(40, 10);
cout<<"0.-Regresar";
```

Y en el apartado de Estado del autobús el usuario podrá ver su historial de las rutas y registrar un nuevo recorrido.

Ilustración 41. EstadoDelAutobus

```
printf("\a");
do{
    do{
        system("cls");
        caja();
        SetConsoleTextAttribute(hCon, 11);
        gotoxy(35, 5);
        cout<<".- ESTADO DEL AUTOBUS -.";
        SetConsoleTextAttribute(hCon, 14);
        gotoxy(35, 6);
        cout<<"1.-Ha llegado el autobus!";
        gotoxy(35, 7);
        cout<<"2.- Ver Historial";
        gotoxy(35, 8);
        cout<<"0.-Regresar"<<endl;
        gotoxy(35, 9);
        opc = validaEntero();
    }while(opc>2 || opc<0);
```

MENUAUTOBUS

Dentro del menú autobús le mostramos al usuario las rutas que guardo, y le preguntamos cuál ruta va a tomar. En caso de que la ruta no exista o esté inactiva se le avisará al usuario. Después le preguntamos si va a subir a la ruta o no.

Ilustración 42.Subir

```
cout<<"Quieres subir a la ruta N."<<ruta->id_ruta<<"?";
gotoxy(35, 6);
cout<<"1.-Subir";
gotoxy(35, 7);
cout<<"2.-No subir";
gotoxy(35, 8);
opc = validaEntero();
```

En caso de subirse se le mostrará consecutivamente las estaciones por las que pasa la ruta y le preguntamos si quiere bajarse en esa estación o si quiere seguir a la siguiente.

Ilustración 43.Bajar

```
do{
    printf("\a");
    system("cls");
    caja();
    SetConsoleTextAttribute(hCon, 14);
    gotoxy(35, 5);
    cout<<"Estacion "<<auxEstacion->nombre;
    gotoxy(35, 6);
    cout<<"1.-Bajar aqui"<<endl;
    gotoxy(35, 7);
    cout<<"2.-continuar"<<endl;
    gotoxy(35, 8);
    opc = validaEntero();
}while(opc>2 || opc<1);
```

Una vez decida bajarse se sumará el tiempo que tiene cada estación para así mostrar la hora en que se baja de la ruta.

Ilustración 44.Horas

```
if(opc==1){
    recorridoActual->estacionBajada = auxEstacion->id_lugar;

    t1 = (recorridoActual->horaSubida[1]+tiempo)/60; //horas en un entero
    t2 = (recorridoActual->horaSubida[1]+tiempo)/60; //Horas en un flotante

    if(t2<1){ //No completo una hora
        recorridoActual->horaBajada[0] = recorridoActual->horaSubida[0];
        recorridoActual->horaBajada[1] += tiempo ;
    }else{
        if(t1>t2){
            recorridoActual->horaBajada[0] += t1-1;
            recorridoActual->horaBajada[1] = recorridoActual->horaSubida[1] + tiempo - (t1-1)*60;
        }else{
            recorridoActual->horaBajada[0] += t1;
            recorridoActual->horaBajada[1] = recorridoActual->horaSubida[1] + tiempo - (t1)*60;
        }
    }

    break;
}
```

En caso de decidir no subirse entonces la hora de subida y de bajada se pondrá en automático en 0, aun así, se guarda en el historial.

REGISTROS

El archivo de registros contiene las Funciones registrarEstacion, registrarRuta, registrarRutasUsuario y registrarUser

REGISTRARUSER

Esta función recibe como parámetro un apuntador de tipo usuario, Genera un apuntador de tipo usuario llamado nuevo y le Genera un nuevo espacio en memoria.

Ya que es un nuevo usuario su estado siempre va a ser activo y en su Apuntador siguiente contendrá nulo.

Ilustración 45.registrarUser

```
1 void registrarUser(Usuario *root){
2     Usuario *nuevo = new Usuario; //Generamos usuario
3     int opc=0;
4     string password, password2;
5     bool bandera = true;
6
7     nuevo->sig=NULL;
8     nuevo->estado=activo; //Los nuevos usuarios siempre estaran activos
```

En caso de que el primer usuario no exista, el id Del usuario será uno coma en caso contrario ser igual al id del último usuario más uno.

Ilustración 46.idUser

```
10 if(primUser==NULL){
11     //Si aun no hay ningun usuario, el ptimero tendra id = 1
12     nuevo->id_usuario = 1;
13 }else{
14     //Si ya existen usuarios, aumentamos de 1 en 1 los id's
15     nuevo->id_usuario = (ultUser->id_usuario)+1;
16 }
```

En caso de que el usuario que está registrando sea administrador se le presentará la opción de registrar a un pasajero o, a un administrador, En caso de que quien registra No sea un administrador el tipo de usuario del nuevo usuario será siempre pasajero

Ilustración 47.Registro

```

21      cout<<"REGISTRO";
22      //Si es admin puede elegir si registrar a un pasajero o a otro admin
23      if(root->TipoUsuario==admin){
24          gotoxy(35, 4);
25          cout<<"Registrar: 1.-Pasajero, 2.-Administrador: ";
26          do{
27              opc = validaEntero(); //Validamos entrada
28          }while(opc!=2 && opc!=1);
29          if(opc==1){
30              nuevo->TipoUsuario=pasajero;
31          }
32          if(opc==2){
33              nuevo->TipoUsuario=admin;
34          }
35      }else{
36          //Cuando es pasajero solo puede registrarse como pasajero
37          nuevo->TipoUsuario = pasajero;
38      }

```

Posteriormente se le pide el nombre y apellido del nuevo usuario además de su sexo.

Después se solicita al usuario que ingrese su password y que lo ingrese una segunda vez para confirmarlo, en caso de que Los password no coincidan Se le notifica al usuario y se solicita nuevamente

Ilustración 48.ValidacionPassword

```

53      do{
54          bandera = true;
55          gotoxy(35, 8);
56          cout<<"Ingresa tu password: ";
57          cin>>nuevo->password;
58          gotoxy(35, 9);
59          cout<<"Confirma tu password: ";
60          cin>>password2;
61
62          if(nuevo->password!=password2){
63              //Hacemos una pequeña confirmacion de password
64              gotoxy(35, 10);
65              cout<<"Los password no coinciden!, intenta nuevamente";
66              bandera = false;
67          }
68      }while(bandera == false);

```

Al finalizar el registro se procede a enlazar el nuevo nodo usuario a la lista de usuarios, en caso de ser el primer usuario registrado, este será el primero y último usuario, en caso contrario se enlazará al final de la lista y se le Etiquetará como último usuario.

Ilustración 49. ListaSimpleUsuarios

```
70 //Los usuarios son una lista simple
71 if(primUser == NULL){
72     primUser = nuevo;
73     ultUser = nuevo;
74 }else{
75     ultUser->sig = nuevo;
76     ultUser = nuevo;
77 }
78 gotoxy(35, 12);
79 cout<<"¡REGISTRO EXITOSO!"<<endl;
```

REGISTRARESTACION

Sólo los usuarios de tipo administrador pueden acceder a esta opción a través del menú administrador, Dentro de ella generamos un nuevo nodo de tipo estación Y le asignamos el id correspondiente De manera seriada.

Ilustración 50. GenerarId

```
93 /*Generamos ID's seriados a las estaciones
94    y lo mostramos el de la que se acaba de generar */
95 if(primEstacion == NULL){
96     nuevo->id_lugar = 1;
97 }else{
98     nuevo->id_lugar = ultEstacion->id_lugar+1;
99 }
```

Posteriormente le mostramos el nuevo id de la nueva estación al usuario y solicitamos el nombre de la estación, también se le asigna a la estación un tiempo en el cual tarda en llegar desde la última estación A la nueva

Finalmente Enlazamos el nuevo nodo estación a la lista circular De estaciones y notificamos que se ha creado exitosamente

Ilustración 51.ListaCircularEstaciones

```
113 //Las estaciones son una lista circular simple
114 if(primEstacion == NULL){
115     primEstacion = nuevo;
116     ultEstacion = nuevo;
117 }else{
118     ultEstacion->sig = nuevo;
119     ultEstacion = nuevo;
120     ultEstacion->sig = primEstacion;
121 }
122 gotoxy(35, 9);
123 cout<<"¡ESTACION REGISTRADA EXITOSAMENTE!"<<endl;
```

REGISTRARRUTA

Al igual que las estaciones, sólo un administrador puede registrar rutas globales, y es necesario que existan al menos 2 estaciones Para que sea posible generar una ruta.

En caso de que esto sea correcto, se genera un Nodo auxiliar igual a la primera estación, Y se genera un nuevo nodo de tipo ruta al cual le asignamos un id de ruta De manera auto incremental con respecto a las rutas ya existentes

Ilustración 52.IdRuta

```
136 //Generamos ID's de rutas seriadados
137 if(primRuta==NULL){
138     nuevo->id_ruta = 1;
139 }else{
140     nuevo->id_ruta = ultRuta->id_ruta+1;
141 }
```

Posteriormente se solicita la hora de inicio de circulación de la ruta y la hora de fin de circulación. Se le Muestra el administrador las estaciones existentes y se les solicita cuál es la primera y última estación que recorrerá la ruta.

Ilustración 53.LlenadoDatos

```

155 imprimirEstaciones(35, 10,0);
156 do{
157     gotoxy(35, 8);
158     cout<<"¿Cual es la primera estación de la ruta?: ";
159     //Pedimos una estacion que exista
160     opc = validaEntero();
161 }while(opc < primEstacion->id_lugar || opc > ultEstacion->id_lugar);
162
163 while(opc!=aux->id_lugar){ //Avanzamos con aux a la que sera la primera estacion y la asignamos
164     aux=aux->sig;
165 }
166 nuevo->primero = aux;
167
168
169 aux = primEstacion; //Reiniciamos aux
170 do{
171     gotoxy(35, 9);
172     cout<<"¿Cual es la ultima estación de la ruta?: ";
173     //Pedimos una estacion que exista
174     opc = validaEntero();
175 }while(opc < primEstacion->id_lugar || opc > ultEstacion->id_lugar);
176
177 while(opc!=aux->id_lugar){ //Avanzamos con aux a la que sera la ultima estacion y la asignamos
178     aux=aux->sig;
179 }
180 nuevo->ultimo = aux;

```

Las listas componen una lista simple de listas, por lo que al finalizar el registro la se enlaza al final de la lista de rutas

Ilustración 54.ListaSimpleRutas

```

182 //Las rutas son una lista de listas simple
183 if(primRuta == NULL){
184     primRuta = nuevo;
185     ultRuta = nuevo;
186 }else{
187     ultRuta->sig = nuevo;
188     ultRuta = nuevo;
189 }

```

REGISTRARRUTASUSUARIO

Esta es una Función diferente a registrar rutas, pues en esta se trabaja con nodos de tipo rutasUsuario; Dentro de esta función el usuario genera la lista que almacena las rutas que tomará él personalmente

Se le muestra la lista de rutas existentes al usuario, Y se le pide que elija una ruta a registrar, en caso de que ya haya registrado esa ruta para su uso se le notificará y se le pedirá otra.

Ilustración 55.RegistrarRutasUsuario

```

211      cout<<"¿Cual ruta quieres registrar? ";
212      cout<<"0.-Regresar";
213
214      do{
215          gotoxy(15, 16);
216          cout<<"N. Ruta: ";
217          opc = validaEntero();
218          if(opc==0){
219              return 0;
220          }
221      }while(opc>ultRuta->id_ruta || opc<1);
222      while(aux!=NULL){
223          if(aux->id_ruta==opc){
224              gotoxy(15, 17);
225              cout<<"La ruta ya esta registrada";
226              Sleep(3000);
227              return 0;
228          }
229          aux=aux->sig;
230      }

```

Se le pide al usuario el día en que la tomará (un valor entre 1 y 30) Y la hora a la que la va a tomar (Un valor entre 1 y 24).

Las rutas del usuario es una lista doblemente enlazada simple, Por lo que al finalizar el registro Se enlaza está a la lista de las rutas del usuario.

Ilustración 56.ListaSimpleRutasUsuario

```

250      if(usuarioActivo->primero==NULL){
251          usuarioActivo->primero=nuevaRuta;
252          usuarioActivo->ultimo=nuevaRuta;
253      }else{
254          nuevaRuta->prev=usuarioActivo->ultimo;
255          usuarioActivo->ultimo->sig = nuevaRuta;
256          usuarioActivo->ultimo = nuevaRuta;
257      }

```


IMPRESIONES

Dentro del archivo impresiones podemos encontrar todas las Funciones que se encargan de imprimir o mostrar en la consola los datos almacenados en las listas, en total son 9 Funciones que son muy parecidas, las diferencias entre ellas son que algunas imprimen un solo nodo todos, dependiendo de las necesidades; Además de que los datos que imprimen son dependiendo de los datos que almacenan sus nodos

Ilustración 57.Impresiones

```
❖ historial (int x, int y, Usuario *usuarioActual) : void
❖ imprimirAdmin (int x, int y, int memoria) : void
❖ imprimirDatosUsuario (Usuario *aux) : void
❖ imprimirEstaciones (int x, int y, int memoria) : void
❖ imprimirPasajeros (int x, int y, int memoria) : void
❖ imprimirRutaPorId (int idRuta) : void
❖ imprimirRutas (int x, int y, int memoria) : void
❖ imprimirRutasUsuario (int x, int y, Usuario *usuarioActual) : int
❖ imprimirUsuarios (int x, int y) : void
```

IMPRIMIRUSUARIOS

Esta función se encarga de imprimir todos los usuarios de la lista usuarios, independientemente de si el usuario es un administrador o un pasajero.

Se compone de dos partes, en la primera imprime los encabezados de la tabla

Ilustración 58.ImpresionUsuarios

```
2      Usuario *aux = primUser;
3      gotoxy(x, y);
4      SetConsoleTextAttribute(hCon, 2);
5      cout<<"ID "<<setw(15);
6      SetConsoleTextAttribute(hCon, 3);
7      cout<<"Nombre "<<setw(15);
8      SetConsoleTextAttribute(hCon, 4);
9      cout<<"Apellido "<<setw(15);
10     SetConsoleTextAttribute(hCon, 5);
11     cout<<"Sexo "<<setw(15);
12     SetConsoleTextAttribute(hCon, 6);
13     cout<<"TipoUsuario "<<setw(15);
14     SetConsoleTextAttribute(hCon, 7);
15     cout<<"Estado "<<setw(15);
16     SetConsoleTextAttribute(hCon, 8);
17     cout<<"Contraseña"<<endl;
```

En la segunda parte contiene un ciclo while que recorre todos los usuarios hasta el final y al mismo tiempo va imprimiendo los datos de los usuarios, entre los datos que imprime se encuentran:

- Id usuario
- nombre
- Apellido
- Sexo
- Tipo de usuario
- Estado
- Contraseña

Ilustración 59. ImpresionUsuarios2

```
18 while(aux!=NULL){
19     y+=1;
20     gotoxy(x, y);
21     SetConsoleTextAttribute(hCon, 2);
22     cout<<aux->id_usuario<<setw(15);
23     SetConsoleTextAttribute(hCon, 3);
24     cout<<aux->nombre<<setw(15);
25     SetConsoleTextAttribute(hCon, 4);
26     cout<<aux->apellido<<setw(15);
27     SetConsoleTextAttribute(hCon, 5);
28     if(aux->sexo == hombre){
29         cout<<"Hombre"<<setw(15);
30     }else{
31         cout<<"Mujer"<<setw(15);
32     }
33     SetConsoleTextAttribute(hCon, 6);
34     if(aux->TipoUsuario == pasajero){
35         cout<<"Pasajero"<<setw(15);
36     }else{
37         cout<<"Administrador"<<setw(15);
38     }
39     SetConsoleTextAttribute(hCon, 7);
40     if(aux->estado == activo){
41         cout<<"Activo"<<setw(15);
42     }else{
43         cout<<"Inactivo"<<setw(15);
44     }
45     SetConsoleTextAttribute(hCon, 8);
46     cout<<aux->password<<endl;
47     aux = aux->sig;
48 }
49 }
```


IMPRIMIRPASAJEROS E IMPRIMIRADMIN

Estas funciones son casi iguales que la de "imprimirUsuarios", con la diferencia de que reciben un parámetro adicional el cual es una variable de tipo entero llamado memoria.

Esta función solo va a imprimir a los usuarios que tengan el tipo de usuario "pasajero" o el valor de 0, o en caso de ser administrador que tengan tipo de usuario "administrador" o el valor de 1.

En caso de que el parámetro "memoria" que se le paso a la función sea igual a 1, se imprimirán las direcciones de memoria de los nodos


Ilustración 60.ImpresionEstados

```
65 
66
67
68
69
70
71
72
73
74
75
```

```
if(memoria == 1){
    SetConsoleTextAttribute(hCon, 7);
    cout<<"Estado "<<setw(15);
    SetConsoleTextAttribute(hCon, 8);
    cout<<"Actual"<<setw(15);
    SetConsoleTextAttribute(hCon, 9);
    cout<<"Siguiente"<<endl;
}else{
    SetConsoleTextAttribute(hCon, 7);
    cout<<"Estado"<<endl;
}
```

...

Ilustración 61.ImpresionMemoria

```
114 
115
116
117
118
119
```

```
if(memoria == 1){
    SetConsoleTextAttribute(hCon, 8);
    cout<<aux<<setw(15);
    SetConsoleTextAttribute(hCon, 9);
    cout<<aux->sig<<endl;
}
```

IMPRIMIRDATOSUSUARIO E IMPRIMIRRUTAPORID

Estas funciones reciben un nodo de tipo usuario o el id de una ruta e imprime los datos contenidos en él, estas funciones no imprimen los encabezados de cada columna, es por esto que puede ser usada con diferentes propósitos y situaciones.

Ilustración 62. ImprimirDatosUsuario

```

198 //Esta funcion imprime los datos del usuario que se le envie
199 void imprimirDatosUsuario(Usuario *aux){
200     SetConsoleTextAttribute(hCon, 2);
201     cout<<aux->id_usuario<<"\t";
202     SetConsoleTextAttribute(hCon, 3);
203     cout<<aux->nombre<<"\t";
204     SetConsoleTextAttribute(hCon, 4);
205     cout<<aux->apellido<<"\t";
206     SetConsoleTextAttribute(hCon, 2);
207     if(aux->sexo == hombre){
208         cout<<"Hombre";
209     }else{
210         cout<<"Mujer";
211     }
212     SetConsoleTextAttribute(hCon, 3);
213     if(aux->TipoUsuario == pasajero){
214         cout<<"Pasajero";
215     }else{
216         cout<<"Administrador";
217     }
218     SetConsoleTextAttribute(hCon, 4);
219     if(aux->estado == activo){
220         cout<<"Activo\t";
221     }else{
222         cout<<"Inactivo\t";
223     }
224 }

```

IMPRIMIRESTACIONES E IMPRIMIRRUTAS

Estas funciones son casi iguales a la función imprimirPasajero e imprimirAdministrador, solamente que aquí van a imprimir todos los datos de todas las estaciones o rutas dependiendo del caso en caso de haber recibido como parámetro un 1 en memoria, se imprimirá la dirección de cada nodo y la dirección de su nodo siguiente, en caso de recibir un 0 en memoria no van a imprimir el valor de la dirección de memoria de ningún nodo.

En caso de no existir lo notificará al usuario.

Ilustración 63. ImprimirRutas

```

230     cout<<"ID "<<setw(10);
231     SetConsoleTextAttribute(hCon, 3);
232     cout<<"Nombre "<<setw(15);
233     SetConsoleTextAttribute(hCon, 4);
234     cout<<"Tiempo "<<setw(15);
235     if(memoria == 1){
236         SetConsoleTextAttribute(hCon, 5);
237         cout<<"Estado "<<setw(15);
238         SetConsoleTextAttribute(hCon, 6);
239         cout<<"Actual"<<setw(15);
240         SetConsoleTextAttribute(hCon, 7);
241         cout<<"Siguiente"<<endl;
242     }else{
243         SetConsoleTextAttribute(hCon, 5);
244         cout<<"Estado"<<endl;
245     }

```

Ilustración 64. ImprimirRutas

```

285 //Esta funcion imprime Todas las rutas
286 void imprimirRutas(int x, int y, int memoria){
287     Ruta *aux = primRuta;
288     gotoxy(x, y);
289     SetConsoleTextAttribute(hCon, 2);
290     cout<<"ID"<<setw(10);
291     SetConsoleTextAttribute(hCon, 3);
292     cout<<"Estado "<<setw(15);
293     SetConsoleTextAttribute(hCon, 4);
294     cout<<"H.Ini"<<setw(15);
295     SetConsoleTextAttribute(hCon, 5);
296     cout<<"H.Fin"<<setw(15);
297     SetConsoleTextAttribute(hCon, 6);
298     cout<<"Prim.E."<<setw(15);
299     if(memoria == 1){
300         SetConsoleTextAttribute(hCon, 7);
301         cout<<"Ult.E. "<<setw(15);
302         SetConsoleTextAttribute(hCon, 8);
303         cout<<"Actual"<<setw(15);
304         SetConsoleTextAttribute(hCon, 9);
305         cout<<"Siguiente"<<endl;
306     }else{
307         SetConsoleTextAttribute(hCon, 7);
308         cout<<"Ult.E."<<endl;
309     }

```

IMPRIMIR RUTAS USUARIO

Esta función recibe como parámetro un nodo de tipo apuntador usuario, En este momento el no de usuario ya debe contener las listas de rutas usuario qué se leyeron de un archivo previamente en buscar la función, Aquí entonces se avanza hasta usuario en la lista principal, se imprimen las cabeceras de la tabla y por medio de otro ciclo while interno Recorre la listade rutas del usuario y las envía a imprimirRutaPorId() De esta manera imprimimos sólo los datos de la ruta enviada.

Ilustración 65.ImprimirRutasUsuario

```

391 //Recorremos todos los nodos mediante un segundo auxiliar
392 while(auxRuta!=NULL){
393     y+=1;
394     gotoxy(x, y);
395     imprimirRutaPorId(auxRuta->id_ruta);
396     cout<<auxRuta->dia<<"\t"<<auxRuta->hora;
397     auxRuta=auxRuta->sig;
398 }

```

HISTORIAL

Esta función es muy parecida a la función imprimirRutasUsuario, La lógica de programación es la misma solo que aquí el nodo de tipo apuntador usuario llega con su apuntador de recorridoUsuario Ya con la lista del usuario que se envió, e imprime los datos del recorrido

Ilustración 66.ImprimirHistorial

```

406 void historial(int x, int y, Usuario *usuarioActual){
407     recorridoUsuario *aux = usuarioActual->primRecorrido;
408     gotoxy(x, y);
409     SetConsoleTextAttribute(hCon, 2);
410     cout<<"ID"<<setw(10);
411     SetConsoleTextAttribute(hCon, 3);
412     cout<<"Id Ruta "<<setw(15);
413     SetConsoleTextAttribute(hCon, 4);
414     cout<<"Estado "<<setw(15);
415     SetConsoleTextAttribute(hCon, 5);
416     cout<<"Estacion Subida"<<setw(20);
417     SetConsoleTextAttribute(hCon, 6);
418     cout<<"Estacion Bajada."<<setw(20);
419     SetConsoleTextAttribute(hCon, 7);
420     cout<<"Hora subida"<<setw(15);
421     SetConsoleTextAttribute(hCon, 8);
422     cout<<"Hora bajada"<<endl;

```

MODIFICACIONES

Las funciones de modificaciones se encuentran en la librería modificaciones.h. Las funciones que trabajamos aquí son 4:

- modificarEstacion
- modificarRuta
- modificarRutaUsuario
- modificarUsuario

MODIFICARESTACION:

Ilustración 67.ModificarEstacion

```

16
17 do{
18     gotoxy(35, 9);
19     opc = validaEntero();
20     if(opc==0){
21         return 0;
22     }
23 }while(opc>2 || opc<1);
24
25 imprimirEstaciones(35,10, 0);
26 gotoxy(35, 20);
27 cout<<"Cual numero de estacion que quieres modificar? ";
28 numEstacion=validaEntero();
29 system("cls");
30 caja();
31 SetConsoleTextAttribute(hCon, 14);
32 do{
33     if(aux->id_lugar == numEstacion){
34         switch(opc){
35             case 1:
36                 gotoxy(35, 5);
37                 cout<<"Cual es el nuevo nombre de la estacion? ";
38                 cin>>nombre;
39                 aux->nombre = nombre;
40                 gotoxy(35, 7);
41                 cout<<"Nombre de la estacion actualizado exitosamente!";
42                 Sleep(3000);
43                 return 0;

```

A esta función solamente puede acceder el administrador. Dentro de esta estación generamos un auxiliar el cual inicia en la primera estación, preguntamos si quiere modificar el nombre o el estado. Después imprimimos todas las estaciones y preguntamos cuál estación quiere modificar.

con el do-while recorremos todas las estaciones para buscar la que quiere modificar el usuario, dependiendo de si eligió modificar el nombre o el estado, preguntamos el nuevo nombre o el nuevo estado.

MODIFICARRUTA:

Ilustración 68.ModificarRuta

```

do{
    gotoxy(35, 15);
    cout<<"Que ruta modificar?";
    opcRuta=validaEntero();
    if(opcRuta<primRuta->id_ruta || opcRuta>ultRuta->id_ruta) cout<<"Ruta no existente"<<endl;
}while(opcRuta<primRuta->id_ruta || opcRuta>ultRuta->id_ruta);
system("cls");
caja();
SetConsoleTextAttribute(hCon, 14);
gotoxy(35, 5);
cout<<"Que quieres modificar?";
gotoxy(35, 6);
cout<<"1.- Estado";
gotoxy(35, 7);
cout<<"2.- Hora Inicio";
gotoxy(35, 8);
cout<<"3.- Hora Fin";
gotoxy(35, 9);
cout<<"4.- Primera Estacion";
gotoxy(35, 10);
cout<<"5.- Ultima Estacion";
do{
    gotoxy(35, 11);
    opcModificar=validaEntero();
}while(opcModificar<1 || opcModificar > 5);
for(int i=1;i<opcRuta;i++){
    modifica = modifica->sig;
}
switch(opcModificar){
    case 1:
        do{
            gotoxy(35, 12);
            cout<<"Ingresa el estado. 0 Inactivo, 1 Activo ";
            valor=validaEntero();

```

A esta función solamente puede entrar el administrador. Dentro de esta función generamos un auxiliar que lo ubicamos en la primera ruta. Imprimimos las rutas y preguntamos qué ruta se quiere modificar. una vez elija una ruta válida le preguntamos si quiere modificar el estado, la hora de inicio, la hora de fin, la primera estación o la última estación y con un switch nos vamos a la opción que vamos a modificar.

MODIFICARRUTAUSUARIO:

A este menú solamente puede acceder el usuario que inicie sesión. generamos un auxiliar que empieza en la primera ruta del usuario.

Imprimimos las rutas del usuario y preguntamos cual quiere modificar, mediante un while comprobamos que exista la ruta.

Ilustración 69.RutaNoEncontrada

```

while(opcRuta!=modifica->id_ruta && modifica!=NULL){ //Con este while recorremos las rutas que tiene el usuario,
//Mientras la id de la ruta modificar sea diferente a la que pidio el usuario y sea diferente de null
    modifica=modifica->sig; //Cambia a la siguiente ruta que tenga guardada el usuario
    if(modifica==NULL){ //Si la ruta es NULL deja de buscar
        gotoxy(35, 17);
        cout<<"Ruta no encontrada"<<endl;
        break;
    }
}

```


En caso de que no exista volvemos a preguntar otra ruta.

Una vez ponga una ruta válida le preguntamos qué quiere modificar: el día de abordaje o la hora de abordaje

Ilustración 70. LlenadoHorario

```
switch(opc2){
    case 1:
        do{
            gotoxy(35, 9);
            cout<<"¿Que dia la vas a tomar? (1-30) ";
            valor=validaEntero();
        }while(valor>30 || valor<1);
        modifica->dia=valor;
        break;
    case 2:
        do{
            gotoxy(35, 9);
            cout<<"¿A que hora lo vas a tomar? (1-24) ";
            valor=validaEntero();
        }while(valor>24 || valor<1);
        modifica->hora=valor;
        break;
}
```

Leemos el nuevo valor y lo guardamos en la ruta a modificar.

modificarUsuario:

A esta función solamente puede acceder el administrador. Primero imprimimos los usuarios y preguntamos cual quiere modificar, verificando que el usuario exista.

Ilustración 71. ModificarUsuario

```
do{
    gotoxy(35, 20);
    cout<<"Cual ID del usuario que quieres modificar? ";
    numUsuario=validaEntero();
}while(numUsuario>ultUser->id_usuario || numUsuario<primUser->id_usuario);
```

Después buscamos al usuario a modificar una vez encontrado preguntamos si quiere modificar el nombre, el apellido, el sexo o su contraseña

Ilustración 72.ModificarNombre

```
case 1:
    //Modificar nombre
    gotoxy(35, 5);
    cout<<"Cual es el nuevo nombre?";
    gotoxy(35, 6);
    cout<<user->nombre<<" => ";
    cin>>user->nombre;
    gotoxy(35, 7);
    cout<<"NOMBRE ACTUALIZADO CORRECTAMENTE!";
    break;
case 2:
    //Modificar nombre
    gotoxy(35, 5);
    cout<<"Cual es el nuevo apellido?";
    gotoxy(35, 6);
    cout<<user->apellido<<" => ";
    cin>>user->apellido;
    gotoxy(35, 7);
    cout<<"APELLIDO ACTUALIZADO CORRECTAMENTE!";
    break;
```

Preguntamos el nuevo valor y lo aplicamos a la estructura. En caso de querer modificar la contraseña la preguntaremos 2 veces, una para preguntar la nueva y la segunda para verificar que es correcta.

Ilustración 73.ModificarPassword

```
do{
    bandera = true;
    gotoxy(35, 5);
    cout<<"Ingresa nuevo password: ";
    cin>>password1;
    gotoxy(35, 6);
    cout<<"Confirma tu password: ";
    cin>>password2;

    if(password1!=password2){
        //Hacemos una pequeña confirmacion de password
        gotoxy(35, 7);
        cout<<"Los password no coinciden!, intenta nuevamente";
        bandera = false;
    }
}while(bandera == false);
```

VALIDACIONES

Dentro de la librería validaciones.h se encuentra la validacion de números enteros, está la usamos para verificar que el usuario ingrese un número en donde lo necesitamos.

la función llamada validaEntero

Ilustración 74. ValidarEnteros

```
int validaEntero(){  
    int entero = 0;  
    int bandera = 0;  
  
    do{  
        bandera = scanf("%d", &entero);  
        fflush(stdin);  
    }while(bandera!=1);  
    fflush(stdin);  
    return entero;  
}
```

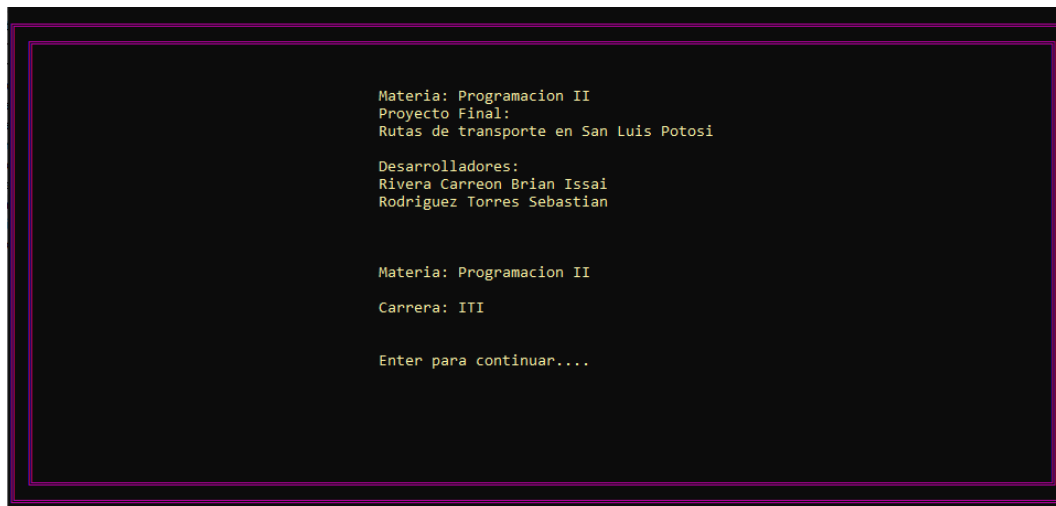
El do-while se ejecutará mientras el scanf devuelve un error causado por el tipo de dato que se ingresa. una vez ingrese un dato válido lo regresara mediante el return

PRESENTACIÓN

Dentro de la librería presentación.h nos encontramos la función HANDLE hCon la cual usamos para poder cambiar el color de texto, la función gotoxy, la cual nos ayuda a mover la posición de impresión a donde le indiquemos.

la función caja imprime la caja color morado impresa alrededor de la consola:

Ilustración 75. Presentacion



```
Materia: Programacion II  
Proyecto Final:  
Rutas de transporte en San Luis Potosi  
  
Desarrolladores:  
Rivera Carreon Brian Issai  
Rodriguez Torres Sebastian  
  
Materia: Programacion II  
Carrera: ITI  
  
Enter para continuar....
```

la función usa gotoxy(), y formatos de printf("%c") dándole el valor del ascii correspondiente a las líneas y los bordes.

Función Presentación:

Ilustración 76.ImpresionPresentación

```
void presentacion(){
    caja();
    gotoxy(42, 5);
    SetConsoleTextAttribute(hCon, 14);
    cout<<"Materia: Programacion II";
    gotoxy(42, 6);
    cout<<"Proyecto Final: ";
    gotoxy(42, 7);
    cout<<"Rutas de transporte en San Luis Potosi";
    gotoxy(42, 9);
    cout<<"Desarrolladores: ";
    gotoxy(42, 10);
    cout<<"Rivera Carreon Brian Issai";
    gotoxy(42, 11);
    cout<<"Rodriguez Torres Sebastian";
    gotoxy(42, 15);
    cout<<"Materia: Programacion II";
    gotoxy(42, 17);
    cout<<"Carrera: ITI";
    gotoxy(42, 20);
    cout<<"Enter para continuar....";
}
```

La función presentación manda imprimir la caja e imprime los datos del proyecto, la materia, el título y los desarrolladores del mismo.

CONCLUSIÓN

Este manual de programador te puede llegar a ayudar para marcar alguna modificación dentro del proyecto o para darle mantenimiento y liberar algo de memoria en los archivos que se generen. Por lo que consideramos que debería ser importante leer el manual de programador y el manual del usuario para conocer al 100% la forma de trabajo del programa, y en caso de querer meter alguna modificación, saber en qué puede afectar modificar algo del código y de qué manera puedes generar alguna nueva función al programa de la forma más sencilla y eficiente.