# Manual Técnico

Fase 1

Laboratorio de Estructuras de Datos Sección C Derek Francisco Orellana Ibáñez 202001151

## Main.cpp

Declaración de variables globales

```
//! Declaración de variables globales
ListaUsuarios::ListaEnlazadaSimple lista_usuarios;
ListaPublicaciones::ListaEnlazadaDoble lista_publicaciones;
MatrizRelacion::MatrizDispersa matriz_relacion;
ListaUsuarios::Usuario *usuario_logeado = nullptr;
```

Se declaran las listas, matrices y usuarios globales las cuales mantendrán los datos accesibles durante toda la ejecución del programa

main()

```
// TODO: MAIN
> int main()...
```

En el método main(), se ejecutará la configuración de la consola, como la creación del usuario administrador agregándolo a la lista de usuarios globales, así como también ejecutando el menú principal para dar inicio al programa.

### Menus.cpp

Menú principal y submenús

```
12 // TODO: Menú Principal
13 > void menuPrincipal() ···
```

Método encargado de mostrar el menú principal.

```
//? Submenús
void menuIniciarSesion()…
```

Método encargado de mostrar el menú para pedir las claves de acceso al usuario, solicitando correo y contraseña. Verificando la validez de los datos.

```
> void menuRegistrarse()…
```

Método encargado de mostrar el menú para pedir los datos del usuario para registrarlo en el programa. Verificando que el correo no exista.

```
> void menuMostrarInformacion()…
```

Método encargado de mostrar la información del estudiante que realizo el proyecto.

#### Menú Administrador

```
// TODO: Menú Administrador
> void menuAdministrador() …
```

Método encargado de mostrarle al usuario con rol de administrador, el menú con las opciones de administración.

```
227 > void adminGestionarUsuarios()...
```

Método encargado de mostrar el menú de gestion de usuarios, el cual lista o elimina los usuarios almacenados.

```
282 > void adminGestionarReportes() ···
```

Método encargado de mostrar el menú de gestion de reportes, el cual genera las graficas y reportes de usuarios, relaciones, publicaciones, entre otros.

```
355 > void adminGestionarTop() ···
```

Método encargado de mostrar el menú de gestion de un top 5 de usuarios.

Menú de usuario

```
406 // TODO: Menú Usuario
407 > void menuUsuario()…
```

Método encargado de mostrar el menú del usuario con el rol de usuario, mostrando las opciones de un usuario normal.

```
465 > void userGestionarPerfil()…
```

Método encargado de mostrar la información del usuario

```
• 517 > void userGestionarSolicitudes()…
```

Método encargado de mostrar la gestion de solicitudes de amistad, aceptarlas, rechazarlas, ver la lista de solicitudes.

```
571 > void userGestionarPublicaciones()...
```

Método encargado de gestionar las publicaciones del usuario, ver el publicaciones del usuario y amigos, crear y eliminar publicaciones.

```
620 > void userGestionarReportes() ···
```

Método encargado de gestionar los reportes del usuario, graficando las listas simples, dobles, circulares y matrices dispersas.

```
733 > void menuMostrarPublicaciones() ···
```

Método encargado de mostrar el menu de las publicaciones realizadas por el usuario.

```
778 > void menuCrearPublicacion() ...
```

Método encargado de pedir el contenido para crear una publicación.

```
800 > void menuEliminarPublicacion() ···
```

Método encargado de eliminar una publicación.

```
862 > void menuEnviarSolicitud() ···
```

Metodo encargado de enviar solicitudes a los usuarios.

```
904 > void menuSolicitudesRecibidas()…
```

Método encargado de mostrar el menú de las solicitudes recibidas.

#### Funciones.h

void RegistrarUsuario(string names, string lastnames, string birthday, string mail, string password)...

Método que pide como parámetros 5 datos de tipo string, encargado de crear un objeto usuario y agregarlo a la lista de usuarios globales.

```
35 > void cargarUsuarios()...
```

Método encargado de realizar una carga masiva de usuarios, solicitando el directorio del archivo JSON con los datos de los usuarios a ingresar.

```
104 > void cargarRelaciones()…
```

Método encargado de realizar una carga masiva de las relaciones de los usuarios cargados anteriormente, solicitando el directorio del archivo JSON con los datos de la relación entre usuarios.

```
185 > void cargarPublicaciones()…
```

Método encargado de realizar una carga masiva de publicaciones de usuarios cargados anteriormente, solicitando el directorio del archivo JSON con los datos de las publicaciones.

```
void crearPublicacion(ListaUsuarios::Usuario usuario, string contenido, string fecha, string hora)...
```

Método que pide como parámetros 4 parámetros encargados de la creación de una publicación del usuario que tenga la sesión activa.

```
262 > string obtenerFecha() ···
```

Método encargado de obtener la fecha actual en formato de dd-mm-aaaa, utilizada en la creación de publicaciones.

```
262 > string obtenerFecha() ···
```

Método encargado de obtener la hora actual en formato de 24h hh:mm:ss, utilizada en la creación de publicaciones.

```
278 > ListaPublicacionesFeed::ListaCircularDoble cargarPublicacionesFeed() ···
```

Método encargado de devolver una lista circular doble con las publicaciones del usuario y de sus amigos.

```
309 > bool verificarSesion() ···
```

Método encargado de verificar la sesión del usuario.

```
318 > void enviarSolicitud(ListaUsuarios::Usuario *usuario) ⋯
```

Método que pide como parámetro el usuario al cual se enviara una solicitud de amistad, encargándose de almacenar el correo del usuario en la lista simple del usuario logeado (emisor), y el correo del usuario logeado (emisor) en el usuario receptor.

```
345 > void aceptarSolicitud(ListaUsuarios::Usuario *receptor) ···
```

Método que pide como parámetro el usuario receptor, encargado de eliminar los usuarios de la lista simple del emisor y el usuario de la pila del receptor, creando una relación en la matriz dispersa.

```
368 > void rechazarSolicitud(ListaUsuarios::Usuario *receptor)...
```

Método que pide como parámetro el usuario receptor, encargado de eliminar los usuarios de la lista simple del emisor y el usuario de la pila del receptor.

```
386 > void eliminarMiPerfil()...
```

Método encargado de eliminar las publicaciones, relaciones con amigos y el usuario como tal, de la lista de usuarios.

```
399 > void eliminarUsuarios()…
```

Método encargado de eliminar las publicaciones, relaciones con amigos y usuarios como tal de la lista de usuarios.

```
423 > void mostrarAmigos()…
```

Método encargado de obtener la lista de amigos y mostrarlos del usuario logeado.

```
429 > void graficarMiFeed()…
```

Metodo encargado de graficar la lista circular doble de las publicaciones del usuario logeado y sus amigos.

```
• 434 > void graficarMiRelacion()…
```

Método encargado de graficar la matriz dispersa de las relaciones del usuario logeado.

## Globales.h

```
extern ListaUsuarios::Usuario *usuario_logeado; You, hace 4 días extern ListaUsuarios::ListaEnlazadaSimple lista_usuarios;
extern ListaPublicaciones::ListaEnlazadaDoble lista_publicaciones;
extern MatrizRelacion::MatrizDispersa matriz_relacion;
```

Variables globales exportadas desde un nuevo archivo para utilizarlos en el resto de los archivos.

#### ListaPublicaciones.h

Struct global

13 > struct Publicacion...

Struct con los atributos y constructor para la creación de publicaciones.

```
22 > Publicacion(int id, string ca, const string &c, const string &f, const string &h)...
```

Constructor de la clase publicación

```
25 // Métodos
26 > void mostrarPublicacion() const...
```

Método encargado de imprimir la información de la publicación

```
Class ListaEnlazadaDoble
```

Private

	rou, nace raids pradutor (rou)
40 >	struct NodoPublicacion ···
49	
50	NodoPublicacion *cabeza;
51	NodoPublicacion *cola;

Struct con los atributos de tipo publicación, nodoPublicacion y constructor. Definiendo así la cabeza y la cola (inicio y fin) de la lista enlazada doble.

```
Public

54  // Constructor

ListaEnlazadaDoble(): cabeza(nullptr), cola(nullptr) {}
```

Constructo de la lista enlazada doble para la creación de las listas de publicaciones.

```
56 // Destructor
57 > ~ListaEnlazadaDoble()...
```

Destructor encargado de eliminar el contenido de la lista para la liberación de memoria.

```
or void agregarPublicacion(const Publicacion &p) ...
```

Método que pide como parámetro una clase publicación, encargada de agregarla a la lista enlazada doble.

```
82 > void eliminarPublicaciones(ListaUsuarios::Usuario *usuario_logeado) ···
```

Método que pide como parámetro un usuario (usuario logeado) que se encargara de eliminar todas las publicaciones del usuario.

```
void eliminarPublicaciones(string correo) ···
```

Método que pide como parámetro el correo de un usuario, encargado de eliminar las publicaciones del correo solicitado.

```
void eliminarPublicacion(int id) …
```

Método que pide como parámetro el id, encargado de eliminar la publicación con un id especifico.

```
Publicacion buscarPublicacion(const int &id) ···
```

Método que pide como parámetro un id, retornando una publicación del id especifico.

```
194 > Publicacion obtenerPublicacion(const int &pos) ···
```

Método que pide como parámetro la posición en la lista, retornando la publicación en la posición especifica.

```
209 > int obtenerUltimoId() const...
```

Método que obtiene el ultimo id de la lista de publicaciones para generar un nuevo id al crear una publicación.

```
220 > int getLength() const...
```

Método que obtiene la cantidad de elementos en la lista.

```
// Métodos para graficar en Graphviz
void graficarPublicaciones(string nombre, string formato) ···
```

Método que pide como parámetros el nombre y el formato para la graficación de la lista enlazada doble utilizando graphviz.

```
void top5UsuariosConMasPublicaciones()…
```

Método encargado de obtener 5 usuarios con más publicaciones creadas.

#### ListaPublicacionesFeed.h

Struct global

14 > struct NodoFeed ...

Struct con parámetros y constructor para el manejo de la lista circular doble

Class ListaCircular Doble

Private

```
class ListaCircularDoble
{
26    private:
27    NodoFeed *cabeza;
28    NodoFeed *cola;
29    NodoFeed *actual;
30
```

Nodos de cabeza, cola y actual para el manejo de la lista circular doble.

```
Public

32  // Constructor

33  ListaCircularDoble(): cabeza(nullptr), cola(nullptr), actual(nullptr) {}
```

Constructor de la lista circular doble

```
35      // Destructor
36 > ~ListaCircularDoble() ···
```

Destructor encargado de eliminar el contenido de la lista para la liberación de memoria.

```
ooid agregarPublicacion(const ListaPublicaciones::Publicacion &p) ···
```

Método que pide como parámetro una publicación, encargado de agregarla a la lista circular doble.

```
89 > void eliminarPublicacion(int id) ···
```

Método que pide como parámetro un id, encargado de eliminar la publicación de la lista circular doble.

```
136 > void mostrarPublicacionActual() ···
```

Método encargado de imprimir la publicación actual de la lista circular doble

```
int obtenerIdActual()…
```

Método encargado de obtener el id de la publicación actual.

163 > void avanzarPublicacion() ···

Método encargado de avanzar a la siguiente publicación de la lista circular doble.

169 > void retrocederPublicacion() ···

Método encargado de retroceder a la publicación anterior de la lista circular doble.

175 > void vaciarLista() ···

Método encargado de vaciar la lista circular doble para la liberación de memoria de forma manual.

void graficarPublicacionesFeed(string nombre, string formato)...

Método que pide como parámetro el nombre y el formato para la graficación en Graphviz la lista circular doble del usuario y sus amigos.

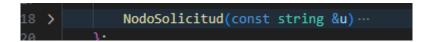
#### ListaSolicitudes.h

Struct global

You, hace 3 dias | 1 author (You)

12 struct NodoSolicitud

Struct con parámetros de correo y nodosSolicitud



Constructor que pide como parámetro el correo del usuario.

```
Class ListaCircularDoble
```

Private

Parámetros para la posición de los nodos, cabeza, actual y cola.

```
Public

29  // Constructor
30  ListaCircularDoble(): cabeza(nullptr), cola(nullptr), actual(nullptr) {}
```

Constructor de la lista circular doble

```
32  // Destructor
33 > ~ListaCircularDoble() ···
```

Destructor encargado de eliminar el contenido de la lista para la liberación de memoria.

```
87 > void mostrarSolicitudActual() ···
```

Método encargado de mostrar el correo de la solicitud actual.

```
97 > string obtenerSolicitudActual() ···
```

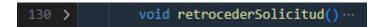
Método que retorna el correo de la solicitud actual.

```
105 > string obtenerSolicitud(int pos) ...
```

Método que retorna el correo de la solicitud en una posición en específico.

```
124 > void avanzarSolicitud()…
```

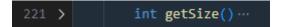
Método para avanzar a la siguiente solicitud.



Método encargado de retroceder a la solicitud anterior.

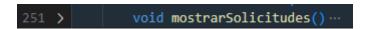
Método encargado de vaciar la lista de forma manual.

Método que pide como parámetro un correo, encargándose de eliminarlo de la lista.



Método que retorna la cantidad de datos de la lista.

Método que retorna un booleano para la verificación de una solicitud, pidiendo como parámetro el correo.



Método encargado de imprimir la lista de solicitudes.

#### ListaUsuarios.h

Struct Global



Struct encargada de crear objetos de tipo usuario.

27	
28	// Constructor
29	Usuario(const int &id,
30	const string &nombres,
31	const string &apellidos,
32	const string &fechaNacimiento,
33	const string &correo,
34	const string &contrasena,
35 >	const string &rol = "user") ···

Constructor del struct que pide como parametros la información del usuario, adicional de un rol, el cual por defecto será user.

```
45 > void mostrarPerfil() const...
```

Método que imprime la información del usuario.

```
54 > bool solicitudRecibidaExiste(const string &correo) ...
```

Método que retorna un booleano y pide como parámetro un correo para verificar que la solicitud recibida exista en la pila del usuario.

```
orreo) ···
```

Método que pide como parámetro un correo que se encarga de agregar el correo a la pila de solicitudes recibidas.

```
79 > void mostrarSolicitudesRecibidas()…
```

Método encargado de imprimir la lista de solicitudes recibidas.

```
93 > void mostrarSolicitudRecibida() ···
```

Método que se encarga de imprimir la primera solicitud recibida.

```
104 > bool solicitudEnviadaExiste(const string &correo) ···
```

Método que retorna un booleano y pide como parámetro un correo para verificar que la solicitud enviada exista en la lista del usuario.

```
void agregarSolicitudEnviada(const string &correo) ···
```

Método que pide como parámetro un correo que se encarga de agregar el correo a la lista de solicitudes enviadas.

```
120 > void mostrarSolicitudesEnviadas()…
```

Método encargado de imprimir la lista de solicitudes enviadas del usuario.

```
// Métodos para graficar
void graficarSolicitudes(string nombre, string formato) ...
```

Método que pide como parámetro el nombre y formato para generar la gráfica de la lista simple de solicitudes enviadas y la pila de solicitudes recibidas.

```
Class ListaEnlazadaSimple

Private

204 > struct NodoUsuario ...
210
211 NodoUsuario *cabeza;
```

Struct del nodoUsuario y declaración de la cabeza de la lista

```
Public

214 ListaEnlazadaSimple(): cabeza(nullptr) {}
```

Constructor de la lista simple

```
216 > ~ListaEnlazadaSimple() ···
```

Destructor encargado de eliminar el contenido de la lista para la liberación de memoria.

```
void agregarUsuario(const Usuario &u)…
```

Método que pide como parámetro un usuario, encargado de agregar al usuario en la lista simple.

```
245 > void listarUsuarios() const...
```

Método encargado de listar a los usuarios imprimiendo la información de cada usuario.

```
256 > Usuario *buscarUsuario(const int &pos) ...
```

Metodo que retorna un usuario pidiendo como parametro la posición en la lista de usuarios.

```
286 > Usuario *buscarUsuario(const string &correo) ...
```

Metodo que retorna un usuario que pide como parametro el correo de un usuario.

```
272 > Usuario *buscarUsuario(const string &correo, const string &contrasena) ...
```

Metodo que retorna un usuario que pide como parametro el correo y contraseña para verificacion de inicio de sesion.

```
● 300 > int getSize()…
```

Metodo que retorna la cantidad de elementos del la lista.



Metodo que retorna el ultimo id de la lista de usuarios.

```
• 338 > void eliminarUsuarios(const Usuario &logeado) ···
```

Metodo que pide como parametro un usuario y elimina toda la lista de usuarios excepto el usuario logeado.

```
324 > bool existeUsuario(const string &correo) ...
```

Metodo que retorna un boleano que pide como parametro el correo para verificar la existencia de un usuario en la lista.

```
365 > void eliminarUsuario(const Usuario *usuario) ···
```

Metodo que pide como parametro un usuario y lo elimina de la lista de usuarios.

```
void agregarSolicitudRecibida(const string &correoEmisor, const string &correoReceptor) ···
```

Metodo que pide como parametro el correo del emisor y receptor y agrega la solicitud a la lista del emisor y a la pila del emisor.

```
void agregarSolicitudEnviada(const string &correoEmisor, const string &correoReceptor) ...
```

Metodo que pide como parametro el correo del emisor y receptor y agrega la solicitud a la lista del emisor y a la pila del emisor.

```
void eliminarSolicitudEnviada(const string &correoEmisor, const string &correoReceptor)...
```

Metodo que pide como parametro el correo del emisor y receptor y elimina la solicitud a la lista del emisor y de la pila del emisor.

```
450 > void eliminarSolicitudes(const string &correo) ···
```

Metodo que pide como parametro un correo y elimina todas las solicitudes de la lista de ese usuario y de la pila de los usuarios a quien le envio solicitud.

```
void graficarUsuarios(string nombre, string formato)...
```

Método que pide como parámetro el nombre y formato para generar la gráfica de la lista simple de usuarios.

## **MatrizRelacion.h**

Class MatrizDispersa

Private

```
You, hace 3 dias | I author (You)

17 > struct NodoRelacion ...
```

Struct del nodo de relación, donde se establecen los parámetros de la relación, la cual consiste en 2 usuarios y 4 nodosRelacion nombrada con las direcciones específicas junto su respectivo constructor.

Constructor de la Matriz dispersa.

Destructor encargado de eliminar el contenido de la matriz dispersa para la liberación de memoria.

```
void agregarRelacion(ListaUsuarios::Usuario *user1, ListaUsuarios::Usuario *user2) ···
```

Método que pide como parámetros 2 usuarios, encargado de agregar la relación entre ellos.

```
98 > NodoRelacion *buscarFila(ListaUsuarios::Usuario *u1) ···
```

Metodo que retorna un NodoRelacion pidiendo como parametro el primer usuario, este se encargara de buscar en las filas al usuario solicitado.

```
112 > NodoRelacion *buscarColumna(ListaUsuarios::Usuario *u2) ...
```

Metodo que retorna un NodoRelacion pidiendo como parametro el segundo usuario, este se encargara de buscar en las columnas al usuario solicitado.

```
126 > bool NodoRelacionExiste(NodoRelacion *fila, ListaUsuarios::Usuario *u2) ...
```

Método que retorna un booleano que pide como parámetros el nodo de la fila y el usuario, el cual se encargara de verificar si existe una relación entre el usuario insertado en la cabecera de la fila y el usuario solicitado.

```
140 > NodoRelacion *insertarFilaCabecera(ListaUsuarios::Usuario *user) ...
```

Método que retorna un NodoRelacion pidiendo como parámetro un usuario, encargado de insertar en las cabeceras de las filas el usuario si este no existe en ellas.

```
167 > NodoRelacion *insertarColumnaCabecera(ListaUsuarios::Usuario *user) ...
```

Método que retorna un NodoRelacion pidiendo como parámetro un usuario, encargado de insertar en las cabeceras de las columnas el usuario si este no existe en ellas.

```
193

194 > void insertarEnFila(NodoRelacion *nuevo, NodoRelacion *filaCabecera) ...

213
```

Método encargado de insertar en el un nodo en la fila de las cabeceras del usuario con el que tiene relación.

```
void insertarEnColumna(NodoRelacion *nuevo, NodoRelacion *columnaCabecera) ···
```

Método encargado de insertar en el un nodo en la columna de las cabeceras del usuario con el que tiene relación.

```
234 > void imprimir() const...
```

Método encargado de imprimir en consola una representación gráfica de la relación de la matriz dispersa.

```
279 > void eliminarRelacionesUsuario(string correo_usuario)…
```

Método que pide como parámetro el correo de un usuario, encargado de eliminar cualquier relación de dicho usuario.

```
bool existeRelacion(string correo1, string correo2) ...
```

Método que retorna un booleano que pide como parámetros 2 correos para verificar la relación entre ambos usuarios.

```
355 > ListaUsuarios::ListaEnlazadaSimple obtenerAmigos(string correo) ···
```

Método que retorna una lista enlazada simple de usuarios pidiendo como parámetro un correo, el cual se encargara de buscar los amigos de dicho usuario respecto a su correo.

```
385 > void top5UsuariosConMenosAmigos() ···
```

Método encargado de imprimir el top 5 de usuarios con menos amigos según la relación de la matriz.

```
void graficarMatrizRelaciones(string nombre, string formato)…
```

Método que pide como parámetro el nombre y formato para generar la gráfica de la matriz dispersa la relación entre los usuarios.

#### Menus.h

Archivo encargado de declarar las funciones que se utilizaran en el archivomenus.cpp.

```
void menuPrincipal();
void menuIniciarSesion();
void menuRegistrarse();
void menuMostrarInformacion();
void menuAdministrador();
void adminGestionarUsuarios();
void adminGestionarReportes();
void adminGestionarTop();
void menuUsuario();
void userGestionarPerfil();
void userGestionarSolicitudes();
void userGestionarPublicaciones();
void userGestionarReportes();
void menuEnviarSolicitud();
void menuEliminarPerfil();
void menuMostrarPublicaciones();
void menuCrearPublicacion();
void menuEliminarPublicacion();
void menuSolicitudesRecibidas();
```