Manual técnico

Índice:

Diagrama de estructuras de datos:

A traves de esta imagen, se puede ver como se organizaron los datos en cada una de las listas.

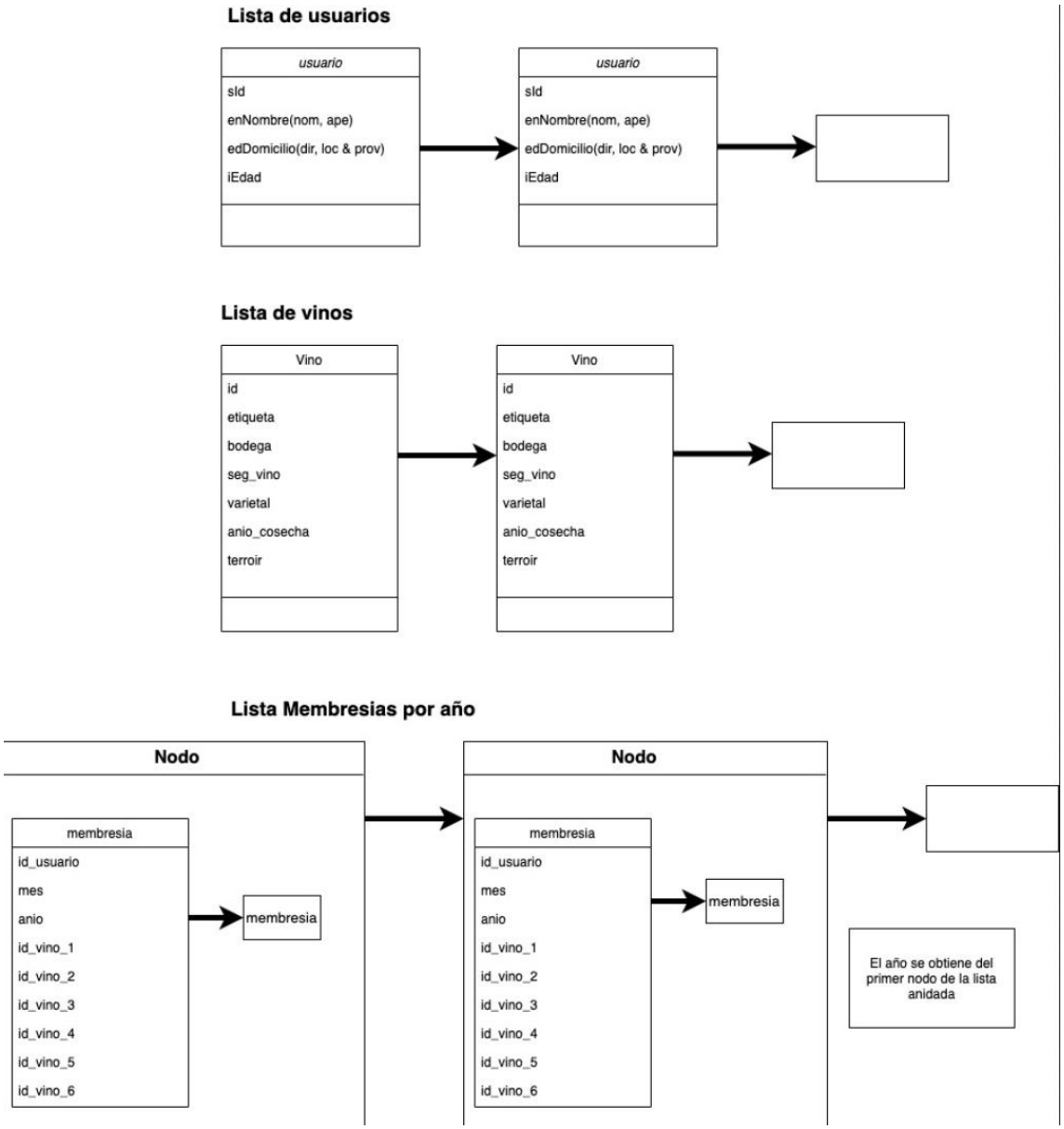
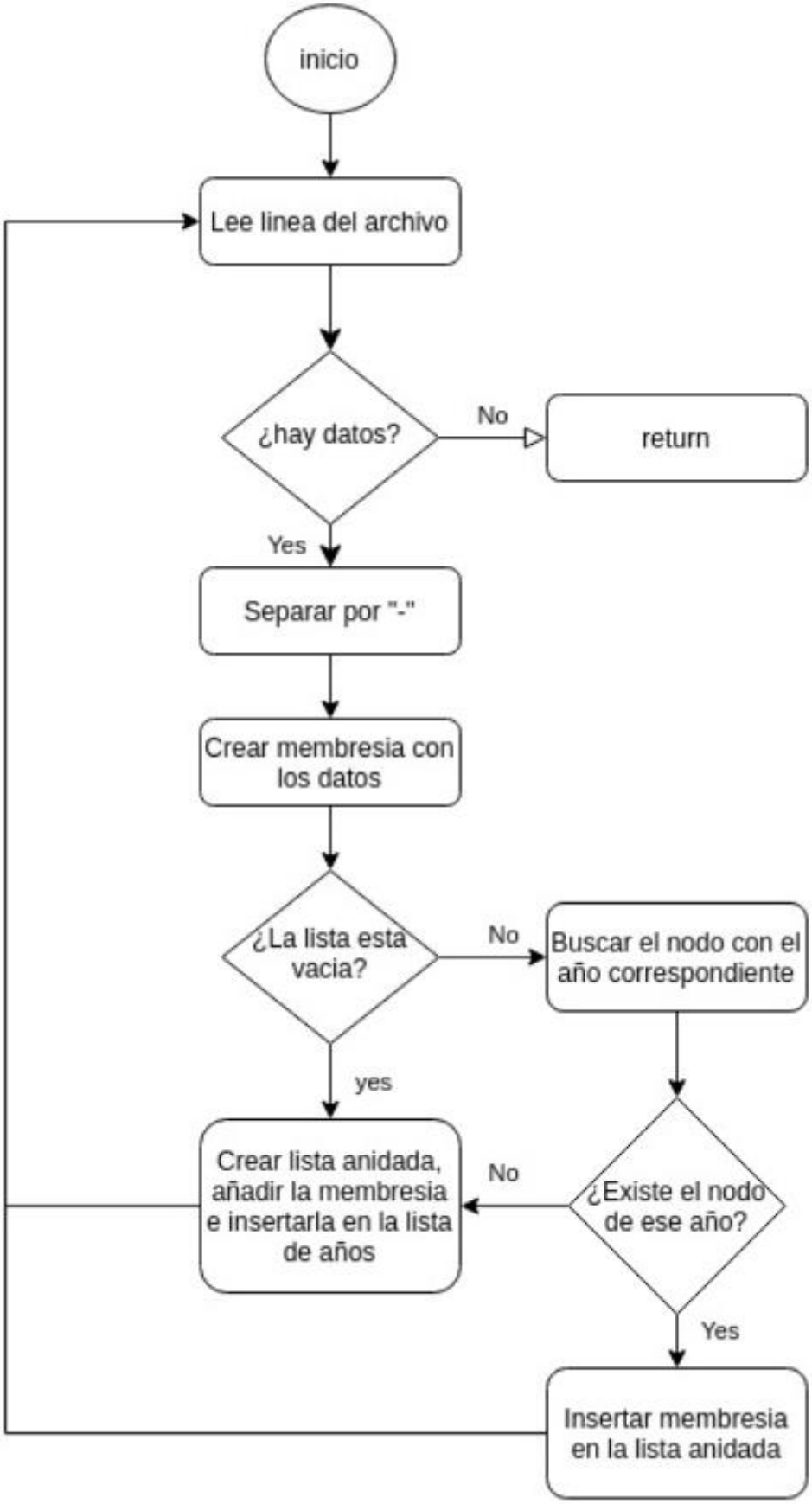


Diagrama de flujo de carga de membresías:

Muestra el algoritmo utilizado para cargar la lista que contiene las membresias.


inicio
Lee linea del archivo
¿hay datos?
No
return
Yes
Separar por "-"
Crear membresia con los datos
¿La lista esta vacia?
No
Buscar el nodo con el año correspondiente
yes
Crear lista anidada, añadir la membresia e insertarla en la lista de años
No
¿Existe el nodo de ese año?
Yes
Insertar membresia en la lista anidada

Estructura de Lista:

```
struct Lista {
   Nodo* inicio;
   int iTamanio_Lista;
};
```

Estructura de Nodo:

```
struct Nodo
{
        ELEMENTO dato;
        Nodo* siguiente;
};
```

Estructura de Usuario:

```
typedef struct {
   std::string sID;
   Nombre* enNombre;
   Domicilio* edDomicilio;
   int iEdad;
} Usuario;
```

Estructura de Nombre/Usuario:

```
typedef struct {
   std::string sApellido;
   std::string sNombre;
} Nombre;
```

Estructura de Domicilio/Usuario:

```
typedef struct {
   std::string sProvincia;
   std::string sLocalidad;
   std::string sDireccion;
} Domicilio;
```

Estructura de loadMembresia:

```
struct Membresia {
   std::string id_usuario;
   std::string mes;
   std::string anio;
   std::string id_vino_1;
   std::string id_vino_2;
   std::string id_vino_3;
   std::string id_vino_4;
   std::string id_vino_5;
   std::string id_vino_6;
};
```

Estructura de Marketing:

```
struct DatoRanking {

   std::string id_vino;

   std::string etiqueta_vino;

   std::string nombre_bodega;

   int contador = 0;

};
```

Aclaración:

El TDA Usuario se debe re-factorizar para eliminar los TDA Nombre y Domicilio que no deberían de estar así, deberían ser structs.

Funciones de Lista:

PRE: La lista no debe haber sido creada.

POST: La lista queda creada y el tamaño de la lista queda seteado en 0.

Lista* crearLista();

PRE: La lista debe haber sido creada.

POST: Devuelve el dato contenido en el iTamaño_Lista de la lista.

int getCantidadDeElementosEnLaLista(Lista*);

PRE: La lista debe haber sido creada.

POST: La lista queda vacía y de tamaño 0.

void vaciarLista(Lista*&);

PRE: La lista debe haber sido creada.

POST: Indica si la lista tiene o no elementos.

bool listaEstaVacia(Lista*);

PRE: La lista debe haber sido creada.

POST: Busca un elemento en la lista y me indica en que posición se encuentra.

int posicionElementoEnLaLista(Lista*, ELEMENTO);

PRE: La lista debe haber sido creada.

POST: Agrego al inicio de la lista 1 elemento.

void insertarElementoAlInicioDeLaLista(Lista*&, ELEMENTO);

PRE: La lista debe haber sido creada.

POST: Agrego 1 elemento a la lista en la posición indicada si es que existe.

void insertarElementoALaLista(Lista*&, int iPosicion, ELEMENTO);

PRE: La lista debe haber sido creada.

POST: Agrego 1 elemento al final de la lista.

void insertarElementoAlFinalDeLaLista(Lista*&, ELEMENTO);

PRE: La lista debe haber sido creada.

POST: Almaceno en una variable el ELEMENTO que poseo en el inicio de la lista.

void obtenerElementoInicialDeLaLista(Lista*, ELEMENTO&);

PRE: La lista debe haber sido creada.

POST: Almaceno en una variable el ELEMENTO que poseo en una posición indicada de la lista si es que existe.

void obtenerElementoDeLaLista(Lista* &, int iPosicion, ELEMENTO&);

PRE: La lista debe haber sido creada.

POST: Almaceno en una variable el ELEMENTO que poseo al final de la lista.

void obtenerElementoFinalDeLaLista(Lista* &, ELEMENTO&);

PRE: La lista debe haber sido creada.

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo en el inicio de la lista.

void eliminarElementoInicialDeLaLista(Lista*&, ELEMENTO&);

PRE: La lista debe haber sido creada.

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo en una posicion indicada de la lista si es que existe.

void eliminarElementoDeLaLista(Lista*&, int iPosicion, ELEMENTO&);

PRE: La lista debe haber sido creada.

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo al final de la lista.

void eliminarElementoFinalDeLaLista(Lista*&, ELEMENTO&);

Funciones de Nodo:

PRE: La lista debe haber sido creada.

POST: Agrego al inicio de la lista 1 elemento.

void insertarAlInicio(Nodo*&, ELEMENTO);

PRE: La lista debe haber sido creada.

POST: Agrego 1 elemento al final de la lista.

void insertarAlFinal(Nodo*&, ELEMENTO);

PRE: La lista debe haber sido creada.

POST: Agrego 1 elemento a la lista en la posición indicada si es que existe.

void insertarElementoEnPosicion(Nodo* &, int iPosicion, ELEMENTO dato);

PRE: La lista debe haber sido creada.

POST: Busca un elemento en la lista y me indica en que posición se encuentra.

int buscarElemento(Nodo*, ELEMENTO);

PRE: La lista debe haber sido creada.

POST: Almaceno en una variable el ELEMENTO que poseo en una posición indicada de la lista si es que existe.

void obtenerElemento(Nodo*, int iPosicion, ELEMENTO&);

PRE: La lista debe haber sido creada.

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo en el inicio de la lista.

void quitarElementoDelInicio(Nodo*&, ELEMENTO&);

PRE: La lista debe haber sido creada.

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo en una posición indicada de la lista si es que existe.

void quitarElementoDePosicion(Nodo*&, int iPosicion, ELEMENTO &);

PRE: La lista debe haber sido creada.

POST: La lista queda vacía.

void vaciarLista(Nodo*&);

Funciones de Usuario:

PRE: El usuario no debe haber sido creado.

POST: El usuario queda creado.

Usuario* crearUsuario(std::string sID, std::string sNombre, std::string sApellido, std::string sDireccion,

std::string sLocalidad, std::string sProvincia, int iEdad);

PRE: El usuario debe haber sido creado.

POST: Muestro los datos del usuario creado.

void mostrarUsuario(Usuario*);

PRE: El usuario debe haber sido creado.

POST: El nombre del usuario es modificado por el dato ingresado en el nombre o el apellido según se indique.

void setNombreUsuario(Usuario*, std::string, void setDatoNombre(Nombre*, std::string));

PRE: El usuario debe haber sido creado.

POST: El domicilio del usuario es modificado por el dato ingresado en la dirección, la localidad o la provincia según se indique.

void setDomicilioUsuario(Usuario*, std::string, void setDatoDomicilio(Domicilio*, std::string));

PRE: El usuario debe haber sido creado.

POST: La edad del usuario es modificada por el dato ingresado.

void setEdadUsuario(Usuario*, int);

PRE: El usuario debe haber sido creado.

POST: Devuelve el dato contenido en el ID del usuario.

std::string getID(Usuario*);

PRE: El usuario debe haber sido creado.

POST: Devuelve el dato contenido en el nombre o el apellido del usuario según se indique.

std::string getNombreUsuario(Usuario*, std::string getDatoNombre(Nombre*));

PRE: El usuario debe haber sido creado.

POST: Devuelve el dato contenido en la dirección, la localidad o la provincia del usuario segun se indique.

std::string getDomicilioUsuario(Usuario*, std::string getDatoDomicilio(Domicilio*));

PRE: El usuario debe haber sido creado.

POST: Devuelve el dato contenido en la iEdad del usuario.

int getEdadUsuario(Usuario*);

PRE: El usuario debe haber sido creado.

POST: El usuario es eliminado.

void destruirUsuario(Usuario*);

Funciones de Nombre/Usuario:

PRE: El nombre no debe haber sido creado.

POST: El nombre queda creado.

Nombre* crearNombre(std::string sApellido, std::string sNombre);

PRE: El nombre debe haber sido creado.

POST: El campo apellido pasa a contener el dato ingresado.

void setApellido(Nombre*, std::string);

PRE: El nombre debe haber sido creado.

POST: El campo nombre pasa a contener el dato ingresado.

void setNombre(Nombre*, std::string);

PRE: El nombre debe haber sido creado.

POST: Devuelve el dato contenido en el campo apellido.

std::string getApellido(Nombre*);

PRE: El nombre debe haber sido creado.

POST: Devuelve el dato contenido en el campo nombre.

std::string getNombre(Nombre*);

PRE: El nombre debe haber sido creado.

POST: El nombre es eliminado.

void destruirNombre(Nombre*);

Funciones de Domicilio/Usuario:

PRE: El domicilio no debe haber sido creado.

POST: El domicilio queda creado.

Domicilio* crearDomicilio(std::string sProvincia, std::string sLocalidad, std::string sDireccion);

PRE: El domicilio debe haber sido creado.

POST: El campo provincia pasa a contener el dato ingresado.

void setProvincia(Domicilio*, std::string);

PRE: El domicilio debe haber sido creado.

POST: El campo localidad pasa a contener el dato ingresado.

void setLocalidad(Domicilio*, std::string);

PRE: El domicilio debe haber sido creado.

POST: El campo dirección pasa a contener el dato ingresado.

void setDireccion(Domicilio*, std::string);

PRE: El domicilio debe haber sido creado.

POST: Devuelve el dato contenido en el campo provincia.

std::string getProvincia(Domicilio*);

PRE: El domicilio debe haber sido creado.

POST: Devuelve el dato contenido en el campo localidad.

std::string getLocalidad(Domicilio*);

PRE: El domicilio debe haber sido creado.

POST: Devuelve el dato contenido en el campo dirección.

std::string getDireccion(Domicilio*);

PRE: El domicilio debe haber sido creado.

POST: El domicilio es eliminado.

void destruirDomicilio(Domicilio*);

Funciones de ArchivoUsuario:

PRE: La lista debe haber sido creada.

POST: La lista tendrá cargados los datos de los usuarios almacenados en el .txt.

void cargarDatosDeUsuarioEnLaLista(const char*, Lista*&);

PRE: La lista debe haber sido creada.

POST: Se muestra por consola los datos de todos los usuarios que hay en la lista.

void mostrarDatosDeLosUsuariosEnLaLista(Lista*&);

Funciones de loadMembresia:

PRE: La lista fue creada con crearLista().

POST: Si el path es correcto, se leerá línea a línea para extraer sus datos, creando una membresía por cada una y se la agregará a la lista.

PATH: Ubicación del archivo.

LISTA: lista donde se almacenaran los datos.

void readFileAndLoad(std::string path, Lista *lista);

PRE: str debe contener los datos necesarios para cargar una membresía.

POST: Se limpia y separa str en cada uno de los datos para la membresía.

STR: Cadena a la cual se va a quitar espacios, tabs y luego separarla.

DEL: Cadena que separa los datos de str.

RETURN: Array[8] string con los datos de la membresía

std::string* splitStrByChar(std::string str, std::string del);

Membresia* getInnerMembresia(Nodo *nodo);

void showMembresiaList(Lista *lista);

Funciones de cargarArchivosEnLista:

PRE: La lista debe haber sido creada.

POST: La lista tendrá cargados los datos de los usuarios almacenados en el .txt.

void cargarDatosDeUsuarioEnLaLista(const char*, Lista*);

PRE: Debe existir el elemento que se desea mostrar y debe ser un usuario.

POST: Muestro un dato de la lista de tipo usuario según la posición de la lista.

void mostrarDatoUsuario(ELEMENTO);

PRE: La lista debe haber sido creada.

POST: La lista tendrá cargados los datos de los vinos almacenados en el .txt.

void cargarCatalogoDeVinosEnLaLista(const char*, Lista*);

PRE: Debe existir el elemento que se desea mostrar y debe ser un vino.

POST: Muestro un dato de la lista de tipo vino según la posición de la lista.

void mostrarDatoVino(ELEMENTO);

Funciones de Marketing:

PRE: Debe existir la lista de la membrecía, de los usuarios y la lista del catálogo.

POST: Devuelve por consola el ranking de los varietales por grupo etario.

void rankingVarietalesPorGrupoEtario(Lista* lMembresia, Lista* lUsuario, Lista* lCatalogos);

PRE: Debe existir la lista de la membresía, la lista del catálogo y 2 variables de tipo int donde almacenar la cantidad de ventas y el año.

POST: Retorna la lista de datos para los rankings.

Lista* listaParaHacerLosRankings(Lista *listaAnioMembresias, Lista *listaVinos, int &maxYear, int &contadorTotalVinos);

PRE: Debe existir la lista de los datos del ranking, función anterior, y las 2 variables de tipo int donde se almacenaron la cantidad de ventas y el año.

POST: Muestra por consola el ranking de vinos del último año.

void rankingVinosUltimoAnio(Lista* listaRanking, int maxYear, int contadorTotalVinos);

PRE: Debe existir la lista de ranking de vinos del último año y una variable de tipo int que indique el año.

POST: Muestra por consola el ranking de bodegas del último año.

void rankingBodegasUltimoAnio(Lista *listaRankingVinos, int maxYear);