



# ALGORITMOS Y ESTRUCTURAS DE DATOS

## *Anexo 1: Código Fuente*

*Universidad Nacional de Lanús, 30 de octubre del 2020*

*Prof. Alejandro Sasin y Diego Cañete*

Nombre	DNI	Mail
<b>Cusato Facundo</b>	36363056	facundocusato12@gmail.com
<b>Fragas Rodrigo</b>	40857822	rodrigofragas11@gmail.com
<b>Fretes Rocio</b>	40916995	rociobh80@gmail.com
<b>Medina Walter</b>	38701467	waltertmedina@gmail.com

# Indice

## APARTADO 1

MAIN.CPP .....	3
Main.cpp .....	4

## APARTADO 2

ARCHIVOS.H .....	5
AdmArchivos.h .....	6
CasaMatriz.h.....	10
ListaCasaMatriz.h .....	13
ListaProvincia.h .....	24
ListaSucursal.h.....	36
Provincia.h.....	47
Sucursal.h .....	51

## APARTADO 3

ARCHIVOS.CPP.....	56
AdmArchivos.cpp .....	57
CasaMatriz.cpp.....	67
ListaCasaMatriz.cpp .....	68
ListaProvincia.cpp.....	77
ListaSucursal.cpp.....	87
Provincia.cpp .....	96
Sucursal.cpp .....	98

# MAIN.CPP

# Main.cpp

---

```
#include <iostream>
#include "AdmArchivos.h"

using namespace std;

int main()
{
    system("color 0E");
    ListaSucursal listaSucursal;
    ListaProvincia listaProvincia;
    ListaCasaMatriz listaCasaMatriz;
    crearListaSucursal(listaSucursal);
    crearListaProvincia(listaProvincia);
    crearListaCasaMatriz(listaCasaMatriz);
    cargarSucursales(listaSucursal, listaProvincia, listaCasaMatriz);
    menu(listaSucursal, listaProvincia, listaCasaMatriz);
    eliminarListaProvincia(listaProvincia);
    eliminarListaCasaMatriz(listaCasaMatriz);
    eliminarListaSucursal(listaSucursal);
    return 0;
}
```

# ARCHIVOS.H

# AdmArchivos.h

---

```
#include "Sucursal.h"
#include "Provincia.h"
#include "CasaMatriz.h"
#include "ListaSucursal.h"
#include "ListaProvincia.h"
#include "ListaCasaMatriz.h"
#include <iostream>
#include <sstream>
#include <fstream>
#include <cstdlib>
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <cstring>
#include <iomanip>

#ifndef ADMARCHIVOS_H_INCLUDED
#define ADMARCHIVOS_H_INCLUDED

#ifndef NULL
#define NULL 0
#endif
```

```
/**
```

PRE: Las listas de sucursal, provincia y casaMatriz deben haber sido creadas.

POST: Las listas quedan cargadas con los datos leídos del archivo de texto.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

ListaProvincia: Lista sobre la cual se invoca la primitiva.

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

RETORNO: No aplica.

\*/

```
void cargarSucursales(ListaSucursal &listaSucursal,ListaProvincia &listaProvincia,ListaCasaMatriz  
&listaCasaMatriz);
```

/\*\*

PRE: Las listas de sucursal, provincia y casaMatriz deben haber sido creadas.

POST: Menu interactivo de opciones para consultas a realizar por el usuario.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

ListaProvincia: Lista sobre la cual se invoca la primitiva.

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

RETORNO: No aplica.

\*/

```
void menu(ListaSucursal &listaSucursal,ListaProvincia &listaProvincia,ListaCasaMatriz  
&listaCasaMatriz);
```

/\*\*

PRE: La lista sucursal debe haber sido creada y cargada con cargarSucursales().

POST: Se obtiene la lista de sucursales cargadas en el sistema en el orden en que fueron leidas desde el archivo.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

RETORNO: No aplica

\*/

```
void obtenerSucursales(ListaSucursal &listaSucursal);
```

/\*\*

PRE: La lista sucursal debe haber sido creada y cargada con cargarSucursales().

POST: Se obtiene el reporte del ranking nacional de sucursales ordenadas por monto de facturacion.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

RETORNO: No aplica.

\*/

```
void rankingNacionalPorMontoFacturacion(ListaSucursal &listaSucursal);
```

/\*\*

PRE: La lista sucursal y provincia deben haber sido creadas y cargadas con cargarSucursales().

POST: Se obtiene el reporte del ranking provincial de sucursales ordenadas por monto de facturacion total por provincia.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se aplica la primitiva.

listaProvincia: Lista sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

```
void rankingProvincialPorMontoFacturacion(ListaSucursal &listaSucursal,ListaProvincia  
&listaProvincia);
```

/\*\*

PRE: La lista sucursal debe haber sido creada y cargada con cargarSucursales().

POST: Se obtiene el reporte del ranking nacional de sucursales ordenadas por cantidad total de articulos vendidos.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

```
void rankingNacionalPorCantArticulosVendidos(ListaSucursal &listaSucursal);
```

/\*\*

PRE: La lista sucursal y provincia deben haber sido creadas y cargadas con cargarSucursales().

POST: Se obtiene el reporte del ranking provincial de sucursales ordenadas por cantidad total de articulos vendidos por provincia.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se aplica la primitiva.

listaProvincia: Lista sobre la cual se aplica la primitiva.



RETORNO: No aplica.

\*/

```
void rankingProvincialPorCantArticulosVendidos(ListaSucursal &listaSucursal, ListaProvincia  
&listaProvincia);
```

/\*\*

PRE: La lista sucursal y casaMatriz deben haber sido creadas y cargadas con cargarSucursales().

POST: Se obtiene el ranking de sucursales ordenadas por rendimiento de cada  
casaMatriz(montoFacturacion/m2)

ATRIBUTOS:

listaSucursal: Lista sobre la cual se aplica la primitiva.

listaCasaMatriz: Lista sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

```
void rankingRendimiento(ListaSucursal &listaSucursal,ListaCasaMatriz &listaCasaMatriz);
```

/\*\*

PRE: La lista sucursal y provincia deben haber sido creadas y cargadas con cargarSucursales().

POST: Se setean los campos del monto de facturacion total por cada provincia.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se aplica la primitiva.

listaProvincia: Lista sobre la cual se aplica la primitiva.

RETORNO: No aplica

\*/

```
void facturacionTotalPorProvincia(ListaSucursal &listaSucursal, ListaProvincia &listaProvincia);
```

/\*\*

PRE: La lista sucursal y provincia deben haber sido creadas y cargadas con cargarSucursales().

POST: Se setean los campos de la cantidad total de articulos vendidos por cada provincia.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se aplica la primitiva.

listaProvincia: Lista sobre la cual se aplica la primitiva.

RETORNO: No aplica

\*/

```

void articulosTotalesVendidosPorProvincia(ListaSucursal &listaSucursal, ListaProvincia
&listaProvincia);

/**

PRE: La lista sucursal y casaMatriz deben haber sido creadas y cargadas con cargarSucursales().
POST: Se setea los campos del rendimiento total por cada casa matriz.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se aplica la primitiva.
listaCasaMatriz: Lista sobre la cual se aplica la primitiva.

RETORNO: No aplica.

*/

void rendimientoTotalPorCasaMatriz(ListaSucursal &listaSucursal, ListaCasaMatriz
&listaCasaMatriz);

#endif // ADMARCHIVOS_H_INCLUDED

```

## CasaMatriz.h

---

```

#ifndef CASAMATRIZ_H_INCLUDED

```

```

typedef struct{
    int idCasaMatriz;
    float rendimiento;
}CasaMatriz;

```

```

#define CASAMATRIZ_H_INCLUDED

```

```

/**

```

DEFINICION DEL TIPO DE DATO PARA MANEJO DE ATRIBUTOS:

```

idCasaMatriz
rendimiento

```

AXIOMAS:

idCasaMatriz > 0

\*/

/\*\*

PRE: La instancia TDA (CasaMatriz) no debe haberse creado pero no debe estar destruida.

POST: Deja la instancia del TDA (CasaMatriz) listo para ser usado.

ATRIBUTOS:

casaMatriz: Instancia sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

void crearCasaMatriz(CasaMatriz &casaMatriz);

/\*\*

PRE: La instancia TDA (CasaMatriz) debe haberse creado pero no debe estar destruida.

POST: Elimina la instancia del TDA y ya no podra ser utilizada.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

void eliminarCasaMatriz(CasaMatriz &casaMatriz);

/\*\*

PRE: La instancia TDA(CasaMatriz) debe haberse creado pero no debe estar destruida.

POST: La casaMatriz queda seteada con el nuevo identificador.

ATRIBUTOS:

casaMatriz: Instancia sobre la cual se aplica la primitiva.

idCasaMatriz: Valor del identificador a asignar a la casaMatriz.

RETORNO: No aplica.

\*/

void setIdCasaMatriz(CasaMatriz &casaMatriz, int idCasaMatriz);

/\*\*

PRE: La instancia TDA(CasaMatriz) debe haberse creado pero no debe estar destruida.

POST: Devuelve el identificador de casaMatriz.

ATRIBUTOS:

casaMatriz: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el identificador de casaMatriz.

\*/

```
int getIdCasaMatriz(CasaMatriz &casaMatriz);
```

/\*\*

PRE: La instancia TDA(CasaMatriz) debe haberse creado pero no debe estar destruida.

POST: La casa matriz queda seteada con el rendimiento.

ATRIBUTOS:

casaMatriz: Instancia sobre la cual se aplica la primitiva.

rendimiento: Rendimiento de la casaMatriz asignado.

RETORNO: No aplica.

\*/

```
void setRendimiento(CasaMatriz &casaMatriz,float rendimiento);
```

/\*\*

PRE: La instancia TDA(CasaMatriz) debe haberse creado pero no debe estar destruida.

POST: Devuelve el rendimiento de la casaMatriz.

ATRIBUTOS:

casaMatriz: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el rendimiento de la casaMatriz.

\*/

```
float getRendimiento(CasaMatriz &casaMatriz);
```

```
#endif // CASAMATRIZ_H_INCLUDED
```

# ListaCasaMatriz.h

---

```
#include "CasaMatriz.h"
#include "ListaSucursal.h"
#include <iomanip>
#include <cstring>
#include <iostream>

#ifndef LISTACASAMATRIZ_H_INCLUDED
#define LISTACASAMATRIZ_H_INCLUDED

/*****
** Definiciones de Tipos de Datos */
/*-----*/

/** tipo enumerado para realizar comparaciones */
enum ResultadoComparacionCasaMatriz {
    CMAYOR,
    CIGUAL,
    CMENOR
};

/** Tipo de Informacion que esta contenida en los Nodos de la
    Lista, identificada como Dato. */
typedef CasaMatriz DatoCasaMatriz;

/** Tipo de Estructura de los Nodos de la Lista. */
struct NodoListaCasaMatriz {
    DatoCasaMatriz dato; // dato almacenado
    NodoListaCasaMatriz* sgte; // puntero al siguiente
};
```

```

/** Tipo de Puntero a los Nodos de la Lista, el cual se usa para recorrer
    la Lista y acceder a sus Datos. */
typedef NodoListaCasaMatriz* PtrNodoListaCasaMatriz;

/** Tipo de Estructura de la Lista */
struct ListaCasaMatriz{
    PtrNodoListaCasaMatriz primeroCasaMatriz;    // puntero al primer nodo de la lista
};

/*****
/** Definicion de Primitivas */
/*-----*/

/**
PRE: La lista no debe haber sido creada.
POST: Lista queda creada y preparada para ser usada.
ATRIBUTOS:
    listaCasaMatriz: Estructura de datos a ser creado.
*/
void crearListaCasaMatriz(ListaCasaMatriz &listaCasaMatriz);

/*-----*/
/**
PRE : Lista Creada con crearLista().
POST: Devuelve true si lista esta vacia, sino devuelve false.
ATRIBUTOS:
    listaCasaMatriz : Lista sobre la cual se invoca la primitiva.

```

\*/

bool listaVaciaCasaMatriz(ListaCasaMatriz &listaCasaMatriz);

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

POST: Devuelve la representacion de lo Siguiente al último Nodo de la lista,  
o sea el valor Null, que en esta implementacion representa el final de  
la lista.

RETORNO: Representación del fin de la lista.

\*/

PtrNodoListaCasaMatriz finCasaMatriz();

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

POST: Devuelve el puntero al primer elemento de la lista, o devuelve fin() si  
esta vacia

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

RETORNO: Puntero al primer nodo.

\*/

PtrNodoListaCasaMatriz primeroCasaMatriz(ListaCasaMatriz &listaCasaMatriz);

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

POST: Devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si  
ptrNodo apuntaba a fin() o si lista esta vacia.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

ptrNodo: Puntero al nodo a partir del cual se requiere el siguiente.

RETORNO: Puntero al nodo siguiente.

\*/

```
PtrNodoListaCasaMatriz siguienteCasaMatriz(ListaCasaMatriz &listaCasaMatriz,  
PtrNodoListaCasaMatriz ptrNodo);
```

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

ptrNodo es un puntero a un nodo de lista.

POST: Devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si

ptrNodo apuntaba al primero o si lista esta vacia.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

ptrNodo: Puntero al nodo a partir del cual se requiere el anterior.

RETORNO: Puntero al nodo anterior.

\*/

```
PtrNodoListaCasaMatriz anteriorCasaMatriz(ListaCasaMatriz &listaCasaMatriz,  
PtrNodoListaCasaMatriz ptrNodo);
```

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si

si lista esta vacia.

ATRIBUTOS:

listaCasaMatriz : Lista sobre la cual se invoca la primitiva.

RETORNO: Puntero al último nodo.

\*/



PtrNodoListaCasaMatriz ultimoCasaMatriz(ListaCasaMatriz &listaCasaMatriz);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo nuevo al principio de la lista con el dato proporcionado  
y devuelve un puntero a ese elemento.

ATRIBUTOS:

listaCasaMatriz : Lista sobre la cual se invoca la primitiva.

dato : Elemento a adicionar al principio de la lista.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaCasaMatriz adicionarPrincipioCasaMatriz(ListaCasaMatriz &listaCasaMatriz,  
DatoCasaMatriz dato);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo despues del apuntado por ptrNodo con el dato  
proporcionado y devuelve un puntero apuntado al elemento insertado.  
Si la lista esta vacía agrega un nodo al principio de esta y devuelve  
un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta  
nada y devuelve fin().

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar.

ptrNodo : Puntero al nodo después del cual se quiere adicionar el dato.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaCasaMatriz adicionarDespuesCasaMatriz(ListaCasaMatriz &listaCasaMatriz,  
DatoCasaMatriz dato, PtrNodoListaCasaMatriz ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo al final de la lista con el dato proporcionado y devuelve un puntero al nodo insertado.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar al final de la lista.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaCasaMatriz adicionarFinalCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

post: Agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo y devuelve un puntero al nodo insertado. Si la lista esta vacia no inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo insertado sera el nuevo primero.

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar.

ptrNodo: Puntero al nodo antes del cual se quiere adicionar el dato.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaCasaMatriz adicionarAntesCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato, PtrNodoListaCasaMatriz ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().

POST: Coloca el dato proporcionado en el nodo apuntado por ptrNodo.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato : elemento a colocar.

ptrNodo : Puntero al nodo del cual se quiere colocar el dato.

\*/

```
void colocarDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz &dato,  
PtrNodoListaCasaMatriz ptrNodo);
```

/\*-----\*/

/\*\*

PRE : Lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().

POST: Devuelve el dato del nodo apuntado por ptrNodo.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento obtenido.

ptrNodo: Puntero al nodo del cual se quiere obtener el dato.

\*/

```
void obtenerDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz &dato,  
PtrNodoListaCasaMatriz ptrNodo);
```

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Elimina el nodo apuntado por ptrNodo. No realiza accion si la lista  
esta vacia o si ptrNodo apunta a fin().

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

ptrNodo: Puntero al nodo que se desea eliminar.

\*/

void eliminarNodoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, PtrNodoListaCasaMatriz ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Si la lista no esta vacia, elimina su nodo primero, sino no realiza accion alguna.

ATRIBUTOS:

listaCasaMatriz : Lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimeroCasaMatriz(ListaCasaMatriz &listaCasaMatriz);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Si la lista no esta vacia elimina su nodo ultimo, sino no realiza accion.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimoCasaMatriz(ListaCasaMatriz &listaCasaMatriz);

/\*-----\*/

/\*\*

PRE: lista creada con crearLista().

POST: Elimina todos los Nodos de la lista quedando destruida e inhabilitada para su uso.

```

    listaCasaMatriz : Lista sobre la cual se invoca la primitiva.
*/

void eliminarListaCasaMatriz(ListaCasaMatriz &listaCasaMatriz);

/*****
/** Definición de Operaciones Adicionales */
/**-----*/

/**
PRE: Lista fue creada con crearLista().
POST: Si el dato se encuentra en la lista, devuelve el puntero al primer nodo
      que lo contiene. Si el dato no se encuentra en la lista devuelve fin().
ATRIBUTOS:
listaCasaMatriz: Lista sobre la cual se invoca la primitiva.
dato: Elemento a localizar.
RETORNO: Puntero al nodo localizado o fin().
*/

PtrNodoListaCasaMatriz localizarDatoCasaMatriz(ListaCasaMatriz
&listaCasaMatriz,DatoCasaMatriz dato);

/**
PRE: Lista fue creada con crearLista().
POST: Si el dato se encuentra en la lista, devuelve el puntero al primer nodo
      que lo contiene. Si el dato no se encuentra en la lista devuelve fin().
ATRIBUTOS:
listaCasaMatriz: Lista sobre la cual se invoca la primitiva.
dato: Elemento a localizar.
RETORNO: Puntero al nodo localizado o fin().
*/

PtrNodoListaCasaMatriz localizarDatoCasaMatriz2(ListaCasaMatriz
&listaCasaMatriz,DatoCasaMatriz dato);

```

/\*-----\*/

/\*\*

PRE: Lista fue creada con crearLista() y cargada con datos ordenados de menor a mayor respecto del sentido progresivo.

POST: agrega a la lista el dato manteniendo el orden pero con multiples valores iguales y devuelve un puntero al nodo insertado.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato : Elemento a insertar.

RETORNO: Puntero al nodo insertado.

\*/

PtrNodoListaCasaMatriz insertarDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato);

/\*\*

PRE: Lista fue creada con crearLista() y cargada con datos ordenados de menor a mayor respecto del sentido progresivo.

POST: Agrega a la lista el dato manteniendo el orden pero con multiples valores iguales y devuelve un puntero al nodo insertado.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento a insertar.

RETORNO: Puntero al nodo insertado.

\*/

PtrNodoListaCasaMatriz insertarDatoCasaMatriz2(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Elimina el dato de la lista, si el mismo se encuentra.

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento a eliminar.

\*/

void eliminarDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Elimina el dato de la lista, si el mismo se encuentra.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento a eliminar.

\*/

void eliminarDatoCasaMatriz2(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Reordena la lista por idCasaMatriz de mayor a menor.

ATRIBUTOS:

listaCasaMatriz : Lista sobre la cual se invoca la primitiva.

\*/

void reordenarPorId(ListaCasaMatriz &listaCasaMatriz);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Reordena la lista por rendimiento total de mayor a menor.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

```

*/
void reordenarPorRendimiento(ListaCasaMatriz &listaCasaMatriz);

/*-----*/
/**
PRE: La lista fue creada con crearLista().
POST: Devuelve la cantidad de datos que tiene la lista.
ATRIBUTOS:
listaCasaMatriz: Lista sobre la cual se invoca la primitiva.
*/
int longitud(ListaCasaMatriz &listaCasaMatriz);
/*-----*/
/**
PRE: La lista fue creada con crearLista().
POST: Devuelve la lista de las provincias.
ATRIBUTOS:
listaCasaMatriz: Lista sobre la cual se invoca la primitiva.
*/
void obtenerListaCasaMatriz(ListaSucursal &listaSucursal, ListaCasaMatriz &listaCasaMatriz);

#endif // LISTACASAMATRIZ_H_INCLUDED

```

## ListaProvincia.h

---

```

#include "Provincia.h"
#include "ListaSucursal.h"
#include <cstring>
#include <iostream>
#include <iomanip>

```



```

#ifndef LISTAPROVINCIA_H_INCLUDED
#include "Provincia.h"
#include "ListaSucursal.h"
#include <cstring>
#include <iostream>
#include <iomanip>

#ifndef LISTAPROVINCIA_H_INCLUDED
#define LISTAPROVINCIA_H_INCLUDED

#ifndef NULL
#define NULL    0
#endif

/*****
** Definiciones de Tipos de Datos */
/*-----*/

/** tipo enumerado para realizar comparaciones */
enum ResultadoComparacionProvincia {
    PMAYOR,
    PIGUAL,
    PMENOR
};

/** Tipo de Informacion que esta contenida en los Nodos de la
    Lista, identificada como Dato. */
typedef Provincia DatoProvincia;

/** Tipo de Estructura de los Nodos de la Lista. */
struct NodoListaProvincia {
    DatoProvincia dato; // dato almacenado

```

```

    NodoListaProvincia* sgte; // puntero al siguiente
};

/** Tipo de Puntero a los Nodos de la Lista, el cual se usa para recorrer
    la Lista y acceder a sus Datos. */
typedef NodoListaProvincia* PtrNodoListaProvincia;

/** Tipo de Estructura de la Lista */
struct ListaProvincia{
    PtrNodoListaProvincia primeroProvincia;    // puntero al primer nodo de la lista
};

/*****
/** Definicion de Primitivas */
/*-----*/

/**
PRE: La lista no debe haber sido creada.
POST: Lista queda creada y preparada para ser usada.
ATRIBUTOS:
    listaProvincia: Estructura de datos a ser creado.
*/
void crearListaProvincia(ListaProvincia &listaProvincia);

/*-----*/
/**
PRE: Lista Creada con crearLista().
POST: Devuelve true si lista esta vacia, sino devuelve false.

```

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

\*/

bool listaVaciaProvincia(ListaProvincia &listaProvincia);

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

POST: Devuelve la representacion de lo Siguiende al último Nodo de la lista,  
o sea el valor Null, que en esta implementacion representa el final de  
la lista.

RETORNO: Representación del fin de la lista.

\*/

PtrNodoListaProvincia finProvincia();

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

POST: Devuelve el puntero al primer elemento de la lista, o devuelve fin() si  
esta vacia

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

RETORNO: Puntero al primer nodo.

\*/

PtrNodoListaProvincia primeroProvincia(ListaProvincia &listaProvincia);

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

POST: Devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si

ptrNodo apuntaba a fin() o si lista esta vacia.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

prtNodo: Puntero al nodo a partir del cual se requiere el siguiente.

RETURN puntero al nodo siguiente.

\*/

PtrNodoListaProvincia siguienteProvincia(ListaProvincia &listaProvincia, PtrNodoListaProvincia ptrNodo);

/\*-----\*/

/\*\*

PRE : Lista Creada con crearLista().

ptrNodo es un puntero a un nodo de lista.

POST: Devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si

ptrNodo apuntaba al primero o si lista esta vacia.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

prtNodo: Puntero al nodo a partir del cual se requiere el anterior.

RETORNO: Puntero al nodo anterior.

\*/

PtrNodoListaProvincia anteriorProvincia(ListaProvincia &listaProvincia, PtrNodoListaProvincia ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si

si lista esta vacia.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

RETORNO: Puntero al último nodo.

\*/

PtrNodoListaProvincia ultimoProvincia(ListaProvincia &listaProvincia);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo nuevo al principio de la lista con el dato proporcionado  
y devuelve un puntero a ese elemento.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar al principio de la lista.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaProvincia adicionarPrincipioProvincia(ListaProvincia &listaProvincia, DatoProvincia  
dato);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo despues del apuntado por ptrNodo con el dato  
proporcionado y devuelve un puntero apuntado al elemento insertado.

Si la lista esta vacía agrega un nodo al principio de esta y devuelve  
un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta  
nada y devuelve fin().

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar.

ptrNodo: Puntero al nodo después del cual se quiere adicionar el dato.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaProvincia adicionarDespuesProvincia(ListaProvincia &listaProvincia, DatoProvincia dato, PtrNodoListaProvincia ptrNodo);

/\*-----\*/

/\*\*

PRE : Lista creada con crearLista().

POST: Agrega un nodo al final de la lista con el dato proporcionado y devuelve un puntero al nodo insertado.

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar al final de la lista.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaProvincia adicionarFinalProvincia(ListaProvincia &listaProvincia, DatoProvincia dato);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo y devuelve un puntero al nodo insertado. Si la lista esta vacia no inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo insertado sera el nuevo primero.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar.

ptrNodo: Puntero al nodo antes del cual se quiere adicionar el dato.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaProvincia adicionarAntesProvincia(ListaProvincia &listaProvincia, DatoProvincia dato, PtrNodoListaProvincia ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().

POST: Coloca el dato proporcionado en el nodo apuntado por ptrNodo.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento a colocar.

ptrNodo: Puntero al nodo del cual se quiere colocar el dato.

\*/

```
void colocarDatoProvincia(ListaProvincia &listaProvincia, DatoProvincia &dato,  
PtrNodoListaProvincia ptrNodo);
```

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().

POST: Devuelve el dato del nodo apuntado por ptrNodo.

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento obtenido.

ptrNodo: Puntero al nodo del cual se quiere obtener el dato.

\*/

```
void obtenerDatoProvincia(ListaProvincia &listaProvincia, DatoProvincia &dato,  
PtrNodoListaProvincia ptrNodo);
```

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Elimina el nodo apuntado por ptrNodo. No realiza accion si la lista  
esta vacia o si ptrNodo apunta a fin().

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

ptrNodo: Puntero al nodo que se desea eliminar.

\*/

```
void eliminarNodoProvincia(ListaProvincia &listaProvincia, PtrNodoListaProvincia ptrNodo);
```

```
/*-----*/
```

```
/**
```

PRE: Lista creada con crearLista().

POST: Si la lista no esta vacia, elimina su nodo primero, sino no realiza accion alguna.

listaProvincia: Lista sobre la cual se invoca la primitiva.

```
*/
```

```
void eliminarNodoPrimeroProvincia(ListaProvincia &listaProvincia);
```

```
/*-----*/
```

```
/**
```

PRE: Lista creada con crearLista().

POST: Si la lista no esta vacia elimina su nodo ultimo, sino no realiza accion.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

```
*/
```

```
void eliminarNodoUltimoProvincia(ListaProvincia &listaProvincia);
```

```
/*-----*/
```

```
/**
```

PRE: Lista creada con crearLista().

POST: Elimina todos los Nodos de la lista quedando destruida e inhabilitada para su uso.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

```
*/
```

```
void eliminarListaProvincia(ListaProvincia &listaProvincia);
```



```

/*****/

/** Definición de Operaciones Adicionales */

/*-----*/

/**

PRE: Lista fue creada con crearLista().

POST: Si el dato se encuentra en la lista, devuelve el puntero al primer nodo
      que lo contiene. Si el dato no se encuentra en la lista devuelve fin().

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento a localizar.

RETORNO: Puntero al nodo localizado o fin().

*/

PtrNodoListaProvincia localizarDatoProvincia(ListaProvincia &listaProvincia ,DatoProvincia dato);

/*-----*/

/**

PRE: Lista fue creada con crearLista() y cargada con datos ordenados de
      menor a mayor respecto del sentido progresivo.

POST: Agrega a la lista el dato manteniendo el orden pero con multiples
      valores iguales y devuelve un puntero al nodo insertado.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: elemento a insertar.

RETORNO: Puntero al nodo insertado.

*/

PtrNodoListaProvincia insertarDatoProvincia(ListaProvincia &listaProvincia, DatoProvincia dato);

/*-----*/

```

/\*\*

PRE: lista fue creada con crearLista() y cargada con datos ordenados de menor a mayor respecto del sentido progresivo.

POST: Agrega a la lista el dato manteniendo el orden pero con multiples valores iguales y devuelve un puntero al nodo insertado.

listaProvincia: Lista sobre la cual se invoca la primitiva.

dato: Elemento a insertar.

RETORNO: Puntero al nodo insertado.

\*/

PtrNodoListaProvincia insertarDatoProvincia2(ListaProvincia &listaProvincia, DatoProvincia dato);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Elimina el dato de la lista, si el mismo se encuentra.

ATRIBUTOS:

listaCasaMatriz: Lista sobre la cual se invoca la primitiva.

dato: Elemento a eliminar.

\*/

void eliminarDatoProvincia(ListaProvincia &listaProvincia, DatoProvincia dato);

/\*-----\*/

/\*\*

PRE: la lista fue creada con crearLista().

POST: Reordena la lista por cantidad total de articulos vendidos de mayor a menor.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

\*/

void reordenarPorcantArticulosVendidos(ListaProvincia &listaProvincia);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Reordena la lista por monto de facturacion de mayor a menor.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

\*/

void reordenarPorMontoTotalDeFacturacion(ListaProvincia &listaProvincia);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST : Devuelve la cantidad de datos que tiene la lista.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

\*/

int longitud(ListaProvincia &listaProvincia);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Devuelve la lista de las provincias con su monto total de facturacion.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

\*/

void obtenerListaProvinciaConTotalMontoFacturacion(ListaSucursal &listaSucursal,ListaProvincia &listaProvincia);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Devuelve la lista de las provincias con su monto total de articulos vendidos.

ATRIBUTOS:

listaProvincia: Lista sobre la cual se invoca la primitiva.

```

*/

void obtenerListaProvinciaConTotalArticulosVendidos(ListaSucursal &listaSucursal, ListaProvincia
&listaProvincia);

#endif // LISTAPROVINCIA_H_INCLUDED

```

## ListaSucursal.h

---

```

#include "Sucursal.h"
#include "Provincia.h"

#ifndef LISTASUCURSAL_H_INCLUDED
#define LISTASUCURSAL_H_INCLUDED

#ifndef NULL
#define NULL 0
#endif

/*****
/* Definiciones de Tipos de Datos */
/*-----*/

/* tipo enumerado para realizar comparaciones */
enum ResultadoComparacionSucursal {
    MAYOR,
    IGUAL,
    MENOR
};

/* Tipo de Informacion que esta contenida en los Nodos de la
Lista, identificada como DatoSucursal. */
typedef Sucursal DatoSucursal;

```

```

/* Tipo de Estructura de los Nodos de la Lista. */
struct NodoListaSucursal {
    DatoSucursal dato; // dato almacenado
    NodoListaSucursal* sgte; // puntero al siguiente
};

/* Tipo de Puntero a los Nodos de la Lista, el cual se usa para recorrer
la Lista y acceder a sus Datos. */
typedef NodoListaSucursal* PtrNodoListaSucursal;

/* Tipo de Estructura de la Lista */
struct ListaSucursal{
    PtrNodoListaSucursal primeroSucursal; // puntero al primer nodo de la lista
};

/*****
/* Definicion de Primitivas */
/*-----*/

/**
PRE : La lista no debe haber sido creada.
POST: Lista queda creada y preparada para ser usada.
ATRIBUTOS:
listaSucursal: Estructura de datos a ser creado.
*/
void crearListaSucursal(ListaSucursal &listaSucursal);

```

/\*-----\*/

/\*\*

PRE : Lista Creada con crearLista().

POST: Devuelve true si lista esta vacia, sino devuelve false.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

\*/

bool listaVaciaSucursal(ListaSucursal &listaSucursal);

/\*-----\*/

/\*\*

PRE : Lista Creada con crearLista().

POST: Devuelve la representacion de lo Siguiendo al último Nodo de la lista,  
o sea el valor Null, que en esta implementacion representa el final de  
la lista.

RETORNO: Representación del fin de la lista.

\*/

PtrNodoListaSucursal finSucursal();

/\*-----\*/

/\*\*

PRE : Lista Creada con crearLista().

POST: Devuelve el puntero al primer elemento de la lista, o devuelve fin() si  
esta vacia

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

RETORNO: Puntero al primer nodo.

\*/

PtrNodoListaSucursal primeroSucursal(ListaSucursal &listaSucursal);

/\*-----\*/

/\*\*

PRE: Lista Creada con crearLista().

POST: Devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si  
ptrNodo apuntaba a fin() o si lista esta vacia.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

ptrNodo : Puntero al nodo a partir del cual se requiere el siguiente.

RETURN puntero al nodo siguiente.

\*/

PtrNodoListaSucursal siguienteSucursal(ListaSucursal &listaSucursal, PtrNodoListaSucursal  
ptrNodo);

/\*-----\*/

/\*\*

PRE : Lista Creada con crearLista().

ptrNodo es un puntero a un nodo de lista.

POST: Devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si  
ptrNodo apuntaba al primero o si lista esta vacia.

ATRIBUTOS:

listaSucursal: lista sobre la cual se invoca la primitiva.

ptrNodo : Puntero al nodo a partir del cual se requiere el anterior.

RETORNO : Puntero al nodo anterior.

\*/

PtrNodoListaSucursal anteriorSucursal(ListaSucursal &listaSucursal, PtrNodoListaSucursal  
ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si

si lista esta vacia.

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

RETORNO: Puntero al último nodo.

\*/

PtrNodoListaSucursal ultimoSucursal(ListaSucursal &listaSucursal);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo nuevo al principio de la lista con el dato proporcionado  
y devuelve un puntero a ese elemento.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar al principio de la lista.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaSucursal adicionarPrincipioSucursal(ListaSucursal &listaSucursal, DatoSucursal dato);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo despues del apuntado por ptrNodo con el dato  
proporcionado y devuelve un puntero apuntado al elemento insertado.

Si la lista esta vacía agrega un nodo al principio de esta y devuelve  
un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta  
nada y devuelve fin().

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar.



ptrNodo: Puntero al nodo después del cual se quiere adicionar el dato.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaSucursal adicionarDespuesSucursal(ListaSucursal &listaSucursal, DatoSucursal dato, PtrNodoListaSucursal ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo al final de la lista con el dato proporcionado y devuelve un puntero al nodo insertado.

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

dato : Elemento a adicionar al final de la lista.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaSucursal adicionarFinalSucursal(ListaSucursal &listaSucursal, DatoSucursal dato);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo y devuelve un puntero al nodo insertado. Si la lista esta vacia no inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo insertado sera el nuevo primero.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a adicionar.

ptrNodo: Puntero al nodo antes del cual se quiere adicionar el dato.

RETORNO: Puntero al nodo adicionado.

\*/

PtrNodoListaSucursal adicionarAntesSucursal(ListaSucursal &listaSucursal, DatoSucursal dato, PtrNodoListaSucursal ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().

POST: Coloca el dato proporcionado en el nodo apuntado por ptrNodo.

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

dato: Elemento a colocar.

ptrNodo : Puntero al nodo del cual se quiere colocar el dato.

\*/

void colocarDatoSucursal(ListaSucursal &listaSucursal, DatoSucursal &dato, PtrNodoListaSucursal ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().

POST: Devuelve el dato del nodo apuntado por ptrNodo.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento obtenido.

ptrNodo: Puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDatoSucursal(ListaSucursal &listaSucursal, DatoSucursal &dato, PtrNodoListaSucursal ptrNodo);

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Elimina el nodo apuntado por ptrNodo. No realiza accion si la lista esta vacia o si ptrNodo apunta a fin().

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

ptrNodo: Puntero al nodo que se desea eliminar.

\*/

```
void eliminarNodoSucursal(ListaSucursal &listaSucursal, PtrNodoListaSucursal ptrNodo);
```

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Si la lista no esta vacia, elimina su nodo primero, sino no realiza accion alguna.

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

\*/

```
void eliminarNodoPrimeroSucursal(ListaSucursal &listaSucursal);
```

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Si la lista no esta vacia elimina su nodo ultimo, sino no realiza accion.

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

\*/

```
void eliminarNodoUltimoSucursal(ListaSucursal &listaSucursal);
```

/\*-----\*/

/\*\*

PRE: Lista creada con crearLista().

POST: Elimina todos los Nodos de la lista quedando destruida e inhabilitada para su uso.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

\*/

void eliminarListaSucursal(ListaSucursal &listaSucursal);

/\*\*\*\*\*\*

/\*\* Definición de Operaciones Adicionales \*/

/\*-----\*/

/\*\*

PRE: Lista fue creada con crearLista().

POST: Si el dato se encuentra en la lista, devuelve el puntero al primer nodo que lo contiene. Si el dato no se encuentra en la lista devuelve fin().

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a localizar.

RETORNO: Puntero al nodo localizado o fin().

\*/

PtrNodoListaSucursal localizarDatoSucursal(ListaSucursal &listaSucursal ,DatoSucursal dato);

/\*-----\*/

/\*\*

PRE: Lista fue creada con crearLista() y cargada con datos ordenados de menor a mayor respecto del sentido progresivo.

POST: Agrega a la lista el dato manteniendo el orden pero con multiples valores iguales y devuelve un puntero al nodo insertado.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a insertar.

RETORNO: Puntero al nodo insertado.

\*/

PtrNodoListaSucursal insertarDato1(ListaSucursal &listaSucursal, DatoSucursal dato);

/\*-----\*/

/\*\*

PRE: Lista fue creada con crearLista() y cargada con datos ordenados de menor a mayor respecto del sentido progresivo.

POST: Agrega a la lista el dato manteniendo el orden pero con multiples valores iguales y devuelve un puntero al nodo insertado.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a insertar.

RETORNO: Puntero al nodo insertado.

\*/

PtrNodoListaSucursal insertarDato2(ListaSucursal &listaSucursal, DatoSucursal dato);

/\*-----\*/

/\*\*

PRE: la lista fue creada con crearLista().

POST: elimina el dato de la lista, si el mismo se encuentra.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a eliminar.

\*/

void eliminarDato1(ListaSucursal &listaSucursal, DatoSucursal dato);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Elimina el dato de la lista, si el mismo se encuentra.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

dato: Elemento a eliminar.

\*/

void eliminarDato2(ListaSucursal &listaSucursal,DatoSucursal dato);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Reordena la lista por monto de facturacion de mayor a menor.

ATRIBUTOS:

listaSucursal : Lista sobre la cual se invoca la primitiva.

\*/

void reordenarPorMonto(ListaSucursal &listaSucursal);

/\*\*

PRE: La lista fue creada con crearLista().

POST: Reordena la lista por cantidad de articulos vendidos de mayor a menor.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

\*/

void reordenarPorCantArticulos(ListaSucursal &listaSucursal);

/\*-----\*/

/\*\*

PRE: La lista fue creada con crearLista().

POST: Devuelve la cantidad de datos que tiene la lista.

ATRIBUTOS:

listaSucursal: Lista sobre la cual se invoca la primitiva.

\*/

int longitudSucursal(ListaSucursal &listaSucursal);

#endif // LISTASUCURSAL\_H\_INCLUDED

## Provincia.h

---

#ifndef PROVINCIA\_H\_INCLUDED

typedef struct{

int idProvincia;

char nombreProvincia[15];

float montoTotal;

int totalArticulosVendidos;

}Provincia;

#define PROVINCIA\_H\_INCLUDED

/\*\*

DEFINICION DEL TIPO DE DATO PARA MANEJO DE ATRIBUTOS:

idProvincia

nombreProvincia

montoTotal

totalArticulosVendidos

AXIOMAS:

idProvincia > 0

\*/

/\*\*

PRE: La instancia TDA (Provincia) no debe haberse creado pero no debe estar destruida.

POST: Deja la instancia del TDA (Provincia) listo para ser usado.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

void crearProvincia(Provincia &provincia);

/\*\*

PRE: La instancia TDA (Provincia) debe haberse creado pero no debe estar destruida.

POST: Elimina la instancia del TDA y ya no podra ser utilizada.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

void eliminarProvincia(Provincia &provincia);

/\*\*

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST: La provincia queda seteada con el nuevo identificador.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

idProvincia: Valor del identificador a asignar a la provincia.

RETORNO: No aplica.

\*/



```
void setIdProvincia(Provincia &provincia, int idProvincia);
```

```
/**
```

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST: Devuelve el identificador de provincia.

ATRIBUTOS: provincia: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el identificador de provincia.

```
*/
```

```
int getIdProvincia(Provincia &provincia);
```

```
/**
```

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST: La provincia queda seteada con el nombreProvincia.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

nombreProvincia: Nombre de la provincia asignado.

RETORNO: No aplica.

```
*/
```

```
void setNombreProvincia(Provincia &provincia, char* nombreProvincia);
```

```
/**
```

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST: Devuelve el nombre de la provincia.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el nombre de la provincia.

```
*/
```

```
char* getNombreProvincia(Provincia &provincia);
```

```
/**
```

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST:

La provincia queda seteada con el monto total de facturacion.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

montoTotal: monto total de facturacion de la provincia asignado.

RETORNO: No aplica.

\*/

void setMontoTotal(Provincia &provincia, float montoTotal);

/\*\*

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST: Devuelve el monto total de facturacion de la provincia.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el monto total de facturacion de la provincia.

\*/

float getMontoTotal(Provincia &provincia);

/\*\*

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST: La provincia queda seteada con el monto total de articulos vendidos.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

totalArticulosVendidos: monto total de articulos vendidos de la provincia asignado.

RETORNO: No aplica.

\*/

void setTotalArticulosVendidos(Provincia &provincia, int totalArticulosVendidos);

/\*\*

PRE: La instancia TDA(Provincia) debe haberse creado pero no debe estar destruida.

POST: Devuelve el monto total de articulos vendidos de la provincia.

ATRIBUTOS:

provincia: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el monto total de articulos vendidos de la provincia.

\*/

```
float getTotalArticulosVendidos(Provincia &provincia);
```

```
#endif // PROVINCIA_H_INCLUDED
```

## Sucursal.h

---

```
#ifndef SUCURSAL_H_INCLUDED
```

```
typedef struct{  
    int codSucursal;  
    char nombreProvincia[20];  
    int cantArticulosVendidos;  
    float montoFacturacion;  
    int m2;  
    int casaMatriz;  
}Sucursal;
```

```
#define SUCURSAL_H_INCLUDED
```

```
/**
```

PRE: La instancia del TDA (Sucursal) no debe haberse creado (crear) ni destruido (destruir) con anterioridad.

POST: Deja la instancia del TDA (Sucursal) listo para ser usado.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

RETORNO: No aplica.

```
*/
```

```
void crearSucursal(Sucursal &sucursal);
```

```
/**
```

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (destruir).

POST: Destruye la instancia del TDA y ya no podrá ser utilizada.

ATRIBUTOS: Sucursal: instancia sobre la cual se aplica la primitiva.

RETORNO: No aplica.

\*/

```
void eliminarSucursal(Sucursal &sucursal);
```

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: La Sucursal queda seteada con el codigo de sucursal.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

codSucursal: Valor del codigo de sucursal a asignar a la sucursal.

RETORNO: No aplica.

\*/

```
void setCodSucursal(Sucursal &sucursal,int codSucursal);
```

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: Devuelve el codigo de sucursal de la sucursal.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el codigo de la sucursal.

\*/

```
int getCodSucursal(Sucursal &sucursal);
```

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: La Sucursal queda seteada con la provincia.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

nombreProvincia: Nombre de la provincia a asignar en la sucursal.

RETORNO: No aplica.

\*/

```
void setNombreProvincia(Sucursal &sucursal,char* nombreProvincia);
```

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: Devuelve la provincia de la sucursal.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el nombre de la provincia de la sucursal.

\*/

```
char* getNombreProvincia(Sucursal &sucursal);
```

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: La Sucursal queda seteada con la cantidad de articulos vendidos.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

cantArticulosVendidos: Cantidad de articulos vendidos a asignar en la sucursal.

RETORNO: No aplica.

\*/

```
void setCantArticulosVendidos(Sucursal &sucursal,int cantArticulosVendidos);
```

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: Devuelve la cantidad de articulos vendidos de la sucursal.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve la cantidad de articulos vendidos de la sucursal.

\*/

int getCantArticulosVendidos(Sucursal &sucursal);

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: La Sucursal queda seteada con el monto de facturacion.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

montoFacturacion: Monto de facturacion a asignar en la sucursal.

RETORNO: No aplica.

\*/

void setMontoFacturacion(Sucursal &sucursal,float montoFacturacion);

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: Devuelve el monto de facturacion de la sucursal.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve el monto de facturacion de la sucursal.

\*/

float getMontoFacturacion(Sucursal &sucursal);

/\*\*

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: La Sucursal queda seteada con la casa matriz a la que pertenece.

ATRIBUTOS:

Sucursal: instancia sobre la cual se aplica la primitiva.

casaMatriz: Casa matriz a asignar en la sucursal.

RETORNO: No aplica.

\*/

```
void setCasaMatriz(Sucursal &sucursal,int casaMatriz);
```

```
/**
```

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: Devuelve la casa matriz de la sucursal.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve la casa matriz de la sucursal.

```
*/
```

```
int getCasaMatriz(Sucursal &sucursal);
```

```
/**
```

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: La Sucursal queda seteada con los metros cuadrados que posee.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

m2: Metros cuadrados a asignar en la sucursal.

RETORNO: No aplica.

```
*/
```

```
void setM2(Sucursal &sucursal,int cm2);
```

```
/**
```

PRE: La instancia del TDA (Sucursal) debe haberse creado (crear) pero no debe estar destruida (eliminar).

POST: Devuelve la cantidad de m2 de la sucursal.

ATRIBUTOS:

Sucursal: Instancia sobre la cual se aplica la primitiva.

RETORNO: Devuelve la cantidad de m2 de la sucursal.

```
*/
```

```
int getM2(Sucursal &sucursal);
```

```
#endif // SUCURSAL_H_INCLUDED
```

# ARCHIVOS.CPP



# AdmArchivos.cpp

---

```
#include "AdmArchivos.h"
```

```
using namespace std;
```

```
void cargarSucursales(ListaSucursal &listaSucursal,ListaProvincia &listaProvincia,ListaCasaMatriz  
&listaCasaMatriz){
```

```
    int idProvincia = 1;
```

```
    ifstream archivo("Sucursales.txt");
```

```
    if(archivo && archivo.is_open()){
```

```
        string linea;
```

```
        Sucursal sucursal; crearSucursal(sucursal);
```

```
        Provincia provincia; crearProvincia(provincia);
```

```
        CasaMatriz casaMatriz; crearCasaMatriz(casaMatriz);
```

```
        stringstream ss;
```

```
        while(archivo.good()){
```

```
            //Codigo de la Sucursal
```

```
            getline(archivo, linea, '-');
```

```
            ss.str(linea);
```

```
            setCodSucursal(sucursal, atoi(linea.c_str()));
```

```
            ss.clear();
```

```
            //Nombre de la Provincia
```

```
            getline(archivo, linea, '-');
```

```
            strcpy(sucursal.nombreProvincia,linea.c_str());
```

```
            strcpy(provincia.nombreProvincia,linea.c_str());
```

```

ss.clear();

//cantidad de articulos vendidos
getline(archivo, linea, '-');
ss.str(linea);
setCantArticulosVendidos(sucursal, atoi(linea.c_str()));
ss.clear();

//Monto de facturacion
getline(archivo, linea, '-');
ss.str(linea);
setMontoFacturacion(sucursal, atof(linea.c_str()));
ss.clear();

//m2
getline(archivo, linea, '-');
ss.str(linea);
setM2(sucursal, atoi(linea.c_str()));
ss.clear();

//casa matriz
getline(archivo, linea);
ss.str(linea);
if(atoi(linea.c_str())==0){
    setCasaMatriz(sucursal,sucursal.codSucursal);
    setIdCasaMatriz(casaMatriz,sucursal.codSucursal);
}
else{
    setCasaMatriz(sucursal, atoi(linea.c_str()));
    setIdCasaMatriz(casaMatriz,atoi(linea.c_str()));
}

```

```

        ss.clear();
    }

    setIdProvincia(provincia,idProvincia);
    if(listaVaciaProvincia(listaProvincia) == true){
        adicionarFinalProvincia(listaProvincia, provincia);
        idProvincia++;
        adicionarFinalSucursal(listaSucursal,sucursal);
    }else{
        if(localizarDatoProvincia(listaProvincia,provincia) == NULL){
            if(idProvincia < 24){
                adicionarFinalProvincia(listaProvincia, provincia);
                idProvincia++;
                adicionarFinalSucursal(listaSucursal,sucursal);
            }
        }
        else{
            adicionarFinalSucursal(listaSucursal,sucursal);
        }
    }

    if(listaVaciaCasaMatriz(listaCasaMatriz) == true){
        adicionarFinalCasaMatriz(listaCasaMatriz,casaMatriz);
    }else{
        if(localizarDatoCasaMatriz(listaCasaMatriz,casaMatriz) == NULL){
            adicionarFinalCasaMatriz(listaCasaMatriz, casaMatriz);
        }
    }
}

```

```

        archivo.close();
    }

}

void menu(ListaSucursal &listaSucursal, ListaProvincia &listaProvincia, ListaCasaMatriz
&listaCasaMatriz){

    int opcion;

    if( listaVaciaSucursal(listaSucursal) == false ){

        while(opcion!= 7){

            system("cls");

            cout << "-----BIENVENIDOS A LA APLICACION-----" << endl;
            cout << "-----UNIVERSIDAD NACIONAL DE LANUS-----" << endl;
            cout << "-----CATEDRA ALGORITMOS Y ESTRUCTURAS DE DATOS-----" <<
endl;
            cout << "-----TRABAJO PRACTICO CUATRIMESTRAL-----" << endl;
            cout << "-----GRUPO 6 -----" << endl;

            cout << "\n MENU DE REPORTE:\n" << endl;
            cout << "\t1. LISTADO DE SUCURSALES A ADMINISTRAR." << endl;
            cout << "\t2. LISTADO DE SUCURSALES QUE MAS FACTURARON A NIVEL NACIONAL." <<
endl;
            cout << "\t3. LISTADO DE SUCURSALES QUE MAS FACTURARON A NIVEL PROVINCIAL." <<
endl;
            cout << "\t4. LISTADO DE SUCURSALES QUE MAS ARTICULOS VENDIERON A NIVEL
NACIONAL" << endl;

```

```

cout << "\t5. LISTADO DE SUCURSALES QUE MAS ARTICULOS VENDIERON A NIVEL
PROVINCIAL" << endl;

cout << "\t6. LISTADO DE SUCURSALES POR RANKING DE RENDIMIENTO" << endl;

cout << "\t7. SALIR DEL APLICATIVO." << endl;


cout << "\n INGRESE LA OPCION DESEADA: ";
cin >> opcion;


switch(opcion){

    case 1:
        obtenerSucursales(listaSucursal);
        break;
    case 2:
        rankingNacionalPorMontoFacturacion(listaSucursal);
        break;
    case 3:
        rankingProvincialPorMontoFacturacion(listaSucursal,listaProvincia);
        break;
    case 4:
        rankingNacionalPorCantArticulosVendidos(listaSucursal);
        break;
    case 5:
        rankingProvincialPorCantArticulosVendidos(listaSucursal,listaProvincia);
        break;
    case 6:
        rankingRendimiento(listaSucursal,listaCasaMatriz);
        break;
    case 7:
        system("cls");

```

```

        cout << "-----" << endl;
        cout << "-- CATEDRA ALGORITMOS Y ESTRUCTURAS DE DATOS --" << endl;
        cout << "----- TRABAJO PRACTICO CUATRIMESTRAL -----" << endl;
        cout << "-----\n" << endl;
        cout << "MUCHAS GRACIAS POR UTILIZAR LA APLICACION" << endl;
        cout << "----- GRUPO 6 -----" << endl;
        break;
    default:
        system("cls");
        cout << "-----" << endl;
        cout << "-- OPCION INGRESADA INEXISTENTE --" << endl;
        cout << "-----\n" << endl;
        cout << "MENSAJE: LA OPCION INGRESADA ES INEXISTENTE, INTENTE NUEVAMENTE.";
        Sleep(3000);
        break;
    }
}
}
}
}

```

```

void obtenerSucursales(ListaSucursal &listaSucursal){
    PtrNodoListaSucursal ptrNodoLista;
    ptrNodoLista = primeroSucursal(listaSucursal);
    system("cls");

    cout << "\n" << setw(16) << "Codigo de Sucursal" << "\t|" << setw(18) << "Provincia" << "\t|" <<
    setw(25) << "Cantidad de Articulos" << "\t|" << setw(25) << "Monto de Facturacion" <<
    "\t|" << setw(12) << "m^2" << "\t|" << setw(18) << "Casa Matriz" << endl;

    while(ptrNodoLista != NULL)
    {

```

```

        cout << "\n" << setw(20) << ptrNodoLista->dato.codSucursal << "\t|" << setw(18)
        << ptrNodoLista->dato.nombreProvincia << "\t|" << setw(25) << ptrNodoLista-
        >dato.cantArticulosVendidos << "\t|" << setw(25) << fixed << setprecision(2) << ptrNodoLista-
        >dato.montoFacturacion << "\t|" << setw(12) << ptrNodoLista->dato.m2 << "\t|" << setw(12)
        << ptrNodoLista->dato.casaMatriz << endl;

        ptrNodoLista = siguienteSucursal(listaSucursal, ptrNodoLista);
    }

    cout << "\n";

    system("pause");
}

```

```

void rankingNacionalPorMontoFacturacion(ListaSucursal &listaSucursal){
    reordenarPorMonto(listaSucursal);
    obtenerSucursales(listaSucursal);
}

```

```

void rankingProvincialPorMontoFacturacion(ListaSucursal &listaSucursal, ListaProvincia
&listaProvincia){
    system("cls");
    facturacionTotalPorProvincia(listaSucursal, listaProvincia);
    reordenarPorMonto(listaSucursal);
    reordenarPorMontoTotalDeFacturacion(listaProvincia);
    obtenerListaProvinciaConTotalMontoFacturacion(listaSucursal, listaProvincia);
    cout << "\n" << endl;
    system("pause");
}

```

```

void rankingNacionalPorCantArticulosVendidos(ListaSucursal &listaSucursal){

```

```

reordenarPorCantArticulos(listaSucursal);
obtenerSucursales(listaSucursal);
}

```

```

void rankingProvincialPorCantArticulosVendidos(ListaSucursal &listaSucursal,ListaProvincia
&listaProvincia){
    system("cls");
    articulosTotalesVendidosPorProvincia(listaSucursal,listaProvincia);
    reordenarPorCantArticulos(listaSucursal);
    reordenarPorcantArticulosVendidos(listaProvincia);
    obtenerListaProvinciaConTotalArticulosVendidos(listaSucursal,listaProvincia);
    cout<<"\n"<<endl;
    system("pause");
}

```

```

void rankingRendimiento(ListaSucursal &listaSucursal,ListaCasaMatriz &listaCasaMatriz){
    system("cls");
    rendimientoTotalPorCasaMatriz(listaSucursal,listaCasaMatriz);
    reordenarPorMonto(listaSucursal);
    reordenarPorRendimiento(listaCasaMatriz);
    obtenerListaCasaMatriz(listaSucursal,listaCasaMatriz);
    cout<<"\n"<<endl;
    system("pause");
}

```

```

void facturacionTotalPorProvincia(ListaSucursal &listaSucursal, ListaProvincia &listaProvincia){

```



```

float sumaMontoProvincia = 0;
PtrNodoListaProvincia ptrNodoProvincia = primeroProvincia(listaProvincia);

while( (!listaVaciaProvincia(listaProvincia)) && (ptrNodoProvincia != finProvincia()) ){

    PtrNodoListaSucursal ptrNodoSucursal = primeroSucursal(listaSucursal);

    while( (!listaVaciaSucursal(listaSucursal)) && (ptrNodoSucursal != finSucursal()) ){

        if( strcmp(getNombreProvincia(ptrNodoProvincia->dato),
getNombreProvincia(ptrNodoSucursal->dato)) == 0 ){
            sumaMontoProvincia += getMontoFacturacion(ptrNodoSucursal->dato);
        }
        ptrNodoSucursal = ptrNodoSucursal->sgte;
    }
    setMontoTotal(ptrNodoProvincia->dato,sumaMontoProvincia);
    sumaMontoProvincia=0;
    ptrNodoProvincia = ptrNodoProvincia->sgte;
}
}

```

```

void articulosTotalesVendidosPorProvincia(ListaSucursal &listaSucursal, ListaProvincia
&listaProvincia){
    float sumaArticulosProvincia = 0;
    PtrNodoListaProvincia ptrNodoProvincia = primeroProvincia(listaProvincia);

    while( (!listaVaciaProvincia(listaProvincia)) && (ptrNodoProvincia != finProvincia()) ){

```

```

PtrNodoListaSucursal ptrNodoSucursal = primeroSucursal(listaSucursal);

while( (!listaVaciaSucursal(listaSucursal)) && (ptrNodoSucursal != finSucursal()) ){

    if( strcmp(getNombreProvincia(ptrNodoProvincia->dato),
getNombreProvincia(ptrNodoSucursal->dato)) == 0 ){
        sumaArticulosProvincia += getCantArticulosVendidos(ptrNodoSucursal->dato);
    }
    ptrNodoSucursal = ptrNodoSucursal->sgte;
}
setTotalArticulosVendidos(ptrNodoProvincia->dato,sumaArticulosProvincia);
sumaArticulosProvincia=0;
ptrNodoProvincia = ptrNodoProvincia->sgte;
}
}

```

```

void rendimientoTotalPorCasaMatriz(ListaSucursal &listaSucursal, ListaCasaMatriz
&listaCasaMatriz){
    float sumaM2Sucursal = 0;
    float sumaMontoFacturacion=0;
    float rendimientoTotal=0;
    PtrNodoListaCasaMatriz ptrNodoCasaMatriz = primeroCasaMatriz(listaCasaMatriz);

    while( (!listaVaciaCasaMatriz(listaCasaMatriz)) && (ptrNodoCasaMatriz != finCasaMatriz()) ){

        PtrNodoListaSucursal ptrNodoSucursal = primeroSucursal(listaSucursal);

        while( (!listaVaciaSucursal(listaSucursal)) && (ptrNodoSucursal != finSucursal()) ){

```

```

        if(getIdCasaMatriz(ptrNodoCasaMatriz->dato)==getCasaMatriz(ptrNodoSucursal-
>dato)){
            sumaM2Sucursal += getM2(ptrNodoSucursal->dato);
            sumaMontoFacturacion += getMontoFacturacion(ptrNodoSucursal->dato);
        }
        ptrNodoSucursal = ptrNodoSucursal->sgte;
    }
    rendimientoTotal=sumaMontoFacturacion/sumaM2Sucursal;
    setRendimiento(ptrNodoCasaMatriz->dato,rendimientoTotal);
    sumaM2Sucursal=0;
    sumaMontoFacturacion=0;
    rendimientoTotal=0;
    ptrNodoCasaMatriz = ptrNodoCasaMatriz->sgte;
}
}

```

## CasaMatriz.cpp

---

```

#include "CasaMatriz.h"

void crearCasaMatriz(CasaMatriz &casaMatriz){
    casaMatriz.idCasaMatriz=0;
    casaMatriz.rendimiento=0;
}

void eliminarCasaMatriz(CasaMatriz &casaMatriz){
    casaMatriz.idCasaMatriz=0;
    casaMatriz.rendimiento=0;
}

void setIdCasaMatriz(CasaMatriz &casaMatriz, int idCasaMatriz){
    casaMatriz.idCasaMatriz=idCasaMatriz;
}

```

```

int getIdCasaMatriz(CasaMatriz &casaMatriz){
    return casaMatriz.idCasaMatriz;
}

void setRendimiento(CasaMatriz &casaMatriz,float rendimiento){
    casaMatriz.rendimiento=rendimiento;
}

float getRendimiento(CasaMatriz &casaMatriz){
    return casaMatriz.rendimiento;
}

```

## ListaCasaMatriz.cpp

---

```

/* TDA Lista
 * Implementación Simplemente Enlazada
 * Archivo : ListaCasaMatriz.cpp
 * Versión : 1.1
 */

#include "ListaCasaMatriz.h"

using namespace std;

/*****
/* Definición de Tipos de Datos para manejo interno */
/*-----*/

/*****
/* Funciones Adicionales */
/*-----*/

/*
pre : ninguna.
post: compara ambos dato1 y dato2, devuelve
      mayor si dato1 es mayor que dato2,
      igual si dato1 es igual a dato2,
      menor si dato1 es menor que dato2.

```

```

dato1 : dato a comparar.
dato2 : dato a comparar.
return resultado de comparar dato1 respecto de dato2.
*/
ResultadoComparacionCasaMatriz compararDatoCasaMatrizPorId(DatoCasaMatriz dato1,
DatoCasaMatriz dato2) {
    if (dato1.idCasaMatriz> dato2.idCasaMatriz) {
        return CMAYOR;
    }
    else if (dato1.idCasaMatriz < dato2.idCasaMatriz) {
        return CMENOR;
    }
    else {
        return CIGUAL;
    }
}

ResultadoComparacionCasaMatriz compararDatoCasaMatrizPorRendimiento(DatoCasaMatriz
dato1, DatoCasaMatriz dato2) {
    if (dato1.rendimiento> dato2.rendimiento) {
        return CMAYOR;
    }
    else if (dato1.rendimiento < dato2.rendimiento) {
        return CMENOR;
    }
    else {
        return CIGUAL;
    }
}

/*****
/* Implementación de Primitivas */
/*-----*/

void crearListaCasaMatriz(ListaCasaMatriz &listaCasaMatriz){
    listaCasaMatriz.primerCasaMatriz=finCasaMatriz();
}

bool listaVaciaCasaMatriz(ListaCasaMatriz &listaCasaMatriz){
    return (primerCasaMatriz(listaCasaMatriz) == finCasaMatriz());
}

```

```
}
```

```
PtrNodoListaCasaMatriz finCasaMatriz(){  
    return NULL;  
}
```

```
PtrNodoListaCasaMatriz primeroCasaMatriz(ListaCasaMatriz &listaCasaMatriz){  
    return listaCasaMatriz.primerCasaMatriz;  
}
```

```
PtrNodoListaCasaMatriz siguienteCasaMatriz(ListaCasaMatriz &listaCasaMatriz,  
PtrNodoListaCasaMatriz ptrNodo){  
/* verifica si la lista está vacia o si ptrNodo es el último */  
    if ((! listaVaciaCasaMatriz(listaCasaMatriz)) && (ptrNodo != finCasaMatriz()))  
        return ptrNodo->sgte;  
    else  
        return finCasaMatriz();  
}
```

```
PtrNodoListaCasaMatriz anteriorCasaMatriz(ListaCasaMatriz &listaCasaMatriz,  
PtrNodoListaCasaMatriz ptrNodo){  
    PtrNodoListaCasaMatriz ptrPrevio = finCasaMatriz();  
    PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(listaCasaMatriz);  
    while (( ptrCursor != finCasaMatriz()) && (ptrCursor != ptrNodo)) {  
        ptrPrevio = ptrCursor;  
        ptrCursor = siguienteCasaMatriz(listaCasaMatriz,ptrCursor);  
    }  
    return ptrPrevio;  
}
```

```
PtrNodoListaCasaMatriz ultimoCasaMatriz(ListaCasaMatriz &listaCasaMatriz){  
/* el último nodo de la lista es el anterior al fin() */  
    return anteriorCasaMatriz(listaCasaMatriz,finCasaMatriz());  
}
```

```
PtrNodoListaCasaMatriz crearNodoLista(DatoCasaMatriz dato) {  
/* reserva memoria para el nodo y luego completa sus datos */  
    PtrNodoListaCasaMatriz ptrAux = new NodoListaCasaMatriz;  
    ptrAux->dato = dato;  
    ptrAux->sgte = finCasaMatriz();  
}
```

```

    return ptrAux;
}

```

```

PtrNodoListaCasaMatriz adicionarPrincipioCasaMatriz(ListaCasaMatriz &listaCasaMatriz,
DatoCasaMatriz dato){
    /* crea el nodo */
    PtrNodoListaCasaMatriz ptrNuevoNodo = crearNodoLista(dato);
    /* lo incorpora al principio de la lista */
    ptrNuevoNodo->sgte = listaCasaMatriz.primerCasaMatriz;
    listaCasaMatriz.primerCasaMatriz = ptrNuevoNodo;
    return ptrNuevoNodo;
}

```

```

PtrNodoListaCasaMatriz adicionarDespuesCasaMatriz(ListaCasaMatriz &listaCasaMatriz,
DatoCasaMatriz dato, PtrNodoListaCasaMatriz ptrNodo){
    PtrNodoListaCasaMatriz ptrNuevoNodo = finCasaMatriz();
    /* si la lista está vacia se adiciona la principio */
    if (listaVacíaCasaMatriz(listaCasaMatriz))
        ptrNuevoNodo = adicionarPrincipioCasaMatriz(listaCasaMatriz,dato);
    else {
        if (ptrNodo != finCasaMatriz()) {
            /* crea el nodo y lo intercala en la lista */
            ptrNuevoNodo = crearNodoLista(dato);
            ptrNuevoNodo->sgte = ptrNodo->sgte;
            ptrNodo->sgte = ptrNuevoNodo;
        }
    }
    return ptrNuevoNodo;
}

```

```

PtrNodoListaCasaMatriz adicionarFinalCasaMatriz(ListaCasaMatriz &listaCasaMatriz,
DatoCasaMatriz dato){
    /* adiciona el dato después del último nodo de la lista */
    return adicionarDespuesCasaMatriz(listaCasaMatriz,dato,ultimoCasaMatriz(listaCasaMatriz));
}

```

```

PtrNodoListaCasaMatriz adicionarAntesCasaMatriz(ListaCasaMatriz &listaCasaMatriz,
DatoCasaMatriz dato, PtrNodoListaCasaMatriz ptrNodo){
    PtrNodoListaCasaMatriz ptrNuevoNodo = finCasaMatriz();
    if (! listaVacíaCasaMatriz(listaCasaMatriz)) {
        if (ptrNodo != primeroCasaMatriz(listaCasaMatriz))

```

```

        ptrNuevoNodo =
adicionarDespuesCasaMatriz(listaCasaMatriz,dato,anteriorCasaMatriz(listaCasaMatriz,ptrNodo));
        else
            ptrNuevoNodo = adicionarPrincipioCasaMatriz(listaCasaMatriz,dato);
    }
    return ptrNuevoNodo;
}

```

```

void colocarDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz &dato,
PtrNodoListaCasaMatriz ptrNodo){
    if ( (! listaVacíaCasaMatriz(listaCasaMatriz)) && (ptrNodo != finCasaMatriz()))
        ptrNodo->dato = dato;
}

```

```

void obtenerDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz &dato,
PtrNodoListaCasaMatriz ptrNodo){
    if ((! listaVacíaCasaMatriz(listaCasaMatriz)) && (ptrNodo != finCasaMatriz()))
        dato = ptrNodo->dato;
}

```

```

void eliminarNodoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, PtrNodoListaCasaMatriz
ptrNodo){
    PtrNodoListaCasaMatriz ptrPrevio;
    /* verifica que la lista no esté vacía y que nodo no sea fin*/
    if ((! listaVacíaCasaMatriz(listaCasaMatriz)) && (ptrNodo != finCasaMatriz())) {
        if (ptrNodo == primeroCasaMatriz(listaCasaMatriz))
            listaCasaMatriz.primerCasaMatriz=
siguienteCasaMatriz(listaCasaMatriz,primeroCasaMatriz(listaCasaMatriz));

        else {
            ptrPrevio = anteriorCasaMatriz(listaCasaMatriz,ptrNodo);
            ptrPrevio->sgte = ptrNodo->sgte;
        }
        // Si el dato es un TDA, acá habría que llamar al destructor.
        delete ptrNodo;
    }
}

```

```

void eliminarNodoPrimeroCasaMatriz(ListaCasaMatriz &listaCasaMatriz){
    while (! listaVacíaCasaMatriz(listaCasaMatriz))

```



```

    eliminarNodoCasaMatriz(listaCasaMatriz,primeroCasaMatriz(listaCasaMatriz));
}

```

```

void eliminarNodoUltimoCasaMatriz(ListaCasaMatriz &listaCasaMatriz){
    while (! listaVaciaCasaMatriz(listaCasaMatriz))
        eliminarNodoCasaMatriz(listaCasaMatriz,ultimoCasaMatriz(listaCasaMatriz));
}

```

```

void eliminarListaCasaMatriz(ListaCasaMatriz &listaCasaMatriz){
/* retira uno a uno los nodos de la lista */
    while (! listaVaciaCasaMatriz(listaCasaMatriz))
        eliminarNodoCasaMatriz(listaCasaMatriz,primeroCasaMatriz(listaCasaMatriz));
}

```

```

PtrNodoListaCasaMatriz localizarDatoCasaMatriz(ListaCasaMatriz
&listaCasaMatriz,DatoCasaMatriz dato){
    bool encontrado = false;
    DatoCasaMatriz datoCursor;
    PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(listaCasaMatriz);
    /* recorre los nodos hasta llegar al último o hasta
    encontrar el nodo buscado */
    while ((ptrCursor != finCasaMatriz()) && (! encontrado)) {
        /* obtiene el dato del nodo y lo compara */
        obtenerDatoCasaMatriz(listaCasaMatriz,datoCursor,ptrCursor);
        if (compararDatoCasaMatrizPorId(datoCursor,dato) == CIGUAL)
            encontrado = true;
        else
            ptrCursor = siguienteCasaMatriz(listaCasaMatriz,ptrCursor);
    }

    /* si no lo encontró devuelve fin */
    if (! encontrado)
        ptrCursor = finCasaMatriz();

    return ptrCursor;
}

```

```

PtrNodoListaCasaMatriz localizarDatoCasaMatriz2(ListaCasaMatriz
&listaCasaMatriz,DatoCasaMatriz dato){

```

```

bool encontrado = false;
DatoCasaMatriz datoCursor;
PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(listaCasaMatriz);
/* recorre los nodos hasta llegar al último o hasta
encontrar el nodo buscado */
while ((ptrCursor != finCasaMatriz()) && (! encontrado)) {
/* obtiene el dato del nodo y lo compara */
obtenerDatoCasaMatriz(listaCasaMatriz,datoCursor,ptrCursor);
if (compararDatoCasaMatrizPorRendimiento(datoCursor,dato) == CIGUAL)
    encontrado = true;
else
    ptrCursor = siguienteCasaMatriz(listaCasaMatriz,ptrCursor);
}

/* si no lo encontró devuelve fin */
if (! encontrado)
    ptrCursor = finCasaMatriz();

return ptrCursor;
}

```

```

PtrNodoListaCasaMatriz insertarDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz,
DatoCasaMatriz dato){
    PtrNodoListaCasaMatriz ptrPrevio = primeroCasaMatriz(listaCasaMatriz);
    PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(listaCasaMatriz);
    PtrNodoListaCasaMatriz ptrNuevoNodo;
    DatoCasaMatriz datoCursor;
    bool ubicado = false;
/* recorre la lista buscando el lugar de la inserción */
while ((ptrCursor != finCasaMatriz()) && (! ubicado)) {
    obtenerDatoCasaMatriz(listaCasaMatriz,datoCursor,ptrCursor);
    // si cambio menor por mayor ordena de menor a mayor
    if (compararDatoCasaMatrizPorId(datoCursor,dato) == CMENOR)
        ubicado = true;

    else {
        ptrPrevio = ptrCursor;
        ptrCursor = siguienteCasaMatriz(listaCasaMatriz,ptrCursor);
    }
}

if (ptrCursor == primeroCasaMatriz(listaCasaMatriz))

```

```

        ptrNuevoNodo = adicionarPrincipioCasaMatriz(listaCasaMatriz,dato);
    else
        ptrNuevoNodo = adicionarDespuesCasaMatriz(listaCasaMatriz,dato,ptrPrevio);

    return ptrNuevoNodo;
}

```

```

PtrNodoListaCasaMatriz insertarDatoCasaMatriz2(ListaCasaMatriz &listaCasaMatriz,
DatoCasaMatriz dato){
    PtrNodoListaCasaMatriz ptrPrevio = primeroCasaMatriz(listaCasaMatriz);
    PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(listaCasaMatriz);
    PtrNodoListaCasaMatriz ptrNuevoNodo;
    DatoCasaMatriz datoCursor;
    bool ubicado = false;
    /* recorre la lista buscando el lugar de la inserción */
    while ((ptrCursor != finCasaMatriz()) && (! ubicado)) {
        obtenerDatoCasaMatriz(listaCasaMatriz,datoCursor,ptrCursor);
        // si cambio menor por mayor ordena de menor a mayor
        if (compararDatoCasaMatrizPorRendimiento(datoCursor,dato) == CMENOR)
            ubicado = true;

        else {
            ptrPrevio = ptrCursor;
            ptrCursor = siguienteCasaMatriz(listaCasaMatriz,ptrCursor);
        }
    }

    if (ptrCursor == primeroCasaMatriz(listaCasaMatriz))
        ptrNuevoNodo = adicionarPrincipioCasaMatriz(listaCasaMatriz,dato);
    else
        ptrNuevoNodo = adicionarDespuesCasaMatriz(listaCasaMatriz,dato,ptrPrevio);

    return ptrNuevoNodo;
}

```

```

void eliminarDatoCasaMatriz(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato){
    /* localiza el dato y luego lo elimina */
    PtrNodoListaCasaMatriz ptrNodo = localizarDatoCasaMatriz(listaCasaMatriz,dato);
    if (ptrNodo != finCasaMatriz())
        eliminarNodoCasaMatriz(listaCasaMatriz,ptrNodo);
}

```

```

void eliminarDatoCasaMatriz2(ListaCasaMatriz &listaCasaMatriz, DatoCasaMatriz dato){
    /* localiza el dato y luego lo elimina */
    PtrNodoListaCasaMatriz ptrNodo = localizarDatoCasaMatriz2(listaCasaMatriz,dato);
    if (ptrNodo != finCasaMatriz())
        eliminarNodoCasaMatriz(listaCasaMatriz,ptrNodo);
}

```

```

void reordenarPorId(ListaCasaMatriz &listaCasaMatriz){
    ListaCasaMatriz temp = listaCasaMatriz;
    PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(temp);
    crearListaCasaMatriz(listaCasaMatriz);
    while ( ptrCursor != finCasaMatriz() ) {
        DatoCasaMatriz dato;
        obtenerDatoCasaMatriz( temp, dato, ptrCursor);
        insertarDatoCasaMatriz( listaCasaMatriz, dato );
        eliminarNodoCasaMatriz( temp, ptrCursor );
        ptrCursor = primeroCasaMatriz(temp);
    }
    eliminarListaCasaMatriz( temp );
}

```

```

void reordenarPorRendimiento(ListaCasaMatriz &listaCasaMatriz){
    ListaCasaMatriz temp = listaCasaMatriz;
    PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(temp);
    crearListaCasaMatriz(listaCasaMatriz);
    while ( ptrCursor != finCasaMatriz() ) {
        DatoCasaMatriz dato;
        obtenerDatoCasaMatriz( temp, dato, ptrCursor);
        insertarDatoCasaMatriz2( listaCasaMatriz, dato );
        eliminarNodoCasaMatriz( temp, ptrCursor );
        ptrCursor = primeroCasaMatriz(temp);
    }
    eliminarListaCasaMatriz( temp );
}

```

```

int longitud(ListaCasaMatriz &listaCasaMatriz){
    PtrNodoListaCasaMatriz ptrCursor = primeroCasaMatriz(listaCasaMatriz);
    int longitud = 0;
    while ( ptrCursor != finCasaMatriz() ) {

```

```

        longitud++;
        ptrCursor = siguienteCasaMatriz(listaCasaMatriz,ptrCursor);
    }
    return longitud;
}

void obtenerListaCasaMatriz(ListaSucursal &listaSucursal,ListaCasaMatriz &listaCasaMatriz){
    PtrNodoListaCasaMatriz ptrNodoCasaMatriz = primeroCasaMatriz(listaCasaMatriz);
    while( (!listaVaciaCasaMatriz(listaCasaMatriz)) && (ptrNodoCasaMatriz != finCasaMatriz()) ){
        cout << "\n> CASA MATRIZ: " << ptrNodoCasaMatriz->dato.idCasaMatriz << " -
RENDIMIENTO: " << ptrNodoCasaMatriz->dato.rendimiento<<"\n" << endl;
        cout << "\n" << setw(16) << "Codigo de Sucursal" << "\t|" << setw(18) << "Provincia" <<
"\t|" << setw(25) << "Cantidad de Articulos" << "\t|" << setw(25) << fixed << setprecision(2) <<
"Monto de Facturacion" << "\t|" << setw(12) << "m^2" << "\t|" << setw(18) << "Casa Matriz" << endl;
        PtrNodoListaSucursal ptrNodoSucursal = primeroSucursal(listaSucursal);
        while( (!listaVaciaSucursal(listaSucursal)) && (ptrNodoSucursal != finSucursal()) ){
            if( getIdCasaMatriz(ptrNodoCasaMatriz->dato)==getIdCasaMatriz(ptrNodoSucursal-
>dato)){
                cout << "\n" << setw(20) << ptrNodoSucursal->dato.codSucursal << "\t|" << setw(18)
<< ptrNodoSucursal->dato.nombreProvincia << "\t|" << setw(25) << ptrNodoSucursal-
>dato.cantArticulosVendidos << "\t|" << setw(25) << ptrNodoSucursal->dato.montoFacturacion <<
"\t|" << setw(12) << ptrNodoSucursal->dato.m2 << "\t|" << setw(12) << ptrNodoSucursal-
>dato.casaMatriz << endl;
            }
            ptrNodoSucursal = siguienteSucursal(listaSucursal,ptrNodoSucursal);
        }
        ptrNodoCasaMatriz = ptrNodoCasaMatriz->sgte;
    }
}

```

## ListaProvincia.cpp

---

```

/* TDA Lista
* Implementación Simplemente Enlazada
* Archivo : ListaProvincia.cpp
* Versión : 1.1
*/

```

```

#include "ListaProvincia.h"

using namespace std;

/*****
/* Definición de Tipos de Datos para manejo interno */
/*-----*/

/*****
/* Funciones Adicionales */
/*-----*/

/*
pre : ninguna.
post: compara ambos dato1 y dato2, devuelve
      mayor si dato1 es mayor que dato2,
      igual si dato1 es igual a dato2,
      menor si dato1 es menor que dato2.

dato1 : dato a comparar.
dato2 : dato a comparar.
return resultado de comparar dato1 respecto de dato2.
*/
ResultadoComparacionProvincia compararDatoProvincia(DatoProvincia dato1, DatoProvincia
dato2) {
    if (dato1.totalArticulosVendidos > dato2.totalArticulosVendidos) {
        return PMAYOR;
    }
    else if (dato1.totalArticulosVendidos < dato2.totalArticulosVendidos) {
        return PMENOR;
    }
    else {
        return PIGUAL;
    }
}

/*
pre : ninguna.
post: compara ambos dato1 y dato2, devuelve
      mayor si dato1 es mayor que dato2,

```

igual si dato1 es igual a dato2,  
menor si dato1 es menor que dato2.

dato1 : dato a comparar.

dato2 : dato a comparar.

return resultado de comparar dato1 respecto de dato2.

\*/

ResultadoComparacionProvincia compararDatoProvincia2(DatoProvincia dato1, DatoProvincia dato2) {

if (dato1.montoTotal > dato2.montoTotal) {

return PMAYOR;

}

else if (dato1.montoTotal < dato2.montoTotal) {

return PMENOR;

}

else {

return PIGUAL;

}

}

/\*-----\*/

/\* Implementación de Primitivas \*/

/\*-----\*/

void crearListaProvincia(ListaProvincia &listaProvincia) {

listaProvincia.primerProvincia= finProvincia();

}

bool listaVaciaProvincia(ListaProvincia &listaProvincia) {

return (primerProvincia(listaProvincia) == finProvincia());

}

PtrNodoListaProvincia finProvincia() {

return NULL;

}

PtrNodoListaProvincia primerProvincia(ListaProvincia &listaProvincia) {

```

    return listaProvincia.primerProvincia;
}

```

```

PtrNodoListaProvincia siguienteProvincia(ListaProvincia &listaProvincia, PtrNodoListaProvincia
ptrNodo) {
    /* verifica si la lista está vacia o si ptrNodo es el último */
    if ((! listaVaciaProvincia(listaProvincia)) && (ptrNodo != finProvincia()))
        return ptrNodo->sgte;
    else
        return finProvincia();
}

```

```

PtrNodoListaProvincia anteriorProvincia(ListaProvincia &listaProvincia, PtrNodoListaProvincia
ptrNodo) {
    PtrNodoListaProvincia ptrPrevio = finProvincia();
    PtrNodoListaProvincia ptrCursor = primeroProvincia(listaProvincia);
    while (( ptrCursor != finProvincia()) && (ptrCursor != ptrNodo)) {
        ptrPrevio = ptrCursor;
        ptrCursor = siguienteProvincia(listaProvincia,ptrCursor);
    }
    return ptrPrevio;
}

```

```

PtrNodoListaProvincia ultimoProvincia(ListaProvincia &listaProvincia) {
    /* el último nodo de la lista es el anterior al fin() */
    return anteriorProvincia(listaProvincia,finProvincia());
}

```

```

PtrNodoListaProvincia crearNodoLista(DatoProvincia dato) {
    /* reserva memoria para el nodo y luego completa sus datos */
    PtrNodoListaProvincia ptrAux = new NodoListaProvincia;
    ptrAux->dato = dato;
    ptrAux->sgte = finProvincia();
    return ptrAux;
}

```



```

PtrNodoListaProvincia adicionarPrincipioProvincia(ListaProvincia &listaProvincia, DatoProvincia
dato) {
    /* crea el nodo */
    PtrNodoListaProvincia ptrNuevoNodo = crearNodoLista(dato);
    /* lo incorpora al principio de la lista */
    ptrNuevoNodo->sgte = listaProvincia.primerProvincia;
    listaProvincia.primerProvincia = ptrNuevoNodo;
    return ptrNuevoNodo;
}

```

```

PtrNodoListaProvincia adicionarDespuesProvincia(ListaProvincia &listaProvincia, DatoProvincia
dato, PtrNodoListaProvincia ptrNodo) {
    PtrNodoListaProvincia ptrNuevoNodo = finProvincia();
    /* si la lista está vacia se adiciona la principio */
    if (listaVaciaProvincia(listaProvincia))
        ptrNuevoNodo = adicionarPrincipioProvincia(listaProvincia,dato);
    else {
        if (ptrNodo != finProvincia()) {
            /* crea el nodo y lo intercala en la lista */
            ptrNuevoNodo = crearNodoLista(dato);
            ptrNuevoNodo->sgte = ptrNodo->sgte;
            ptrNodo->sgte = ptrNuevoNodo;
        }
    }
    return ptrNuevoNodo;
}

```

```

PtrNodoListaProvincia adicionarFinalProvincia(ListaProvincia &listaProvincia, DatoProvincia dato) {
    /* adiciona el dato después del último nodo de la lista */
    return adicionarDespuesProvincia(listaProvincia,dato,ultimoProvincia(listaProvincia));
}

```

```

PtrNodoListaProvincia adicionarAntesProvincia(ListaProvincia &listaProvincia,DatoProvincia
dato,PtrNodoListaProvincia ptrNodo) {
    PtrNodoListaProvincia ptrNuevoNodo = finProvincia();
    if (! listaVaciaProvincia(listaProvincia)) {
        if (ptrNodo != primerProvincia(listaProvincia))
            ptrNuevoNodo =
adicionarDespuesProvincia(listaProvincia,dato,anteriorProvincia(listaProvincia,ptrNodo));
        else

```

```

        ptrNuevoNodo = adicionarPrincipioProvincia(listaProvincia,dato);
    }
    return ptrNuevoNodo;
}

```

```

void colocarDatoProvincia(ListaProvincia &listaProvincia, DatoProvincia &dato,
PtrNodoListaProvincia ptrNodo) {
    if ( (! listaVaciaProvincia(listaProvincia)) && (ptrNodo != finProvincia()))
        ptrNodo->dato = dato;
}

```

```

void obtenerDatoProvincia(ListaProvincia &listaProvincia,DatoProvincia &dato,
PtrNodoListaProvincia ptrNodo) {
    if ((! listaVaciaProvincia(listaProvincia)) && (ptrNodo != finProvincia()))
        dato = ptrNodo->dato;
}

```

```

void eliminarNodoProvincia(ListaProvincia &listaProvincia, PtrNodoListaProvincia ptrNodo) {
    PtrNodoListaProvincia ptrPrevio;
    /* verifica que la lista no esté vacia y que nodo no sea fin*/
    if ((! listaVaciaProvincia(listaProvincia)) && (ptrNodo != finProvincia())) {
        if (ptrNodo == primeroProvincia(listaProvincia))
            listaProvincia.primerProvincia=
siguienteProvincia(listaProvincia,primeroProvincia(listaProvincia));

        else {
            ptrPrevio = anteriorProvincia(listaProvincia,ptrNodo);
            ptrPrevio->sgte = ptrNodo->sgte;
        }
        // Si el dato es un TDA, acá habría que llamar al destructor.
        delete ptrNodo;
    }
}

```

```

void eliminarNodoPrimeroProvincia(ListaProvincia &listaProvincia) {
    if (! listaVaciaProvincia(listaProvincia))
        eliminarNodoProvincia(listaProvincia,primeroProvincia(listaProvincia));
}

```

```

void eliminarNodoUltimoProvincia(ListaProvincia &listaProvincia) {
    if (! listaVaciaProvincia(listaProvincia))
        eliminarNodoProvincia(listaProvincia,ultimoProvincia(listaProvincia));
}

```

```

void eliminarListaProvincia(ListaProvincia &listaProvincia) {
    /* retira uno a uno los nodos de la lista */
    while (! listaVaciaProvincia(listaProvincia))
        eliminarNodoProvincia(listaProvincia,primeroProvincia(listaProvincia));
}

```

```

PtrNodoListaProvincia localizarDatoProvincia(ListaProvincia &listaProvincia ,DatoProvincia dato){
    bool encontrado = false;
    DatoProvincia datoCursor;
    PtrNodoListaProvincia ptrCursor = primeroProvincia(listaProvincia);

    /* recorre los nodos hasta llegar al último o hasta
    encontrar el nodo buscado */
    while ((ptrCursor != finProvincia()) && (! encontrado)) {
        /* obtiene el dato del nodo y lo compara */
        obtenerDatoProvincia(listaProvincia,datoCursor,ptrCursor);
        if (strcmp(datoCursor.nombreProvincia, dato.nombreProvincia) == 0)
            encontrado = true;
        else
            ptrCursor = siguienteProvincia(listaProvincia,ptrCursor);
    }

    /* si no lo encontró devuelve fin */
    if (! encontrado)
        ptrCursor = finProvincia();

    return ptrCursor;
}

```

```

PtrNodoListaProvincia insertarDatoProvincia(ListaProvincia &listaProvincia, DatoProvincia dato){
    PtrNodoListaProvincia ptrPrevio = primeroProvincia(listaProvincia);
    PtrNodoListaProvincia ptrCursor = primeroProvincia(listaProvincia);

```

```

PtrNodoListaProvincia ptrNuevoNodo;
DatoProvincia datoCursor;
bool ubicado = false;
/* recorre la lista buscando el lugar de la inserción */
while ((ptrCursor != finProvincia()) && (! ubicado)) {
    obtenerDatoProvincia(listaProvincia,datoCursor,ptrCursor);
    // si cambio menor por mayor ordena de menor a mayor
    if (compararDatoProvincia(datoCursor,dato) == PMENOR)
        ubicado = true;

    else {
        ptrPrevio = ptrCursor;
        ptrCursor = siguienteProvincia(listaProvincia,ptrCursor);
    }
}

if (ptrCursor == primeroProvincia(listaProvincia))
    ptrNuevoNodo = adicionarPrincipioProvincia(listaProvincia,dato);
else
    ptrNuevoNodo = adicionarDespuesProvincia(listaProvincia,dato,ptrPrevio);

return ptrNuevoNodo;
}

```

```

PtrNodoListaProvincia insertarDatoProvincia2(ListaProvincia &listaProvincia, DatoProvincia dato){
    PtrNodoListaProvincia ptrPrevio = primeroProvincia(listaProvincia);
    PtrNodoListaProvincia ptrCursor = primeroProvincia(listaProvincia);
    PtrNodoListaProvincia ptrNuevoNodo;
    DatoProvincia datoCursor;
    bool ubicado = false;
    /* recorre la lista buscando el lugar de la inserción */
    while ((ptrCursor != finProvincia()) && (! ubicado)) {
        obtenerDatoProvincia(listaProvincia,datoCursor,ptrCursor);
        // si cambio menor por mayor ordena de menor a mayor
        if (compararDatoProvincia2(datoCursor,dato) == PMENOR)
            ubicado = true;

        else {
            ptrPrevio = ptrCursor;
            ptrCursor = siguienteProvincia(listaProvincia,ptrCursor);
        }
    }
}

```

```

        }
    }

    if (ptrCursor == primeroProvincia(listaProvincia))
        ptrNuevoNodo = adicionarPrincipioProvincia(listaProvincia,dato);
    else
        ptrNuevoNodo = adicionarDespuesProvincia(listaProvincia,dato,ptrPrevio);

    return ptrNuevoNodo;

}

void eliminarDatoProvincia(ListaProvincia &listaProvincia, DatoProvincia dato){
    /* localiza el dato y luego lo elimina */
    PtrNodoListaProvincia ptrNodo = localizarDatoProvincia(listaProvincia,dato);
    if (ptrNodo != finProvincia())
        eliminarNodoProvincia(listaProvincia,ptrNodo);
}

void reordenarPorcantArticulosVendidos(ListaProvincia &listaProvincia){
    ListaProvincia temp = listaProvincia;
    PtrNodoListaProvincia ptrCursor = primeroProvincia(temp);
    crearListaProvincia(listaProvincia);
    while ( ptrCursor != finProvincia() ) {
        DatoProvincia dato;
        obtenerDatoProvincia( temp, dato, ptrCursor);
        insertarDatoProvincia( listaProvincia, dato );
        eliminarNodoProvincia( temp, ptrCursor );
        ptrCursor = primeroProvincia(temp);
    }
    eliminarListaProvincia( temp );
}

void reordenarPorMontoTotalDeFacturacion(ListaProvincia &listaProvincia){
    ListaProvincia temp = listaProvincia;
    PtrNodoListaProvincia ptrCursor = primeroProvincia(temp);
    crearListaProvincia(listaProvincia);
    while ( ptrCursor != finProvincia() ) {
        DatoProvincia dato;

```

```

        obtenerDatoProvincia( temp, dato, ptrCursor);
        insertarDatoProvincia2( listaProvincia, dato );
        eliminarNodoProvincia( temp, ptrCursor );
        ptrCursor = primeroProvincia(temp);
    }
    eliminarListaProvincia( temp );
}

int longitud(ListaProvincia &listaProvincia){
    PtrNodoListaProvincia ptrCursor = primeroProvincia(listaProvincia);
    int longitud = 0;
    while ( ptrCursor != finProvincia() ) {
        longitud++;
        ptrCursor = siguienteProvincia(listaProvincia,ptrCursor);
    }
    return longitud;
}

void obtenerListaProvinciaConTotalMontoFacturacion(ListaSucursal &listaSucursal,Listaprovincia
&listaProvincia){
    PtrNodoListaProvincia ptrNodoProvincia = primeroProvincia(listaProvincia);
    while( (!listaVacíaProvincia(listaProvincia)) && (ptrNodoProvincia != finProvincia()) ){
        cout << "\n> PROVINCIA: " << ptrNodoProvincia->dato.idProvincia << " - NOMBRE
PROVINCIA: " << ptrNodoProvincia->dato.nombreProvincia << " - MONTO TOTAL DE
FACTURACION: " << ptrNodoProvincia->dato.montoTotal<< "\n" << endl;
        cout << "\n" << setw(16) << "Codigo de Sucursal" << "\t|" << setw(18) << "Provincia" <<
"\t|" << setw(25) << "Cantidad de Articulos" << "\t|" << setw(25) << "Monto de Facturacion" <<
"\t|" << setw(12) << "m^2" << "\t|" << setw(18) << "Casa Matriz" << endl;
        PtrNodoListaSucursal ptrNodoSucursal = primeroSucursal(listaSucursal);
        while( (!listaVacíaSucursal(listaSucursal)) && (ptrNodoSucursal != finSucursal()) ){
            if( strcmp(getNombreProvincia(ptrNodoProvincia->dato),
getNombreProvincia(ptrNodoSucursal->dato)) == 0 ){
                cout << "\n" << setw(20) << ptrNodoSucursal->dato.codSucursal << "\t|" << setw(18)
<< ptrNodoSucursal->dato.nombreProvincia << "\t|" << setw(25) << ptrNodoSucursal-
>dato.cantArticulosVendidos << "\t|" << setw(25) << ptrNodoSucursal->dato.montoFacturacion <<
"\t|" << setw(12) << ptrNodoSucursal->dato.m2 << "\t|" << setw(12) << ptrNodoSucursal-
>dato.casaMatriz << endl;
            }
            ptrNodoSucursal = siguienteSucursal(listaSucursal,ptrNodoSucursal);
        }
        cout << "\n" << endl;
    }
}

```

```

        ptrNodoProvincia = ptrNodoProvincia->sgte;
    }
}

void obtenerListaProvinciaConTotalArticulosVendidos(ListaSucursal &listaSucursal, ListaProvincia
&listaProvincia){
    PtrNodoListaProvincia ptrNodoProvincia = primeroProvincia(listaProvincia);
    while( (!listaVaciaProvincia(listaProvincia)) && (ptrNodoProvincia != finProvincia()) ){
        cout << "\n> PROVINCIA: " << ptrNodoProvincia->dato.idProvincia << " - NOMBRE
PROVINCIA: " << ptrNodoProvincia->dato.nombreProvincia << " - TOTAL ARTICULOS VENDIDOS: "
<<ptrNodoProvincia->dato.totalArticulosVendidos<<"\n" << endl;
        cout << "\n" << setw(16) << "Codigo de Sucursal" << "\t|" << setw(18) << "Provincia" <<
"\t|" << setw(25) << "Cantidad de Articulos" << "\t|" << setw(25) << fixed << setprecision(2) <<
"Monto de Facturacion" << "\t|" << setw(12) << "m^2" << "\t|" << setw(18) << "Casa Matriz" << endl;
        PtrNodoListaSucursal ptrNodoSucursal = primeroSucursal(listaSucursal);
        while( (!listaVaciaSucursal(listaSucursal)) && (ptrNodoSucursal != finSucursal()) ){
            if( strcmp(getNombreProvincia(ptrNodoProvincia->dato),
getNombreProvincia(ptrNodoSucursal->dato)) == 0 ){
                cout << "\n" << setw(20) << ptrNodoSucursal->dato.codSucursal << "\t|" << setw(18)
<< ptrNodoSucursal->dato.nombreProvincia << "\t|" << setw(25) << ptrNodoSucursal-
>dato.cantArticulosVendidos << "\t|" << setw(25) << ptrNodoSucursal->dato.montoFacturacion <<
"\t|" << setw(12) << ptrNodoSucursal->dato.m2 << "\t|" << setw(12) << ptrNodoSucursal-
>dato.casaMatriz << endl;
            }
            ptrNodoSucursal = siguienteSucursal(listaSucursal, ptrNodoSucursal);
        }
        cout << "\n" << endl;
        ptrNodoProvincia = ptrNodoProvincia->sgte;
    }
}

```

## ListaSucursal.cpp

---

```

/* TDA Lista
* Implementación Simplemente Enlazada
* Archivo : ListaSucursal.cpp
* Versión : 1.1
*/

```

```

#include "ListaSucursal.h"

/*****
/* Definición de Tipos de Datos para manejo interno */
/*-----*/

/*****
/* Funciones Adicionales */
/*-----*/

/*
pre : ninguna.
post: compara ambos dato1 y dato2, devuelve
      mayor si dato1 es mayor que dato2,
      igual si dato1 es igual a dato2,
      menor si dato1 es menor que dato2.

dato1 : dato a comparar.
dato2 : dato a comparar.
return resultado de comparar dato1 respecto de dato2.
*/
ResultadoComparacionSucursal compararDatoMonto(DatoSucursal dato1, DatoSucursal dato2) {
    if (dato1.montoFacturacion > dato2.montoFacturacion) {
        return MAYOR;
    }
    else if (dato1.montoFacturacion < dato2.montoFacturacion) {
        return MENOR;
    }
    else {
        return IGUAL;
    }
}

ResultadoComparacionSucursal compararDatoCantArticulos(DatoSucursal dato1, DatoSucursal
dato2) {
    if (dato1.cantArticulosVendidos > dato2.cantArticulosVendidos) {
        return MAYOR;
    }
    else if (dato1.cantArticulosVendidos < dato2.cantArticulosVendidos ) {

```



```

        return MENOR;
    }
    else {
        return IGUAL;
    }
}

/*****
/* Implementación de Primitivas */
/*-----*/

void crearListaSucursal(ListaSucursal &listaSucursal) {
    listaSucursal.primeroSucursal = finSucursal();
}

bool listaVaciaSucursal(ListaSucursal &listaSucursal) {
    return (primeroSucursal(listaSucursal) == finSucursal());
}

PtrNodoListaSucursal finSucursal() {
    return NULL;
}

PtrNodoListaSucursal primeroSucursal(ListaSucursal &listaSucursal) {
    return listaSucursal.primeroSucursal;
}

PtrNodoListaSucursal siguienteSucursal(ListaSucursal &listaSucursal, PtrNodoListaSucursal
ptrNodo) {
    /* verifica si la lista está vacia o si ptrNodo es el último */
    if ((! listaVaciaSucursal(listaSucursal)) && (ptrNodo != finSucursal()))
        return ptrNodo->sgte;
    else
        return finSucursal();
}

```

```

PtrNodoListaSucursal anteriorSucursal(ListaSucursal &listaSucursal, PtrNodoListaSucursal ptrNodo)
{
    PtrNodoListaSucursal ptrPrevio = finSucursal();
    PtrNodoListaSucursal ptrCursor = primeroSucursal(listaSucursal);
    while (( ptrCursor != finSucursal()) && (ptrCursor != ptrNodo)) {
        ptrPrevio = ptrCursor;
        ptrCursor = siguienteSucursal(listaSucursal,ptrCursor);
    }
    return ptrPrevio;
}

```

```

PtrNodoListaSucursal ultimoSucursal(ListaSucursal &listaSucursal) {
    /* el último nodo de la lista es el anterior al fin() */
    return anteriorSucursal(listaSucursal,finSucursal());
}

```

```

PtrNodoListaSucursal crearNodoLista(DatoSucursal dato) {
    /* reserva memoria para el nodo y luego completa sus datos */
    PtrNodoListaSucursal ptrAux = new NodoListaSucursal;
    ptrAux->dato = dato;
    ptrAux->sgte = finSucursal();
    return ptrAux;
}

```

```

PtrNodoListaSucursal adicionarPrincipioSucursal(ListaSucursal &listaSucursal, DatoSucursal dato) {
    /* crea el nodo */
    PtrNodoListaSucursal ptrNuevoNodo = crearNodoLista(dato);
    /* lo incorpora al principio de la lista */
    ptrNuevoNodo->sgte = listaSucursal.primerosSucursal;
    listaSucursal.primerosSucursal = ptrNuevoNodo;
    return ptrNuevoNodo;
}

```

```

PtrNodoListaSucursal adicionarDespuesSucursal(ListaSucursal &listaSucursal, DatoSucursal dato,
PtrNodoListaSucursal ptrNodo) {
    PtrNodoListaSucursal ptrNuevoNodo = finSucursal();
    /* si la lista está vacia se adiciona la principio */

```

```

    if (listaVaciaSucursal(listaSucursal))
        ptrNuevoNodo = adicionarPrincipioSucursal(listaSucursal,dato);
    else {
        if (ptrNodo != finSucursal()) {
            /* crea el nodo y lo intercala en la lista */
            ptrNuevoNodo = crearNodoLista(dato);
            ptrNuevoNodo->sgte = ptrNodo->sgte;
            ptrNodo->sgte = ptrNuevoNodo;
        }
    }
    return ptrNuevoNodo;
}

```

```

PtrNodoListaSucursal adicionarFinalSucursal(ListaSucursal &listaSucursal, DatoSucursal dato) {
    /* adiciona el dato después del último nodo de la lista */
    return adicionarDespuesSucursal(listaSucursal,dato,ultimoSucursal(listaSucursal));
}

```

```

PtrNodoListaSucursal adicionarAntesSucursal(ListaSucursal &listaSucursal,DatoSucursal
dato,PtrNodoListaSucursal ptrNodo) {
    PtrNodoListaSucursal ptrNuevoNodo = finSucursal();
    if (! listaVaciaSucursal(listaSucursal)) {
        if (ptrNodo != primeroSucursal(listaSucursal))
            ptrNuevoNodo =
adicionarDespuesSucursal(listaSucursal,dato,anteriorSucursal(listaSucursal,ptrNodo));
        else
            ptrNuevoNodo = adicionarPrincipioSucursal(listaSucursal,dato);
    }
    return ptrNuevoNodo;
}

```

```

void colocarDatoSucursal(ListaSucursal &listaSucursal, DatoSucursal &dato, PtrNodoListaSucursal
ptrNodo) {
    if ( (! listaVaciaSucursal(listaSucursal)) && (ptrNodo != finSucursal()))
        ptrNodo->dato = dato;
}

```

```

void obtenerDatoSucursal(ListaSucursal &listaSucursal, DatoSucursal &dato, PtrNodoListaSucursal
ptrNodo) {
    if ((! listaVaciaSucursal(listaSucursal)) && (ptrNodo != finSucursal()))
        dato = ptrNodo->dato;
}

```

```

void eliminarNodoSucursal(ListaSucursal &listaSucursal, PtrNodoListaSucursal ptrNodo) {
    PtrNodoListaSucursal ptrPrevio;
    /* verifica que la lista no esté vacía y que nodo no sea fin*/
    if ((! listaVaciaSucursal(listaSucursal)) && (ptrNodo != finSucursal())) {
        if (ptrNodo == primeroSucursal(listaSucursal))
            listaSucursal.primeroSucursal =
siguienteSucursal(listaSucursal, primeroSucursal(listaSucursal));

        else {
            ptrPrevio = anteriorSucursal(listaSucursal, ptrNodo);
            ptrPrevio->sgte = ptrNodo->sgte;
        }
        // Si el dato es un TDA, acá habría que llamar al destructor.
        delete ptrNodo;
    }
}

```

```

void eliminarNodoPrimeroSucursal(ListaSucursal &listaSucursal) {
    if (! listaVaciaSucursal(listaSucursal))
        eliminarNodoSucursal(listaSucursal, primeroSucursal(listaSucursal));
}

```

```

void eliminarNodoUltimoSucursal(ListaSucursal &listaSucursal) {
    if (! listaVaciaSucursal(listaSucursal))
        eliminarNodoSucursal(listaSucursal, ultimoSucursal(listaSucursal));
}

```

```

void eliminarListaSucursal(ListaSucursal &listaSucursal) {
    /* retira uno a uno los nodos de la lista */
    while (! listaVaciaSucursal(listaSucursal))
        eliminarNodoSucursal(listaSucursal, primeroSucursal(listaSucursal));
}

```

```
PtrNodoListaSucursal localizarDato1(ListaSucursal &listaSucursal,DatoSucursal dato) {
```

```
    bool encontrado = false;
    DatoSucursal datoCursor;
    PtrNodoListaSucursal ptrCursor = primeroSucursal(listaSucursal);
    /* recorre los nodos hasta llegar al último o hasta
    encontrar el nodo buscado */
    while ((ptrCursor != finSucursal()) && (! encontrado)) {
        /* obtiene el dato del nodo y lo compara */
        obtenerDatoSucursal(listaSucursal,datoCursor,ptrCursor);
        if (compararDatoMonto(datoCursor,dato) == IGUAL)
            encontrado = true;
        else
            ptrCursor = siguienteSucursal(listaSucursal,ptrCursor);
    }

    /* si no lo encontró devuelve fin */
    if (! encontrado)
        ptrCursor = finSucursal();

    return ptrCursor;
}
```

```
PtrNodoListaSucursal localizarDato2(ListaSucursal &listaSucursal,DatoSucursal dato) {
```

```
    bool encontrado = false;
    DatoSucursal datoCursor;
    PtrNodoListaSucursal ptrCursor = primeroSucursal(listaSucursal);

    /* recorre los nodos hasta llegar al último o hasta
    encontrar el nodo buscado */
    while ((ptrCursor != finSucursal()) && (! encontrado)) {
        /* obtiene el dato del nodo y lo compara */
        obtenerDatoSucursal(listaSucursal,datoCursor,ptrCursor);
        if (compararDatoCantArticulos(datoCursor,dato) == IGUAL)
            encontrado = true;
        else
            ptrCursor = siguienteSucursal(listaSucursal,ptrCursor);
    }
```

```

    }

    /* si no lo encontró devuelve fin */
    if (! encontrado)
        ptrCursor = finSucursal();

    return ptrCursor;
}

void eliminarDato1(ListaSucursal &listaSucursal,DatoSucursal dato) {
    /* localiza el dato y luego lo elimina */
    PtrNodoListaSucursal ptrNodo = localizarDato1(listaSucursal,dato);
    if (ptrNodo != finSucursal())
        eliminarNodoSucursal(listaSucursal,ptrNodo);
}

void eliminarDato2(ListaSucursal &listaSucursal,DatoSucursal dato) {
    /* localiza el dato y luego lo elimina */
    PtrNodoListaSucursal ptrNodo = localizarDato2(listaSucursal,dato);
    if (ptrNodo != finSucursal())
        eliminarNodoSucursal(listaSucursal,ptrNodo);
}

PtrNodoListaSucursal insertarDato1(ListaSucursal &listaSucursal,DatoSucursal dato) {
    PtrNodoListaSucursal ptrPrevio = primeroSucursal(listaSucursal);
    PtrNodoListaSucursal ptrCursor = primeroSucursal(listaSucursal);
    PtrNodoListaSucursal ptrNuevoNodo;
    DatoSucursal datoCursor;
    bool ubicado = false;
    /* recorre la lista buscando el lugar de la inserción */
    while ((ptrCursor != finSucursal()) && (! ubicado)) {
        obtenerDatoSucursal(listaSucursal,datoCursor,ptrCursor);
        // si cambio menor por mayor ordena de menor a mayor
        if (compararDatoMonto(datoCursor,dato) == MENOR)
            ubicado = true;
        else {
            ptrPrevio = ptrCursor;
            ptrCursor = siguienteSucursal(listaSucursal,ptrCursor);
        }
    }
}

```

```

    }
    if (ptrCursor == primeroSucursal(listaSucursal))
        ptrNuevoNodo = adicionarPrincipioSucursal(listaSucursal,dato);
    else
        ptrNuevoNodo = adicionarDespuesSucursal(listaSucursal,dato,ptrPrevio);

    return ptrNuevoNodo;
}

```

```

PtrNodoListaSucursal insertarDato2(ListaSucursal &listaSucursal,DatoSucursal dato) {
    PtrNodoListaSucursal ptrPrevio = primeroSucursal(listaSucursal);
    PtrNodoListaSucursal ptrCursor = primeroSucursal(listaSucursal);
    PtrNodoListaSucursal ptrNuevoNodo;
    DatoSucursal datoCursor;
    bool ubicado = false;
    /* recorre la lista buscando el lugar de la inserción */
    while ((ptrCursor != finSucursal()) && (! ubicado)) {
        obtenerDatoSucursal(listaSucursal,datoCursor,ptrCursor);
        // si cambio menor por mayor ordena de menor a mayor
        if (compararDatoCantArticulos(datoCursor,dato) == MENOR)
            ubicado = true;

        else {
            ptrPrevio = ptrCursor;
            ptrCursor = siguienteSucursal(listaSucursal,ptrCursor);
        }
    }

    if (ptrCursor == primeroSucursal(listaSucursal))
        ptrNuevoNodo = adicionarPrincipioSucursal(listaSucursal,dato);
    else
        ptrNuevoNodo = adicionarDespuesSucursal(listaSucursal,dato,ptrPrevio);

    return ptrNuevoNodo;
}

```

```

void reordenarPorMonto(ListaSucursal &listaSucursal) {
    ListaSucursal temp = listaSucursal;
    PtrNodoListaSucursal ptrCursor = primeroSucursal(temp);

```

```

crearListaSucursal(listaSucursal);
while ( ptrCursor != finSucursal() ) {
    DatoSucursal dato;
    obtenerDatoSucursal( temp, dato, ptrCursor);
    insertarDato1( listaSucursal, dato );
    eliminarNodoSucursal( temp, ptrCursor );
    ptrCursor = primeroSucursal(temp);
}
eliminarListaSucursal( temp );
}

void reordenarPorCantArticulos(ListaSucursal &listaSucursal) {
    ListaSucursal temp = listaSucursal;
    PtrNodoListaSucursal ptrCursor = primeroSucursal(temp);
    crearListaSucursal(listaSucursal);
    while ( ptrCursor != finSucursal() ) {
        DatoSucursal dato;
        obtenerDatoSucursal( temp, dato, ptrCursor);
        insertarDato2( listaSucursal, dato );
        eliminarNodoSucursal( temp, ptrCursor );
        ptrCursor = primeroSucursal(temp);
    }
    eliminarListaSucursal( temp );
}

```

```

int longitudSucursal(ListaSucursal &listaSucursal){
    PtrNodoListaSucursal ptrCursor = primeroSucursal(listaSucursal);
    int longitud = 0;
    while ( ptrCursor != finSucursal() ) {
        longitud++;
        ptrCursor = siguienteSucursal(listaSucursal,ptrCursor);
    }
    return longitud;
}

```

## Provincia.cpp

---



```

#include "Provincia.h"
#include <iostream>
#include <string.h>

void crearProvincia(Provincia &provincia){
    provincia.idProvincia=0;
    strcpy(provincia.nombreProvincia,"");
    provincia.montoTotal=0;
    provincia.totalArticulosVendidos=0;
}

void eliminarProvincia(Provincia &provincia){
    provincia.idProvincia=0;
    strcpy(provincia.nombreProvincia,"");
    provincia.montoTotal=0;
    provincia.totalArticulosVendidos=0;
}

void setIdProvincia(Provincia &provincia, int idProvincia){
    provincia.idProvincia=idProvincia;
}

int getIdProvincia(Provincia &provincia){
    return provincia.idProvincia;
}

void setNombreProvincia(Provincia &provincia, char* nombreProvincia){
    strcpy(provincia.nombreProvincia,nombreProvincia);
}

char* getNombreProvincia(Provincia &provincia){
    return provincia.nombreProvincia;
}

void setMontoTotal(Provincia &provincia,float montoTotal){
    provincia.montoTotal=montoTotal;
}

float getMontoTotal(Provincia &provincia){
    return provincia.montoTotal;
}

```

```
void setTotalArticulosVendidos(Provincia &provincia, int totalArticulosVendidos){
    provincia.totalArticulosVendidos=totalArticulosVendidos;
}
```

```
float getTotalArticulosVendidos(Provincia &provincia){
    return provincia.totalArticulosVendidos;
}
```

## Sucursal.cpp

---

```
#include "Sucursal.h"
#include <cctype>
#include <string.h>
```

```
void crearSucursal(Sucursal &sucursal){
    sucursal.codSucursal=0;
    strncpy(sucursal.nombreProvincia, "", 15);
    sucursal.cantArticulosVendidos=0;
    sucursal.montoFacturacion=0;
    sucursal.m2=0;
    sucursal.casaMatriz=0;
}
```

```
void eliminarSucursal(Sucursal &sucursal){
    sucursal.codSucursal=0;
    strncpy(sucursal.nombreProvincia, "", 15);
    sucursal.cantArticulosVendidos=0;
    sucursal.montoFacturacion=0;
    sucursal.m2=0;
    sucursal.casaMatriz=0;
}
```

```
void setCodSucursal(Sucursal &sucursal,int codSucursal){
    sucursal.codSucursal=codSucursal;
}
```

```
int getCodSucursal(Sucursal &sucursal){
```

```

        return sucursal.codSucursal;
    }

    void setNombreProvincia(Sucursal &sucursal,char* nombreProvincia){
        strcpy(sucursal.nombreProvincia,nombreProvincia);
    }

    char* getNombreProvincia(Sucursal &sucursal){
        return sucursal.nombreProvincia;
    }

    void setCantArticulosVendidos(Sucursal &sucursal,int cantArticulosVendidos){
        sucursal.cantArticulosVendidos=cantArticulosVendidos;
    }

    int getCantArticulosVendidos(Sucursal &sucursal){
        return sucursal.cantArticulosVendidos;
    }

    void setMontoFacturacion(Sucursal &sucursal,float montoFacturacion){
        sucursal.montoFacturacion=montoFacturacion;
    }

    float getMontoFacturacion(Sucursal &sucursal){
        return sucursal.montoFacturacion;
    }

    void setCasaMatriz(Sucursal &sucursal,int casaMatriz){
        sucursal.casaMatriz=casaMatriz;
    }

    int getCasaMatriz(Sucursal &sucursal){
        return sucursal.casaMatriz;
    }

    void setM2(Sucursal &sucursal,int m2){
        sucursal.m2=m2;
    }

    int getM2(Sucursal &sucursal){
        return sucursal.m2;
    }

```