|  |  |
| --- | --- |
| Nombre: | **Emanuel Nigro, Rolando A. Rapali** |
| DNI: | 35375416, 38355972 |
| Mail: | [**emanuel.nigro@gmail.com**](mailto:emanuel.nigro@gmail.com)**,** [**rolandoa.rapali@gmail.com**](mailto:rolandoa.rapali@gmail.com) |



ALGORITMOS Y ESTRUCTURAS DE DATOS

Trabajo Práctico de Diagnóstico

Universidad Nacional de Lanús, 29/10/2020

**Enunciado 3**

**Desarrollo de la solucion 3**

**Manual de usuario 3**

**TDA.H 10**

**TDA.CPP 17**

**ListaSucursales.h 22**

**ListaSucursales.cpp 28**

**Main.cpp 38**

**Generalidades.h 44**

**Generalidades.cpp 48**

**Enunciado:**

Se pide elaborar:

● Un listado con el orden de las sucursales que más facturaron a nivel nacional y en

cada provincia, (incluir los totales por provincia).

● Un ranking con las sucursales que mayor cantidad de artículos vendieron a nivel

nacional y por provincia.

● Un ranking de rendimiento donde se busca las sucursales que mejores rendimientos

por metro cuadrado tienen, ordenando por el cociente de facturación/metros

cuadrados del local.

**Desarrollo de la solución:**

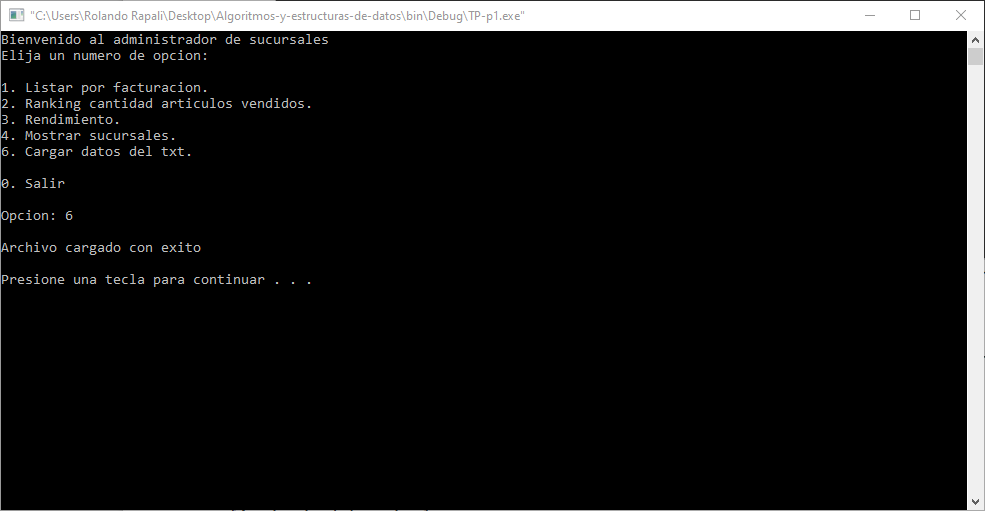
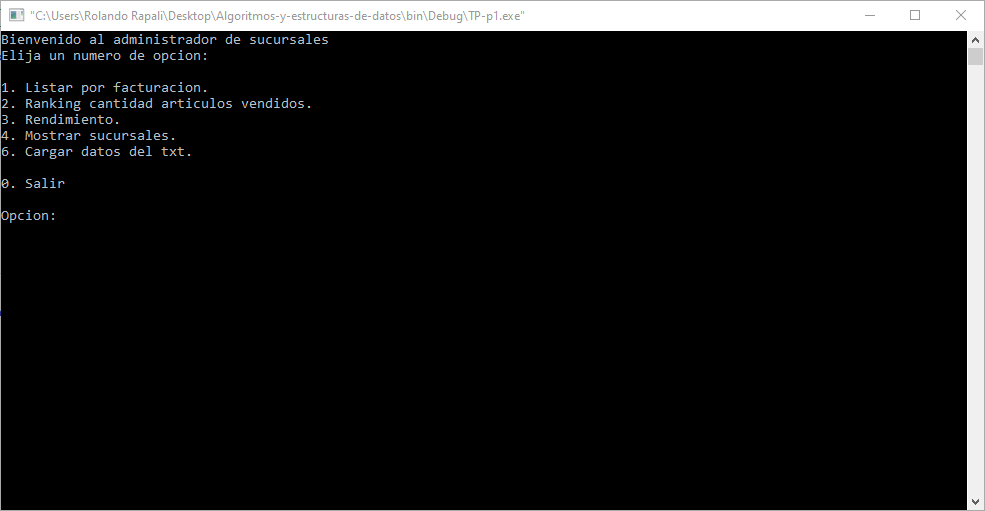
Para la solución del problema se encaro hacer una lista y cargarlo con los datos del txt. Luego esta lista con un tamaño delimitado se pasa a ordenar con los distintos métodos desarrollados y se separan en: Nacional, Provincial, y ,cuando es por rendimiento, según su casa matriz. Las ventajas de utilizar una lista es el dinamismo que presta en el ingreso de los datos.   
**Emanuel Nigro** se encargo de ordenar y mostrar los distintos rankings(Nacionales,Provinciales y por Rendimiento). **Rolando A. Rapali** se encargo de desarrollar lo correspondiente con la lista y la carga de datos a traves de un txt.

**Manual de usuario:**

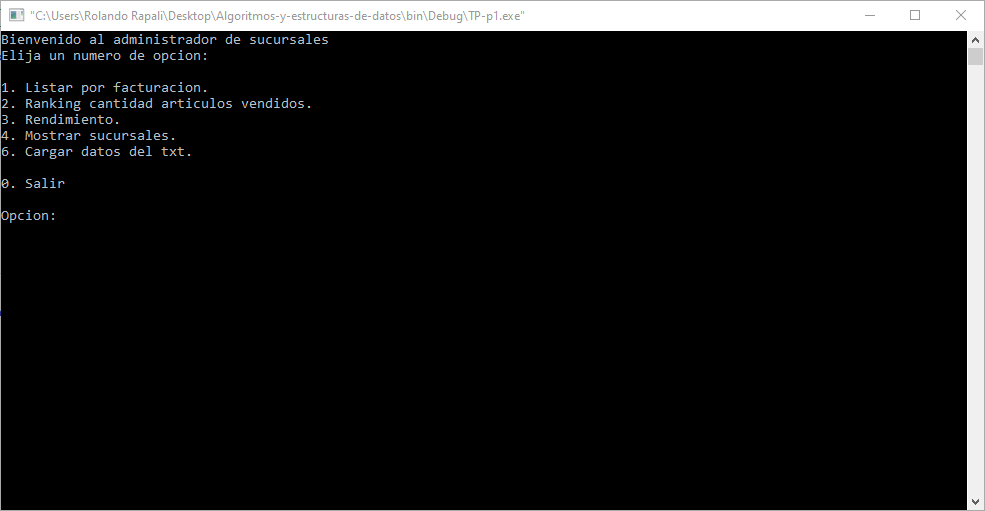
El proyecto permitira al usuario cargar un txt y ver el balance de las distintas sucursales separandolas por: nacionales, provinciales y por rendimiento.

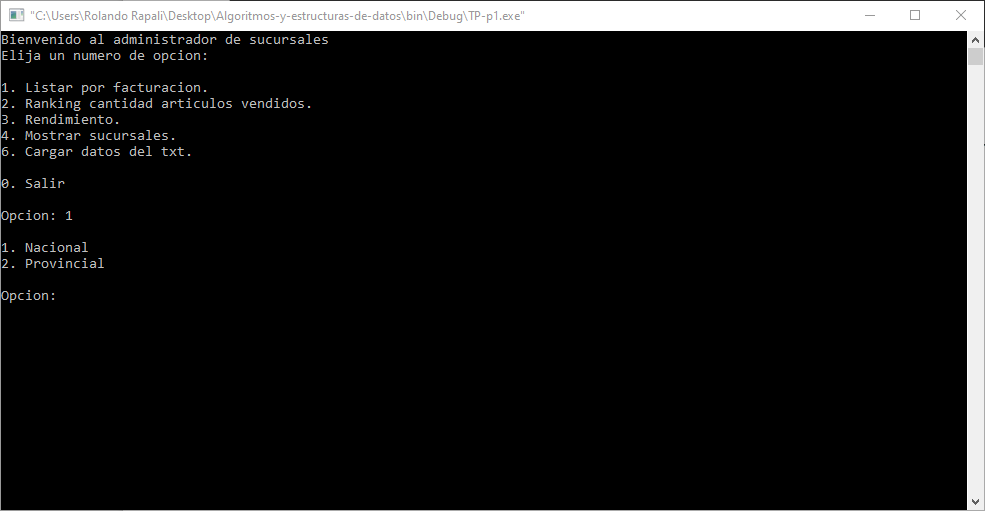
Para el correcto funcionamiento del programa:

1° Se tiene que cargar con el txt que se llame “ejemplo-sucursales.txt” con la opcion “Cargar datos del txt”:

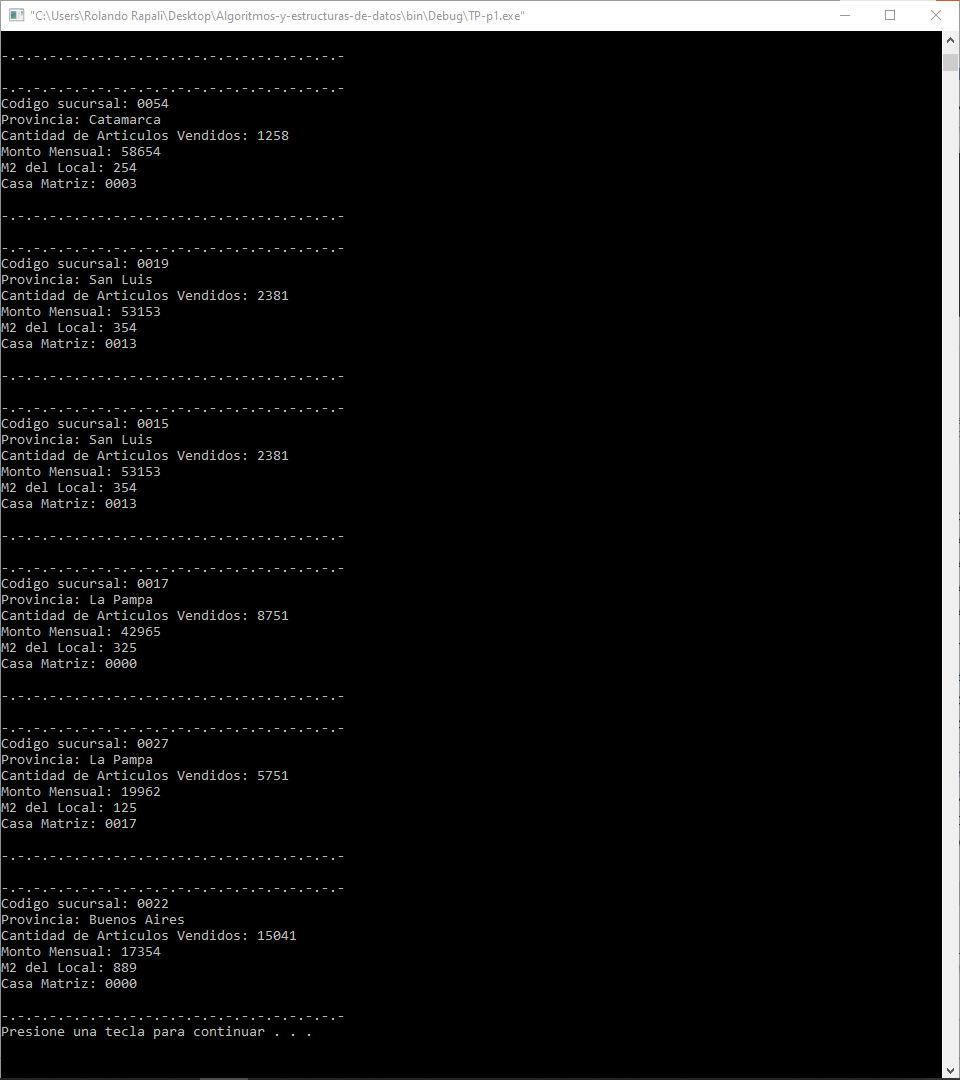


2°Luego se puede proceder a hacer lo que se desee pero si vamos en orden primero pasamos a mostrar el Listado por facturacion:

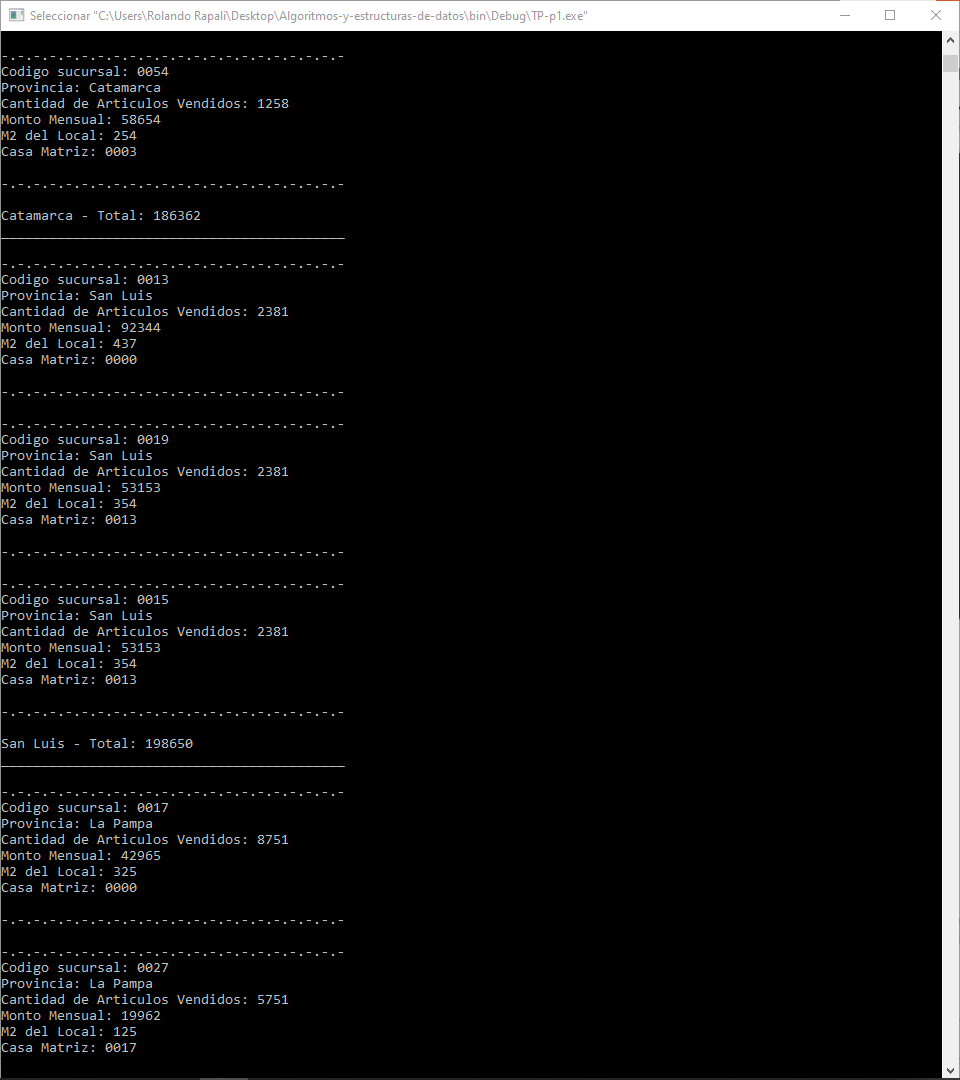


3° Luego de seleccionar el “Listado por facturación” se nos da a elegir entre “Nacional” y “Provincial”:

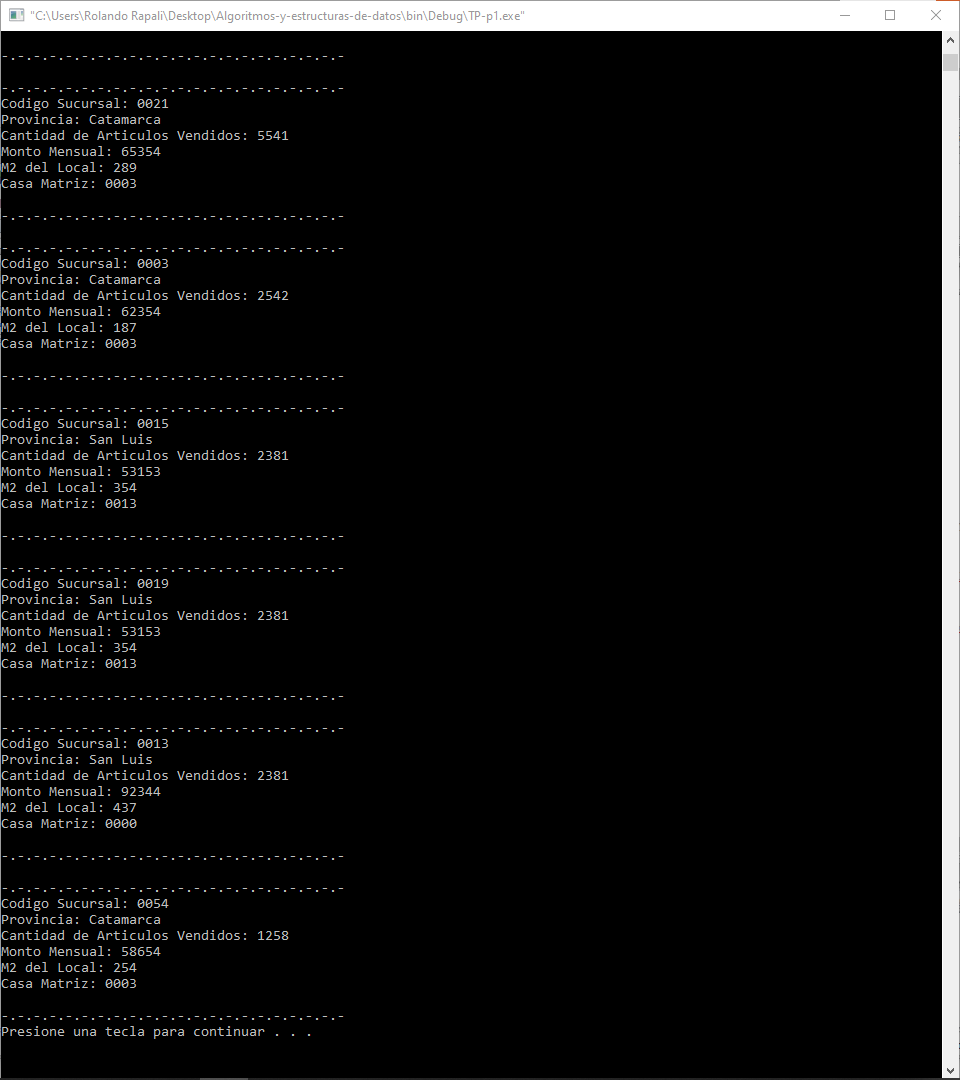
4-A° Elegimos por **Nacional** el cual nos mostrara un **ranking de las sucursales** que tienen el mayor **monto mensual**:



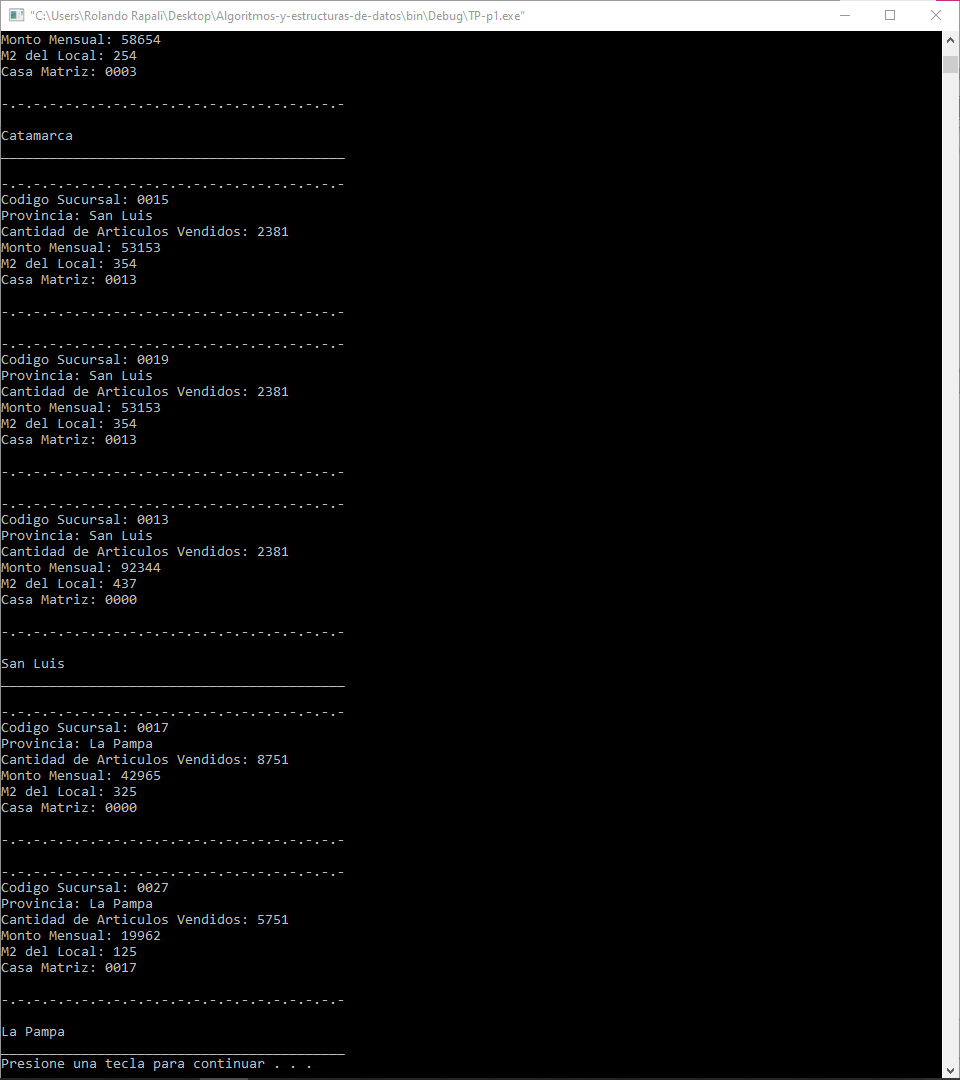
4-B° Elegimos por **Provincial** el cual nos mostrara un **ranking de las sucursales** que tienen el mayor **monto mensual**:

****

5-A° Elegimos por **Nacional** el cual nos mostrara un **ranking de las sucursales** que tienen el mayor **cantidad de artículos vendidos**:

****

5-B° Elegimos por **Provincial** el cual nos mostrara un **ranking de las sucursales** que tienen el mayor **cantidad de artículos vendidos**:



**TDA.h**

#ifndef TDA\_H\_INCLUDED

#define TDA\_H\_INCLUDED

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/\*\*

Definicion del Tipo de Dato para manejo de las Sucursales

Atributos:

\* codigo,

\* provincia,

\* cantidad de articulos,

\* monto mensual,

\* metros cuadrados,

\* casa matriz

Axiomas:

\* codigo: tiene que poseer 4 chars

\* provincia: tiene que ingresarse una provincia valida

\* cantidad\_articulos > 0

\* monto\_mensual > 0

\* m\_cuadrados > 0

\* casa\_matriz: tiene que poseer 4 chars

\*/

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

/\*\* Definiciones de Tipos de Datos \*/

/\*\*-----------------------------------\*/

/\*\* Tipo de Estructura de la Sucursal \*/

struct EstructuraSucursales

{

char codigo[5];

char provincia[20];

int cantidad\_articulos;

float monto\_mensual;

float m\_cuadrados;

char casa\_matriz[5];

EstructuraSucursales(char codigoInput[5],char provinciaInput[20],int articulosInput, float montoMensualInput, float mCuadradosInput, char casa\_matrizInput[5]){

strcpy(codigo,codigoInput);

strcpy(provincia,provinciaInput);

cantidad\_articulos = articulosInput;

monto\_mensual = montoMensualInput;

m\_cuadrados = mCuadradosInput;

strcpy(casa\_matriz,casa\_matrizInput);

}

};

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

/\*\* Definicion de Primitivas \*/

/\*\*-----------------------------------\*/

/\*\*

PRE : La sucursal no debe haber sido creada.

POST: La sucursal queda creada y lista para ser usada.

todos los parametros son ingresados por el usuario.

\*/

void nuevoSucursal(EstructuraSucursales \*sucursal,char codigo[5], char provincia[20],int articulos, float montoMensual, float mCuadrados, char casaMatriz[5]);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: La Sucursal es eliminada.

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void destruirSucursal(EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: Devuelve el dato contenido en el campo codigo.

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

char \*getCodigo (EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

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

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void setCodigo (EstructuraSucursales\* sucursal, char NewCodigo[5]);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: Devuelve el dato contenido en el campo provincia.

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

char \*getProvincia(EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

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

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void setProvincia(EstructuraSucursales\* sucursal,char NewProvincia[20]);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: Devuelve el dato contenido en el campo CantidadArticulos.

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

int getCantidadArticulos(EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

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

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void setCantidadArticulos(EstructuraSucursales\* sucursal, int NewCantidad);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: Devuelve el dato contenido en el campo MontoMensual.

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

float getMontoMensual(EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

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

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void setMontoMensual (EstructuraSucursales\* sucursal, float NewMontoMensual);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: Devuelve el dato contenido en el campo MCuadrados.

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

float getMCuadrados(EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

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

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void setMCuadrados (EstructuraSucursales\* sucursal,float NewMetros);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: Devuelve el dato contenido en el campo CasaMatriz.

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

char \*getCasaMatriz(EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

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

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void setCasaMatriz(EstructuraSucursales\* sucursal, char NewCasaMatriz[5]);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*

PRE : Sucursal creada con EstructuraSucursal().

POST: Se pasa a mostrar todos los campos que contiene la Sucursal

Sucursal : Instancia sobre la cual se invoca la primitiva

\*/

void mostrarSucursal (EstructuraSucursales\* sucursal);

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

#endif // TDA\_H\_INCLUDED

**TDA.CPP**

#include <iostream>

#include <conio.h>

#include <string.h>

#include "TDA.h"

using namespace std;

struct EstructuraSucursales;

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

/\*\* Implementacion de Primitivas \*/

/\*\*-----------------------------------\*/

void nuevoSucursal(EstructuraSucursales \*sucursal,char codigo[5], char provincia[20],int articulos, float montoMensual, float mCuadrados, char casaMatriz[5])

{

strcpy(sucursal->codigo,codigo);

strcpy(sucursal->provincia,provincia);

sucursal->cantidad\_articulos = articulos;

sucursal->monto\_mensual = montoMensual;

sucursal->m\_cuadrados = mCuadrados;

strcpy(sucursal->casa\_matriz,casaMatriz);

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void destruirSucursal(EstructuraSucursales\* sucursal)

{

delete sucursal;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

char \*getCodigo (EstructuraSucursales\* sucursal)

{

return sucursal->codigo;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void setCodigo (EstructuraSucursales\* sucursal, char NewCodigo[5])

{

strcpy(sucursal->codigo,NewCodigo);

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

char \*getProvincia (EstructuraSucursales\* sucursal)

{

return sucursal->provincia;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void setProvincia(EstructuraSucursales\* sucursal,char NewProvincia[20])

{

strcpy(sucursal->provincia,NewProvincia);

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

int getCantidadArticulos(EstructuraSucursales\* sucursal)

{

return sucursal->cantidad\_articulos;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void setCantidadArticulos(EstructuraSucursales\* sucursal, int NewCantidad)

{

sucursal->cantidad\_articulos=NewCantidad;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

float getMontoMensual(EstructuraSucursales\* sucursal)

{

return sucursal->monto\_mensual;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void setMontoMensual (EstructuraSucursales\* sucursal, float NewMontoMensual)

{

sucursal->monto\_mensual=NewMontoMensual;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

float getMCuadrados(EstructuraSucursales\* sucursal)

{

return sucursal->m\_cuadrados;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void setMCuadrados (EstructuraSucursales\* sucursal,float NewMetros)

{

sucursal->m\_cuadrados=NewMetros;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

char \*getCasaMatriz(EstructuraSucursales\* sucursal)

{

return sucursal->casa\_matriz;

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void setCasaMatriz(EstructuraSucursales\* sucursal, char NewCasaMatriz[5])

{

strcpy(sucursal->casa\_matriz,NewCasaMatriz);

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

void mostrarSucursal (EstructuraSucursales\* sucursal)

{

cout<<"\n-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-\n";

cout<<"Codigo sucursal: "<<sucursal->codigo<<endl;

cout<<"Provincia: "<<sucursal->provincia<<endl;

cout<<"Cantidad de Articulos Vendidos: "<<sucursal->cantidad\_articulos<<endl;

cout<<"Monto Mensual: "<<sucursal->monto\_mensual<<endl;

cout<<"M2 del Local: "<<sucursal->m\_cuadrados<<endl;

cout<<"Casa Matriz: "<<sucursal->casa\_matriz<<endl;

cout<<"\n-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-\n";

}

/\*---------------------------------------------------------------------------------------------------------------------------------------------------------\*/

**ListasSucursales.h**

#ifndef LISTAENSUCURSALES\_H\_INCLUDED

#define LISTAENSUCURSALES\_H\_INCLUDED

#include "TDA.h"

/\*\*

Definicion del Tipo de Dato para manejo de Listas Sucursales:

Atributos:

\* EstructuraSucursales \*sucursales,

\* ListaSucursales \*siguiente.

Axiomas:

\* Sucursal != NULL

\* siguiente = NULL

\*/

/\*\*

Definicion del Tipo de Dato para manejo de Listas Sucursales:

Atributos:

\* ListaSucursales \*inicio

\* tamanio

Axiomas:

\* ListaSucursales \*inicio != NULL

\* tamanio > -1

\*/

/\*\* Tipo de Estructura Lista de Sucursales \*/

struct ListaSucursales

{

struct EstructuraSucursales \* sucursales;

ListaSucursales \* siguiente;

};

/\*\* Tipo de Estructura Lista Encadenada \*/

struct Listaenc{

ListaSucursales \* inicio;

int tam;

};

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

/\*\* Definicion de Primitivas \*/

/\*\*-----------------------------------\*/

/\*\*

PRE : La lista no debe haber sido creada.

POST: La lista queda creada y lista para ser usada.

lista: Instancia sobre la cual se invoca la primitiva

\*/

Listaenc\* crearLista();

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tiene que tener una lista inicializada con crearLista()

POST: La lista es vaciada si posee contenidos

lista: Instancia sobre la cual se invoca la primitiva

\*/

void liberarLista(Listaenc\* lista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tiene que tener una lista inicializada con crearLista() y datos cargados

POST: Da verdadero o falso dependiendo de la inicializacion de esta

lista: Instancia sobre la cual se invoca la primitiva

\*/

bool estaInicializado(Listaenc\* lista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tiene que tener una lista inicializada con crearLista() y datos cargados

POST: Da verdadero o falso dependiendo de la inicializacion de esta

sucursal: Instancia sobre la cual se invoca la primitiva

\*/

bool estaInicializado(ListaSucursales\* sucursal);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tiene que tener una lista inicializada con crearLista()

POST: Da verdadero o falso dependiendo de si la lista posee elementos o no

lista: Instancia sobre la cual se invoca la primitiva

\*/

bool estaVacia(Listaenc\* lista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tiene que tener una lista inicializada con crearLista()

POST: Devuelve una Nueva Sucursal del tipo ListaSucursal

sucursal: Instancia sobre la cual se invoca la primitiva

proximo: Puntero que apunta hacia la proxima locacion la cual seria nula

\*/

ListaSucursales\* crearSucur(EstructuraSucursales \* sucursal,ListaSucursales\* proximo);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tiene que tener una lista inicializada con crearLista()

POST: Devuelve una lista con un elemento insertado en el principio.

lista: Instancia sobre la cual se invoca la primitiva

proximo: Puntero que apunta hacia la proxima locacion la cual seria nula

\*/

void insertarInicio(Listaenc\* lista,EstructuraSucursales \* sucursal);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tiene que tener una lista inicializada con crearLista() y una posicion elegida

POST: Devuelve una lista con un elemento insertado en la posicion deseada

sucursal: Instancia sobre la cual se invoca la primitiva

nacional: Instancia sobre la cual se invoca la primitiva

\*/

void insertar(Listaenc\* nacional,EstructuraSucursales \* sucursal, int pos);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Debe poseer una Lista creada con crearLista() y datos ingresados con insertar()

POST: Devuelve la lista con el elemento en la posicion inicial eliminado

lista: Instancia sobre la cual se invoca la primitiva

sucursal: Instancia sobre la cual se invoca la primitiva

\*/

void removerInicio(Listaenc\* lista, EstructuraSucursales \* sucursal);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Debe poseer una Lista creada con crearLista() y datos ingresados con insertar()

POST: Devuelve la lista con el elemento en aquella posicion eliminado

lista: Instancia sobre la cual se invoca la primitiva

sucursal: Instancia sobre la cual se invoca la primitiva

\*/

void remover(Listaenc\* lista, EstructuraSucursales \* sucursal, int pos);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Debe poseer una Lista creada con crearLista() y datos ingresados

POST: Devuelve el elemento que se encuentra en esa posicion

lista: Instancia sobre la cual se invoca la primitiva

\*/

void obtenerElemento(Listaenc\* lista, EstructuraSucursales\*\* sucursal, int pos);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : Tener una lista creada con crearLista();

POST: Devuelve el dato del campo tamanio de la lista

lista: Instancia sobre la cual se invoca la primitiva

\*/

void obtenerTamanio(Listaenc\* lista, int\* tam);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : La Lista creada con crearLista() y con elementos insertados con insertar()

POST: Se pasa a mostrar todos los nodos que contiene la Lista

lista: Instancia sobre la cual se invoca la primitiva

\*/

void imprimir(Listaenc\* lista);

/\*------------------------------------------------------------------------------------\*/

#endif // LISTAENSUCURSALES\_H\_INCLUDED

**ListasSucursales.cpp**

#include "ListaSucursales.h"

#include "TDA.h"

#include <iostream>

#include <stdlib.h>

using namespace std;

Listaenc\* crearLista()

{

Listaenc \* lista = new Listaenc();

if(lista == NULL)

return NULL;

lista->inicio = NULL;

lista->tam = 0;

return lista;

}

/\*------------------------------------------------------------------------------------\*/

void liberarLista(Listaenc\* lista)

{

if(estaInicializado(lista))

{

while(!estaVacia(lista))

remover(lista, NULL, 0);

delete lista;

lista = NULL;

}

else

{

cout<<"no se encuentra inicializada la lista(Liberarlista)"<<endl;

}

}

/\*------------------------------------------------------------------------------------\*/

bool estaInicializado(Listaenc\* lista)

{

bool Inicializado = true;

if(lista == NULL)

{

Inicializado = false;

}

return Inicializado;

}

/\*------------------------------------------------------------------------------------\*/

bool estaInicializado(ListaSucursales\* sucursal)

{

bool Inicializado = true;

if(sucursal == NULL)

{

Inicializado = false;

}

return Inicializado;

}

/\*------------------------------------------------------------------------------------\*/

bool estaVacia(Listaenc\* lista)

{

bool estaVacia = true;

if(estaInicializado(lista))

{

if (!lista->inicio == NULL)

estaVacia = false;

return estaVacia;

}

estaVacia = false;

return estaVacia;

}

/\*------------------------------------------------------------------------------------\*/

ListaSucursales\* crearSucur(EstructuraSucursales \* sucursal,ListaSucursales\* proximo)

{

ListaSucursales\* nueva\_Sucursal = new ListaSucursales;

if (nueva\_Sucursal == NULL)

return NULL;

nueva\_Sucursal->sucursales = sucursal;

nueva\_Sucursal->siguiente = proximo;

return nueva\_Sucursal;

}

/\*------------------------------------------------------------------------------------\*/

void insertarInicio(Listaenc\* lista,EstructuraSucursales \* sucursal)

{

if(estaInicializado(lista))

{

ListaSucursales \*nuevoNodo = crearSucur(sucursal,lista->inicio);

if(estaInicializado(nuevoNodo))

{

lista->inicio = nuevoNodo;

lista->tam++;

}

else

{

cout<<"El nodo es nulo"<<endl;

}

}

else

{

cout<<"no se encuentra inicializada la lista(InsertarInicio)"<<endl;

}

}

/\*------------------------------------------------------------------------------------\*/

void insertar(Listaenc\* nacional,EstructuraSucursales \* sucursal, int pos)

{

if((estaInicializado(nacional)) && (!(pos < 0 || pos > nacional->tam)))

{

if (pos == 0)

{

insertarInicio(nacional, sucursal);

}

else

{

ListaSucursales \*nuevoNodo;

ListaSucursales \*aux;

aux = nacional->inicio;

for(int i = 0; i < pos-1; i++)

{

aux = aux->siguiente;

}

nuevoNodo = crearSucur(sucursal, aux->siguiente);

if (estaInicializado(nuevoNodo))

{

aux->siguiente = nuevoNodo;

nacional->tam++;

}

}

}

else

{

cout<<"no se encuentra inicializada la lista(Insertar)"<<endl;

}

}

/\*------------------------------------------------------------------------------------\*/

void removerInicio(Listaenc\* lista, EstructuraSucursales \* sucursal)

{

if(estaInicializado(lista) && !(estaVacia(lista)))

{

ListaSucursales \*aux = lista->inicio;

if (sucursal != NULL)

sucursal = aux->sucursales;

lista->inicio = aux->siguiente;

delete aux;

aux = NULL;

lista->tam--;

}

}

/\*------------------------------------------------------------------------------------\*/

void remover(Listaenc\* lista, EstructuraSucursales \* sucursal, int pos)

{

if(estaInicializado(lista) && !(estaVacia(lista)) && !(pos < 0 || pos >= lista->tam))

{

ListaSucursales \*anterior, \*actual;

if (pos == 0)

{

removerInicio(lista, sucursal);

}

else

{

anterior = NULL;

actual = lista->inicio;

for(int i = 0; i < pos; i++)

{

anterior = actual;

actual = actual->siguiente;

}

anterior->siguiente = actual->siguiente;

if (sucursal != NULL)

sucursal = actual->sucursales;

delete actual;

actual = NULL;

lista->tam--;

}

}

}

/\*------------------------------------------------------------------------------------\*/

void obtenerElemento(Listaenc\* lista, EstructuraSucursales\*\* sucursal, int pos)

{

if(estaInicializado(lista) && (!estaVacia(lista) && (!(pos < 0 || pos >= lista->tam))) && (!(sucursal == NULL)))

{

ListaSucursales \*aux;

aux = lista->inicio;

for(int i = 0; i < pos; i++)

{

aux = aux->siguiente;

}

\*sucursal = aux->sucursales;

}

else

{

cout<<"no se encuentra inicializada la lista(obtener Elmento)"<<endl;

}

}

/\*------------------------------------------------------------------------------------\*/

void obtenerTamanio(Listaenc\* lista, int\* tam)

{

if((estaInicializado(lista)) && (!(tam == NULL)))

{

\*tam = lista->tam;

}

else

{

cout<<"no esta inicializado o el tamanio es nulo(Obtener tamanio)"<<endl;

}

}

/\*------------------------------------------------------------------------------------\*/

void imprimir(Listaenc\* lista)

{

if(lista->inicio==NULL)

{

cout<<endl<<"No hay datos cargados"<<endl<<endl;

}

else

{

system("cls");

int qtdeElementos;

obtenerTamanio(lista, &qtdeElementos);

EstructuraSucursales \* sucursal;

for(int i = 0; i < qtdeElementos; i++)

{

obtenerElemento(lista, &sucursal, i);

mostrarSucursal(sucursal);

}

cout<<endl;

}

system ("pause");

system("cls");

}

/\*------------------------------------------------------------------------------------\*/

**Main.CPP**

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <conio.h>

#include "TDA.h"

#include "ListaSucursales.h"

#include "generalidades.h"

using namespace std;

int main()

{

/// Declaracion e inicializacion de variables

int opcion;

Listaenc\* miLista = crearLista();

do

{

cout<<"Bienvenido al administrador de sucursales\n";

cout<<"Elija un numero de opcion:\n\n" ;

cout<<"1. Listar por facturacion.\n";

cout<<"2. Ranking cantidad articulos vendidos.\n";

cout<<"3. Rendimiento.\n";

cout<<"4. Mostrar sucursales.\n";

cout<<"6. Cargar datos del txt.\n\n";

cout<<"0. Salir\n\n";

cout<<"Opcion: ";

cin>>opcion;

switch(opcion)

{

case 1:

cout<<endl<<"1. Nacional"<<endl;

cout<<"2. Provincial"<<endl;

cout<<endl<<"Opcion: ";

cin>>opcion;

if(opcion==1)

{

ordenarNacionalFacturacion(miLista);

break;

}

else if(opcion==2)

{

ordenarProvinciaFacturacion(miLista);

break;

}

else

{

system("cls");

break;

}

case 2:

cout<<endl<<"1. Nacional"<<endl;

cout<<"2. Provincial"<<endl;

cout<<endl<<"Opcion: ";

cin>>opcion;

if(opcion==1)

{

ordenarNacionalCantArt(miLista);

break;

}

else if(opcion==2)

{

ordenarProvinciaCantArt(miLista);

break;

}

else

{

system("cls");

break;

}

case 3:

ordenarPorRendimiento(miLista);

break;

case 4:

imprimir(miLista);

break;

case 6:

cargarDatos(miLista);

break;

case 0:

system("cls");

cout<<"\Hasta luego!!\n";

liberarLista(miLista);

break;

default:

cout<<endl<<"Opcion no encontrada \n"<<endl;

system ("pause");

system("cls");

}

}

while(opcion!=0);

return 0;

}

**Generalidades.h**

#ifndef GENERALIDADES\_H\_INCLUDED

#define GENERALIDADES\_H\_INCLUDED

#include "ListaSucursales.h"

#include "TDA.h"

/\*\*

Definicion del Tipo de Dato para manejo del Rendimiento

Atributos:

\* codSuc

\* casaMatriz

\* rendimiento

Axiomas:

\* codSuc tiene que poseer 4 chars

\* casaMatriz tiene que poseer 4 chars

\* rendimiento > 0

\*/

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

/\*\* Definiciones de Tipos de Datos \*/

/\*\*-----------------------------------\*/

/\*\* Tipo de Estructura del Rendimiento \*/

typedef struct

{

char codSuc[5];

char casaMatriz[4];

float rendimiento=0;

} Rendimiento;

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

/\*\* Definicion de Primitivas \*/

/\*\*-----------------------------------\*/

/\*\*

PRE : La lista debe haber sido creada.

POST: La lista queda creada y cargada con los datos del txt ingresado

lista: Instancia sobre la cual se invoca la primitiva

\*/

void cargarDatos(Listaenc\* miLista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : La lista debe haber sido creada y debe haber sido cargada con cargarDatos()

POST: Devuelve una lista ordenada de Nacional mayor a menor en base al campo MontoMensual

lista: Instancia sobre la cual se invoca la primitiva

\*/

void ordenarNacionalFacturacion(Listaenc\* miLista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : La lista debe haber sido creada y debe haber sido cargada con cargarDatos()

POST: Devuelve una lista ordenada de Provincial mayor a menor en base al campo MontoMensual

lista: Instancia sobre la cual se invoca la primitiva

\*/

void ordenarProvinciaFacturacion(Listaenc\* miLista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : La lista debe haber sido creada y debe haber sido cargada con cargarDatos()

POST: Devuelve una lista ordenada de Nacional mayor a menor en base al campo Cantidad de Articulos

lista: Instancia sobre la cual se invoca la primitiva

\*/

void ordenarNacionalCantArt(Listaenc\* miLista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : La lista debe haber sido creada y debe haber sido cargada con cargarDatos()

POST: Devuelve una lista ordenada de Provincial mayor a menor en base al campo Cantidad de Articulos

lista: Instancia sobre la cual se invoca la primitiva

\*/

void ordenarProvinciaCantArt(Listaenc\* miLista);

/\*------------------------------------------------------------------------------------\*/

/\*\*

PRE : La lista debe haber sido creada y debe haber sido cargada con cargarDatos()

POST: Devuelve una lista ordenada mayor a menor en base al Rendimiento de las casa matrices

lista: Instancia sobre la cual se invoca la primitiva

\*/

void ordenarPorRendimiento(Listaenc\* miLista);

/\*------------------------------------------------------------------------------------\*/

#endif // GENERALIDADES\_H\_INCLUDED

**Generalidades.CPP**

#include "generalidades.h"

#include "ListaSucursales.h"

#include "TDA.h"

#include <iostream>

#include <fstream>

using namespace std;

const char \*provincias[23]= {"Misiones","Buenos Aires","Catamarca","Jujuy","San Luis","San Juan","Chaco","Chubut","Cordoba","Corriente","Entre Rios","Formosa","La Pampa", "La Rioja", "Mendoza","Neuquen","Rio Negro","Salta","Santa Cruz", "Santa Fe", "Santiago del Estero", "Tierra de Fuego", "Tucuman"};

void cargarDatos(Listaenc\* miLista)

{

if(miLista->inicio==NULL)

{

ifstream archivo;

string texto;

int cont = 0;

int articulos,n;

float montoMensual,metrosCuadrados;

char provincia[20],numeroCasaMatriz[5],codigo[5];

int posicion = 0;

archivo.open("ejemplo-sucursales.txt",ios::in);

if(archivo.fail())

{

cout<<endl<<"No se puede abrir el archivo"<<endl<<endl;

exit(1);

}

while(archivo >> ws,getline(archivo,texto,'-'))

{

switch(cont)

{

case 0:

strcpy(codigo,texto.c\_str());

break;

case 1:

strcpy(provincia,texto.c\_str());

break;

case 2:

articulos = atoi(texto.c\_str());

break;

case 3:

montoMensual = atof(texto.c\_str());

break;

case 4:

metrosCuadrados = atof(texto.c\_str());

break;

case 5:

strcpy(numeroCasaMatriz,texto.c\_str());

break;

}

if(cont == 5)

{

EstructuraSucursales\* Sucursal = new EstructuraSucursales(codigo,provincia,articulos,montoMensual,metrosCuadrados,numeroCasaMatriz);

insertar(miLista,Sucursal,posicion);

posicion++;

cont = -1;

}

cont++;

}

archivo.close();

cout<<endl<<"Archivo cargado con exito"<<endl<<endl;

}

else

{

cout<<endl<<"El archivo ya se encuentra cargado"<<endl<<endl;

}

system ("pause");

system("cls");

}

/\*------------------------------------------------------------------------------------\*/

void ordenarNacionalFacturacion(Listaenc\* miLista)

{

if(estaVacia(miLista))

{

cout<<endl<<"No hay datos cargados"<<endl<<endl;

}

else

{

system("cls");

int tamanioLista,i,j;

obtenerTamanio(miLista,&tamanioLista);

EstructuraSucursales\* auxiliar;

EstructuraSucursales\* sucursal[tamanioLista];

for(i=0; i<tamanioLista; i++)

{

obtenerElemento(miLista,&sucursal[i],i);

}

for(i=0; i<tamanioLista; i++)

{

for(j=0; j<tamanioLista; j++)

{

if(sucursal[i]->monto\_mensual>sucursal[j]->monto\_mensual)

{

auxiliar=sucursal[i];

sucursal[i]=sucursal[j];

sucursal[j]=auxiliar;

}

}

}

for(i=0; i<tamanioLista; i++)

{

mostrarSucursal(sucursal[i]);

}

}

system ("pause");

system("cls");

}

/\*------------------------------------------------------------------------------------\*/

void ordenarProvinciaFacturacion(Listaenc\* miLista)

{

if(estaVacia(miLista))

{

cout<<endl<<"No hay datos cargados"<<endl<<endl;

}

else

{

system("cls");

int tamanioLista,i,j;

obtenerTamanio(miLista,&tamanioLista);

EstructuraSucursales\* auxiliar;

EstructuraSucursales\* sucursal[tamanioLista];

for(i=0; i<tamanioLista; i++)

{

obtenerElemento(miLista,&sucursal[i],i);

}

for(i=0; i<tamanioLista; i++)

{

for(j=0; j<tamanioLista; j++)

{

if(sucursal[i]->monto\_mensual>sucursal[j]->monto\_mensual)

{

auxiliar=sucursal[i];

sucursal[i]=sucursal[j];

sucursal[j]=auxiliar;

}

}

}

float totalProvincias;

bool encontrado;

for(i=0; i<23; i++)

{

encontrado=false;

totalProvincias=0;

for(j=0; j<tamanioLista; j++)

{

if(strcmp(provincias[i], sucursal[j]->provincia)==0)

{

mostrarSucursal(sucursal[j]);

totalProvincias+=sucursal[j]->monto\_mensual;

encontrado=true;

}

}

if(encontrado)

{

cout<<endl<<provincias[i]<<" - "<<"Total: "<<totalProvincias<<endl<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

}

}

}

system ("pause");

system("cls");

}

/\*------------------------------------------------------------------------------------\*/

void ordenarProvinciaCantArt(Listaenc\* miLista)

{

if(miLista->inicio==NULL)

{

cout<<endl<<"No hay datos cargados"<<endl<<endl;

}

else

{

system("cls");

int tamanioLista,i,j;

obtenerTamanio(miLista,&tamanioLista);

EstructuraSucursales\* auxiliar;

EstructuraSucursales\* sucursal[tamanioLista];

for(i=0; i<tamanioLista; i++)

{

obtenerElemento(miLista,&sucursal[i],i);

}

for(i=0; i<tamanioLista; i++)

{

for(j=0; j<tamanioLista; j++)

{

if(sucursal[i]->cantidad\_articulos>sucursal[j]->cantidad\_articulos)

{

auxiliar=sucursal[i];

sucursal[i]=sucursal[j];

sucursal[j]=auxiliar;

}

}

}

float totalProvincias;

bool encontrado;

for(i=0; i<23; i++)

{

encontrado=false;

for(j=0; j<tamanioLista; j++)

{

if(strcmp(provincias[i], sucursal[j]->provincia)==0)

{

mostrarSucursal(sucursal[j]);

encontrado=true;

}

}

if(encontrado)

{

cout<<endl<<provincias[i];

cout<<endl<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

}

}

}

system ("pause");

system("cls");

}

/\*------------------------------------------------------------------------------------\*/

void ordenarNacionalCantArt(Listaenc\* miLista)

{

if(miLista->inicio==NULL)

{

cout<<endl<<"No hay datos cargados"<<endl<<endl;

}

else

{

system("cls");

int tamanioLista,i,j;

obtenerTamanio(miLista,&tamanioLista);

EstructuraSucursales\* auxiliar;

EstructuraSucursales\* sucursal[tamanioLista];

for(i=0; i<tamanioLista; i++)

{

obtenerElemento(miLista,&sucursal[i],i);

}

for(i=0; i<tamanioLista; i++)

{

for(j=0; j<tamanioLista; j++)

{

if(sucursal[i]->cantidad\_articulos>sucursal[j]->cantidad\_articulos)

{

auxiliar=sucursal[i];

sucursal[i]=sucursal[j];

sucursal[j]=auxiliar;

}

}

}

for(i=0; i<tamanioLista; i++)

{

mostrarSucursal(sucursal[i]);

}

}

system ("pause");

system("cls");

}

/\*------------------------------------------------------------------------------------\*/

void ordenarPorRendimiento(Listaenc\* miLista)

{

if(estaVacia(miLista))

{

cout<<endl<<"No hay datos cargados"<<endl<<endl;

}

else

{

system("cls");

int tamanioLista,i,j;

obtenerTamanio(miLista,&tamanioLista);

EstructuraSucursales\* auxiliar;

EstructuraSucursales\* sucursal[tamanioLista];

Rendimiento rendimientoFranquicia[tamanioLista],aux;

for(i=0; i<tamanioLista; i++)

{

obtenerElemento(miLista,&sucursal[i],i);

}

cout<<"Rendimiento por metros cuadrados"<<endl;

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

bool encontrado;

for(i=0; i<tamanioLista; i++)

{

float rendimiento=0;

rendimiento=sucursal[i]->monto\_mensual/sucursal[i]->m\_cuadrados;

if(strcmp(sucursal[i]->casa\_matriz,"0000")==0)

{

strcpy(rendimientoFranquicia[i].codSuc,sucursal[i]->codigo);

rendimientoFranquicia[i].rendimiento=rendimiento;

}

else

{

encontrado=false;

for(j=0; j<tamanioLista; j++)

{

if(strcmp(sucursal[i]->casa\_matriz,rendimientoFranquicia[j].casaMatriz)==0)

{

rendimientoFranquicia[j].rendimiento+=rendimiento;

encontrado=true;

}

}

if(!encontrado)

{

strcpy(rendimientoFranquicia[i].casaMatriz,sucursal[i]->casa\_matriz);

rendimientoFranquicia[i].rendimiento=rendimiento;

}

}

}

for(i=0; i<tamanioLista; i++)

{

for(j=tamanioLista-1; j>=i; j--)

{

if(rendimientoFranquicia[i].rendimiento<rendimientoFranquicia[j].rendimiento)

{

aux=rendimientoFranquicia[i];

rendimientoFranquicia[i]=rendimientoFranquicia[j];

rendimientoFranquicia[j]=aux;

}

}

for(j=0; j<tamanioLista; j++)

{

if(strcmp(rendimientoFranquicia[i].codSuc, sucursal[j]->codigo)==0||strcmp(rendimientoFranquicia[i].casaMatriz,sucursal[j]->casa\_matriz)==0 && strcmp(sucursal[j]->casa\_matriz,"0000")!=0)

{

mostrarSucursal(sucursal[j]);

}

}

if(rendimientoFranquicia[i].rendimiento!=0)

{

cout<<endl<<"Rendimiento: "<<rendimientoFranquicia[i].rendimiento;

cout<<endl<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

}

}

}

system ("pause");

system("cls");

}

/\*------------------------------------------------------------------------------------\*/