

ALGORITMOS Y ESTRUCTURAS DE DATOS

Trabajo practico grupal

Universidad Nacional de Lanús, Fecha: 30 de Octubre del 2020

**Integrantes:**

Nehuen Verges, DNI: 39407041, Mail: nehuenverges@hotmail.com

Pablo Agustín Charczuk, DNI: 36167991, Mail: pablochark@gmail.com

Miguel San Martín , DNI: 40639931, Mail: miguelshakur97@gmail.com

Esteban Emanuel Rodríguez, DNI: 39450435, Mail: e.rodriguez95@hotmail.com

**Enunciado**

En la siguiente documentación vamos a ir explicando el programa desarrollado. En el mismo se nos encomendó realizar un ranking de sucursales. El programa consta de tres rankings principales, uno para la facturación de las sucursales, otro para la cantidad de artículos vendidos por sucursal y el ultimo es el rendimiento. En el ranking de facturación y de cantidad de artículos vendidos podemos encontrar dos maneras distintas de visualización. Puede ser de manera individual de las sucursales a nivel nacional o en caso de querer, se puede visualizar con un listado separado según un ranking interno por provincia. Por último, nos encontramos con el ranking de rendimiento según casa matriz, cuyo procedimiento y funcionamiento se va a explicar detalladamente más adelante.

**Desarrollo de la solución**

Antes de iniciar el desarrollo se decidió la utilización de una lista simplemente enlazada como estructura para los datos a manipular.

Habiendo decidido la estructura a utilizar, el primer inconveniente que se presento fue la carga de datos. Estos debían ser tomados desde un archivo de texto y no por consola, ya que el programa lo único que hace es la visualización ordenada de los rankings.

Los datos en el archivo de texto se presentan de la siguiente manera:

**0054-Catamarca-1258-42654,12-254,00-0003-**

Se nos había aclarado en un principio el significado de cada valor para una fácil comprensión. Los cuales vamos a mostrar cómo deben ser leídos.

**Numero de sucursal: 0054**

**Provincia: Catamarca**

**Cantidad de artículos vendidos: 1258**

**Facturación: 42654,12**

**Área de la sucursal (cm2): 254**

**Casa matriz la cual pertenece: 0003**

Sabiendo cómo interpretar los datos entregados, había que transformar los mismos en parte de nuestra estructura para su debida manipulación.

Lo que realizamos fue “cortes” en la línea de texto para separar los datos deseables.

A modo de ejemplo se va a mostrar el funcionamiento de cómo obtener el número de sucursal y almacenarlo dentro de un elemento de nuestra lista. Este procedimiento se encuentra en el archivo Sucursal.cpp y el mismo se llama cargarSucursal.

El primer paso es leer la línea completa y almacenarla en una variable llamada “lectura”.

**leerLineaSucursal(&lectura, fSucursal)**

Luego determinar el largo de la línea obtenida.

**int largo = lectura.size()**

El siguiente paso es rastrear y determinar la ubicación del primer guion (-). De esta manera vamos a saber que, desde el inicio de la línea hasta el guion, obtendremos el número de sucursal.

**pos = lectura.find('-')**

Habiendo encontrado el guion, el próximo paso es realizar el corte apropiado para la obtención y almacenamiento del dato como un atributo de un elemento de nuestra lista. En este caso se utiliza la instrucción atoi para convertir el dato obtenido en un entero.

**int idSuc = atoi(lectura.substr(0,pos).c\_str())**

Como último paso, se guarda el resto de la línea de texto sobrescribiendo nuestra variable “lectura” para poder utilizarla más adelante, pero con una salvedad. El guion que utilizamos para realizar el corte no va a ser guardado, quedando el mismo eliminado de la línea.

**lectura = lectura.substr(pos+1,largo).c\_str()**

Estos pasos se repiten hasta terminar la línea completa y obtener todos los datos deseados para luego pasar a la siguiente línea de texto, hasta que finalmente ya no queden más datos. Cada una de las sucursales obtenidas del archivo de texto, se guarda al principio de la lista para luego ser ordenada según corresponda.

En caso de realizar una errónea carga de datos en el archivo de texto, en la lista de nuestro programa se van a cargar valores incorrectos provocando visualización incorrecta o causando fallas en el ordenamiento de los elementos. Por ejemplo, todas las líneas de texto deben terminar con el guion medio (-) para que se pueda realizar correctamente los cortes internos para separar los campos.

Habiendo realizado el correcto almacenamiento de datos en nuestra lista, lo siguiente a resolver fue ordenar los elementos según el usuario lo desee. Siendo una lista simplemente enlazada, había que tener en consideración que solamente se podía recorrer la misma en una sola dirección. Esto quiere decir que un elemento de la lista podía apuntar solamente al siguiente, sin posibilidad de volver al elemento anterior. En caso de ser el último, este directamente no apunta a nada, por lo tanto, se entiende que es el final de la lista.

Para poder realizar el ordenamiento de la lista se creó una estructura tipo enum la cual al comparar dos elementos nos va a indicar con más facilidad cual es mayor o si son iguales, la comparación se realiza a través de una función que será seteada en la lista al momento de ser creada. La declaración de dicha estructura la podemos encontrar en el archivo lista.h.

Este ordenamiento se ejecuta dentro del procedimiento reordenar (declarado en lista.cpp) que, con ayuda de una lista temporal, ejecuta las comparaciones entre elementos para luego ser almacenados dentro de nuestra lista inicial. Por último, la lista temporal creada es eliminada dentro del procedimiento.

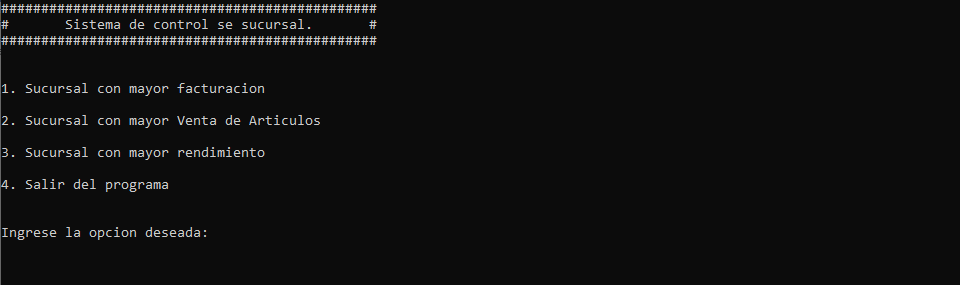
Como último de los cálculos a realizar fue el rendimiento de las casas madres (casa matriz). Para poder lograrlo, se decidió crear una nueva lista donde se va a almacenar un acumulador de la facturación y los metros cuadrados correspondientes a las sucursales que trabajan para la misma casa matriz. Dentro del procedimiento “void calculoRendimiento” en el archivo Nucleo.cpp se realiza la incorporación de las casas matriz a la lista correspondiente. A su vez, con ayuda de dos punteros, uno para recorrer la lista completa de sucursales y otro para el almacenamiento de los datos seleccionados en la lista de casas matriz, se intenta ubicar los parámetros indicados anteriormente en las sucursales correspondientes para almacenarlos en los acumuladores de la nueva lista. Finalmente, cuando ya esté completa la lista de casas matriz, se utiliza el procedimiento “resutladoRendimiento” para realizar el cociente entre la facturación y el área total. Dicho procedimiento también realizar la impresión de los valores.

Por ultimo antes de realizar la impresión, se creó un archivo llamado Nucleo.cpp (con su correspondiente Nucleo.h) el cual es un intermediario entre el archivo main y los procedimientos. El cometido de este archivo es descomprimir y ordenar el contenido de main, en el cual podemos encontrar los procedimientos para la impresión de cada lista, el cálculo de rendimiento y las comparaciones entre elementos.

**Manual de usuario**

Al iniciar el Sistema de control de sucursal, la primera pantalla que nos encontramos, muestra las opciones indicadas con números.

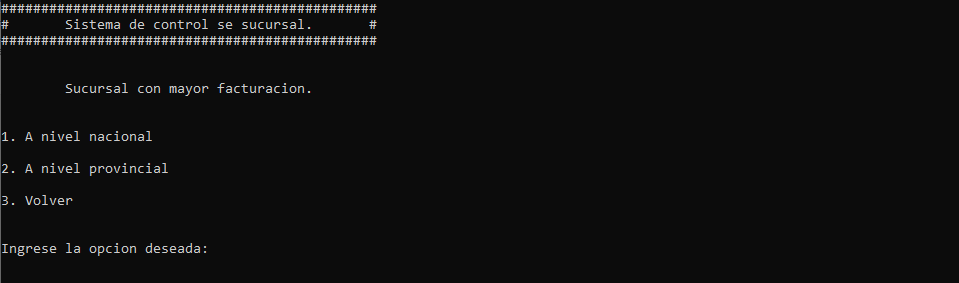
Para acceder a cada una, simplemente seleccionamos el numero deseado.



En las siguientes demostraciones vamos a ir explicando paso a paso cada una de las opciones en pantalla.

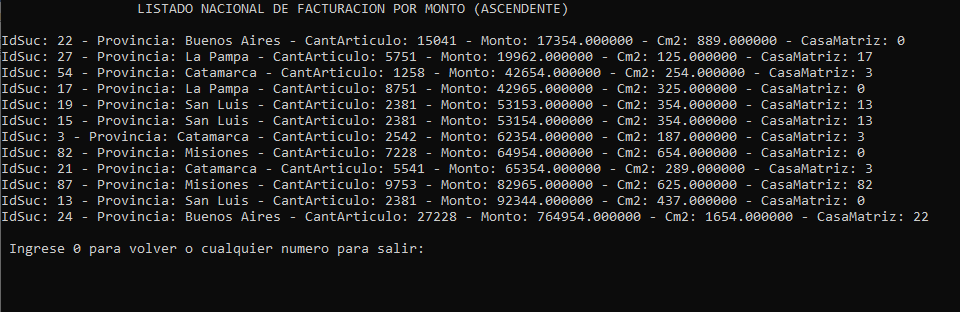
Opción 1: Sucursal con mayor facturación.

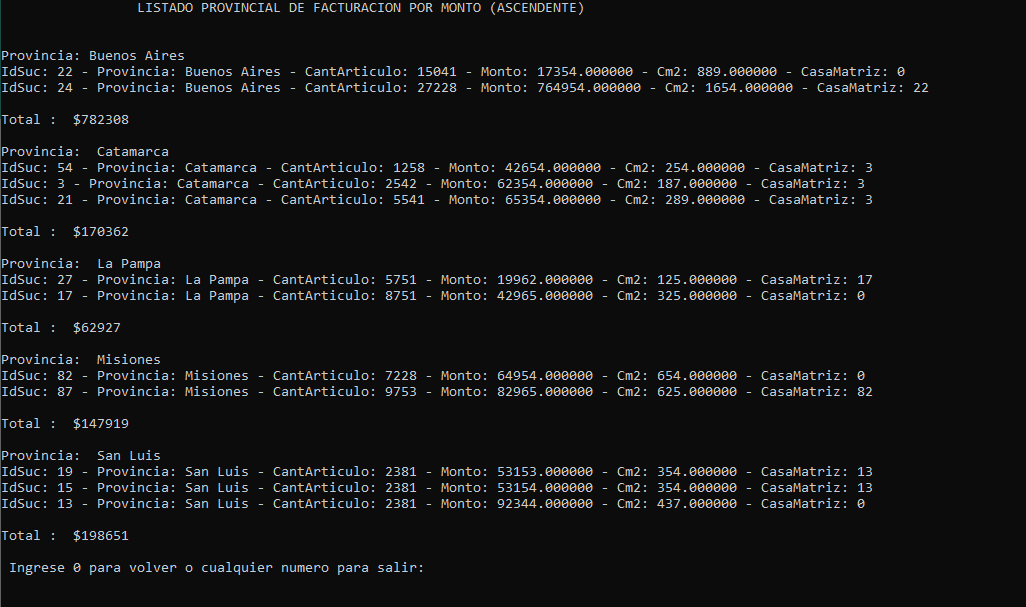
Si seleccionamos esta opción vamos a pasar al siguiente menú, en la cual podemos elegir entre dos opciones de visualización de datos. Siendo la opción 1 para ver las sucursales ordenadas de manera ascendente en un ranking nacional y la opción 2 también con un orden ascendente, pero con un ranking según sea la provincia. Para volver al menú principal utilizaremos la opción 3.



A continuación, se mostrará imagen de cómo es la visualización de datos con ambas opciones. Dentro de cada opción, para volver al menú principal ingresamos el número 0 y para salir del programa utilizaremos un numero distinto de 0.

Ranking de facturación a nivel nacional:

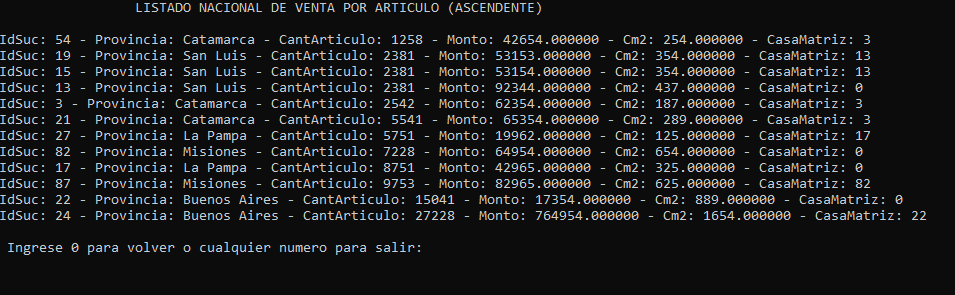


Ranking de facturación a nivel provincial:

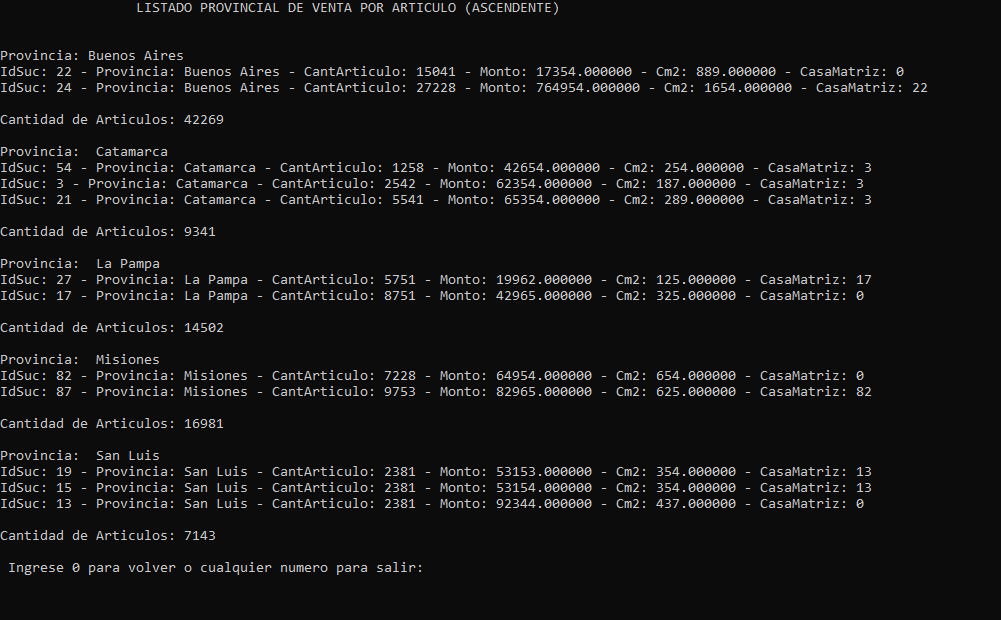
Opción 2: Sucursal con mayor venta de artículos.

Como en la opción anterior, también vamos a tener un ranking con ordenamiento ascendente pudiendo elegir la visualización de los datos, siendo la opción 1 el listado nacional y la opción 2 el listado por provincia.

Ranking de venta de artículos a nivel nacional:



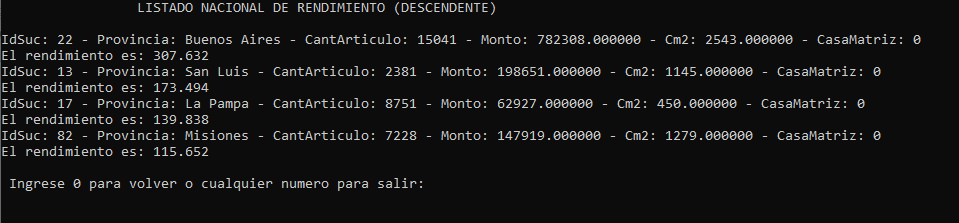
Ranking de venta de artículos a nivel provincial:



Opción 3: Sucursal con mayor rendimiento

Con esta opción vamos a poder visualizar de manera descendente el rendimiento, pero a diferencia de los anteriores listados, este se va a presentar por casas matriz (dueño). Es decir que las sucursales que responden al mismo dueño se van a encontrar evaluadas de manera conjunta. El rendimiento se puede obtener mediante el cociente de la facturación con los metros cuadrados (Facturación / Metros2), teniendo en cuenta el total de sucursales que responden a la misma casa matriz.

En la siguiente imagen se muestra de manera ilustrativa como es la visualización de los datos.



**Anexo**

**Main.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <iomanip> |
|  |

|  |
| --- |
| #include <string> |
|  |

|  |
| --- |
| #include "entidades/Sucursal.h" |
|  |

|  |
| --- |
| #include "entidades/Provincia.h" |
|  |

|  |
| --- |
| #include "Listas/Lista.h" |
|  |

|  |
| --- |
| #include "entidades/Nucleo.h" |
|  |

|  |
| --- |
| using namespace std; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int opc; |
|  |

|  |
| --- |
| void consulta(), menu(),facturacion(),ventas(), rendimiento(), volver(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void consulta(){ |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
| cout <<"###############################################"<< endl; |
|  |

|  |
| --- |
| cout << "#\tSistema de control se sucursal. #" << endl; |
|  |

|  |
| --- |
| cout <<"###############################################"<< endl; |
|  |

|  |
| --- |
| cout << "\n\n1. Sucursal con mayor facturacion" << endl; |
|  |

|  |
| --- |
| cout << "\n2. Sucursal con mayor Venta de Articulos" << endl; |
|  |

|  |
| --- |
| cout << "\n3. Sucursal con mayor rendimiento" << endl; |
|  |

|  |
| --- |
| cout << "\n4. Salir del programa" << endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| cout << "\n\nIngrese la opcion deseada: "; cin>>opc; |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
| switch(opc){ |
|  |

|  |
| --- |
| case 1: facturacion(); break; |
|  |

|  |
| --- |
| case 2: ventas(); break; |
|  |

|  |
| --- |
| case 3: rendimiento(); volver(); break; |
|  |

|  |
| --- |
| case 4: break; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| default : consulta(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void facturacion(){ |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| cout <<"###############################################"<< endl; |
|  |

|  |
| --- |
| cout << "#\tSistema de control se sucursal. #" << endl; |
|  |

|  |
| --- |
| cout <<"###############################################"<< endl; |
|  |

|  |
| --- |
| cout << "\n\n\tSucursal con mayor facturacion." << endl; |
|  |

|  |
| --- |
| cout << "\n\n1. A nivel nacional" << endl; |
|  |

|  |
| --- |
| cout << "\n2. A nivel provincial" << endl; |
|  |

|  |
| --- |
| cout << "\n3. Volver" << endl; |
|  |

|  |
| --- |
| cout << "\n\nIngrese la opcion deseada: "; cin>>opc; |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
| switch(opc){ |
|  |

|  |
| --- |
| case 1: cout <<" \t \t LISTADO NACIONAL DE FACTURACION POR MONTO (ASCENDENTE)"<<endl << endl; lstNacionalPorMonto(); volver(); break; |
|  |

|  |
| --- |
| case 2: cout <<" \t \t LISTADO PROVINCIAL DE FACTURACION POR MONTO (ASCENDENTE)"<<endl << endl; lstProvincialPorMonto(); volver(); break; |
|  |

|  |
| --- |
| case 3: consulta(); break; |
|  |

|  |
| --- |
| default : facturacion(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void ventas(){ |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| cout <<"###############################################"<< endl; |
|  |

|  |
| --- |
| cout << "#\tSistema de control se sucursal. #" << endl; |
|  |

|  |
| --- |
| cout <<"###############################################"<< endl; |
|  |

|  |
| --- |
| cout << "\n\n\tSucursal con mayor Venta de Articulos" << endl; |
|  |

|  |
| --- |
| cout << "\n\n1. A nivel nacional" << endl; |
|  |

|  |
| --- |
| cout << "\n2. A nivel provincial" << endl; |
|  |

|  |
| --- |
| cout << "\n3. Volver" << endl; |
|  |

|  |
| --- |
| cout << "\n\nIngrese la opcion deseada: "; cin>>opc; |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
| switch(opc){ |
|  |

|  |
| --- |
| case 1: cout <<" \t \t LISTADO NACIONAL DE VENTA POR ARTICULO (ASCENDENTE)"<<endl << endl; lstNacionalPorArticulo(); volver(); break; |
|  |

|  |
| --- |
| case 2: cout <<" \t \t LISTADO PROVINCIAL DE VENTA POR ARTICULO (ASCENDENTE)"<<endl << endl; lstProvincialPorArticulo(); volver(); break; |
|  |

|  |
| --- |
| case 3: consulta(); break; |
|  |

|  |
| --- |
| default : ventas(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void rendimiento(){ |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
| cout <<" \t \t LISTADO NACIONAL DE RENDIMIENTO (DESCENDENTE)"<<endl << endl; lstRendimiento(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void volver(){ |
|  |

|  |
| --- |
| cout << "\n Ingrese 0 para volver o cualquier numero para salir: "; cin>>opc; |
|  |

|  |
| --- |
| system("cls"); |
|  |

|  |
| --- |
| switch(opc){ |
|  |

|  |
| --- |
| case 0: consulta(); break; |
|  |

|  |
| --- |
| default : break; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int main() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| consulta(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return 0; |
|  |

}

**Nucleo.cpp**

|  |
| --- |
| // #include "entidades/Sucursal.h" |
|  |

|  |
| --- |
| // #include "entidades/Provincia.h" |
|  |

|  |
| --- |
| // #include "Listas/Lista.h" |
|  |

|  |
| --- |
| #include "../entidades/Nucleo.h" |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <iomanip> |
|  |

|  |
| --- |
| using namespace std; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void ResultadoRendimiento(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| float resultado=0; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
| while ( ptrCursor != fin() ) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| resultado=(getMonto(\*(Sucursal\*)ptrCursor->ptrDato)/getCm2(\*(Sucursal\*)ptrCursor->ptrDato)); |
|  |

|  |
| --- |
| std::cout<< toString(\*(Sucursal\*)ptrCursor->ptrDato)<< std::endl << "El rendimiento es: " << resultado << std::endl; |
|  |

|  |
| --- |
| ptrCursor=ptrCursor->sgte; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ResultadoComparacion compararArticulo(PtrDato ptrDato1, PtrDato ptrDato2) { |
|  |

|  |
| --- |
| if (getCantArticulo(\*(Sucursal\*)ptrDato1) >getCantArticulo(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| return MAYOR; |
|  |

|  |
| --- |
| else if (getCantArticulo(\*(Sucursal\*)ptrDato1) < getCantArticulo(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| return MENOR; |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| return IGUAL; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ResultadoComparacion compararMonto(PtrDato ptrDato1, PtrDato ptrDato2) { |
|  |

|  |
| --- |
| if (getMonto(\*(Sucursal\*)ptrDato1) >getMonto(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| return MAYOR; |
|  |

|  |
| --- |
| else if (getMonto(\*(Sucursal\*)ptrDato1) < getMonto(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| return MENOR; |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| return IGUAL; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ResultadoComparacion compararMontoProv(PtrDato ptrDato1, PtrDato ptrDato2) { |
|  |

|  |
| --- |
| if (getProvincia(\*(Sucursal\*)ptrDato1)>getProvincia(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return MAYOR; |
|  |

|  |
| --- |
| }else |
|  |

|  |
| --- |
| if (getProvincia(\*(Sucursal\*)ptrDato1)<getProvincia(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return MENOR; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| compararMonto(ptrDato1,ptrDato2); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| ResultadoComparacion compararArticuloProv(PtrDato ptrDato1, PtrDato ptrDato2) { |
|  |

|  |
| --- |
| if (getProvincia(\*(Sucursal\*)ptrDato1)>getProvincia(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return MAYOR; |
|  |

|  |
| --- |
| }else |
|  |

|  |
| --- |
| if (getProvincia(\*(Sucursal\*)ptrDato1)<getProvincia(\*(Sucursal\*)ptrDato2)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return MENOR; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| compararArticulo(ptrDato1,ptrDato2); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ResultadoComparacion compararMt2Fact(PtrDato ptrDato1, PtrDato ptrDato2) { |
|  |

|  |
| --- |
| float cos1=getCm2(\*(Sucursal\*)ptrDato1)/ getMonto(\*(Sucursal\*)ptrDato1); |
|  |

|  |
| --- |
| float cos2=getCm2(\*(Sucursal\*)ptrDato2)/ getMonto(\*(Sucursal\*)ptrDato2); |
|  |

|  |
| --- |
| if (cos1 > cos2) |
|  |

|  |
| --- |
| return MAYOR; |
|  |

|  |
| --- |
| else if (cos1<cos2) |
|  |

|  |
| --- |
| return MENOR; |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| return IGUAL; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void lstNacionalPorMonto(){ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Lista listaSuc; |
|  |

|  |
| --- |
| crearLista(listaSuc,compararMonto); |
|  |

|  |
| --- |
| cargarSucursal( listaSuc); |
|  |

|  |
| --- |
| reordenar(listaSuc); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ImprimirListaNacional(listaSuc); |
|  |

|  |
| --- |
| eliminarListaSucursal(listaSuc); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void lstProvincialPorMonto(){ |
|  |

|  |
| --- |
| Lista listaSuc; |
|  |

|  |
| --- |
| crearLista(listaSuc,compararMontoProv); |
|  |

|  |
| --- |
| cargarSucursal( listaSuc); |
|  |

|  |
| --- |
| reordenar(listaSuc); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ImprimirListaProvincialMonto(listaSuc); |
|  |

|  |
| --- |
| eliminarListaSucursal(listaSuc); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void lstProvincialPorArticulo(){ |
|  |

|  |
| --- |
| Lista listaSuc; |
|  |

|  |
| --- |
| crearLista(listaSuc,compararArticuloProv); |
|  |

|  |
| --- |
| cargarSucursal(listaSuc); |
|  |

|  |
| --- |
| reordenar(listaSuc); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ImprimirListaProvincialArticulo(listaSuc); |
|  |

|  |
| --- |
| eliminarListaSucursal(listaSuc); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void lstNacionalPorArticulo(){ |
|  |

|  |
| --- |
| Lista listaSuc; |
|  |

|  |
| --- |
| crearLista(listaSuc,compararArticulo); |
|  |

|  |
| --- |
| cargarSucursal(listaSuc); |
|  |

|  |
| --- |
| reordenar(listaSuc); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ImprimirListaNacional(listaSuc); |
|  |

|  |
| --- |
| eliminarListaSucursal(listaSuc); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void lstRendimiento(){ |
|  |

|  |
| --- |
| Lista listaFact; |
|  |

|  |
| --- |
| Lista listaSuc; |
|  |

|  |
| --- |
| crearLista(listaSuc,compararMt2Fact); |
|  |

|  |
| --- |
| cargarSucursal(listaSuc); |
|  |

|  |
| --- |
| crearLista(listaFact,compararMt2Fact); |
|  |

|  |
| --- |
| calculoRendimiento(listaSuc,listaFact); |
|  |

|  |
| --- |
| reordenar(listaFact); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ResultadoRendimiento(listaFact); |
|  |

|  |
| --- |
| eliminarListaSucursal(listaFact); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void ImprimirListaNacional(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
| while ( ptrCursor != fin() ) { |
|  |

|  |
| --- |
| std::cout<< toString(\*(Sucursal\*)ptrCursor->ptrDato)<<std::endl; |
|  |

|  |
| --- |
| ptrCursor=ptrCursor->sgte; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void ImprimirListaProvincialMonto(Lista &lista) { |
|  |

|  |
| --- |
| PtrNodoLista ptrPrevio = primero(lista); |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Provincia: " << getProvincia(\*(Sucursal\*)ptrPrevio->ptrDato)<<std::endl; |
|  |

|  |
| --- |
| float resultado = 0; |
|  |

|  |
| --- |
| float aux=0; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int flag = 0; |
|  |

|  |
| --- |
| while ( ptrCursor != fin() ) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| resultado+=(getMonto(\*(Sucursal\*)ptrPrevio->ptrDato)); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if(flag==1) { |
|  |

|  |
| --- |
| resultado-=(getMonto(\*(Sucursal\*)ptrPrevio->ptrDato)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if(getProvincia(\*(Sucursal\*)ptrPrevio->ptrDato)!=getProvincia(\*(Sucursal\*)ptrCursor->ptrDato)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Total : $" << resultado <<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Provincia: " << getProvincia(\*(Sucursal\*)ptrCursor->ptrDato)<<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| resultado = 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout<< toString(\*(Sucursal\*)ptrCursor->ptrDato)<<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ptrPrevio=ptrCursor; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ptrCursor=ptrCursor->sgte; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| flag++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| aux=(getMonto(\*(Sucursal\*)ptrPrevio->ptrDato)); |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Total : $" << resultado+aux <<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void ImprimirListaProvincialArticulo(Lista &lista) { |
|  |

|  |
| --- |
| PtrNodoLista ptrPrevio = primero(lista); |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Provincia: " << getProvincia(\*(Sucursal\*)ptrPrevio->ptrDato)<<std::endl; |
|  |

|  |
| --- |
| float resultado = 0; |
|  |

|  |
| --- |
| float aux=0; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int flag = 0; |
|  |

|  |
| --- |
| while ( ptrCursor != fin() ) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| resultado+=(getCantArticulo(\*(Sucursal\*)ptrPrevio->ptrDato)); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if(flag==1) { |
|  |

|  |
| --- |
| resultado-=(getCantArticulo(\*(Sucursal\*)ptrPrevio->ptrDato)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if(getProvincia(\*(Sucursal\*)ptrPrevio->ptrDato)!=getProvincia(\*(Sucursal\*)ptrCursor->ptrDato)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Cantidad de Articulos: " << resultado <<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Provincia: " << getProvincia(\*(Sucursal\*)ptrCursor->ptrDato)<<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| resultado = 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout<< toString(\*(Sucursal\*)ptrCursor->ptrDato)<<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ptrPrevio=ptrCursor; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ptrCursor=ptrCursor->sgte; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| flag++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| aux=(getCantArticulo(\*(Sucursal\*)ptrPrevio->ptrDato)); |
|  |

|  |
| --- |
| std::cout <<std::endl <<"Cantidad de Articulos: " << resultado+aux <<std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void eliminarListaSucursal(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* retira uno a uno los nodos de la lista \*/ |
|  |

|  |
| --- |
| while (! listaVacia(lista)){ |
|  |

|  |
| --- |
| borrarSucursal((Sucursal\*)primero(lista)->ptrDato); |
|  |

|  |
| --- |
| eliminarNodo(lista,primero(lista)); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void calculoRendimiento (Lista &listaSuc, Lista &listaCasMat) { |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(listaSuc); |
|  |

|  |
| --- |
| while (ptrCursor != fin()) { |
|  |

|  |
| --- |
| if (getCasaMatriz(\*(Sucursal\*)ptrCursor->ptrDato)==0) { |
|  |

|  |
| --- |
| adicionarPrincipio(listaCasMat,(Sucursal\*)ptrCursor->ptrDato); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| ptrCursor=ptrCursor->sgte; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| PtrNodoLista ptrCursorSuc = primero(listaSuc); |
|  |

|  |
| --- |
| PtrNodoLista ptrCursorCasMat = primero(listaCasMat); |
|  |

|  |
| --- |
| float facturacion=0; |
|  |

|  |
| --- |
| float mt2=0; |
|  |

|  |
| --- |
| while (ptrCursorCasMat != fin()) { |
|  |

|  |
| --- |
| facturacion=getMonto(\*(Sucursal\*)ptrCursorCasMat->ptrDato); |
|  |

|  |
| --- |
| mt2=getCm2(\*(Sucursal\*)ptrCursorCasMat->ptrDato); |
|  |

|  |
| --- |
| while(ptrCursorSuc != fin()) { |
|  |

|  |
| --- |
| if (getCodSucursal(\*(Sucursal\*)ptrCursorCasMat->ptrDato) == getCasaMatriz(\*(Sucursal\*)ptrCursorSuc->ptrDato)){ |
|  |

|  |
| --- |
| mt2+=getCm2(\*(Sucursal\*)ptrCursorSuc->ptrDato); |
|  |

|  |
| --- |
| facturacion+=getMonto(\*(Sucursal\*)ptrCursorSuc->ptrDato); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| ptrCursorSuc=ptrCursorSuc->sgte; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| ptrCursorSuc = primero(listaSuc); |
|  |

|  |
| --- |
| setCm2(\*(Sucursal\*)ptrCursorCasMat->ptrDato,mt2); |
|  |

|  |
| --- |
| setMonto(\*(Sucursal\*)ptrCursorCasMat->ptrDato,facturacion); |
|  |

|  |
| --- |
| ptrCursorCasMat=ptrCursorCasMat->sgte; |
|  |

|  |
| --- |
| } |
|  |

}

**Nucleo.h**

|  |
| --- |
| #ifndef NUCLEO\_H\_INCLUDED |
|  |

|  |
| --- |
| #define NUCLEO\_H\_INCLUDED |
|  |

|  |
| --- |
| #include "../entidades/Sucursal.h" |
|  |

|  |
| --- |
| #include "../entidades/Provincia.h" |
|  |

|  |
| --- |
| #include "../Listas/Lista.h" |
|  |

|  |
| --- |
| /\* Axiomas resultadoComparacion: |
|  |

|  |
| --- |
| pre: Tener dos punteros de Datos |
|  |

|  |
| --- |
| post: Devuelve un menor mayor o igual de la comparacion |
|  |

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

|  |
| --- |
| ResultadoComparacion compararArticulo(PtrDato ptrDato1, PtrDato ptrDato2); |
|  |

|  |
| --- |
| ResultadoComparacion compararMonto(PtrDato ptrDato1, PtrDato ptrDato2); |
|  |

|  |
| --- |
| ResultadoComparacion compararMontoProv(PtrDato ptrDato1, PtrDato ptrDato2); |
|  |

|  |
| --- |
| ResultadoComparacion compararArticuloProv(PtrDato ptrDato1, PtrDato ptrDato2); |
|  |

|  |
| --- |
| ResultadoComparacion compararIdProvincia(PtrDato ptrDato1, PtrDato ptrDato2); |
|  |

|  |
| --- |
| ResultadoComparacion compararMt2Fact(PtrDato ptrDato1, PtrDato ptrDato2); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Axiomas : |
|  |

|  |
| --- |
| pre: Tener el programa ejecutandose |
|  |

|  |
| --- |
| post: Muestra la salida de datos |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void lstNacionalPorMonto(); |
|  |

|  |
| --- |
| void lstProvincialPorMonto(); |
|  |

|  |
| --- |
| void lstNacionalPorArticulo(); |
|  |

|  |
| --- |
| void lstProvincialPorArticulo(); |
|  |

|  |
| --- |
| void lstRendimiento(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Axiomas : |
|  |

|  |
| --- |
| pre: La lista debe existir y tener contenido |
|  |

|  |
| --- |
| post: Imprime por consola la lista |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void ImprimirListaNacional(Lista &lista); |
|  |

|  |
| --- |
| void ImprimirListaProvincialMonto(Lista &lista); |
|  |

|  |
| --- |
| void ImprimirListaProvincialArticulo(Lista &lista); |
|  |

|  |
| --- |
| void ResultadoRendimiento(Lista &lista); |
|  |

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

|  |
| --- |
| pre: Debe existir una lista de Sucursales |
|  |

|  |
| --- |
| post: Elimina la lista de sucursales |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void eliminarListaSucursal(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| pre: Debe existir una lista de sucursales (llena de datos) y una lista de casas matrices (vacia) |
|  |

|  |
| --- |
| post: Una lista de casas matrices con la suma de la facturacion y de los cm2 de todas las sucursales |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void calculoRendimiento (Lista &listaSuc, Lista &listaCasMat); |
|  |

|  |
| --- |
|  |
|  |

#endif // NUCLEO\_H\_INCLUDED

**Lista.h**

|  |
| --- |
| /\* TDA Lista |
|  |

|  |
| --- |
| \* Implementación Simplemente Enlazada |
|  |

|  |
| --- |
| \* Archivo : Lista.h |
|  |

|  |
| --- |
| \* Versión : 1.1 |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| #ifndef \_\_LISTA\_H\_\_ |
|  |

|  |
| --- |
| #define \_\_LISTA\_H\_\_ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #ifndef NULL |
|  |

|  |
| --- |
| #define NULL 0 |
|  |

|  |
| --- |
| #endif |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| /\* Definiciones de Tipos de Datos \*/ |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* tipo enumerado para realizar comparaciones \*/ |
|  |

|  |
| --- |
| enum ResultadoComparacion { |
|  |

|  |
| --- |
| MAYOR, |
|  |

|  |
| --- |
| IGUAL, |
|  |

|  |
| --- |
| MENOR |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
| /\* Tipo de Informacion que esta contenida en los Nodos de la |
|  |

|  |
| --- |
| Lista, identificada como Dato. \*/ |
|  |

|  |
| --- |
| typedef void\* PtrDato; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Tipo de Estructura de los Nodos de la Lista. \*/ |
|  |

|  |
| --- |
| struct NodoLista { |
|  |

|  |
| --- |
| PtrDato ptrDato; // dato almacenado |
|  |

|  |
| --- |
| NodoLista\* sgte; // puntero al siguiente |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| typedef ResultadoComparacion (\*PFComparacion)(PtrDato , PtrDato); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Tipo de Puntero a los Nodos de la Lista, el cual se usa para recorrer |
|  |

|  |
| --- |
| la Lista y acceder a sus Datos. \*/ |
|  |

|  |
| --- |
| typedef NodoLista\* PtrNodoLista; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Tipo de Estructura de la Lista \*/ |
|  |

|  |
| --- |
| struct Lista{ |
|  |

|  |
| --- |
| PtrNodoLista primero; // puntero al primer nodo de la lista |
|  |

|  |
| --- |
| PFComparacion compara; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| /\* Definicion de Primitivas \*/ |
|  |

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

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| pre : la lista no debe haber sido creada. |
|  |

|  |
| --- |
| post: lista queda creada y preparada para ser usada. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : estructura de datos a ser creado. |
|  |

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

|  |
| --- |
| void crearLista(Lista &lista, PFComparacion compara ); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : lista Creada con crearLista(). |
|  |

|  |
| --- |
| post: Devuelve true si lista esta vacia, sino devuelve false. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

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

|  |
| --- |
| bool listaVacia(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| 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. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return representación del fin de la lista. |
|  |

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

|  |
| --- |
| PtrNodoLista fin(); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : lista Creada con crearLista(). |
|  |

|  |
| --- |
| post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si |
|  |

|  |
| --- |
| esta vacia |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| return puntero al primer nodo. |
|  |

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

|  |
| --- |
| PtrNodoLista primero(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| 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. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : 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. |
|  |

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

|  |
| --- |
| PtrNodoLista siguiente(Lista &lista, PtrNodoLista 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. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| prtNodo : puntero al nodo a partir del cual se requiere el anterior. |
|  |

|  |
| --- |
| return puntero al nodo anterior. |
|  |

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

|  |
| --- |
| PtrNodoLista anterior(Lista &lista, PtrNodoLista ptrNodo); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : lista creada con crearLista(). |
|  |

|  |
| --- |
| post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si |
|  |

|  |
| --- |
| si lista esta vacia. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| return puntero al último nodo. |
|  |

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

|  |
| --- |
| PtrNodoLista ultimo(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| 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. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| dato : elemento a adicionar al principio de la lista. |
|  |

|  |
| --- |
| return puntero al nodo adicionado. |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarPrincipio(Lista &lista, PtrDato ptrDato); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| 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(). |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| dato : elemento a adicionar. |
|  |

|  |
| --- |
| ptrNodo : puntero al nodo después del cual se quiere adicionar el dato. |
|  |

|  |
| --- |
| return puntero al nodo adicionado. |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarDespues(Lista &lista, PtrDato ptrDato, PtrNodoLista 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. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| dato : elemento a adicionar al final de la lista. |
|  |

|  |
| --- |
| return puntero al nodo adicionado. |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarFinal(Lista &lista, PtrDato ptrDato); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| 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. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| dato : elemento a adicionar. |
|  |

|  |
| --- |
| ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato. |
|  |

|  |
| --- |
| return puntero al nodo adicionado. |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarAntes(Lista &lista, PtrDato ptrDato, PtrNodoLista 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(). |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| ptrNodo : puntero al nodo que se desea eliminar. |
|  |

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

|  |
| --- |
| void eliminarNodo(Lista &lista, PtrNodoLista ptrNodo); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : lista creada con crearLista(). |
|  |

|  |
| --- |
| post: si la lista no esta vacia, elimina su nodo primero, sino no realiza |
|  |

|  |
| --- |
| accion alguna. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

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

|  |
| --- |
| void eliminarNodoPrimero(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : lista creada con crearLista(). |
|  |

|  |
| --- |
| post: si la lista no esta vacia elimina su nodo ultimo, |
|  |

|  |
| --- |
| sino no realiza accion. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

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

|  |
| --- |
| void eliminarNodoUltimo(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : lista creada con crearLista(). |
|  |

|  |
| --- |
| post: elimina todos los Nodos de la lista quedando destruida e inhabilitada |
|  |

|  |
| --- |
| para su uso. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| /\* 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(). |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| dato : elemento a localizar. |
|  |

|  |
| --- |
| return puntero al nodo localizado o fin(). |
|  |

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

|  |
| --- |
| PtrNodoLista localizarDato(Lista &lista , PtrDato ptrDato); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| 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. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| dato : elemento a insertar. |
|  |

|  |
| --- |
| return puntero al nodo insertado. |
|  |

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

|  |
| --- |
| PtrNodoLista insertarDato(Lista &lista, PtrDato ptrDato); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : la lista fue creada con crearLista(). |
|  |

|  |
| --- |
| post : elimina el dato de la lista, si el mismo se encuentra. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

|  |
| --- |
| dato : elemento a eliminar. |
|  |

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

|  |
| --- |
| void eliminarDato(Lista &lista, PtrDato ptrDato); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : la lista fue creada con crearLista(). |
|  |

|  |
| --- |
| post : reordena la lista. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

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

|  |
| --- |
| void reordenar(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
| pre : la lista fue creada con crearLista(). |
|  |

|  |
| --- |
| post : devuelve la cantidad de datos que tiene la lista. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lista : lista sobre la cual se invoca la primitiva. |
|  |

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

|  |
| --- |
| int longitud(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

#endif

**Lista.cpp**

|  |
| --- |
| /\* TDA Lista |
|  |

|  |
| --- |
| \* Implementación Simplemente Enlazada |
|  |

|  |
| --- |
| \* Archivo : Lista.cpp |
|  |

|  |
| --- |
| \* Versión : 1.1 |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include "Lista.h" |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <iomanip> |
|  |

|  |
| --- |
| #include "../Entidades/Sucursal.h" |
|  |

|  |
| --- |
| #include "../entidades/Provincia.h" |
|  |

|  |
| --- |
| #include "../entidades/Nucleo.h" |
|  |

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

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

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

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| /\* Implementación de Primitivas \*/ |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| void crearLista(Lista &lista, PFComparacion compara) { |
|  |

|  |
| --- |
| lista.primero = fin(); |
|  |

|  |
| --- |
| lista.compara = compara; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| bool listaVacia(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return (primero(lista) == fin()); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista fin() { |
|  |

|  |
| --- |
| return NULL; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista primero(Lista &lista) { |
|  |

|  |
| --- |
| return lista.primero; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista siguiente(Lista &lista, PtrNodoLista ptrNodo) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* verifica si la lista está vacia o si ptrNodo es el último \*/ |
|  |

|  |
| --- |
| if ((! listaVacia(lista)) && (ptrNodo != fin())) |
|  |

|  |
| --- |
| return ptrNodo->sgte; |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| return fin(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista anterior(Lista &lista, PtrNodoLista ptrNodo) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| PtrNodoLista ptrPrevio = fin(); |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| while (( ptrCursor != fin()) && (ptrCursor != ptrNodo)) { |
|  |

|  |
| --- |
| ptrPrevio = ptrCursor; |
|  |

|  |
| --- |
| ptrCursor = siguiente(lista,ptrCursor); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return ptrPrevio; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista ultimo(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* el último nodo de la lista es el anterior al fin() \*/ |
|  |

|  |
| --- |
| return anterior(lista,fin()); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista crearNodoLista(PtrDato ptrDato) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* reserva memoria para el nodo y luego completa sus datos \*/ |
|  |

|  |
| --- |
| PtrNodoLista ptrAux = new NodoLista; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ptrAux->ptrDato = ptrDato; |
|  |

|  |
| --- |
| ptrAux->sgte = fin(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return ptrAux; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarPrincipio(Lista &lista, PtrDato ptrDato) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* crea el nodo \*/ |
|  |

|  |
| --- |
| PtrNodoLista ptrNuevoNodo = crearNodoLista(ptrDato); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* lo incorpora al principio de la lista \*/ |
|  |

|  |
| --- |
| ptrNuevoNodo->sgte = lista.primero; |
|  |

|  |
| --- |
| lista.primero = ptrNuevoNodo; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return ptrNuevoNodo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarDespues(Lista &lista, PtrDato dato, PtrNodoLista ptrNodo) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| PtrNodoLista ptrNuevoNodo = fin(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* si la lista está vacia se adiciona la principio \*/ |
|  |

|  |
| --- |
| if (listaVacia(lista)) |
|  |

|  |
| --- |
| ptrNuevoNodo = adicionarPrincipio(lista,dato); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| else { |
|  |

|  |
| --- |
| if (ptrNodo != fin()) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* crea el nodo y lo intercala en la lista \*/ |
|  |

|  |
| --- |
| ptrNuevoNodo = crearNodoLista(dato); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ptrNuevoNodo->sgte = ptrNodo->sgte; |
|  |

|  |
| --- |
| ptrNodo->sgte = ptrNuevoNodo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return ptrNuevoNodo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarFinal(Lista &lista, PtrDato dato) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* adiciona el dato después del último nodo de la lista \*/ |
|  |

|  |
| --- |
| return adicionarDespues(lista,dato,ultimo(lista)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista adicionarAntes(Lista &lista, PtrDato dato, PtrNodoLista ptrNodo) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| PtrNodoLista ptrNuevoNodo = fin(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (! listaVacia(lista)) { |
|  |

|  |
| --- |
| if (ptrNodo != primero(lista)) |
|  |

|  |
| --- |
| ptrNuevoNodo = adicionarDespues(lista,dato,anterior(lista,ptrNodo)); |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| ptrNuevoNodo = adicionarPrincipio(lista,dato); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return ptrNuevoNodo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void eliminarNodo(Lista &lista, PtrNodoLista ptrNodo) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| PtrNodoLista ptrPrevio; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* verifica que la lista no esté vacia y que nodo no sea fin\*/ |
|  |

|  |
| --- |
| if ((! listaVacia(lista)) && (ptrNodo != fin())) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (ptrNodo == primero(lista)) |
|  |

|  |
| --- |
| lista.primero = siguiente(lista,primero(lista)); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| else { |
|  |

|  |
| --- |
| ptrPrevio = anterior( lista , ptrNodo ); |
|  |

|  |
| --- |
| ptrPrevio->sgte = ptrNodo->sgte; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| // Si el dato es un TDA, acá habría que llamar al destructor. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| delete ptrNodo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void eliminarNodoPrimero(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (! listaVacia(lista)) |
|  |

|  |
| --- |
| eliminarNodo(lista,primero(lista)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void eliminarNodoUltimo(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (! listaVacia(lista)) |
|  |

|  |
| --- |
| eliminarNodo(lista,ultimo(lista)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista localizarDato(Lista &lista, PtrDato ptrDato) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| bool encontrado = false; |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* recorre los nodos hasta llegar al último o hasta |
|  |

|  |
| --- |
| encontrar el nodo buscado \*/ |
|  |

|  |
| --- |
| while ((ptrCursor != fin()) && (! encontrado)) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (lista.compara(ptrCursor->ptrDato,ptrDato) == IGUAL) |
|  |

|  |
| --- |
| encontrado = true; |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| ptrCursor = siguiente(lista,ptrCursor); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* si no lo encontró devuelve fin \*/ |
|  |

|  |
| --- |
| if (! encontrado) |
|  |

|  |
| --- |
| ptrCursor = fin(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return ptrCursor; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| void eliminarDato(Lista &lista, PtrDato ptrDato) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* localiza el dato y luego lo elimina \*/ |
|  |

|  |
| --- |
| PtrNodoLista ptrNodo = localizarDato(lista,ptrDato); |
|  |

|  |
| --- |
| if (ptrNodo != fin()) |
|  |

|  |
| --- |
| eliminarNodo(lista,ptrNodo); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| PtrNodoLista insertarDato(Lista &lista, PtrDato ptrDato) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| PtrNodoLista ptrPrevio = primero(lista); |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
| PtrNodoLista ptrNuevoNodo; |
|  |

|  |
| --- |
| bool ubicado = false; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* recorre la lista buscando el lugar de la inserción \*/ |
|  |

|  |
| --- |
| while ((ptrCursor != fin()) && (! ubicado)) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (lista.compara(ptrCursor->ptrDato,ptrDato) == MAYOR) |
|  |

|  |
| --- |
| ubicado = true; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| else { |
|  |

|  |
| --- |
| ptrPrevio = ptrCursor; |
|  |

|  |
| --- |
| ptrCursor = siguiente(lista,ptrCursor); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (ptrCursor == primero(lista)) |
|  |

|  |
| --- |
| ptrNuevoNodo = adicionarPrincipio(lista,ptrDato); |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| ptrNuevoNodo = adicionarDespues(lista,ptrDato,ptrPrevio); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return ptrNuevoNodo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| void reordenar(Lista &lista) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Lista temp = lista; |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(temp); |
|  |

|  |
| --- |
| crearLista(lista, lista.compara); |
|  |

|  |
| --- |
| while ( ptrCursor != fin() ) { |
|  |

|  |
| --- |
| insertarDato( lista, ptrCursor->ptrDato ); |
|  |

|  |
| --- |
| eliminarNodo( temp, ptrCursor ); |
|  |

|  |
| --- |
| ptrCursor = primero(temp); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| eliminarListaSucursal( temp ); |
|  |

|  |
| --- |
| } |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| int longitud(Lista &lista){ |
|  |

|  |
| --- |
| PtrNodoLista ptrCursor = primero(lista); |
|  |

|  |
| --- |
| int longitud = 0; |
|  |

|  |
| --- |
| while ( ptrCursor != fin() ) { |
|  |

|  |
| --- |
| longitud++; |
|  |

|  |
| --- |
| ptrCursor = siguiente( lista, ptrCursor); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return longitud; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

**Sucursal.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <sstream> |
|  |

|  |
| --- |
| #include <stdio.h> |
|  |

|  |
| --- |
| #include <string> |
|  |

|  |
| --- |
| #include "Sucursal.h" |
|  |

|  |
| --- |
| #include "../Listas/Lista.h" |
|  |

|  |
| --- |
| #include <stdlib.h> |
|  |

|  |
| --- |
| using namespace std; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void crearSucursal(Sucursal &sucursal ,int codSucursal,string provincia, int cantArticulo, float monto, float cm2,int casaMatriz ){ |
|  |

|  |
| --- |
| sucursal.codSucursal = codSucursal; |
|  |

|  |
| --- |
| sucursal.provincia = provincia; |
|  |

|  |
| --- |
| sucursal.cantArticulo= cantArticulo; |
|  |

|  |
| --- |
| sucursal.monto = monto; |
|  |

|  |
| --- |
| sucursal.cm2 = cm2; |
|  |

|  |
| --- |
| sucursal.casaMatriz = casaMatriz; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void borrarSucursal(Sucursal \*sucursal) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| delete sucursal; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int getCodSucursal(Sucursal &sucursal){ |
|  |

|  |
| --- |
| return sucursal.codSucursal; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void setCodSucursal(Sucursal &sucursal, int codSucursal){ |
|  |

|  |
| --- |
| sucursal.codSucursal = codSucursal; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //char |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| string getProvincia(Sucursal &sucursal) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return sucursal.provincia; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void setProvincia(Sucursal &sucursal, string provincia) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| sucursal.provincia = provincia; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int getCantArticulo(Sucursal &sucursal) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return sucursal.cantArticulo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void setCantArtciulo(Sucursal &sucursal, int cantArticulo) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| sucursal.cantArticulo = cantArticulo; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| float getMonto(Sucursal &sucursal) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return sucursal.monto; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void setMonto(Sucursal &sucursal, float monto) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| sucursal.monto = monto; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| float getCm2(Sucursal &sucursal) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return sucursal.cm2; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void setCm2(Sucursal &sucursal, float cm2) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| sucursal.cm2 = cm2; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int getCasaMatriz(Sucursal &sucursal) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return sucursal.casaMatriz; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void setCasaMatriz(Sucursal &sucursal, int casaMatriz) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| sucursal.casaMatriz = casaMatriz; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void cargarSucursal(Lista &lista) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| FILE\* fSucursal; |
|  |

|  |
| --- |
| string lectura=""; |
|  |

|  |
| --- |
| int pos=0; |
|  |

|  |
| --- |
| //fSucursal = fopen("C:\\Users\\Equipo\\Documents\\AYED-TP\\archivo.txt","r"); |
|  |

|  |
| --- |
| fSucursal = fopen("archivos.txt","r"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| while(!feof(fSucursal)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| //leemos la linea |
|  |

|  |
| --- |
| leerLineaSucursal(&lectura, fSucursal); |
|  |

|  |
| --- |
| //cout<< lectura<< endl; |
|  |

|  |
| --- |
| //se obtiene el largo de la linea |
|  |

|  |
| --- |
| int largo = lectura.size(); |
|  |

|  |
| --- |
| //encontramos el primer guion |
|  |

|  |
| --- |
| pos = lectura.find('-'); |
|  |

|  |
| --- |
| //cortamos y casteamos desde el inicio de la linea hasta el guion |
|  |

|  |
| --- |
| int idSuc = atoi(lectura.substr(0,pos).c\_str()); |
|  |

|  |
| --- |
| //conservamos el resto de la linea a partir del primer guion hasta el final |
|  |

|  |
| --- |
| lectura= lectura.substr(pos+1,largo).c\_str(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*se repite el proceso anterior |
|  |

|  |
| --- |
| recorte de provincia |
|  |

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

|  |
| --- |
| pos = lectura.find('-'); |
|  |

|  |
| --- |
| string prov = lectura.substr(0,pos); |
|  |

|  |
| --- |
| lectura= lectura.substr(pos+1,largo).c\_str(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*recorte de cant articulo\*/ |
|  |

|  |
| --- |
| pos = lectura.find('-'); |
|  |

|  |
| --- |
| int cant = atoi(lectura.substr(0,pos).c\_str()); |
|  |

|  |
| --- |
| lectura= lectura.substr(pos+1,largo).c\_str(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*recorte de monto\*/ |
|  |

|  |
| --- |
| pos = lectura.find('-'); |
|  |

|  |
| --- |
| float mont = atof(lectura.substr(0,pos).c\_str()); |
|  |

|  |
| --- |
| lectura= lectura.substr(pos+1,largo).c\_str(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*centimetros cuadrados\*/ |
|  |

|  |
| --- |
| pos = lectura.find('-'); |
|  |

|  |
| --- |
| float cm = atof(lectura.substr(0,pos).c\_str()); |
|  |

|  |
| --- |
| lectura= lectura.substr(pos+1,largo).c\_str(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*casa matriz\*/ |
|  |

|  |
| --- |
| pos = lectura.find('-'); |
|  |

|  |
| --- |
| int casa = atoi(lectura.substr(0,pos).c\_str()); |
|  |

|  |
| --- |
| lectura= lectura.substr(pos+1,largo).c\_str(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // cout<< idSuc<< prov<< cant<< mont<< cm<< casa<<endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Sucursal \*sucursal = new Sucursal; |
|  |

|  |
| --- |
| crearSucursal(\*sucursal,idSuc,prov,cant,mont,cm,casa); |
|  |

|  |
| --- |
| // cout <<toString(\*sucursal)<<endl; |
|  |

|  |
| --- |
| adicionarPrincipio(lista,sucursal); |
|  |

|  |
| --- |
| lectura=""; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| fclose(fSucursal); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void leerLineaSucursal(string \*destino, FILE\* fSucursal) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| char buffer=0; |
|  |

|  |
| --- |
| int i; |
|  |

|  |
| --- |
| i=0; |
|  |

|  |
| --- |
| while (buffer!=10 && fread(&buffer,1,1,fSucursal)==1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(buffer!=10) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| // cout<< buffer<< endl; |
|  |

|  |
| --- |
| \*destino += buffer; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| i++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| string toString(Sucursal &sucursal) { |
|  |

|  |
| --- |
| string dato="NULL\n"; |
|  |

|  |
| --- |
| dato = "IdSuc: "+ std::to\_string(getCodSucursal(sucursal)) + |
|  |

|  |
| --- |
| " - Provincia: " + getProvincia(sucursal)+ |
|  |

|  |
| --- |
| " - CantArticulo: "+ std::to\_string(getCantArticulo(sucursal))+ |
|  |

|  |
| --- |
| " - Monto: "+ std::to\_string(getMonto(sucursal))+ |
|  |

|  |
| --- |
| " - Cm2: "+ std::to\_string(getCm2(sucursal))+ |
|  |

|  |
| --- |
| " - CasaMatriz: " + std::to\_string(getCasaMatriz(sucursal)); |
|  |

|  |
| --- |
| return dato; |
|  |

}

**Sucursal.h**

|  |
| --- |
| #ifndef SUCURSAL\_H\_INCLUDED |
|  |

|  |
| --- |
| #define SUCURSAL\_H\_INCLUDED |
|  |

|  |
| --- |
| #include <stdio.h> |
|  |

|  |
| --- |
| #include <stdlib.h> |
|  |

|  |
| --- |
| #include <string> |
|  |

|  |
| --- |
| #include "../Listas/Lista.h" |
|  |

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

|  |
| --- |
| Definicion de tipo de estructura Sucursal |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| typedef struct { |
|  |

|  |
| --- |
| int codSucursal; |
|  |

|  |
| --- |
| std::string provincia; |
|  |

|  |
| --- |
| int cantArticulo; |
|  |

|  |
| --- |
| float monto; |
|  |

|  |
| --- |
| float cm2; |
|  |

|  |
| --- |
| int casaMatriz; |
|  |

|  |
| --- |
| }Sucursal; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: Se recibe una instancia de sucursal |
|  |

|  |
| --- |
| post: Devuelve la instancia con toda la informacion de ella \*/ |
|  |

|  |
| --- |
| void crearSucursal(Sucursal &sucursal,int codSucursal,char provincia, int cantArticulo, float monto, int cm2,int casaMatriz ); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: Debe haber una sucursal creada |
|  |

|  |
| --- |
| post: elimina una sucursal existente \*/ |
|  |

|  |
| --- |
| void borrarSucursal(Sucursal \*sucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: DEVUELVE CODIGO DE SUCURSAL \*/ |
|  |

|  |
| --- |
| int getCodSucursal(Sucursal &sucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: ASIGNA EL VALOR A CODSUCURSAL A LA INSTANCIA DE SUCURSAL \*/ |
|  |

|  |
| --- |
| void setCodSucursal(Sucursal &sucursal, int codSucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: DEVUELVE LA PROVINCIA \*/ |
|  |

|  |
| --- |
| std::string getProvincia(Sucursal &sucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: ASIGNA EL VALOR A PROVINCIA A LA INSTANCIA DE SUCURSAL\*/ |
|  |

|  |
| --- |
| void setProvincia(Sucursal &sucursal, std::string provincia); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: DEVUELVE CANTIDAD DE ARTICULO \*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int getCantArticulo(Sucursal &sucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: ASIGNA EL VALOR A CANT ARTICULO A LA INSTANCIA DE SUCURSAL \*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void setCantArtciulo(Sucursal &sucursal, int cantArticulo); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: DEVUELVE MONTO \*/ |
|  |

|  |
| --- |
| float getMonto(Sucursal &sucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: ASIGNA EL VALOR A MONTO A LA INSTANCIA DE SUCURSAL \*/ |
|  |

|  |
| --- |
| void setMonto(Sucursal &sucursal, float monto); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: DEVUELVE cm2 \*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| float getCm2(Sucursal &sucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: ASIGNA EL VALOR A CM2 A LA INSTANCIA DE SUCURSAL \*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void setCm2(Sucursal &sucursal, float cm2); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: DEVUELVE Casa Matriz \*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int getCasaMatriz(Sucursal &sucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: LA INSTANCIA DE TDA SUCURSAL DEBE HABERSE CREADO PERO NO DEBE ESTAR DESTRUIDA. |
|  |

|  |
| --- |
| post: ASIGNA EL VALOR A CASA MATRIZ A LA INSTANCIA DE SUCURSAL \*/ |
|  |

|  |
| --- |
| void setCasaMatriz(Sucursal &sucursal, int casaMatriz); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: DEBE EXISTIR UN ARCHIVO DE LECTURA DE SUCURSALES Y UNA LISTA DE SUCURSALES VACIA |
|  |

|  |
| --- |
| post: LEER EL ARCHIVO, DEVOLVER UNA LISTA DE SUCURSALES CARGADA CON TODA LA INFORMACION\*/ |
|  |

|  |
| --- |
| void cargarSucursal(Lista &lista); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: TIENE QUE ESTAR ABIERTO EL ARCHIVO EN MODO LECTURA Y TENER UN STRING DONDE RECIBIR EL CONTENIDO |
|  |

|  |
| --- |
| post: DEVUELVE UNA LINEA DE LECTURA DEL ARCHIVO\*/ |
|  |

|  |
| --- |
| void leerLineaSucursal(std::string \*destino, FILE\* fSucursal); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* pre: EXISTIR UNA SUCURSAL CREADA Y CON DATOS |
|  |

|  |
| --- |
| post: DEVUELVE STRING CON TODOS LOS DATOS DE LA SUCURSAL\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::string toString(Sucursal &sucursal); |
|  |

|  |
| --- |
|  |
|  |

#endif // SUCURSAL\_H\_INCLUDED