Trabajo Práctico

“Gestión de Stock”

Segunda Entrega: “Presentación Final”

* Algoritmos y Estructuras de Datos
* Ing. Diego Azcurra – Ing. Damián Santos
* UNLA
* 2016

|  |  |  |
| --- | --- | --- |
| Padrón | Nombre | Evaluación Individual |
| 39.748.283 | Bustos, Rita Rocío de los Ángeles |  |
| 39.493.425 | Galvan, Gabriel Nazareno |  |
|  | Galarza, Diego |  |
| 34836313 | Luque Oviedo , Leandro Nicolas |  |
| Evaluación Trabajo |  | |

OPERACIONES A REALIZAR

**TP Gestión de Stock**

Operaciones a realizar:

1. Crear todas las estructuras a utilizar (todas las listas y pila)

Lista: Art.,Calle, Piso, etc

Pila:Ubicación

Esto sería de la parte 4Requerimientos (en el TP) el punto 1

Crear: el config, el de ingreso y solicitud

1. Leer el archivo Configuracion.conf y almacenarlo en una estructura (cargar los parámetros).
2. Cargar el Maestro de Articulos.
3. Manejo de Archivos en Paralelo (Archivos:Ingresos.db y Solicitudes.db). Se los

almacena y se los compara.

1. Procesamiento de Stock o Ingreso de Artículos (PUNTO 3 en la parte de Requerimientos en el TP)

* Determinar en primer lugar la ubicación en el depósito.

1. Si el artículo **ya se encuentra**, se buscará **almacenar la cantidad** solicitada en la ubicación ya utilizada.
2. Si es un **artículo nuevo**, se **solicitará una nueva ubicación**.

(*Recordar!! El orden de asignación de ubicaciones libres en las calles es de atrás hacia adelante y de abajo para arriba. Si no hay ubicaciones disponibles, se podrá construir una nueva calle.)*

Es decir:

* Se buscara primero por CALLE. La cual era una LISTA por lo que se recorre desde la primera a la última.
* Posteriormente se procederá a buscar por PISOS de la respectiva calle. Los cuales son trabajados como una LISTA, es decir se recorre desde el primer piso hasta el mayor.

Recordar lo del rango determinado por la Configuración

* Por último, la UBICACIÓN asignada siempre será la última ya que es una PILA DE UBICACIONES. Esta asignación será posible si se encuentra entre el rango disponible.
* Asimismo, se trabaja que ver cómo trabajar lo del indice.

1. Procesamiento Carga Sucursal o Retiro de Stock (PUNTO 2 en la parte de Requerimientos en el TP)

* Se realiza accediendo a la ubicación asignada al artículo en el depósito. (RECORDAR!! *“No se puede acceder a todas las calles (el acceso a cada calle es de tipo LIFO) para buscar un artículo, usar algún tipo de índice para que la determinación de la ubicación se realice en forma directa.”)*
* Posteriormente a su ubicación, se evalua el articulo pedido (Esto es lo que una vez hablamos cuando hicimos esa hoja borrador enfrente de la biblioteca).
* Si encuentra el artículo: Se evaluara, estimara o calculara la cantidad pedida respecto a la cantidad existente.
* Si es menor, simplemente le resta a la cantidad existente.
* Si es igual, se resta a la cantidad exitente y se elimina el artículo del depósito.
* Si es mayor, se realiza se resta a la cantidad existente (se entregaría la cantidad que hay), se eliminaría el artículo de Deposito y también se deja constancia en Faltantes.

En todos los casos se indica un registro nuevo en Ventas (para poder mostrar posteriormente el txt de Ventas) y se registra la carga en el Camión.El cual:

*“Si se supera el peso máximo, se deberá solicitar un camión nuevo. Si por ejemplo, debo despachar 1000 kg de un artículo y en el camión actual sólo me queda capacidad para 500 kg, despacho ese camión y solicito uno nuevo donde arranco la carga con los 1000 kg.”*

Es decir, se verificaba los kg de carga disponible y se evaluaba si al reparto lo hacia ese camión u otro:

* Si supera el peso máximo: se solicita un camión nuevo.
* Si no supera el peso máximo: realiza el pedido.
* Si le alcanza para llevar un porcentaje del pedido: lleva ese porcentaje y lo que falta lo lleva otro camión.

Cada camión enviado deberá ser registrado indicando el número, la carga total en kg y el porcentaje de ocupación (carga total / KC).

* Si no se encuentra el artículo o no se pudo completar la cantidad: Se deberá guardar la información en Faltantes y eliminar el artículo del Depósito.

1. Registrar la información en los archivos txt: (ordenar y crear)

* **Saldos detallados.txt:** (Nro Calle, piso, ubicación, codArticulo, cantidad)

Ordenado por Nro Calle asc, piso asc, ubicación asc.

* **Saldos agrupados.txt:.** (codArticulo, cantidad, Acumulado.)

Ordenado por cantidad del Artículo en forma descendente.

Acumulado es el peso acumulado de los artículos que se van presentando en el archivo.

* **Faltantes.txt:** reporte indicando los pedidos que no se pudieron cumplir en forma total o parcial. (codSucursal, codArticulo, cantidadOriginal, cantidadCumplida, %cumplimiento.)

Ordenado por codigoSucursal, codArticulo.

* **Ventas.txt:** (codSucursal, montoTotal)

Ordenado por montoTotal desc.

* **Utilización de camiones.** (nroCamion, cargaTotal, porcentajeOcupación.)

Ordenado por nroCamion desc.

1. Liberar memoria de las estructuras antes de cerrar.

ARCHIVOS DEL PROYECTO

Archivos.h

* **ARTDEPOSITO.h**

#ifndef \_\_ARTDEPOSITO\_H\_\_

#define \_\_ARTDEPOSITO\_H\_\_

#include "ListaPiso.h"

//TIPO DE INFORMACION CONTENIDA EN LA ESTRUCTURA NODOPISO IDENTIFICADA COMO PTRNODOPISODEP

typedef NodoPiso \*PtrNodoPisoDep;

//DEFINICION DE LOS TIPOS DE DATOS

typedef struct ArtDeposito{

int codArt;

PtrNodoPisoDep pisoDep;

};

//DEFINICION DE PRIMITIVAS

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia del TDA ArtDeposito no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo ArtDeposito.

\*POST-CONDICIONES: Instancia del TDA ArtDeposito creada y lista para ser utilizada. (VACIA)

\*ArtDeposito: Estructura sobre la cual se aplica la primitiva.

\*/

void crearArtDep(ArtDeposito &artDep);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo ArtDeposito, ademas de un entero y un puntero.

\*POST-CONDICIONES: Se creara la instancia de tipo ArtDeposito con los valores establecidos por el usuario.

\*ArtDeposito: Estructura en la cual se invoca la primitiva.

\*/

void crearArtDep(ArtDeposito &artDep,int codart,PtrNodoPisoDep pisoDep);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* DESTRUCTOR

\*PRE-CONDICIONES: La instancia TDA ArtDeposito tiene que estar creada y no tiene que estar destruida.

\*POST-CONDICION: La instancia TDA ArtDeposito sera destruida y ya no podra utilizarse.

\*ArtDeposito: Estructura sobre la cual se invoca la primitiva.

\*/

void destruirArtDep(ArtDeposito &artDep);

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

/\* SET DEL CODARTDEP

\*PRE-CONDICION:La instancia ArtDeposito debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo codart en ArtDeposito queda seteado con el codigo especificado.

\*ArtDeposito: instancia sobre la que se invoca la primitiva.

\*codart: valor a asignar de codArt del articulo.

\*/

void setCodArtDep(ArtDeposito &artDep,int codart);

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

/\* SET PISODEP

\*PRE-CONDICION:La instancia ArtDeposito debe estar creada y no debe estar destruida.

\*POST-CONDICION:El puntero piso en ArtDeposito queda seteado con el piso especificado.

\*ArtDeposito: instancia sobre la que se invoca la primitiva.

\*pisoDep: valor a asignar del piso.

\*/

void setPisoDep(ArtDeposito &artDep,PtrNodoPisoDep pisoDep);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* GET CODARTDEP

\*PRE-CONDICION: La instancia ArtDeposito debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo codart de Articulo.

\*ArtDeposito:Instancia sobre la que se invoca la primitiva.

\*/

int getCodArtDep(ArtDeposito &artDep);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* GET PISODEP

\*PRE-CONDICION: la instancia stock debe ser creado y no debe estar destruida..

\*POST-CONDICION: devuelve el puntero al piso donde del articulo indicado en ArtDeposito

\*ArtDeposito: instancia sobre la que se invoca la primitiva

\*/

PtrNodoPisoDep getPisoDep(ArtDeposito &artDep);

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

#endif // \_\_ARTDEPOSITO\_\_H\_\_

* **ARTICULOS.h**

//TDA DE LOS ARTICULOS

#ifndef \_\_ARTICULO\_H\_\_

#define \_\_ARTICULO\_H\_\_

//DEFINICION DE LOS TIPOS DE DATOS

typedef struct Articulo{

int codart;

float precio;

char descripcion[30];

};

//DEFINICION DE PRIMITIVAS

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia del TDA Articulo no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Articulo.

\*POST-CONDICIONES: Instancia del TDA Articulo creada y lista para ser utilizada. (VACIA)

\*Articulo: Estructura sobre la cual se aplica la primitiva.

\*/

void constructorArt(Articulo &articulo);

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Articulo, ademas de un char, un entero y un float.

\*POST-CONDICIONES: Se creara la instancia de tipo Articulo con los valores establecidos por el usuario.

\*Articulo: Estructura en la cual se invoca la primitiva.

\*/

void constructorArt(Articulo &articulo,int codart,float precio, char descripcion[]);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* SET DEL CODART

\*PRE-CONDICION:La instancia Articulo debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo codart en Articulo queda seteado con el codigo especificado.

\*Articulo: instancia sobre la que se invoca la primitiva.

\*codart: valor a asignar de codArt del articulo.

\*/

void setCodArt(Articulo &articulo,int codart);

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

/\* SET DEL PRECIO

\*PRE-CONDICION:La instancia Articulo debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo precio en Articulo queda seteado con el precio especificado.

\*Articulo: instancia sobre la que se invoca la primitiva.

\*precio: valor a asignar de precio del articulo.

\*/

void setPrecio(Articulo &articulo,float precio);

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

/\* SET DE DESCRIPCION

\*PRE-CONDICION:La instancia Articulo debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo descripcion en Articulo queda seteado con la descripcion especificada.

\*Articulo: instancia sobre la que se invoca la primitiva.

\*descripcion: valor a asignar de descripcion del articulo.

\*/

void setDescripcion(Articulo &articulo,char descripcion);

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

/\* GET CODART

\*PRE-CONDICION: La instancia articulo debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo codart de Articulo.

\*Articulo:Instancia sobre la que se invoca la primitiva.

\*/

int getCodArt(Articulo &articulo);

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

/\* GET PRECIO

\*PRE-CONDICION: La instancia articulo debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo precio de Articulo.

\*Articulo:Instancia sobre la que se invoca la primitiva.

\*/

float getPrecio(Articulo &articulo);

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

/\* GET DESCRIPCION

\*PRE-CONDICION: La instancia articulo debe ser creado y no debe estar destruida..

\*POST-CONDICION:Devuelve el atributo descripcion de Articulo

\*Articulo:Instancia sobre la que se invoca la primitiva

\*/

char\* getDescripcion(Articulo &articulo);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* DESTRUCTOR

\*PRE-CONDICIONES: La instancia TDA Articulo tiene que estar creada y no tiene que estar destruida.

\* POST-CONDICION: La instancia TDA Articulo sera destruida y ya no podra utilizarse.

\* Articulo: Estructura sobre la cual se invoca la primitiva.

\*/

void destructorArt(Articulo &articulo);

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

#endif // \_\_ARTICULO\_H\_\_

* **ARTUBICACION.h**

#ifndef \_\_ARTUBICACION\_H\_\_

#define \_\_ARTUBICACION\_H\_\_

#include "Articulo.h"

//TIPO DE INFORMACION CONTENIDA EN LA ESTRUCTURA ARTICULO IDENTIFICADA COMO ART

typedef Articulo Art;

//DEFINICION DE LOS TIPOS DE DATOS

typedef struct ArtUbicacion{

Art datoArt;

int c;

int p;

int u;

float cantArtUb;

};

//DEFINICION DE PRIMITIVAS

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia del TDA ArtUbicacion no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Artubicacion.

\*POST-CONDICIONES: Instancia del TDA ArtUbicacion creada y lista para ser utilizada. (VACIA)

\*ArtUbicacion: Estructura sobre la cual se aplica la primitiva.

\*/

void constructorArtUb(ArtUbicacion &artUbicacion);

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo ArtUbicacion, ademas de tres enteros y un float.

\*POST-CONDICIONES: Se creara la instancia de tipo ArtUbicacion con los valores establecidos por el usuario.

\*ArtUbicacion: Estructura en la cual se invoca la primitiva.

\*/

void constructorArtUb(ArtUbicacion &artUbicacion,int c,int p,int u,float cantArtUb);

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo ArtUbicacion, ademas de tres enteros y un float.

\*POST-CONDICIONES: Se creara la instancia de tipo ArtUbicacion con los valores establecidos por el usuario y un articulo.

\*ArtUbicacion: Estructura en la cual se invoca la primitiva.

\*/

void constructorArtUb(ArtUbicacion &artUbicacion,Art &datoArtaux,int c,int p,int u,float cantArtUb);

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

/\* SET C

\*PRE-CONDICION:La instancia ArtUbicacion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo c en ArtUbicacion queda seteado con el c (calle) especificado.

\*ArtUbicacion: instancia sobre la que se invoca la primitiva.

\*c: valor a asignar de c (calle).

\*/

void setC(ArtUbicacion &artUbicacion,int c);

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

/\* SET P

\*PRE-CONDICION:La instancia ArtUbicacion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo p en ArtUbicacion queda seteado con el p (piso) especificado.

\*ArtUbicacion: instancia sobre la que se invoca la primitiva.

\*p: valor a asignar de p(piso).

\*/

void setP(ArtUbicacion &artUbicacion,int p);

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

/\* SET U

\*PRE-CONDICION:La instancia ArtUbicacion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo u en ArtUbicacion queda seteado con el u (ubicacion) especificado.

\*ArtUbicacion: instancia sobre la que se invoca la primitiva.

\*u: valor a asignar de u(ubicacion).

\*/

void setU(ArtUbicacion &artUbicacion,int u);

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

/\* SET DATOART

\*PRE-CONDICION:La instancia ArtUbicacion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo datoArt en ArtUbicacion queda seteado con el datoArt especificado.

\*ArtUbicacion: instancia sobre la que se invoca la primitiva.

\*datoArt: valor a asignar de datoArt.

\*/

void setDatoArt(ArtUbicacion &artUbicacion,Art &datoArt);

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

/\* SET CANTARTUB

\*PRE-CONDICION:La instancia ArtUbicacion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo cantArtUb en ArtUbicacion queda seteado con el cantArtUb especificado.

\*ArtUbicacion: instancia sobre la que se invoca la primitiva.

\*cantArtUb: valor a asignar de cantArtUb.

\*/

void setCantArtUb(ArtUbicacion &artUbicacion,float cantArtUb);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* GET C

\*PRE-CONDICION: La instancia ArtDUbicacion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo c.

\*ArtDeposito:Instancia sobre la que se invoca la primitiva.

\*/

int getC(ArtUbicacion &artUbicacion);

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

/\* GET P

\*PRE-CONDICION: La instancia ArtDUbicacion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo p.

\*ArtDeposito:Instancia sobre la que se invoca la primitiva.

\*/

int getP(ArtUbicacion &artUbicacion);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* GET U

\*PRE-CONDICION: La instancia ArtDUbicacion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo u.

\*ArtDeposito:Instancia sobre la que se invoca la primitiva.

\*/

int getU(ArtUbicacion &artUbicacion);

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

/\* GET DATOART

\*PRE-CONDICION: La instancia ArtDUbicacion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo datoart.

\*ArtDeposito:Instancia sobre la que se invoca la primitiva.

\*/

Art getDatoArt(ArtUbicacion &artUbicacion);

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

/\* GET CANTARTUB

\*PRE-CONDICION: La instancia ArtDUbicacion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo cantArtUb.

\*ArtDeposito:Instancia sobre la que se invoca la primitiva.

\*/

float getCantArtUb(ArtUbicacion &artUbicacion);

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

/\* DESTRUCTOR

\*PRE-CONDICIONES: La instancia TDA ArtUbicacion tiene que estar creada y no tiene que estar destruida.

\*POST-CONDICION: La instancia TDA ArtUbicacion sera destruida y ya no podra utilizarse.

\*ArtUbicacion: Estructura sobre la cual se invoca la primitiva.

\*/

void destructorArtUb(ArtUbicacion &artUbicacion);

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

#endif // \_\_ARTUBICACION\_H\_\_

* **CAMION.h**

#ifndef \_\_CAMION\_H\_\_

#define \_\_CAMION\_H\_\_

//DEFINICION DE LOS TIPOS DE DATOS

typedef struct Camion{

int id\_camion;

float carga\_total;

float porcentaje;

};

//DEFINICION DE PRIMITIVAS

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia del TDA Camion no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Camion.

\*POST-CONDICIONES: Instancia del TDA Camion creada y lista para ser utilizada. (VACIA)

\*Camion: Estructura sobre la cual se aplica la primitiva.

\*/

void crearCamion(Camion &camion);

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Camion, ademas de un entero y tres float.

\*POST-CONDICIONES: Se creara la instancia de tipo Camion con los valores establecidos por el usuario.

\*Camion: Estructura en la cual se invoca la primitiva.

\*/

void crearCamion(Camion &camion,int idCamion,float Ct,float porc);

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

/\* DESTRUCTOR

\*PRE-CONDICIONES: La instancia TDA Camion tiene que estar creada y no tiene que estar destruida.

\*POST-CONDICION: La instancia TDA Camion sera destruida y ya no podra utilizarse.

\*Camion: Estructura sobre la cual se invoca la primitiva.

\*/

void destruirCamion(Camion &camion);

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

/\* GET GETID

\*PRE-CONDICION: La instancia Camion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo id.

\*Camion:Instancia sobre la que se invoca la primitiva.

\*/

int getId\_Camion(Camion &camion);

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

/\* GET CARGA\_T

\*PRE-CONDICION: La instancia Camion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo carga\_total.

\*Camion:Instancia sobre la que se invoca la primitiva.

\*/

float getCarga\_T(Camion &camion);

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

/\* GET PORCENTAJE

\*PRE-CONDICION: La instancia Camion debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo porcentaje.

\*Camion:Instancia sobre la que se invoca la primitiva.

\*/

float getPorcentaje(Camion &camion);

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

/\* SET ID\_CAMION

\*PRE-CONDICION:La instancia Camion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo idcamion en Camion queda seteado con el idcamion especificado.

\*Camion: instancia sobre la que se invoca la primitiva.

\*idcamion: valor a asignar de idcamion.

\*/

void setId\_Camion(Camion &camion,int idCamion);

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

/\* SET CARGA\_T

\*PRE-CONDICION:La instancia Camion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo ct en Camion queda seteado con la carga total especificada.

\*Camion: instancia sobre la que se invoca la primitiva.

\*ct: valor a asignar de ct.

\*/

void setCarga\_T(Camion &camion,float cT);

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

/\* SET PORCENTAJE

\*PRE-CONDICION:La instancia Camion debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo porc en Camion queda seteado con el porcentaje especificado.

\*Camion: instancia sobre la que se invoca la primitiva.

\*porc: valor a asignar de porcentaje.

\*/

void setPorcentaje(Camion &camion,float porc);

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

#endif // \_\_CAMION\_H\_\_

* **CONFIGURACION.h**

#ifndef \_\_CONFIGURACION\_H\_\_

#define \_\_CONFIGURACION\_H\_\_

//DEFINICION DE LOS TIPOS DE DATOS

typedef struct Config{

int configP;

int configU;

float configKC;

};

//DEFINICION DE PRIMITIVAS

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia del TDA Config no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Config.

\*POST-CONDICIONES: Instancia del TDA Config creada y lista para ser utilizada. (VACIA)

\*Config: Estructura sobre la cual se aplica la primitiva.

\*/

void constructorConfig(Config &config);

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Config, ademas de dos entero y un float.

\*POST-CONDICIONES: Se creara la instancia de tipo Config con los valores establecidos por el usuario.

\*Config: Estructura en la cual se invoca la primitiva.

\*/

void constructorConfig(Config &config,int p,int u, float kc);

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

/\* SET CONFIGP

\*PRE-CONDICION:La instancia Config debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo p en Config queda seteado con la cantidad de pisos especificado.

\*Config: instancia sobre la que se invoca la primitiva.

\*p: valor a asignar de pisos.

\*/

void setConfigP(Config &config,int p);

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

/\* SET CONFIGU

\*PRE-CONDICION:La instancia Config debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo u en Config queda seteado con la cantidad de ubicaciones especificado.

\*Config: instancia sobre la que se invoca la primitiva.

\*u: valor a asignar de ubicaciones.

\*/

void setConfigU(Config &config,int u);

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

/\* SET CONFIGKC

\*PRE-CONDICION:La instancia Config debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo kc en Config queda seteado con la cantidad de kilos por camion especificado.

\*Config: instancia sobre la que se invoca la primitiva.

\*kc: valor a asignar de kilos por camion.

\*/

void setConfigKC(Config &config,float kc);

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

/\* GET CONFIGP

\*PRE-CONDICION: La instancia Config debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo p.

\*Config:Instancia sobre la que se invoca la primitiva.

\*/

int getConfigP(Config &config);

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

/\* GET CONFIGU

\*PRE-CONDICION: La instancia Config debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo u.

\*Config:Instancia sobre la que se invoca la primitiva.

\*/

int getConfigU(Config &config);

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

/\* GET CONFIGKC

\*PRE-CONDICION: La instancia Config debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo kc.

\*Config:Instancia sobre la que se invoca la primitiva.

\*/

float getConfigKC(Config &config);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* DESTRUCTOR

\*PRE-CONDICIONES: La instancia TDA Config tiene que estar creada y no tiene que estar destruida.

\*POST-CONDICION: La instancia TDA Config sera destruida y ya no podra utilizarse.

\*Config: Estructura sobre la cual se invoca la primitiva.

\*/

void destructorConfig(Config &config);

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

#endif // \_\_CONFIGURACION\_H\_\_

* **FALTANTES.h**

#ifndef \_\_FALTANTES\_H\_\_

#define \_\_FALTANTES\_H\_\_

//DEFINICION DE LOS TIPOS DE DATOS

typedef struct Faltantes{

int codsucursal;

int codArt;

float cantTot;

float cantEnv;

int porc;

};

//DEFINICION DE PRIMITIVAS

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia del TDA Faltantes no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Faltantes.

\*POST-CONDICIONES: Instancia del TDA Faltantes creada y lista para ser utilizada. (VACIA)

\*Faltantes: Estructura sobre la cual se aplica la primitiva.

\*/

void crearFaltante(Faltantes &faltantes);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Faltantes, ademas de tres enteros y dos float.

\*POST-CONDICIONES: Se creara la instancia de tipo Faltantes con los valores establecidos por el usuario.

\*Faltantes: Estructura en la cual se invoca la primitiva.

\*/

void crearFaltante(Faltantes &faltantes,int codSucursal,int codart,float cantot,float cantenv,int porc);

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

/\* DESTRUCTOR

\*PRE-CONDICIONES: La instancia TDA Faltantes tiene que estar creada y no tiene que estar destruida.

\* POST-CONDICION: La instancia TDA Faltantes sera destruida y ya no podra utilizarse.

\* Faltantes: Estructura sobre la cual se invoca la primitiva.

\*/

void destructorFaltantes(Faltantes &faltantes);

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

/\* GET CODART

\*PRE-CONDICION: La instancia Faltantes debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo codart de Faltantes.

\*Faltantes:Instancia sobre la que se invoca la primitiva.

\*/

int getCodArt(Faltantes &faltantes);

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

/\* GET CODSUCURSAL

\*PRE-CONDICION: La instancia Faltantes debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo codsucursal de Faltantes.

\*Faltantes:Instancia sobre la que se invoca la primitiva.

\*/

int getCodsucursal(Faltantes &faltantes);

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

/\* GET CANTTOT

\*PRE-CONDICION: La instancia Faltantes debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo cantTot de Faltantes.

\*Faltantes:Instancia sobre la que se invoca la primitiva.

\*/

float getCantTot(Faltantes &faltantes);

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

/\* GET CANTENV

\*PRE-CONDICION: La instancia Faltantes debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo cantenv de Faltantes.

\*Faltantes:Instancia sobre la que se invoca la primitiva.

\*/

float getCantEnv(Faltantes &faltantes);

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

/\* GET PORC

\*PRE-CONDICION: La instancia Faltantes debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo porc de Faltantes.

\*Faltantes:Instancia sobre la que se invoca la primitiva.

\*/

int getPorc(Faltantes &faltantes);

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

/\* SET DEL CODSUCURSAL

\*PRE-CONDICION:La instancia Faltantes debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo codSucursal en Faltantes queda seteado con el codsucursal especificado.

\*Faltantes: instancia sobre la que se invoca la primitiva.

\*codsucursal: valor a asignar de codSucursal.

\*/

void setCodsuccursal(Faltantes &faltante,int codSucursal);

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

/\* SET DEL CODART

\*PRE-CONDICION:La instancia Faltantes debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo codart en Faltantes queda seteado con el codart especificado.

\*Faltantes: instancia sobre la que se invoca la primitiva.

\*codart: valor a asignar de codArt.

\*/

void setCodArt(Faltantes &faltantes,int codart);

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

/\* SET DEL CANTTOT

\*PRE-CONDICION:La instancia Faltantes debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo cantot en Faltantes queda seteado con el canttot especificado.

\*Faltantes: instancia sobre la que se invoca la primitiva.

\*cantot: valor a asignar de cantot.

\*/

void setCantTot(Faltantes &faltantes,float cantot);

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

/\* SET DE CANTENV

\*PRE-CONDICION:La instancia Faltantes debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo cantenv en Faltantes queda seteado con el cantenv especificado.

\*Faltantes: instancia sobre la que se invoca la primitiva.

\*cantenv: valor a asignar de cantenv.

\*/

void setCantEnv(Faltantes &faltantes,float cantenv);

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

/\* SET DEL PORC

\*PRE-CONDICION:La instancia Faltantes debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo porc en Faltantes queda seteado con el porc especificado.

\*Faltantes: instancia sobre la que se invoca la primitiva.

\*porc: valor a asignar de porc.

\*/

void setPorc(Faltantes &faltantes,int porc);

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

#endif // \_\_FALTANTES\_H\_\_

* **FUNYPROD.h**

#ifndef \_\_CONFIGURACION\_H\_\_

#define \_\_CONFIGURACION\_H\_\_

//DEFINICION DE LOS TIPOS DE DATOS

typedef struct Config{

int configP;

int configU;

float configKC;

};

//DEFINICION DE PRIMITIVAS

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

/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia del TDA Config no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Config.

\*POST-CONDICIONES: Instancia del TDA Config creada y lista para ser utilizada. (VACIA)

\*Config: Estructura sobre la cual se aplica la primitiva.

\*/

void constructorConfig(Config &config);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* CONSTRUCTOR

\*PRE-CONDICIONES: La instancia no debe estar creada con anterioridad.

Constructor solo acepta instancias de tipo Config, ademas de dos entero y un float.

\*POST-CONDICIONES: Se creara la instancia de tipo Config con los valores establecidos por el usuario.

\*Config: Estructura en la cual se invoca la primitiva.

\*/

void constructorConfig(Config &config,int p,int u, float kc);

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

/\* SET CONFIGP

\*PRE-CONDICION:La instancia Config debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo p en Config queda seteado con la cantidad de pisos especificado.

\*Config: instancia sobre la que se invoca la primitiva.

\*p: valor a asignar de pisos.

\*/

void setConfigP(Config &config,int p);

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

/\* SET CONFIGU

\*PRE-CONDICION:La instancia Config debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo u en Config queda seteado con la cantidad de ubicaciones especificado.

\*Config: instancia sobre la que se invoca la primitiva.

\*u: valor a asignar de ubicaciones.

\*/

void setConfigU(Config &config,int u);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* SET CONFIGKC

\*PRE-CONDICION:La instancia Config debe estar creada y no debe estar destruida.

\*POST-CONDICION:El atributo kc en Config queda seteado con la cantidad de kilos por camion especificado.

\*Config: instancia sobre la que se invoca la primitiva.

\*kc: valor a asignar de kilos por camion.

\*/

void setConfigKC(Config &config,float kc);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* GET CONFIGP

\*PRE-CONDICION: La instancia Config debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo p.

\*Config:Instancia sobre la que se invoca la primitiva.

\*/

int getConfigP(Config &config);

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

/\* GET CONFIGU

\*PRE-CONDICION: La instancia Config debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo u.

\*Config:Instancia sobre la que se invoca la primitiva.

\*/

int getConfigU(Config &config);

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

/\* GET CONFIGKC

\*PRE-CONDICION: La instancia Config debe ser creado y no debe estar destruida.

\*POST-CONDICION:Devuelve el atributo kc.

\*Config:Instancia sobre la que se invoca la primitiva.

\*/

float getConfigKC(Config &config);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* DESTRUCTOR

\*PRE-CONDICIONES: La instancia TDA Config tiene que estar creada y no tiene que estar destruida.

\*POST-CONDICION: La instancia TDA Config sera destruida y ya no podra utilizarse.

\*Config: Estructura sobre la cual se invoca la primitiva.

\*/

void destructorConfig(Config &config);

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

#endif // \_\_CONFIGURACION\_H\_\_

* **LDEPOSITO.h**

#ifndef \_\_LDEPOSITO\_H\_\_

#define \_\_LDEPOSITO\_H\_\_

#include "ArtDeposito.h"

#ifndef NULL

#define NULL 0

#endif

//TIPO DE INFORMACION CONTENIDA EN LA ESTRUCTURA ARTDEPOSITO IDENTIFICADA COMO ARTDEP

typedef ArtDeposito ArtDep;

//TIPO DE ESTRUCTURA DE LOS NODOS DE LA LISTA

struct NodoListaDeposito{

ArtDep datoArtDep;

NodoListaDeposito \*sgteAD;

};

// TIPO DE PUNTERO A LOS NODOS DE LA LISTA, EL CUAL SE USA PARA RECORRER LA LISTA Y ACCEDER A SUS DATOS

typedef NodoListaDeposito \*PtrNodoListaDeposito;

// TIPO DE ESTRUCTURA DE LA LISTA

struct ListaArtDeposito{

PtrNodoListaDeposito primerArtDeposito;

};

//DEFINICION DE PRIMITIVAS

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

/\* CREAR LISTA

\*PRE-CONDICION:La lista no debe haber sido creada.

\*POST-CONDICION:La lista queda creada y preparada para ser usada.

\*listaArtDep : estructura de datos a ser creado.

\*/

void crearListaArtDeposito(ListaArtDeposito &listaArtDep);

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

/\* LISTA VACIA

\*PRE-CONDICION:Lista Creada con crearListaArtDeposito().

\*POST-CONDICION: Devuelve true si lista esta vacia, sino devuelve false.

\*listaArtDep: lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(ListaArtDeposito &listaArtDep);

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

/\* FINAD

\*PRE-CONDICION: Lista Creada con crearListaArtDeposito().

\*POST-CONDICION: 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.

\*/

PtrNodoListaDeposito finAD();

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

/\* PRIMERO

\*PRE-CONDICION: Lista Creada con crearListaArtDeposito().

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

\*listaArtDep : lista sobre la cual se invoca la primitiva.

\*return puntero al primer nodo.

\*/

PtrNodoListaDeposito primero(ListaArtDeposito &listaArtDep);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* SIGUIENTE

PRE-CONDICION: Lista Creada con crearListaArtDep().

POST-CONDICION: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.

\*prtNodoListaDep : puntero al nodo a partir del cual se requiere el siguiente.

\*return puntero al nodo siguiente.

\*/

PtrNodoListaDeposito siguiente(ListaArtDeposito &listaArtDep,PtrNodoListaDeposito ptrNodoListaDep);

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

/\* ANTERIOR

\*PRE-CONDICION:Lista Creada con crearListaArtDep().

PtrNodoListaDep es un puntero a un nodo de lista.

\*POST-CONDICION: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si

ptrNodo apuntaba al primero o si lista esta vacia.

\*listaArtDep :Lista sobre la cual se invoca la primitiva.

\*prtNodoListaDep : puntero al nodo a partir del cual se requiere el anterior.

\*return puntero al nodo anterior.

\*/

PtrNodoListaDeposito anterior(ListaArtDeposito &listaArtDep,PtrNodoListaDeposito ptrNodoListaDep);

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

/\* ULTIMO

\*PRE-CONDICION: lista creada con crearListaArtDep().

\*POST-CONDICION: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si ista esta vacia.

\*listaArtDep : lista sobre la cual se invoca la primitiva.

\*return puntero al último nodo.

\*/

PtrNodoListaDeposito ultimo(ListaArtDeposito &listaArtDep);

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

PtrNodoListaDeposito crearNodoLista(ArtDep datoArtDep);

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

/\* ADICIONAR AL PRINCIPIO

\*PRE-CONDICION: lista creada con crearListaArtDep().

\*POST-CONDICION: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

\*listaArtDep: lista sobre la cual se invoca la primitiva.

\*datoArtDep: elemento a adicionar al principio de la lista.

\*return puntero al nodo adicionado.

\*/

PtrNodoListaDeposito adicionarPrincipio(ListaArtDeposito &listaArtDep,ArtDep datoArtDep);

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

/\* ADICIONAR DESPUES

\*PRE-CONDICION: lista creada con crearListaArtDep().

\*POST-CONDICION: 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().

\*listaArtDep : lista sobre la cual se invoca la primitiva.

\*datoArtDep: elemento a adicionar.

\*ptrNodoListaDep: puntero al nodo después del cual se quiere adicionar el dato.

\*return puntero al nodo adicionado.

\*/

PtrNodoListaDeposito adicionarDespues(ListaArtDeposito &listaArtDep,ArtDep datoArtDep, PtrNodoListaDeposito ptrNodoListaDep);

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

/\* ADICIONAR FINAL

\*PRE-CONDICION: lista creada con crearListaArtDep().

\*POST-CONDICION: agrega un nodo al final de la lista con el dato proporcionado y devuelve un puntero al nodo insertado.

\*listaArtDep : lista sobre la cual se invoca la primitiva.

\*datoArtDep : elemento a adicionar al final de la lista.

\*return puntero al nodo adicionado.

\*/

PtrNodoListaDeposito adicionarFinal(ListaArtDeposito &listaArtDep, ArtDep datoArtDep);

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

/\* COLOCAR DATO

\*PRE-CONDICION: lista creada con crearListaArtDep(), no vacia. ptrNodo es distinto de fin().

\*POST-CONDICION: coloca el dato proporcionado en el nodo apuntado por ptrNodo.

\*lista : lista sobre la cual se invoca la primitiva.

\*dato : elemento a colocar.

\*ptrNodo : puntero al nodo del cual se quiere colocar el dato.

\*/

void colocarDato(ListaArtDeposito &listaArtDep,ArtDep &datoArtDep, PtrNodoListaDeposito ptrNodoListaDep);

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

/\* OBTENER DATO

\*PRE-CONDICION: lista creada con crearListaArtDep(), no vacia. ptrNodo es distinto de fin().

\*POST-CONDICION: devuelve el dato del nodo apuntado por ptrNodo.

\*listaArtDep: lista sobre la cual se invoca la primitiva.

\*datoArtDep: elemento obtenido.

\*ptrNodoListaDep : puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDato(ListaArtDeposito &listaArtDep,ArtDep &datoArtDep, PtrNodoListaDeposito ptrNodoListaDep);

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

/\* ELIMINAR NODO

\*PRE-CONDICION: lista creada con crearListaArtDep().

\*POST-CONDICION: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista

esta vacia o si ptrNodo apunta a fin().

\*listaArtDep: lista sobre la cual se invoca la primitiva.

\*ptrNodoListaDep: puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(ListaArtDeposito &listaArtDep,PtrNodoListaDeposito ptrNodoListaDep);

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

/\* ELIMINAR NODO PRIMERO

\*PRE-CONDICION:Lista creada con crearListaArtDep().

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

\*listaArtDep : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(ListaArtDeposito &listaArtDep);

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

/\* ELIMINAR NODO ULTIMO

\*PRE-CONDICION: lista creada con crearListaArtDep().

\*POST-CONDICION: si la lista no esta vacia elimina su nodo ultimo, sino no realiza accion.

\*listaArtDep : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(ListaArtDeposito &listaArtDep);

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

/\* ELIMINAR LISTA

\*PRE-CONDICION:Lista creada con crearListaArtDep().

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

\*listaArtDep: lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(ListaArtDeposito &listaArtDep);

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

#endif // \_\_LDEPOSITO\_H\_\_

* **LISTAARTICULO.h**

#ifndef \_\_LISTAARTICULO\_H\_\_

#define \_\_LISTAARTICULO\_H\_\_

#include "Articulo.h"

#ifndef NULL

#define NULL 0

#endif

//TIPO DE INFORMACION CONTENIDA EN LA ESTRUCTURA ARTICULO IDENTIFICADA COMO LART

typedef Articulo LArt;

//TIPO DE ESTRUCTURA DE LOS NODOS DE LA LISTA

struct NodoListaArt{

LArt datoLArt;

NodoListaArt \*sgteArt;

};

// TIPO DE PUNTERO A LOS NODOS DE LA LISTA, EL CUAL SE USA PARA RECORRER LA LISTA Y ACCEDER A SUS DATOS

typedef NodoListaArt \*PrtNodoListaArt;

// TIPO DE ESTRUCTURA DE LA LISTA

struct ListaArt{

PrtNodoListaArt primeroListaArt;

};

//DEFINICION DE PRIMITIVAS

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

/\* CREAR LISTA ART

\*PRE-CONDICION:La lista no debe haber sido creada.

\*POST-CONDICION:La lista queda creada y preparada para ser usada.

\*listaA : estructura de datos a ser creado.

\*/

void crearListaArt(ListaArt &listaA);

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

/\* LISTA VACIA

\*PRE-CONDICION:Lista Creada con crearListaArt().

\*POST-CONDICION: Devuelve true si lista esta vacia, sino devuelve false.

\*listaA: lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(ListaArt &listaA);

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

/\* FINA

\*PRE-CONDICION: Lista Creada con crearListaArt().

\*POST-CONDICION: 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.

\*/

PrtNodoListaArt finA();

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

/\* PRIMERO

\*PRE-CONDICION: Lista Creada con crearListaArt().

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

\*listaA : lista sobre la cual se invoca la primitiva.

\*return puntero al primer nodo.

\*/

PrtNodoListaArt primero(ListaArt &listaA);

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

/\* SIGUIENTE

PRE-CONDICION: Lista Creada con crearListaArt().

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

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

\*listaA : lista sobre la cual se invoca la primitiva.

\*prtNodoListaArt : puntero al nodo a partir del cual se requiere el siguiente.

\*return puntero al nodo siguiente.

\*/

PrtNodoListaArt siguiente(ListaArt &listaA,PrtNodoListaArt ptrNodoListaArt);

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

/\* ANTERIOR

\*PRE-CONDICION:Lista Creada con crearListaArt().

PtrNodoListaArt es un puntero a un nodo de lista.

\*POST-CONDICION: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si

ptrNodo apuntaba al primero o si lista esta vacia.

\*listaA :Lista sobre la cual se invoca la primitiva.

\*prtNodoListaArt : puntero al nodo a partir del cual se requiere el anterior.

\*return puntero al nodo anterior.

\*/

PrtNodoListaArt anterior(ListaArt &listaA,PrtNodoListaArt ptrNodoListaArt);

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

/\* ULTIMO

\*PRE-CONDICION: lista creada con crearListaArt().

\*POST-CONDICION: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si ista esta vacia.

\*listaA : lista sobre la cual se invoca la primitiva.

\*return puntero al último nodo.

\*/

PrtNodoListaArt ultimo(ListaArt &listaA);

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

PrtNodoListaArt crearNodoLista(LArt datoArticulo);

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

/\* ADICIONAR AL PRINCIPIO

\*PRE-CONDICION: lista creada con crearListaArt().

\*POST-CONDICION: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

\*listaA: lista sobre la cual se invoca la primitiva.

\*datoArticulo: elemento a adicionar al principio de la lista.

\*return puntero al nodo adicionado.

\*/

PrtNodoListaArt adicionarPrincipio(ListaArt &listaA,LArt datoArticulo);

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

/\* ADICIONAR DESPUES

\*PRE-CONDICION: lista creada con crearListaArt().

\*POST-CONDICION: agrega un nodo despues del apuntado por ptrNodoListaArt 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().

\*listaArt : lista sobre la cual se invoca la primitiva.

\*datoArticulo: elemento a adicionar.

\*ptrNodoListaArt: puntero al nodo después del cual se quiere adicionar el dato.

\*return puntero al nodo adicionado.

\*/

PrtNodoListaArt adicionarDespues(ListaArt &listaA,LArt datoArticulo, PrtNodoListaArt ptrNodoListaArt);

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

/\* ADICIONAR FINAL

\*PRE-CONDICION: lista creada con crearListaArt().

\*POST-CONDICION: agrega un nodo al final de la lista con el dato proporcionado y devuelve un puntero

al nodo insertado.

\*listaA: lista sobre la cual se invoca la primitiva.

\*datoArticulo : elemento a adicionar al final de la lista.

\*return puntero al nodo adicionado.

\*/

PrtNodoListaArt adicionarFinal(ListaArt &listaA, LArt datoArticulo);

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

/\* COLOCAR DATO

\*PRE-CONDICION: lista creada con crearListaArt(), no vacia. ptrNodo es distinto de fin().

\*POST-CONDICION: coloca el dato proporcionado en el nodo apuntado por ptrNodoListaArt.

\*lista : lista sobre la cual se invoca la primitiva.

\*dato : elemento a colocar.

\*ptrNodoListaArt : puntero al nodo del cual se quiere colocar el dato.

\*/

void colocarDato(ListaArt &listaA,LArt &datoArticulo, PrtNodoListaArt ptrNodoListaArt);

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

/\* OBTENER DATO

\*PRE-CONDICION: lista creada con crearListaArt(), no vacia. ptrNodo es distinto de fin().

\*POST-CONDICION: devuelve el dato del nodo apuntado por ptrNodo.

\*listaA: lista sobre la cual se invoca la primitiva.

\*datoArticulo: elemento obtenido.

\*ptrNodoListaArt : puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDato(ListaArt &listaA,LArt &datoArticulo, PrtNodoListaArt ptrNodoListaArt);

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

/\* ELIMINAR NODO

\*PRE-CONDICION: lista creada con crearListaArt().

\*POST-CONDICION: elimina el nodo apuntado por ptrNodoListaArt. No realiza accion si la lista

esta vacia o si ptrNodo apunta a fin().

\*listaA: lista sobre la cual se invoca la primitiva.

\*ptrNodoListaArt: puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(ListaArt &listaA,PrtNodoListaArt ptrNodoListaArt);

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

/\* ELIMINAR NODO PRIMERO

\*PRE-CONDICION:Lista creada con crearListaArt().

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

\*listaA : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(ListaArt &listaA);

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

/\* ELIMINAR NODO ULTIMO

\*PRE-CONDICION: lista creada con crearListaArt().

\*POST-CONDICION: si la lista no esta vacia elimina su nodo ultimo, sino no realiza accion.

\*listaA : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(ListaArt &listaA);

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

/\* ELIMINAR LISTA

\*PRE-CONDICION:Lista creada con crearListaArt().

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

\*listaA: lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(ListaArt &listaA);

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

#endif // \_\_LISTAARTICULO\_H\_\_

* **LISTACALLE.h**

#ifndef \_\_LISTACALLE\_H\_\_

#define \_\_LISTACALLE\_H\_\_

#ifndef NULL

#define NULL 0

#endif

#include "ListaPiso.h"

//TIPO DE INFORMACION CONTENIDA EN LA ESTRUCTURA LISTAPISO IDENTIFICADA COMO LISTPISO

typedef ListaPiso ListPiso;

//TIPO DE ESTRUCTURA DE LOS NODOS DE LA LISTA

struct NodoCalle{

ListPiso listPiso ;// puntero al piso

NodoCalle \*sgteC;//puntero al nodo siguente

};

// TIPO DE PUNTERO A LOS NODOS DE LA LISTA, EL CUAL SE USA PARA RECORRER LA LISTA Y ACCEDER A SUS DATOS

typedef NodoCalle \*PtrNodoCalle;// puntero de la estructura nodo

// TIPO DE ESTRUCTURA DE LA LISTA

struct ListaCalle{

PtrNodoCalle primeroCalle;//declaro el PtrNodoCalle como el tipo de la variable primero

};

//DEFINICION DE PRIMITIVAS

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* CREAR LISTA CALLE

\*PRE-CONDICION:La lista no debe haber sido creada.

\*POST-CONDICION:La lista queda creada y preparada para ser usada.

\*listaC : estructura de datos a ser creado.

\*/

void crearListaCalle(ListaCalle &listaC);

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

/\* LISTA VACIA

\*PRE-CONDICION:Lista Creada con crearListaCalle().

\*POST-CONDICION: Devuelve true si lista esta vacia, sino devuelve false.

\*listaC: lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(ListaCalle &listaC);

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

/\* FINC

\*PRE-CONDICION: Lista Creada con crearListaCalle().

\*POST-CONDICION: 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.

\*/

PtrNodoCalle finC();

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

/\* PRIMERO

\*PRE-CONDICION: Lista Creada con crearListaCalle().

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

\*listaC : lista sobre la cual se invoca la primitiva.

\*return puntero al primer nodo.

\*/

PtrNodoCalle primero(ListaCalle &listaC);

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

/\* SIGUIENTE

PRE-CONDICION: Lista Creada con crearListaCalle().

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

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

\*listaC : lista sobre la cual se invoca la primitiva.

\*prtNodoCalle : puntero al nodo a partir del cual se requiere el siguiente.

\*return puntero al nodo siguiente.

\*/

PtrNodoCalle siguiente(ListaCalle &listaC,PtrNodoCalle ptrNodoCalle);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* ANTERIOR

\*PRE-CONDICION:Lista Creada con crearListaCalle().

PtrNodoCalle es un puntero a un nodo de lista.

\*POST-CONDICION: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si

ptrNodo apuntaba al primero o si lista esta vacia.

\*listaC :Lista sobre la cual se invoca la primitiva.

\*prtNodoCalle : puntero al nodo a partir del cual se requiere el anterior.

\*return puntero al nodo anterior.

\*/

PtrNodoCalle anterior(ListaCalle &listaC, PtrNodoCalle ptrNodoC);

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

/\* ULTIMO

\*PRE-CONDICION: lista creada con crearListaCalle().

\*POST-CONDICION: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si ista esta vacia.

\*listaC : lista sobre la cual se invoca la primitiva.

\*return puntero al último nodo.

\*/

PtrNodoCalle ultimo(ListaCalle &listaC);

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

PtrNodoCalle crearNodoLista(ListPiso listPiso);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* ADICIONAR AL PRINCIPIO

\*PRE-CONDICION: lista creada con crearListaCalle().

\*POST-CONDICION: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

\*listaC: lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a adicionar al principio de la lista.

\*return puntero al nodo adicionado.

\*/

PtrNodoCalle adicionarPrincipio(ListaCalle &listaC, ListPiso listPiso);

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

/\* ADICIONAR DESPUES

\*PRE-CONDICION: lista creada con crearListaCalle().

\*POST-CONDICION: agrega un nodo despues del apuntado por ptrNodoListaArt 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().

\*listaC : lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a adicionar.

\*ptrNodoCalle: puntero al nodo después del cual se quiere adicionar el dato.

\*return puntero al nodo adicionado.

\*/

PtrNodoCalle adicionarDespues(ListaCalle &listaC, ListPiso listPiso, PtrNodoCalle ptrNodoC);

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

/\* ADICIONAR FINAL

\*PRE-CONDICION: lista creada con crearListaCalle().

\*POST-CONDICION: agrega un nodo al final de la lista con el dato proporcionado y devuelve un puntero

al nodo insertado.

\*listaC: lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a adicionar al final de la lista.

\*return puntero al nodo adicionado.

\*/

PtrNodoCalle adicionarFinal(ListaCalle &listaC,ListPiso listPiso);

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

/\* ADICIONAR ANTES

\*PRE-CONDICION: lista creada con crearListaCalle().

\*POST-CONDICION: 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 ptrNodoCalle apunta al primero, el nodo insertado sera el nuevo primero.

\*listaC : lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a adicionar.

\*PtrNodoCalle: puntero al nodo antes del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoCalle adicionarAntes(ListaCalle &listaC, ListPiso listPiso, PtrNodoCalle ptrNodoC);

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

/\* COLOCAR DATO

\*PRE-CONDICION: lista creada con crearListaCalle(), no vacia. ptrNodoCalle es distinto de fin().

\*POST-CONDICION: coloca el dato proporcionado en el nodo apuntado por ptrNodoCalle.

\*listaC: lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a colocar.

\*ptrNodoCalle : puntero al nodo del cual se quiere colocar el dato.

\*/

void colocarDato(ListaCalle &listaC,ListPiso listPiso, PtrNodoCalle ptrNodoC);

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

/\* OBTENER DATO

\*PRE-CONDICION: lista creada con crearListaCalle(), no vacia. ptrNodo es distinto de fin().

\*POST-CONDICION: devuelve el dato del nodo apuntado por ptrNodoCalle.

\*listaC: lista sobre la cual se invoca la primitiva.

\*listPiso: elemento obtenido.

\*ptrNodoCalle : puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDato(ListaCalle &listaC, ListPiso listPiso, PtrNodoCalle ptrNodoC);

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

/\* ELIMINAR NODO

\*PRE-CONDICION: lista creada con crearListaCalle().

\*POST-CONDICION: elimina el nodo apuntado por ptrNodoCalle. No realiza accion si la lista

esta vacia o si ptrNodo apunta a fin().

\*listaC: lista sobre la cual se invoca la primitiva.

\*ptrNodoCalle: puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(ListaCalle &listaC, PtrNodoCalle ptrNodoC);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* ELIMINAR NODO PRIMERO

\*PRE-CONDICION:Lista creada con crearListaCalle().

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

\*listaC : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(ListaCalle &listaC);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* ELIMINAR NODO ULTIMO

\*PRE-CONDICION: lista creada con crearListaCalle().

\*POST-CONDICION: si la lista no esta vacia elimina su nodo ultimo, sino no realiza accion.

\*listaC : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(ListaCalle &listaC);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* ELIMINAR LISTA

\*PRE-CONDICION:Lista creada con crearListaCalle().

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

\*listaC: lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(ListaCalle &listaC);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//DEFINICION DE OPERACIONES ADICIONALES

/\* LOCALIZAR DATO

\*PRE-CONDICION: lista fue creada con crearListaCalle().

\*POST-CONDICION: 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().

\*listaC : lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a localizar.

return puntero al nodo localizado o fin().

\*/

PtrNodoCalle localizarDato(ListaCalle &listaC , ListPiso listPiso);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* INSERTAR DATO

\*PRE-CONDICION: lista fue creada con crearListaCalle() y cargada con datos ordenados de menor a mayor

respecto del sentido progresivo.

\*POST-CONDICION: agrega a la lista el dato manteniendo el orden pero con multiples valores iguales y

devuelve un puntero al nodo insertado.

\*listaC : lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a insertar.

\*return puntero al nodo insertado.

\*/

PtrNodoCalle insertarDato(ListaCalle &listaC, ListPiso listPiso);

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

/\* ELIMINAR DATO

\*PRE-CONDICION: la lista fue creada con crearListaCalle().

\*POST-CONDICION: elimina el dato de la lista, si el mismo se encuentra.

\*listaC: lista sobre la cual se invoca la primitiva.

\*listPiso: elemento a eliminar.

\*/

void eliminarDato(ListaCalle &listaC, ListPiso listPiso);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\* REORDENAR

\*PRE-CONDICION: la lista fue creada con crearListaCalle().

\*POST-CONDICION: reordena la lista.

\*listaC : lista sobre la cual se invoca la primitiva.

\*/

void reordenar(ListaCalle &listaC);

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

/\*

\*PRE-CONDICION: la lista fue creada con crearListaCalle().

\*POST-CONDICION: devuelve la cantidad de datos que tiene la lista.

\*listaC: lista sobre la cual se invoca la primitiva.

\*/

int longitud(ListaCalle &listaC);

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

#endif // \_\_LISTACALLE\_H\_\_

* **LISTAFALTANTES.h**

#ifndef \_\_LISTAFALTANTES\_H\_\_

#define \_\_LISTAFALTANTES\_H\_\_

#include "Faltantes.h"

#ifndef NULL

#define NULL 0

#endif

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

typedef Faltantes Faltante;

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

struct NodoFaltante{

Faltante datoFaltante;

NodoFaltante \*sgteF;

};

/\* Tipo de Puntero a los Nodos de la Lista\*/

typedef NodoFaltante \*PtrNodoFaltante;

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

struct ListaFaltante{

PtrNodoFaltante primerFaltante;

};

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

/\*

pre : la lista no debe haber sido creada.

post: lista queda creada y preparada para ser usada.

listaF : estructura de datos a ser creado.

\*/

void crearListaFaltante(ListaFaltante &listaF);

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

/\*

pre : lista Creada anteriormente.

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

listaF : lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(ListaFaltante &listaF);

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

/\*

pre : lista debe estar creada.

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null.

return: representación del fin de la lista.

\*/

PtrNodoFaltante finF();

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

/\*

pre : lista debe haber sido creada

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

esta vacia

listaF : Lista sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoFaltante primero(ListaFaltante &listaF);

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

/\*

pre : lista debe estar creada.

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

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

listaF : lista sobre la cual se invoca la primitiva.

prtNodoFaltante : puntero al nodo a partir del cual se requiere el siguiente.

return puntero al nodo siguiente.

\*/

PtrNodoFaltante siguiente(ListaFaltante &listaF,PtrNodoFaltante ptrNodoFaltante);

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

/\*

pre : lista Creada anteriormente.

ptrNodoFaltante es un puntero a un nodo de lista.

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

el punter apuntaba al primero o si lista esta vacia.

listaF : lista sobre la cual se invoca la primitiva.

prtNodoFaltante : puntero al nodo a partir del cual se requiere el anterior.

return puntero al nodo anterior.

\*/

PtrNodoFaltante anterior(ListaFaltante &listaF,PtrNodoFaltante ptrNodoFaltante);

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

/\*

pre : lista creada con anterioridad

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

si lista esta vacia.

listaF : lista sobre la cual se invoca la primitiva.

return puntero al último nodo.

\*/

PtrNodoFaltante ultimo(ListaFaltante &listaF);

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

/\*

pre : lista creada con anterioridad

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

si lista esta vacia.

datoFaltante : dato faltante ya viene creado de antes.

return crea un nodo en la lista.

\*/

PtrNodoFaltante crearNodoLista(Faltante datoFaltante);

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

/\*

pre : lista creada

post: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

listaF : lista sobre la cual se invoca la primitiva.

datFaltanteo : elemento a adicionar al principio de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoFaltante adicionarPrincipio(ListaFaltante &listaF,Faltante datoFaltante);

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

/\*

pre : lista creada

post: agrega un nodo despues del apuntado por puntero con el dato

proporcionado y devuelve un puntero apuntado al elemento insertado.

listaF : lista sobre la cual se invoca la primitiva.

datoFaltante : elemento a adicionar.

ptrNodoFaltante : puntero al nodo después del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoFaltante adicionarDespues(ListaFaltante &listaF,Faltante datoFaltante, PtrNodoFaltante ptrNodoFaltante);

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

/\*

pre : lista creada

post: agrega un nodo al final de la lista con el dato proporcionado y devuelve

un puntero al nodo insertado.

listaf : lista sobre la cual se invoca la primitiva.

datoFaltante : elemento a adicionar al final de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoFaltante adicionarFinal(ListaFaltante &listaF, Faltante datoFaltante);

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

/\*

pre : lista creada, no vacia. puntero al nodo es distinto de fin().

post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.

listaF : lista sobre la cual se invoca la primitiva.

datoFaltante : elemento a colocar.

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

\*/

void colocarDato(ListaFaltante &listaF,Faltante &datoFaltante, PtrNodoFaltante ptrNodoFaltante);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: devuelve el dato del nodo apuntado por ptrNodo.

listaF : lista sobre la cual se invoca la primitiva.

datoF: elemento obtenido.

ptrNodoFaltante : puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDato(ListaFaltante &listaF,Faltante &datoFaltante, PtrNodoFaltante ptrNodoFaltante);

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

/\*

pre : lista creada

post: elimina el nodo apuntado por el puntero. No realiza accion si la lista

esta vacia o si puntero apunta a fin().

listaF : lista sobre la cual se invoca la primitiva.

ptrNodoFaltante : puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(ListaFaltante &listaF,PtrNodoFaltante ptrNodoFaltante);

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

/\*

pre : lista creada

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

accion alguna.

listaF : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(ListaFaltante &listaF);

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

/\*

pre : lista creada.

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

sino no realiza accion.

listaF : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(ListaFaltante &listaF);

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

/\*

pre : lista creada

post: elimina todos los Nodos de la lista quedando destruida.

listaF : lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(ListaFaltante &listaF);

#endif // \_\_LISTAFALTANTES\_H\_\_

* **LISTAPESOACUMULADO.h**

#ifndef \_\_LISTAPESOACUMULADO\_H\_\_

#define \_\_LISTAPESOACUMULADO\_H\_\_

#ifndef NULL

#define NULL 0

#endif

#include "PesoAcumulado.h"

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

typedef PesoAcum PesoA;

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

struct NodoListaPAcum{

PesoA datoPesoA;

NodoListaPAcum \*sgtePA;

};

/\* Tipo de Puntero a los Nodos de la Lista\*/

typedef NodoListaPAcum \*PtrNodoListaPA;

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

struct ListaPesoAcum{

PtrNodoListaPA primerPesoAcum;

};

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

/\*

pre : la lista no debe haber sido creada.

post: lista queda creada y preparada para ser usada.

listaPesoAcum : estructura de datos a ser creado.

\*/

void crearListaPesoAcum(ListaPesoAcum &listaPA);

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

/\*

pre : lista Creada anteriormente.

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

listaPA : lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(ListaPesoAcum &listaPA);

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

/\*

pre : lista debe estar creada.

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null.

return: representación del fin de la lista.

\*/

PtrNodoListaPA finPA();

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

/\*

pre : lista debe haber sido creada

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

esta vacia

listaPA : Lista sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoListaPA primero(ListaPesoAcum &listaPA);

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

/\*

pre : lista debe estar creada.

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

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

listaPA : lista sobre la cual se invoca la primitiva.

prtNodoListaPA : puntero al nodo a partir del cual se requiere el siguiente.

return puntero al nodo siguiente.

\*/

PtrNodoListaPA siguiente(ListaPesoAcum &listaPA,PtrNodoListaPA ptrNodoListaPA);

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

/\*

pre : lista Creada anteriormente.

ptrNodoListaPA es un puntero a un nodo de lista.

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

el punter apuntaba al primero o si lista esta vacia.

listaPA : lista sobre la cual se invoca la primitiva.

prtNodoListaPA : puntero al nodo a partir del cual se requiere el anterior.

return puntero al nodo anterior.

\*/

PtrNodoListaPA anterior(ListaPesoAcum &listaPA,PtrNodoListaPA ptrNodoListaPA);

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

/\*

pre : lista creada con anterioridad

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

si lista esta vacia.

listaPA : lista sobre la cual se invoca la primitiva.

return puntero al último nodo.

\*/

PtrNodoListaPA ultimo(ListaPesoAcum &listaPA);

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

/\*

pre : lista creada con anterioridad

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

si lista esta vacia.

datoPesoA : dato pesoA ya viene creado de antes.

return crea un nodo en la lista.

\*/

PtrNodoListaPA crearNodoLista(PesoA datoPesoA);

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

/\*

pre : lista creada

post: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

listaPA: lista sobre la cual se invoca la primitiva.

datPesoA : elemento a adicionar al principio de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoListaPA adicionarPrincipio(ListaPesoAcum &listaPA,PesoA datoPesoA);

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

/\*

pre : lista creada

post: agrega un nodo despues del apuntado por puntero con el dato

proporcionado y devuelve un puntero apuntado al elemento insertado.

listaPA : lista sobre la cual se invoca la primitiva.

datoPesoA : elemento a adicionar.

ptrNodoListaPA : puntero al nodo después del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoListaPA adicionarDespues(ListaPesoAcum &listaPA,PesoA datoPesoA, PtrNodoListaPA ptrNodoListaPA);

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

/\*

pre : lista creada

post: agrega un nodo al final de la lista con el dato proporcionado y devuelve

un puntero al nodo insertado.

listaPA : lista sobre la cual se invoca la primitiva.

datoPesoA : elemento a adicionar al final de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoListaPA adicionarFinal(ListaPesoAcum &listaPA, PesoA datoPesoA);

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

/\*

pre : lista creada, no vacia. puntero al nodo es distinto de fin().

post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.

listaPA : lista sobre la cual se invoca la primitiva.

datoPesoA : elemento a colocar.

ptrNodoListaPA : puntero al nodo del cual se quiere colocar el dato.

\*/

void colocarDato(ListaPesoAcum &listaPA,PesoA &datoPesoA, PtrNodoListaPA ptrNodoListaPA);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: devuelve el dato del nodo apuntado por ptrNodo.

listaPA : lista sobre la cual se invoca la primitiva.

datoPesoA: elemento obtenido.

ptrNodoListaPA : puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDato(ListaPesoAcum &listaPA,PesoA &datoPesoA, PtrNodoListaPA ptrNodoListaPA);

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

/\*

pre : lista creada

post: elimina el nodo apuntado por el puntero. No realiza accion si la lista

esta vacia o si puntero apunta a fin().

listaPA: lista sobre la cual se invoca la primitiva.

ptrNodoListaPA : puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(ListaPesoAcum &listaPA,PtrNodoListaPA ptrNodoListaPA);

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

/\*

pre : lista creada

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

accion alguna.

listaPA : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(ListaPesoAcum &listaPA);

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

/\*

pre : lista creada.

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

sino no realiza accion.

listaPA : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(ListaPesoAcum &listaPA);

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

/\*

pre : lista creada

post: elimina todos los Nodos de la lista quedando destruida.

listaPA : lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(ListaPesoAcum &listaPA);

#endif // \_\_LISTAPESOACUMULADO\_H\_\_

* **LISTAPISO.h**

#ifndef \_\_LISTAPISO\_H\_\_

#define \_\_LISTAPISO\_H\_\_

#ifndef NULL

#define NULL 0

#endif

#include "PilaArtUb.h"

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

typedef PilaArtUb PilUb;

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

struct NodoPiso{

PilUb pilUb;// puntero a la pila

NodoPiso \*sgteP;//puntero al nodo siguente

};

/\* Tipo de Puntero a los Nodos de la Lista\*/

typedef NodoPiso \*PtrNodoPiso;// puntero de la estructura nodo

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

struct ListaPiso{

PtrNodoPiso primeroPiso;//declaro el PtrNodoCalle como el tipo de la variable primero

};

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

/\*

pre : la lista no debe haber sido creada.

post: lista queda creada y preparada para ser usada.

listaP : estructura de datos a ser creado.

\*/

void crearListaPiso(ListaPiso &listaP);

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

/\*

pre : lista Creada anteriormente.

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

listaP : lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(ListaPiso &listaP);

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

/\*

pre : lista debe estar creada.

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null.

return: representación del fin de la lista.

\*/

PtrNodoPiso finP();

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

/\*

pre : lista debe haber sido creada

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

esta vacia

listaP : Lista sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoPiso primero(ListaPiso &listaP);

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

/\*

pre : lista debe estar creada.

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

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

listaP : lista sobre la cual se invoca la primitiva.

prtNodoP : puntero al nodo a partir del cual se requiere el siguiente.

return puntero al nodo siguiente.

\*/

PtrNodoPiso siguiente(ListaPiso &listaP, PtrNodoPiso ptrNodoP);

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

/\*

pre : lista creada con anterioridad

post: añade un dato a la pila

piluB: dato que se añade a la pila.

return crea un nodo en la lista.

\*/

PtrNodoPiso crearNodoLista(PilUb pilUb);

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

/\*

pre : lista Creada anteriormente.

ptrNodoP es un puntero a un nodo de lista.

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

el punter apuntaba al primero o si lista esta vacia.

listaP : lista sobre la cual se invoca la primitiva.

prtNodoP : puntero al nodo a partir del cual se requiere el anterior.

return puntero al nodo anterior.

\*/

PtrNodoPiso anterior(ListaPiso &listaP, PtrNodoPiso ptrNodoP);

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

/\*

pre : lista creada con anterioridad

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

si lista esta vacia.

listaP : lista sobre la cual se invoca la primitiva.

return puntero al último nodo.

\*/

PtrNodoPiso ultimo(ListaPiso &listaP);

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

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

/\*

pre : lista creada

post: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

listaP: lista sobre la cual se invoca la primitiva.

Pilub : elemento a adicionar al principio de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoPiso adicionarPrincipio(ListaPiso &listaP, PilUb pilUb);

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

/\*

pre : lista creada

post: agrega un nodo despues del apuntado por puntero con el dato

proporcionado y devuelve un puntero apuntado al elemento insertado.

listaP : lista sobre la cual se invoca la primitiva.

Pilub : elemento a adicionar.

ptrNodoP : puntero al nodo después del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoPiso adicionarDespues(ListaPiso &listaP, PilUb pilUb, PtrNodoPiso ptrNodoP);

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

/\*

pre : lista creada

post: agrega un nodo al final de la lista con el dato proporcionado y devuelve

un puntero al nodo insertado.

listaP : lista sobre la cual se invoca la primitiva.

Pilub : elemento a adicionar al final de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoPiso adicionarFinal(ListaPiso &listaP, PilUb pilUb);

/\*

pre : lista creada

post: agrega un nodo antes en la lista con el dato proporcionado y devuelve

un puntero al nodo insertado.

listaP : lista sobre la cual se invoca la primitiva.

Pilub : elemento a adicionar al final de la lista.

ptrNodoP

return puntero al nodo adicionado.

\*/

PtrNodoPiso adicionarAntes(ListaPiso &listaP, PilUb pilUb, PtrNodoPiso ptrNodoP);

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

/\*

pre : lista creada, no vacia. puntero al nodo es distinto de fin().

post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.

listaP : lista sobre la cual se invoca la primitiva.

dpiluB : elemento a colocar.

ptrNodoP : puntero al nodo del cual se quiere colocar el dato.

\*/

void colocarDato(ListaPiso &listaP, PilUb &pilUb, PtrNodoPiso ptrNodoP);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: devuelve el dato del nodo apuntado por ptrNodo.

listaF : lista sobre la cual se invoca la primitiva.

datoF: elemento obtenido.

ptrNodoFaltante : puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDato(ListaPiso &listaP, PilUb &pilUb, PtrNodoPiso ptrNodoP);

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

/\*

pre : lista creada

post: elimina el nodo apuntado por el puntero. No realiza accion si la lista

esta vacia o si puntero apunta a fin().

listaP : lista sobre la cual se invoca la primitiva.

ptrNodoP : puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(ListaPiso &listaP, PtrNodoPiso ptrNodoP);

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

/\*

pre : lista creada

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

accion alguna.

listaP : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(ListaPiso &listaP);

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

/\*

pre : lista creada.

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

sino no realiza accion.

listaP : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(ListaPiso &listaP);

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

/\*

pre : lista creada

post: elimina todos los Nodos de la lista quedando destruida.

listaP : lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(ListaPiso &listaP);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: Localiza el dato y devuelve un puntero apuntando a ese dato

listap : lista sobre la cual se invoca la primitiva.

pilUb: elemento a localizar

\*/

PtrNodoPiso localizarDato(ListaPiso &listaP , PilUb pilUb);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: inserta el dato y devuelve un puntero apuntando a ese dato

listap : lista sobre la cual se invoca la primitiva.

pilUb: elemento a localizar

\*/

PtrNodoPiso insertarDato(ListaPiso &listaP, PilUb pilUb);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: elimina el dato

listap : lista sobre la cual se invoca la primitiva.

pilUb: elemento a eliminar

\*/

void eliminarDato(ListaPiso &listaP, PilUb pilUb);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: reordena la lista

listap : lista sobre la cual se invoca la primitiva.

\*/

void reordenar(ListaPiso &listaP);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: obtiene la longitud de la lista.

listap : lista sobre la cual se invoca la primitiva.

\*/

int longitud(ListaPiso &listaP);

#endif // \_\_LISTAPISO\_H\_\_

* **LISTAVENTAS.h**

#ifndef \_\_LISTAVENTAS\_H\_\_

#define \_\_LISTAVENTAS\_H\_\_

#ifndef NULL

#define NULL 0

#endif // NULL

#include "Ventas.h"

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

typedef Ventas Venta;

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

struct NodoListaVentas{

Venta datoVenta;

NodoListaVentas \*sgteV;

};

/\* Tipo de Puntero a los Nodos de la Lista\*/

typedef NodoListaVentas \*PtrNodoListaVentas;

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

struct ListaVentas{

PtrNodoListaVentas primeroListaVentas;

};

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

/\*

pre : la lista no debe haber sido creada.

post: lista queda creada y preparada para ser usada.

listaV: estructura de datos a ser creado.

\*/

void crearListaVenta(ListaVentas &listaV);

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

/\*

pre : lista Creada anteriormente.

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

listaV : lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(ListaVentas &listaV);

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

/\*

pre : lista debe estar creada.

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null.

return: representación del fin de la lista.

\*/

PtrNodoListaVentas finV();

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

/\*

pre : lista debe haber sido creada

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

esta vacia

listaV : Lista sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoListaVentas primero(ListaVentas &listaV);

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

/\*

pre : lista debe estar creada.

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

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

listaV : lista sobre la cual se invoca la primitiva.

prtNodoListaVent : puntero al nodo a partir del cual se requiere el siguiente.

return puntero al nodo siguiente.

\*/

PtrNodoListaVentas siguiente(ListaVentas &listaV,PtrNodoListaVentas ptrNodoListaVent);

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

/\*

pre : lista Creada anteriormente.

ptrNodoListaVentas es un puntero a un nodo de lista.

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

el punter apuntaba al primero o si lista esta vacia.

listav : lista sobre la cual se invoca la primitiva.

prtNodoListaVentas : puntero al nodo a partir del cual se requiere el anterior.

return puntero al nodo anterior.

\*/

PtrNodoListaVentas anterior(ListaVentas &listaV,PtrNodoListaVentas ptrNodoListaVent);

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

/\*

pre : lista creada con anterioridad

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

si lista esta vacia.

listaV : lista sobre la cual se invoca la primitiva.

return puntero al último nodo.

\*/

PtrNodoListaVentas ultimo(ListaVentas &listaV);

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

/\*

pre : lista creada con anterioridad

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

si lista esta vacia.

datoVenta: dato faltante ya viene creado de antes.

return crea un nodo en la lista.

\*/

PtrNodoListaVentas crearNodoLista(Venta datoVenta);

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

/\*

pre : lista creada

post: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

listaF : lista sobre la cual se invoca la primitiva.

datFaltanteo : elemento a adicionar al principio de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoListaVentas adicionarPrincipio(ListaVentas &listaV,Venta datoVenta);

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

/\*

pre : lista creada

post: agrega un nodo despues del apuntado por puntero con el dato

proporcionado y devuelve un puntero apuntado al elemento insertado.

listav : lista sobre la cual se invoca la primitiva.

datoVenta : elemento a adicionar.

ptrNodoListaVent: puntero al nodo después del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoListaVentas adicionarDespues(ListaVentas &listaV,Venta datoVenta, PtrNodoListaVentas ptrNodoListaVent);

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

/\*

pre : lista creada

post: agrega un nodo al final de la lista con el dato proporcionado y devuelve

un puntero al nodo insertado.

listaV: lista sobre la cual se invoca la primitiva.

datoVenta : elemento a adicionar al final de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoListaVentas adicionarFinal(ListaVentas &listaV, Venta datoVenta);

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

/\*

pre : lista creada, no vacia. puntero al nodo es distinto de fin().

post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.

listaV : lista sobre la cual se invoca la primitiva.

datoVenta : elemento a colocar.

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

\*/

void colocarDato(ListaVentas &listaV,Venta &datoVenta, PtrNodoListaVentas ptrNodoListaVent);

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

/\*

pre : lista creada ,no vacia. puntero al nodo es distinto de fin().

post: devuelve el dato del nodo apuntado por ptrNodo.

listaV : lista sobre la cual se invoca la primitiva.

datoVentas: elemento obtenido.

ptrNodoListaVent : puntero al nodo del cual se quiere obtener el dato.

\*/

void obtenerDato(ListaVentas &listaV,Venta &datoVenta, PtrNodoListaVentas ptrNodoListaVent);

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

/\*

pre : lista creada

post: elimina el nodo apuntado por el puntero. No realiza accion si la lista

esta vacia o si puntero apunta a fin().

listaV : lista sobre la cual se invoca la primitiva.

ptrNodoListaVent : puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(ListaVentas &listaV,PtrNodoListaVentas ptrNodoListaVent);

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

/\*

pre : lista creada

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

accion alguna.

listaV : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(ListaVentas &listaV);

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

/\*

pre : lista creada

post: Elimina nodo ultimo de la lista.

listaV : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(ListaVentas &listaV);

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

/\*

pre : lista creada

post: elimina todos los Nodos de la lista quedando destruida.

listaV : lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(ListaVentas &listaV);

#endif // \_\_LISTAVENTAS\_H\_\_

* **PESOACUMULADO.h**

#ifndef \_\_PESOACUMULADO\_H\_\_

#define \_\_PESOACUMULADO\_H\_\_

/\*Definición del Tipo de Dato\*/

typedef struct PesoAcum{

int codArt;

float cant;

};

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

/\*

pre: la instancia no debe ser creado ni estar destruido.

post: Crea la instancia con los datos en 0.

pesoAcum: instancia sobre la que se invoca la primitiva

\*/

void crearPesoAcum(PesoAcum &pesoAcum);

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

/\*

pre: la instancia no debe ser creado ni estar destruido.

post: Crea la instancia con los datos seteados segun lo que se le ingresa.

pesoAcum: instancia sobre la que se invoca la primitiva

codart,cant. Variables que se utilizan en la funcion

\*/

void crearPesoAcum(PesoAcum &pesoAcum,int codart,float cant);

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

/\*

pre: la instancia debe existir.

post: elimina la instancia.

pesoAcum: instancia sobre la que se invoca la primitiva

\*/

void destruirPesoAcum(PesoAcum &pesoAcum);

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

/\*

pre: la instancia debe existir.

post: Se le setea un valor asignado a codart.

pesoAcum: instancia sobre la que se invoca la primitiva

cordart: Variable que se utiliza en la funcion.

\*/

void setCodArt(PesoAcum &pesoAcum,int codart);

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

/\*

pre: la instancia debe existir.

post: Se le setea un valor asignado a cant

pesoAcum: instancia sobre la que se invoca la primitiva

cant: Variable que se utiliza en la funcion.

\*/

void setCant(PesoAcum &pesoAcum,float cant);

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

/\*

pre: la instancia debe existir.

post: obtenemos el valor de codart, de la instancia.

pesoAcum: instancia sobre la que se invoca la primitiva

\*/

int getCodArt(PesoAcum &pesoAcum);

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

/\*

pre: la instancia debe existir.

post: obtenemos el valor de cantidad, de la instancia.

pesoAcum: instancia sobre la que se invoca la primitiva

\*/

float getCant(PesoAcum &pesoAcum);

#endif // \_\_PESOACUMULADO\_H\_\_

* **PILAARTUB.h**

#ifndef \_\_PILAARTUB\_H\_\_

#define \_\_PILAARTUB\_H\_\_

#include "ArtUbicacion.h"

#ifndef NULL

#define NULL 0

#endif

/\* Tipo de Estructura de los Nodos de la Pila. \*/

typedef ArtUbicacion ArtUb;

struct NodoPilaArtUb{

ArtUb datoArtUb;

NodoPilaArtUb \*sgteU;

};

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

la pila y acceder a sus Datos. \*/

typedef NodoPilaArtUb \*PtrNodoPilaArtUb;

/\* Tipo de Estructura de la Pila\*/

struct PilaArtUb{

PtrNodoPilaArtUb top;

};

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

/\*

pre : la pila no debe haber sido creada.

post: pila queda creada y preparada para ser usada.

pilaArtUb : estructura de datos a ser creado.

\*/

void crearPila(PilaArtUb &pilaArtUb);

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

/\*

pre : pila creada con crearpila().

post: elimina el nodo apuntado por ptrNodo. No realiza accion si la pila

esta vacia o si ptrNodo apunta a fin().

pila : pila sobre la cual se invoca la primitiva.

pilaArtUb : puntero al nodo que se desea eliminar.

\*/

void destruir(PilaArtUb &pilaArtUb);

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

/\*

pre : pila Creada con crearpila().

post: Devuelve true si pila esta vacia, sino devuelve false.

pilaArtUb : pila sobre la cual se invoca la primitiva.

\*/

bool pilaVacia(PilaArtUb &pilaArtUb);

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

/\*

pre : tiene que haber una pila creada.

post: Se agrega el dato a la pila.

pilaArtUb : pila sobre la cual se invoca la primitiva.

\*/

void push(PilaArtUb &pilaArtUb,ArtUb datoArtUb);

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

/\*

pre : tiene que haber una pila creada.

post: Se saca un nodo el dato a la pila.

pilaArtUb : pila sobre la cual se invoca la primitiva.

\*/

ArtUb pop(PilaArtUb &pilaArtUb);

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

/\*

pre : pila debe estar creada.

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

si pila esta vacia.

pilaArtUb : pila sobre la cual se invoca la primitiva.

\*/

PtrNodoPilaArtUb finU();

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

/\*

pre : pila debe estar creada .

post: devuelve el puntero al top de la pila

pilaArtUb : pila sobre la cual se invoca la primitiva.

\*/

PtrNodoPilaArtUb top(PilaArtUb &pilaArtUb);

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

#endif // \_\_PILAARTUB\_H\_\_

* **PILACAMION.h**

#ifndef \_\_PILACAMION\_H\_\_

#define \_\_PILACAMION\_H\_\_

#include "Camion.h"

#ifndef NULL

#define NULL 0

#endif

/\* Tipo de Estructura de los Nodos de la Pila. \*/

typedef Camion Cam;

struct NodoPilaCamion{

Cam datoCamion;

NodoPilaCamion \*sgteCa;

};

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

la pila y acceder a sus Datos. \*/

typedef NodoPilaCamion \*PtrNodoPilaCamion;

/\* Tipo de Estructura de la Pila\*/

struct PilaCamion{

PtrNodoPilaCamion top;

};

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

/\*

pre : la pila no debe haber sido creada.

post: pila queda creada y preparada para ser usada.

pilaCamion : estructura de datos a ser creado.

\*/

void crearPila(PilaCamion &pilaCamion);

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

/\*

pre : pila debe estar creada.

post: elimina el nodo apuntado por ptrNodo. No realiza accion si la pila

esta vacia o si ptrNodo apunta a fin().

pilaCamion : pila sobre la cual se invoca la primitiva.

\*/

void destruir(PilaCamion &pilaCamion);

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

/\*

pre : pila debe estar creada.

post: Devuelve true si pila esta vacia, sino devuelve false.

pilaCamion : pila sobre la cual se invoca la primitiva.

\*/

bool pilaVacia(PilaCamion &pilaCamion);

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

/\*

pre : tiene que haber una pila creada.

post: Se saca un nodo el dato a la pila.

pilaCamion : pila sobre la cual se invoca la primitiva.

cam datocamion : tipo de creo una instancia de tip camion

\*/

void push(PilaCamion &pilaCamion,Cam datoCamion);

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

/\*

pre : tiene que haber una pila creada.

post: Se saca un nodo el dato a la pila.

pilaCamion : pila sobre la cual se invoca la primitiva.

\*/

Cam pop(PilaCamion &pilaCamion);

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

/\*

pre : pila debe estar creada .

post: devuelve el puntero al top de la pila

pilaCamion : pila sobre la cual se invoca la primitiva.

\*/

PtrNodoPilaCamion top(PilaCamion &pilaCamion);

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

/\*

pre : pila debe estar creada .

post: Si es nulo es el fin de pila camion

\*/

PtrNodoPilaCamion finCa();

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

#endif // \_\_PILACAMION\_H\_\_

* **REPOSICION.h**

#ifndef \_\_REPOSICION\_H\_\_

#define \_\_REPOSICION\_H\_\_

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

typedef struct Reposicion{

int cod\_articulo;

int cantidad;

char hora[150];

};

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

/\* pre : la reposicion no haya sido creada

post: la reposicion queda creada y preparada para ser usada.

reposicion : estructura de datos a ser creado.

\*/

void crearReposicion(Reposicion &reposicion);

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

/\*pre : reposicion no haya sido creada

post: reposicion queda creada y preparada para ser usada.

reposicion : estructura de datos a ser creado.

cod\_articulo,cantidad,hora variables que se van a utilizar

\*/

void crearReposicion(Reposicion &reposicion,int CodArt,int Cant,char hora[]);

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

/\*pre : Que no se haya destruido la reposicion

post:reposicion destruida

reposicion : estructura de datos a ser creado.\*/

void destruirReposicion(Reposicion &reposicion);

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

/\*

pre : se quiere obtener el codigo articulo de la estructura

post: codigo de articulo obtenido

reposicion : estructura de datos a ser creado.

\*/

int getCodArt(Reposicion &reposicion);

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

/\*

pre : se quiere obtener el cantidad de la estructura

post: cantidad obtenido

reposicion : estructura de datos a ser creado.

\*/

int getCant(Reposicion &reposicion);

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

/\*

pre : se quiere obtener la hora de la estructura

post: hora obtenido

reposicion : estructura de datos a ser creado.

\*/

char\* getHora(Reposicion &reposicion);

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

/\*

pre : se quiere setear codigo articulo de la estructura

post: codigo de articulo seteado

reposicion : estructura de datos a ser creado.

\*/

void setCodArt(Reposicion &reposicion,int codart);

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

/\*

pre : se quiere setear cantidad de la estructura

post: cantidad seteado

reposicion : estructura de datos a ser creado.

\*/

void setCant(Reposicion &reposicion,int cant);

#endif // \_\_REPOSICION\_H\_\_

* **SOLICITUD.h**

#ifndef \_\_SOLICITUD\_H\_\_

#define \_\_SOLICITUD\_H\_\_

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

typedef struct Solicitud{

int cod\_suc;

int cod\_articulo;

int cantidad;

char hora[150];

};

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

/\* pre : la Solicitus no haya sido creada

post: Solicitud queda creada y preparada para ser usada.

Solicitud : estructura de datos a ser creado.

\*/

void crearSolicitud(Solicitud &solicitud);

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

/\*pre : Solicitud no haya sido creada

post: Solicitud queda creada y preparada para ser usada.

Solicitud : estructura de datos a ser creado.

codsuc,codart,cant,hora variables que se van a utilizar

\*/

void crearSolicitud(Solicitud &solicitud,int CodSuc,int CodArt,int cant,char hora[]);

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

/\*

pre : La solicitud no debe haber sido destruida.

post: La solicitud es eliminada completamente.

Solicitud : estructura de datos a ser creado.

codsuc,codart,cant,hora variables que se van a utilizar

\*/

void destruirSolicitud(Solicitud &solicitud);

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

/\*

pre : se quiere obtener el codigo sucursal de la estructura

post: codigo de sucursal obtenido

solicitud : estructura de datos a ser creado.

\*/

int getCodSuc(Solicitud &solicitud);

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

/\*

pre : se quiere obtener el codigo articulo de la estructura

post: codigo articulo obtenido

solicitud : estructura de datos a ser creado.

\*/

int getCodArt(Solicitud &solicitud);

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

/\*

pre : se quiere obtener la cantidad de la estructura

post: cantidad obtenido

solicitud : estructura de datos a ser creado.

\*/

int getCant(Solicitud &solicitud);

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

/\*

pre : se quiere obtener la hora de la estructura

post: hora obtenido

solicitud : estructura de datos a ser creado.

\*/

char\* getHora(Solicitud &solicitud);

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

/\*

pre : codigo de sucursal no debe haber sido seteada.

post: codigo de sucursal seteado

Solicitud : estructura de datos a ser creado.

codsuc variabla a utilizar

\*/

void setCodSuc(Solicitud &solicitud,int CodSuc);

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

/\*

pre : codigo de articulo no debe haber sido seteada.

post: codigo de articulo seteado

Solicitud : estructura de datos a ser creado.

codart variable a utilizar

\*/

void setCodArt(Solicitud &solicitud,int CodArt);

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

/\*

pre : cantidad no debe haber sido seteada.

post: cantidad seteado

Solicitud : estructura de datos a ser creado.

cant variable a utilizar

\*/

void setCantidad(Solicitud &Solicitud,int Cant);

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

/\*

pre : Hora no debe ser seteada

post: Hora queda seteada

Solicitud : estructura de datos a ser creado.

h variable a utilizar

\*/

void setHora(Solicitud &solicitud,char h[]);

#endif // \_\_SOLICITUD\_H\_\_

* **VENTAS.h**

#ifndef \_\_VENTAS\_H\_\_

#define \_\_VENTAS\_H\_\_

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

typedef struct Ventas{

int codSucursal;

float montoTotal;

};

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

/\*

pre : la Venta no haya sido creada

post: venta queda creada y preparada para ser usada.

Venta : estructura de datos a ser creado.

\*/

void crearVentas(Ventas &ventas);

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

/\*

pre : la venta no debe haber sido creada.

post: venta queda creada y preparada para ser usada.

Venta : estructura de datos a ser creado.

\*/

void crearVentas(Ventas &ventas,int codSuc,float montTot);

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

/\*

pre : la venta no debe haber sido destruida.

post: La venta es eliminada completamente.

Venta : estructura de datos a ser creado.

codsuc, montot: variables a utilizar

\*/

void destruirVentas(Ventas &ventas);

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

/\*

pre : El Codigo sucursal no esta seteado

post: codigo sucursal queda seteado

Venta : estructura de datos a ser creado.

codsuc: entero con el cual se setea el codigo sucursal

\*/

void setCodSuc(Ventas &ventas,int codsuc);

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

/\*

pre : El Monto total no esta seteado

post: Monto total queda seteado

venta : estructura de datos a ser creado.

monTot: entero con el cual se setea el monto

\*/

void setMonTot(Ventas &ventas,float monTot);

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

/\*

pre : se quiere obtener el codigo sucursal de la estructura

post: codigo sucursal obtenido

venta : estructura de datos a ser creado.

\*/

int getCodSuc(Ventas &ventas);

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

/\*

pre : se quiere obtener el codigo sucursal de la estructura

post: Monto total obtenido

venta : estructura de datos a ser creado.

\*/

float getMonTot(Ventas &ventas);

#endif // \_\_VENTAS\_H\_\_

Archivos.cpp

* **Main.Cpp**

#include <iostream>

#include <fstream>

#include <stdio.h>

#include <conio.h>

#include <string>

#include <time.h>

#include <cstring>

#include <fstream>

#include <stdlib.h>

#include <ctype.h>

#include "FunYProd.h"

using namespace std;

int main(){

//\*\*\*\*\*\*\*\*\*\*\*INSTANCIAS DE TDA\*\*\*\*\*\*\*\*\*\*\*\*\*

Config configuracion;//instancia de tda configuracion

constructorConfig(configuracion);

ListaArt listArt;//lista de articulos para cargar el maestro

crearListaArt(listArt);

ListaCalle listaCalle;//lista de calles la cual contendra lista de pisos y pila de ubicaciones con diferentes artículos.

crearListaCalle(listaCalle);

ListaArtDeposito listaArtDep;//lista de articulos cargados en el deposito

crearListaArtDeposito(listaArtDep);

ListaFaltante listaFaltante;//lista de articulos faltanes con las que no se pudieron responder a las solicitudes

crearListaFaltante(listaFaltante);

Reposicion reposicion;//instacia de ingreso,estructura para almacenar datos de ingreso

crearReposicion(reposicion);

Solicitud solicitud;//instancia de solicitud,estructura para almacenar datos de solicitud

crearSolicitud(solicitud);

Camion camion;//camion que lleve la solicitud

crearCamion(camion);

PilaCamion pilaCamion;//pila de camiones que fueron despachados

crearPila(pilaCamion);

ListaPesoAcum listaPesoAcum;//lista para alamacenar los saldos agrupados

crearListaPesoAcum(listaPesoAcum);

ListaVentas listaVentas;//lista para almacenar las ventas

crearListaVenta(listaVentas);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CARGA CONFIGURACION\*\*\*\*\*\*\*\*\*\*\*\*

cargaConfig(configuracion);//CARGA ARCHIVO CONFIGURACION

//\*\*\*\*\*\*\*\*\*\*\*\*CARGA MASTER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

cargaMaster(listArt);//CARGA EL MASTER DE ARTICULOS

//\*\*\*\*\*\*\*\*\*\*\*COMPARACION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FILE \*ptrING; //Puntero de archivo INGRESO.

FILE \*ptrSOL; //Puntero de archivo SOLICITUDES.

ptrING=fopen("ingresos.db","r"); // LOS ABRIMOS EN EL MAIN PARA CERRARLOS

ptrSOL=fopen("solicitudes.db","r"); // DENTRO DEL MISMO.

int contador=0;

int paso=0;

bool sinfin=true;

//A modo de prueba, leemos la primer linea de cada archivo.

while(sinfin==true){

while (!feof(ptrING) && !feof(ptrSOL)) {

contador++;

paso=calcularTiempo(ptrING,ptrSOL,solicitud,reposicion);

if(paso==1){

ingresoStock(configuracion,listaCalle,listArt,listaArtDep,reposicion);

}

else{

solicitudStock(configuracion,listaCalle,listArt,listaArtDep,listaFaltante,solicitud,camion,pilaCamion,listaPesoAcum,listaVentas); //listaVentast

}

cout <<" "<< contador << endl;

}

if (!feof(ptrSOL) && feof(ptrING)){

contador++;

cout <<" "<< contador << endl;

leerSolicitud(ptrSOL,solicitud); solicitudStock(configuracion,listaCalle,listArt,listaArtDep,listaFaltante,solicitud,camion,pilaCamion,listaPesoAcum,listaVentas); //listaVentast

}

if (feof(ptrSOL) && !feof(ptrING)){

contador++;

cout <<" "<< contador << endl;

leerIngreso(ptrING,reposicion);

ingresoStock(configuracion,listaCalle,listArt,listaArtDep,reposicion); }

if((feof(ptrING)) && (feof(ptrSOL))){

sinfin=false; }

}//WHILE SINFIN

printf("TERMINO EL PROGRAMA. Cerrando archivos\n");

fclose(ptrING);// CERRAMOS LOS ARCHIVOS AL FINAL DEL PROGRAMA

fclose(ptrSOL);// //

printf("Archivos cerrados\n");

system("cls");

//\*\*\*\*\*\*\*\*ARCHIVOS DE TEXTO\*\*\*\*\*\*\*\*\*

crearSaldosDetalladosTxt(listaCalle);

ordenamientofaltantes(listaFaltante);

crearPesoAcumulado(listaCalle,listaPesoAcum);

ordenamientoSaldosagrupados(listaPesoAcum);

crearPesoAcumuladoTxt(listaPesoAcum);

ordenarlista(listaVentas);

crearVentaTxt(listaVentas);

crearCamionTxt(pilaCamion);

crarFaltanteTxt(listaFaltante);

return 0;

}

* **FunYProd.Cpp**

#include <cstdlib>

#include <stdio.h>

#include <conio.h>

#include <string>

#include <time.h>

#include <cstring>

#include <fstream>

#include <stdlib.h>

#include <ctype.h>

#include <iostream>

#include "Configuracion.h"

#include "FunYProd.h"

#include "LDeposito.h"

#include "ListaArticulo.h"

#include "ListaCalle.h"

#include "ListaFaltantes.h"

#include "ListaPesoAcumulado.h"

#include "ListaPiso.h"

#include "ListaVentas.h"

#include "PilaArtUb.h"

#include "PilaCamion.h"

#include "Reposicion.h"

#include "Solicitud.h"

using namespace std;

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

void cargaMaster(ListaArt &listArt){

//1 recorrer el archivo

//2 en variables aux guardamos el valor de la linea

//3 creamos un instancia de articulos

//4 a esa instancia le pasamos los datos convertidos

//5 mientras sea fin de archivo adicionar a la lista

ifstream arcArticulos("articulos.db");

Articulo artAux;

constructorArt(artAux);

char linea[128];

char\* cpToken2;

if(arcArticulos.fail()){

cerr << "Error al abrir el archivo articulos\n" << endl;}

else{

while(!arcArticulos.eof() {

arcArticulos.getline(linea, sizeof(linea));

cpToken2 = strtok (linea, ";");

setCodArt(artAux,atoi(cpToken2));

cpToken2= strtok(NULL,",");

float ent=atof(cpToken2);

cpToken2=strtok(NULL,";");

float dec=atof(cpToken2);

setPrecio(artAux,(ent+(dec/100)));

cpToken2= strtok(NULL,"\n");

strcpy(artAux.descripcion,cpToken2);

adicionarFinal(listArt,artAux); } }

arcArticulos.close(); }

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

void cargaConfig(Config &config){

ifstream archConfig("configuracion.conf");

char linea[12];

char\* cpToken;

if(archConfig.fail()){

cerr << "Error al abrir el archivo CONFIGURACION\n" << endl;}

else{

while(!archConfig.eof()){

archConfig.getline(linea, sizeof(linea));

cpToken = strtok (linea, ";");

setConfigP(config,atoi(cpToken));

cpToken= strtok(NULL,";");

setConfigU(config,atoi(cpToken));

cpToken=strtok(NULL,"\n");

setConfigKC(config,atof(cpToken)); } }

archConfig.close(); }

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

int calcularTiempo(FILE \*ptrING,FILE \*ptrSOL,Solicitud &solicitud,Reposicion &reposicion){

//Variables que se usan en el primer paso.

//cadenaI Se usa para guardar la linea que se lee del archivo.

//calTIEMPO se usa para guardar en forma de char, el tiempo. Ej: 08:00:00

char cadenaI[50],calTiempo[20];

char\* cpToken; //cpToken se utiliza para guardar caracteres desde cierto punto hasta cierto punto en una linea de caracteres

int PosIniING,h,m,s,hh,mm,stingreso; //los POS(posicion del cursor), son antes y despues de leer en el archivo

//Variables que se usan en el segundo paso.

char cadenaS[18];

char\* cpToken2;

int hh2,mm2,stsolicitud,pos2;

PosIniING=ftell(ptrING);

//LEE LA PRIMER CADENA EN EL ARCHIVO INGRESO Y GUARDAMOS EN cadenaI

fscanf (ptrING, "%s", cadenaI);

cpToken = strtok (cadenaI, ";"); //Desde la cadenaI leemos del principio hasta ";"

// obtiene el primer campo del registro y lo guarda en cpToken

reposicion.cod\_articulo= atoi(cpToken); //Transformamos en INT y guardamos en CodArt

cpToken = strtok (NULL, ";"); //Continuamos leyendo en cadenaI pero ahora desde NULL.

reposicion.cantidad= atoi(cpToken); //Transformamos en FLOAT

cpToken = strtok (NULL, ";");

strcpy (reposicion.hora, cpToken);

// obtiene el tercer campo... HORA

;//contiene en forma de cadena la hora

strcpy (calTiempo, cpToken); // COPIA EL CONTENIDO DE CPTOKEN EN LA CADENA

//printf("%s \n",calTiempo);

cpToken= strtok(calTiempo,":");

h=atoi(cpToken);

cpToken= strtok (NULL,":");

m=atoi(cpToken);

cpToken= strtok (NULL,":");

s=atoi(cpToken);

hh=h\*3600;

mm=m\*60;

stingreso=hh+mm+s; //TRANSFORMA LA HORA EN SEGUNDOS EJ: 08:01:37 = 28897 SEGS.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FIN INGRESO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* pos2=ftell(ptrSOL);

fscanf (ptrSOL, "%s", cadenaS) ;

cpToken2 = strtok (cadenaS, ";"); // obtiene el primer campo dela cadena y lo guarda en cpToken

setCodSuc(solicitud,atoi(cpToken2));//Lo guarda en

cpToken2 = strtok (NULL, ";");// Obtiene el CodART

setCodArt(solicitud,atoi(cpToken2));

cpToken2 = strtok (NULL, ";"); //obtiene el segundo campo... cantidad

setCantidad(solicitud,atof(cpToken2)); //Lo guardo en la estructura

cpToken2 = strtok (NULL, ";"); //Obtiene la HORA

strcpy (solicitud.hora, cpToken2); //Lo guarda en la estructura.

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

strcpy (calTiempo, cpToken2); //En CalTiempo, guardamos la Hora que estaba en cptoken2 para trabajarla.

cpToken2= strtok(calTiempo,":");

h=atoi(cpToken2);

cpToken2= strtok (NULL,":");

m=atoi(cpToken2);

cpToken2= strtok (NULL,":");

s=atoi(cpToken2);

hh2=h\*3600;

mm2=m\*60;

stsolicitud=hh2+mm2+s; //vALOR EN SEGUNDOS DE ESA HORA.

if (stingreso<=stsolicitud){

fseek(ptrSOL, pos2, SEEK\_SET);

return 1; }

else {

fseek( ptrING, PosIniING, SEEK\_SET);

return 2; } }

void leerSolicitud(FILE \*ptrSOL,Solicitud &solicitud){

char cadenaS[18];

char\* cpToken2;

fscanf (ptrSOL, "%s", cadenaS) ;

cpToken2 = strtok (cadenaS, ";"); // obtiene el primer campo dela cadena y lo guarda en cpToken

setCodSuc(solicitud,atoi(cpToken2));//Lo guarda en

cpToken2 = strtok (NULL, ";");// Obtiene el CodART

setCodArt(solicitud,atoi(cpToken2));

cpToken2 = strtok (NULL, ";"); //obtiene el segundo campo... cantidad

setCantidad(solicitud,atof(cpToken2)); //Lo guardo en la estructura

cpToken2 = strtok (NULL, ";"); //Obtiene la HORA

strcpy (solicitud.hora, cpToken2); //Lo guarda en la estructura.

cout << solicitud.cod\_articulo<< " " <<solicitud.cod\_suc<< " " <<solicitud.cantidad<< " " <<solicitud.hora<<endl; }

void leerIngreso(FILE \*ptrING,Reposicion &reposicion){

char cadenaI[50];

char\* cpToken;

fscanf (ptrING, "%s", cadenaI);

cpToken = strtok (cadenaI, ";");

reposicion.cod\_articulo= atoi(cpToken);

cpToken = strtok (NULL, ";");

reposicion.cantidad= atoi(cpToken);

cpToken = strtok (NULL, ";");

strcpy (reposicion.hora, cpToken);

cout << reposicion.cod\_articulo << " " << reposicion.cantidad << " " << reposicion.hora<< endl; }

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

//BUSCA Y DEVUELVE EL ARTICULO DE LA LISTA ARTICULO

Articulo buscarArticulo(ListaArt &listaArt, int codart){

PrtNodoListaArt ptrArticulo = primero(listaArt);

bool encontrado=false;

while(ptrArticulo != NULL && encontrado != true){

if ( getCodArt(ptrArticulo->datoLArt) == codart){

encontrado = true; }

else{

ptrArticulo = siguiente(listaArt,ptrArticulo); } }

return (ptrArticulo->datoLArt); }

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

void ingresoStock(Config &configuracion, ListaCalle &listaC, ListaArt &listaArt, ListaArtDeposito &listaArtDep,Reposicion &reposicion){

cout<<"INGRESO DE ARTICULO"<<endl;

cout<<"Cod. Articulo: "<< getCodArt(reposicion) <<endl;

cout<<"Cantidad: "<< getCant(reposicion) << endl;

int codArti=0;

if(listaVacia(listaArtDep)){

// SI ESE ARTICULO NUEVO, ES EL PRIMERO AGREGADO

PrtNodoListaArt ptrArt= primero(listaArt);

codArti=reposicion.cod\_articulo;

bool encont=false;

while (ptrArt !=NULL && encont !=true){

if (getCodArt(ptrArt->datoLArt)==codArti){ //COMPROBAR EXISTENCIA

encont=true;

}

else {

ptrArt=siguiente(listaArt, ptrArt);

}

}

if(encont==true){

Articulo art1 = buscarArticulo(listaArt,getCodArt(reposicion));

ArtUbicacion articuloU1;

constructorArtUb(articuloU1,art1,1,1,1,getCant(reposicion));

PilaArtUb pilaA1;

crearPila(pilaA1);

push(pilaA1,articuloU1);

ListaPiso piso1;

crearListaPiso(piso1);

adicionarFinal(piso1,pilaA1);

adicionarFinal(listaC, piso1);

ArtDeposito artDep1;

crearArtDep(artDep1);

//CORRECCION

PtrNodoPisoDep pisoDep=primero(piso1);

setPisoDep(artDep1,pisoDep);

//

setCodArtDep(artDep1, getCodArt(reposicion));

//setPisoDep(artDep1, primero(primero(listaC)->piso1));

adicionarFinal(listaArtDep,artDep1);

cout<<"Primer articulo guardado exitosamente"<<endl;

}

else{

cout<<"El articulo no se encuentra"<<endl;

}

}

else{

bool encontrado=false; //FLAG DE REFERENCIA DE ARTICULO ENCONTRADO

//DETERMINAR EN PRIMER LUGAR LA UBICACION EN EL DEPOSITO

PtrNodoListaDeposito ptrNodoListaDeposito = primero(listaArtDep); //

while(ptrNodoListaDeposito!= NULL && encontrado != true){ // BUSCO EL ARTICULO

if ( getCodArtDep(ptrNodoListaDeposito->datoArtDep) == getCodArt(reposicion)){ // PARA VER SI YA EXISTE

encontrado = true; // EN DEPOSITO,

} // PERO REALIZO LA BUSQUEDA

else{ // A TRAVES DEL INDICE

ptrNodoListaDeposito = siguiente(listaArtDep,ptrNodoListaDeposito); // (ListaArtDeposito)

} //

}

//SI EL ARTICULO EXISTE, SE ALMACENARA LA CANTIDAD EN LA UBICACION YA UTILIZADA

if (encontrado == true){

PilaArtUb pilaAux; //CREO UNA PILA AUXILIAR PARA GUARDAR LOS

crearPila(pilaAux); //ARTICULOS QUE VOY SACANDO DEL DEPOSITO

bool encontradoP= false; //FLAG DE REFERENCIA DE ENCONTRADO EN LA PILA

PtrNodoPisoDep ptrPiso = getPisoDep(ptrNodoListaDeposito->datoArtDep);//NODO DE PISO DONDE ESTA EL ARTICULO

Articulo artEnDep;

constructorArt(artEnDep);

while ( !pilaVacia(ptrPiso->pilUb) && encontradoP != true){ //ME POSICIONO EN EL NODO DEL ARTICULOUBICACION

artEnDep = getDatoArt((top(ptrPiso->pilUb))->datoArtUb); //OBTENGO EL ARTICULO APUNTADO

if( getCodArt(artEnDep) == getCodArt(reposicion) ) { //SI EL COD DEL ARTICULO EN TOP

float cantAct = getCantArtUb((top(ptrPiso->pilUb))->datoArtUb) + getCant(reposicion); //COINCIDE CON EL DEL INGRESO

setCantArtUb(((top(ptrPiso->pilUb))->datoArtUb),cantAct); //LE AGREGO LA CANTIDAD AL

encontradoP= true; //ARTICULO EXISTENTE

cout<<"\tActualizacion de stock de articulo existente"<<endl;

cout<<cantAct<<getCantArtUb((top(ptrPiso->pilUb))->datoArtUb)<<endl;

}

else{

push(pilaAux, pop(ptrPiso->pilUb) ); //SACA LOS ARTICULOS Y LOS PONE EN LA PILA AUXILIAR

}

}

while(!pilaVacia(pilaAux) ){

push((ptrPiso->pilUb), pop(pilaAux)); //METE LOS ARTICULOS SACADOS

}

}

//SI EL ARTICULO NO EXISTE EN DEPOSITO: SE SOLICITARA UNA NUEVA POSICIÓN

else{

bool aRegist = false; //FLAG PARA ARTICULO REGISTRADO

int contadorC=0,contadorP=0; //POSTERIORMENTE SON UTILIZADOS DE CONTADORES DE CALLES Y DE PISOS

int longUb=0;

PrtNodoListaArt ptrArt= primero(listaArt);

codArti=reposicion.cod\_articulo;

bool encont=false;

while (ptrArt !=NULL && encont !=true){

if (getCodArt(ptrArt->datoLArt)==codArti){ //COMPROBAR EXISTENCIA

encont=true;

}

else {

ptrArt=siguiente(listaArt, ptrArt);

}

}

if(encont==true){

Articulo artUbic = buscarArticulo(listaArt, getCodArt(reposicion));

ArtUbicacion artNUb; //GUARDARA EL ARTUBIC QUE ES EL ARTICULO A UBICAR

constructorArtUb(artNUb,artUbic,0,0,0,getCant(reposicion)); //VALORES

ArtDeposito artDepNuevo; //GUARDA LA POSICION DEL ARTICULO NUEVO

crearArtDep(artDepNuevo);

ListaPiso listPisoN; //ESTO ES POR SI TENGO QUE INSERTAR UN PISO NUEVO

crearListaPiso(listPisoN);

PilaArtUb ubiNueva; //Y ASI TAMBIEN POR SI NECESITO UNA NUEVA UBICACION

crearPila(ubiNueva);

PtrNodoCalle ptrC = primero(listaC);

while(ptrC != NULL && !aRegist){

contadorC++;

PtrNodoPiso ptrP = primero(ptrC->listPiso);

while(ptrP != NULL && !aRegist){ //RECORRO LISTA PISOS

contadorP++;

if (!pilaVacia(ptrP->pilUb)){

longUb=ptrP->pilUb.top->datoArtUb.u;

}

if (longUb < getConfigU(configuracion)&& !aRegist){ //MIENTRAS LA LONGITUD ES MENOR A LO DETERMINADO EN CONFIGURACION Y SEA DISTINYO DE AREGIST

setC(artNUb,contadorC); setP(artNUb,contadorP);

setU(artNUb,longUb+1);

push(ptrP->pilUb,artNUb); //COLOCO EL ARTNUB EN LA PILA DE UBICACIONES

setCodArtDep(artDepNuevo, getCodArt(reposicion)); //

setPisoDep(artDepNuevo, ptrP); //PARA EL NUEVO INDICE adicionarFinal(listaArtDep,artDepNuevo); //

aRegist = true; //CAMBIO EL FLAG

cout<<artNUb.datoArt.codart<<" "<<artNUb.c<<" "<<getP(artNUb)<<" "<<getU(artNUb)<<" "<<getCantArtUb(artNUb)<<endl;

cout<<"Nuevo articulo guardado exitosamente"<<endl;

}

//}

ptrP = siguiente(ptrC->listPiso,ptrP);

}

//SI NO HAY LUGAR, AÑADO UN NUEVO PISO PARA ESA CALLE

if(aRegist==false && contadorP<getConfigP(configuracion)){

setC(artNUb,contadorC); //

setP(artNUb,contadorP + 1); //SETEO LAS COORDENADAS A INGRESAR

setU(artNUb,1); //

push(ubiNueva,artNUb);

PtrNodoPisoDep ptrP = adicionarFinal(ptrC->listPiso,ubiNueva); //AGREGO EL PISO

setCodArtDep(artDepNuevo, getCodArt(reposicion));

setPisoDep(artDepNuevo, ptrP); //AGREGO EL NUEVO ARTICULO AL INDICE

adicionarFinal(listaArtDep,artDepNuevo);

aRegist = true;

cout<<"\tNuevo piso creado "<<contadorC<<", nuevo articulo guardado exitosamente"<<endl;

}

contadorP=0;

ptrC = siguiente(listaC, ptrC);

}

// SI NECESITO CREAR UNA NUEVA CALLE

if(aRegist==false){

setC(artNUb,contadorC + 1); //SETEO LAS COORDENADAS DEL NUEVO ARTICULO setP(artNUb,1);

setU(artNUb,1);

push(ubiNueva,artNUb);

PtrNodoPisoDep ptrPiso2 = adicionarFinal(listPisoN,ubiNueva); //AGREGO UN NODO DE UBICACION A LA LISTPISON

adicionarFinal(listaC, listPisoN);

setCodArtDep(artDepNuevo, getCodArt(reposicion)); //

setPisoDep(artDepNuevo, ptrPiso2); //AGREGO EL NUEVO ARTICULO AL INDICE

adicionarFinal(listaArtDep,artDepNuevo); //

cout<<"\Nueva calle: "<<contadorC+1<<". Nuevo articulo guardado exitosamente."<<endl;

}

}

else{

cout<<"El articulo no se encuentra"<<endl;

} } } }

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

void solicitudStock(Config &configuracion,ListaCalle &listaCalle,ListaArt &listArt,ListaArtDeposito &listaArtDep,ListaFaltante &listaFaltante,Solicitud &solicitud,Camion &camion,PilaCamion &pilaCamion,ListaPesoAcum &listaPesoAcum,ListaVentas &listaVentas){

float monto=0;

int contador=0;

int cantidadCumplida=0;

int faltanteCantidad=0;

bool eliminarN = false;

ArtUbicacion artaux;

constructorArtUb(artaux);

PrtNodoListaArt ptr=primero(listArt);

PtrNodoListaDeposito ptrdep=primero(listaArtDep);

PtrNodoPiso nodo=ptrdep->datoArtDep.pisoDep;

PtrNodoPilaArtUb ptrPila=top(ptrdep->datoArtDep.pisoDep->pilUb);

PilaArtUb pilaAux;

crearPila(pilaAux);

bool articuloEnDep = false;

while(ptrdep != NULL && articuloEnDep != true){

if(getCodArt(solicitud) == ptrdep->datoArtDep.codArt){

articuloEnDep = true;

}else{

ptrdep = siguiente(listaArtDep,ptrdep); } }

if(articuloEnDep == true){

while(!pilaVacia(ptrdep->datoArtDep.pisoDep->pilUb )){

artaux=pop(ptrdep->datoArtDep.pisoDep->pilUb);

//cout << artaux.datoArt.codart <<"ACACACACA"<< solicitud.cod\_articulo<<endl;

//SI "SON IGUALES LAS ID" COMPROBAMOS QUE HAYA SUFIENTE EN STOCK

if ((artaux.datoArt.codart == solicitud.cod\_articulo)){

// cout << "HAY COINCIDENCIA" << endl;//NOS ALCANZA? SI

if (getCantArtUb(artaux)>solicitud.cantidad){

//ENTONCES, HACEMOS LA RESTA.

artaux.cantArtUb-=solicitud.cantidad;

//VENTAS, CAMION, ETC.

//Ya tenemos lo que falta, ahora enviamos lo que tenemos al camion.

Solicitud solaux;

crearSolicitud(solaux,solicitud.cod\_suc,solicitud.cod\_articulo,solicitud.cantidad,solicitud.hora);

//Cargamos camion.

cargaCamiones(pilaCamion,solaux,configuracion);

//Cargamos la venta

monto=solicitud.cantidad\*artaux.datoArt.precio;

cargaVentas(listaVentas,solaux,monto);

//Pusheamos en la pila auxiliar con los datos ya modificados.

push(pilaAux,artaux);

}

else if(getCantArtUb(artaux)<=solicitud.cantidad){

printf("NO ALCANZA\n");

//ENVIAR LO QUE TENEMOS.

//Primero restamos para calcular DIFERENCIA FALTATE.

faltanteCantidad=solicitud.cantidad-artaux.cantArtUb;

cantidadCumplida=artaux.cantArtUb;

//Ya tenemos lo que falta, ahora enviamos lo que tenemos al camion.

Solicitud solaux;  
crearSolicitud(solaux,solicitud.cod\_suc,solicitud.cod\_articulo,cantidadCumplida,solicitud.hora)

//Cargamos camion.

cargaCamiones(pilaCamion,solaux,configuracion);

//Cargamos la venta

monto=artaux.cantArtUb\*artaux.datoArt.precio;

cargaVentas(listaVentas,solaux,monto);

//Creo el faltante.

float cantidadCumplida2=cantidadCumplida;

float porcentaje=cantidadCumplida\*100/solicitud.cantidad;

//Añadimos el faltante a la lista.

Faltante faltaaux;

crearFaltante(faltaaux,solicitud.cod\_suc,solicitud.cod\_articulo,cantidadCumplida2,solicitud.cantidad,porcentaje);

cargarFaltante(listaFaltante,faltaaux);

//QUITAR ARTICULO DE LA PILA

//CAMBIAMOS CPU DE Los OTROS ARTICULO EN LA PILA.

eliminarN = true; } }

else{ push(pilaAux,artaux); } } while(!pilaVacia(pilaAux)){

artaux=pop(pilaAux);

contador++;

setU(artaux,contador);

push(ptrdep->datoArtDep.pisoDep->pilUb,artaux);

cout << "Aca GUARDO un ARTICULO EN UBICACION "<< ptrdep->datoArtDep.pisoDep->pilUb.top->datoArtUb.u << endl;

//getCantArtUb(artaux) << endl;

}

if(eliminarN == true){

eliminarNodo(listaArtDep,ptrdep); } }

if(articuloEnDep == false){

//NO ESTA EN DEPOSITO

cout << "NO ESTA EN DEPOSITO, AÑADIMOS A FALTANTES" << endl;

Faltante faltaux2;

crearFaltante(faltaux2,solicitud.cod\_suc,solicitud.cod\_articulo,solicitud.cantidad,0,0.00);

adicionarFinal(listaFaltante,faltaux2); } }

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

void crearPesoAcumulado(ListaCalle &listaC,ListaPesoAcum &listaPesoAcum){

PtrNodoCalle ptrC=primero(listaC);

PtrNodoPiso ptrPiso=primero(ptrC->listPiso);

PtrNodoPilaArtUb ptrPila=top(ptrPiso->pilUb);

ArtUbicacion artAux;

constructorArtUb(artAux);

PilaArtUb pilAux;

crearPila(pilAux);

PesoAcum pesoAcumAux;

if(listaVacia(listaC)==true){

cout<<"LISTA CAlle VACIA"<<endl;}

else{

//RECORRO LA LISTA DE CALLE

while(!listaVacia(listaC)&&ptrC!=NULL){

if (listaVacia(ptrC->listPiso)){

cout<<"LISta PIso Vacia"<<endl; }

else{

//RECORRO LA LISTA DE PIso

ptrPiso=primero(ptrC->listPiso);

while(!listaVacia(ptrC->listPiso)&& ptrPiso!=NULL){

//RECORRO LA PILA DE ARTICULOS

if(pilaVacia(ptrPiso->pilUb)){

cout<<"PILA ART VACIA"<<endl; }

else{

while(!pilaVacia(ptrPiso->pilUb)){

//hago el pop a la auxiliar

artAux=pop(ptrPiso->pilUb);

crearPesoAcum(pesoAcumAux,artAux.datoArt.codart,artAux.cantArtUb);

adicionarFinal(listaPesoAcum,pesoAcumAux);

push(pilAux,artAux); }

while(!pilaVacia(pilAux)){

artAux=pop(pilAux);

push(ptrPiso->pilUb,artAux); } }

ptrPiso=siguiente(ptrC->listPiso,ptrPiso); } }

ptrC=siguiente(listaC,ptrC); }}}

void cargarFaltante(ListaFaltante &listaFaltante,Faltantes &faltante){

PtrNodoFaltante ptrF=primero(listaFaltante);

if(listaVacia(listaFaltante)){

Faltantes f1;

crearFaltante(f1,faltante.codsucursal,faltante.codArt,faltante.cantTot,faltante.cantEnv,faltante.porc);

adicionarPrincipio(listaFaltante,f1);}

else{ Faltantes f2; crearFaltante(f2,faltante.codsucursal,faltante.codArt,faltante.cantTot,faltante.cantEnv,faltante.porc);

adicionarFinal(listaFaltante,f2);}}

void cargaVentas(ListaVentas &listaVentas,Solicitud &solicitud,float &monto){

bool encontrado=false;

PtrNodoListaVentas ptrLV=primero(listaVentas);

if (listaVacia(listaVentas)){

Venta v1;

crearVentas(v1,solicitud.cod\_suc,monto);

adicionarPrincipio(listaVentas,v1);}

else {

while(ptrLV!=NULL && encontrado!=true){

if(solicitud.cod\_suc==ptrLV->datoVenta.codSucursal){

encontrado=true;

ptrLV->datoVenta.montoTotal+=monto; }

ptrLV=siguiente(listaVentas,ptrLV);}

if (encontrado==false){

Venta v2;

crearVentas(v2,solicitud.cod\_suc,monto);

adicionarFinal(listaVentas,v2);}}}

void cargaCamiones(PilaCamion &pilacamion,Solicitud &solicitud,Config &config){

PtrNodoPilaCamion ptrPC=top(pilacamion);

float KxC=getConfigKC(config);//carga maxima del camion dado por la configuracion

float solCant=getCant(solicitud);//cantidad de la solicitud

Camion auxCam;

Camion auxCam1;

Camion auxCam2;

Camion auxCam3;

if (pilaVacia(pilacamion)){

crearCamion(auxCam,1,solCant,((solCant\*100)/KxC));

push(pilacamion,auxCam);

cout << "PILA CAMION VACIA Creamos un camion con la solicitud\n" << endl;}

else { int IDAux=getId\_Camion(ptrPC->datoCamion);

if (getCarga\_T(ptrPC->datoCamion)>=KxC){

printf("No me alcanza");

crearCamion(auxCam1,IDAux+1,solCant,((solCant)\*100)/KxC);

push(pilacamion,auxCam1);

cout<<"Creamos un camion nuevo, el camion del top estaba lleno\n"<<endl; }

else{ if(getCarga\_T(ptrPC->datoCamion)<KxC){

cout<<"El camion del top de la pila estaba CASI lleno\n"<<endl;

//saco el camion de la lista para utilizarlo

Camion camAux;

camAux=pop(pilacamion);

float suma=solCant+getCarga\_T(camAux);

float sumaPorcentaje=((solCant\*100)/KxC)+getPorcentaje(camAux);

int idCamionPop=getId\_Camion(camAux);

//si la cantidad de la solicitud y la cantidad q ya tenia el camion no superan la cantidad maxima

if(suma<=KxC){

//creo el camion y lo cargo

crearCamion(auxCam2,idCamionPop,suma,sumaPorcentaje);

push(pilacamion,auxCam2);

cout<<"Al camion que estaba en el top lo cargamos con otra solicitud \n"<<endl; } //si la suma de la cantidad de solicitud con lo cargado en el camion supera la carga maxima

else {//sino crear un nuevo camion y pushear el q saque

if(suma>KxC){//si la suma es mayor a la que soporta un camion

//creo un camion un el mismo id del camion q saque, con la caga maxima y su porcentaje

push(pilacamion,camAux);

crearCamion(auxCam3,idCamionPop+1,solCant,((solCant\*100)/KxC));

push(pilacamion,auxCam3);

// despacho un camion con el resto de la carga y su porcentaje

cout<<"Al camion que estaba en el top lo cargamos con otra solicitud y creamos uno nuevo porque excedio el limite\n"<<endl; } } } } }}

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

void ordenamientoSaldosagrupados(ListaPesoAcum &listaPesoAcum){

NodoListaPAcum\* temporalhead=primero(listaPesoAcum);

int temporal;

int contar;

float temporal2;

float temporal3;

while (temporalhead != NULL) {

temporalhead=temporalhead->sgtePA;

contar++; }

temporalhead= primero(listaPesoAcum);

for (int i=0;i< contar ;i++) {

while (temporalhead ->sgtePA) {

if (temporalhead->datoPesoA.codArt > temporalhead->sgtePA->datoPesoA.codArt ) {

temporal = temporalhead->datoPesoA.codArt;

temporalhead->datoPesoA.codArt=temporalhead->sgtePA->datoPesoA.codArt;

temporalhead->sgtePA->datoPesoA.codArt=temporal;

temporal2 = temporalhead->datoPesoA.cant;

temporalhead->datoPesoA.cant=temporalhead->sgtePA->datoPesoA.cant;

temporalhead->sgtePA->datoPesoA.cant=temporal2; }

else temporalhead=temporalhead->sgtePA; }

temporalhead = primero(listaPesoAcum); } }

void ordenarlista(ListaVentas &listaventas){

NodoListaVentas\* temporalhead= primero(listaventas);

int temporal;

int contar=0;

while (temporalhead != NULL) {

temporalhead=temporalhead->sgteV;

contar++; }

temporalhead= primero(listaventas);

for (int i=0;i< contar ;i++) {

while (temporalhead ->sgteV) {

if (temporalhead->datoVenta.montoTotal < temporalhead->sgteV->datoVenta.montoTotal) {

temporal = temporalhead->datoVenta.codSucursal;

temporalhead->datoVenta.codSucursal=temporalhead->sgteV->datoVenta.codSucursal;

temporalhead->sgteV->datoVenta.codSucursal=temporal;

temporal = temporalhead->datoVenta.montoTotal;

temporalhead->datoVenta.montoTotal=temporalhead->sgteV->datoVenta.montoTotal;

temporalhead->sgteV->datoVenta.montoTotal=temporal; }

else temporalhead=temporalhead->sgteV; }

temporalhead = primero(listaventas); }}

void ordenamientofaltantes(ListaFaltante &listaFaltante){

NodoFaltante\* temporalhead= primero(listaFaltante);

int temporal;

int contar=0;

float temporal1=0;

float temporal2=0;

bool moverpuntero=false;

while (temporalhead != NULL) {

temporalhead=temporalhead->sgteF;

contar++; }

temporalhead= primero(listaFaltante);

for (int i=0;i< contar ;i++) {

while (temporalhead ->sgteF) {

moverpuntero=false;

if (temporalhead->datoFaltante.codsucursal >temporalhead->sgteF->datoFaltante.codsucursal) {

temporal = temporalhead->datoFaltante.codsucursal;

temporalhead->datoFaltante.codsucursal=temporalhead->sgteF->datoFaltante.codsucursal;

temporalhead->sgteF->datoFaltante.codsucursal=temporal;

temporal = temporalhead->datoFaltante.codArt;

temporalhead->datoFaltante.codArt=temporalhead->sgteF->datoFaltante.codArt;

temporalhead->sgteF->datoFaltante.codArt=temporal;

temporal1 = temporalhead->datoFaltante.cantTot;

temporalhead->datoFaltante.cantTot=temporalhead->sgteF->datoFaltante.cantTot;

temporalhead->sgteF->datoFaltante.cantTot=temporal1;

temporal2 = temporalhead->datoFaltante.cantEnv;

temporalhead->datoFaltante.cantEnv=temporalhead->sgteF->datoFaltante.cantEnv;

temporalhead->sgteF->datoFaltante.cantEnv=temporal2;

temporal = temporalhead->datoFaltante.porc;

temporalhead->datoFaltante.porc=temporalhead->sgteF->datoFaltante.porc;

temporalhead->sgteF->datoFaltante.porc=temporal;

moverpuntero =true; }

else if (temporalhead->datoFaltante.codsucursal == temporalhead ->sgteF->datoFaltante.codsucursal) {

if (temporalhead->datoFaltante.codArt > temporalhead->sgteF->datoFaltante.codArt) {

temporal = temporalhead->datoFaltante.codsucursal;

temporalhead->datoFaltante.codsucursal=temporalhead->sgteF->datoFaltante.codsucursal;

temporalhead->sgteF->datoFaltante.codsucursal=temporal;

temporal = temporalhead->datoFaltante.codArt;

temporalhead->datoFaltante.codArt=temporalhead->sgteF->datoFaltante.codArt;

temporalhead->sgteF->datoFaltante.codArt=temporal;

temporal1 = temporalhead->datoFaltante.cantTot;

temporalhead->datoFaltante.cantTot=temporalhead->sgteF->datoFaltante.cantTot;

temporalhead->sgteF->datoFaltante.cantTot=temporal1;

temporal2 = temporalhead->datoFaltante.cantEnv;

temporalhead->datoFaltante.cantEnv=temporalhead->sgteF->datoFaltante.cantEnv;

temporalhead->sgteF->datoFaltante.cantEnv=temporal2;

temporal = temporalhead->datoFaltante.porc;

temporalhead->datoFaltante.porc=temporalhead->sgteF->datoFaltante.porc;

temporalhead->sgteF->datoFaltante.porc=temporal;

moverpuntero=true; } }

if (moverpuntero==false){

temporalhead=temporalhead->sgteF; } }

temporalhead = primero(listaFaltante); } }

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

void crearSaldosDetalladosTxt(ListaCalle &listaC){

PtrNodoCalle ptrC=primero(listaC);

PtrNodoPiso ptrPiso=primero(ptrC->listPiso);

PtrNodoPilaArtUb ptrPila=top(ptrPiso->pilUb);

ArtUbicacion artAux;

constructorArtUb(artAux);

PilaArtUb pilAux;

crearPila(pilAux);

ofstream SaldosDetalladosTxt("Saldos\_Detallados.txt");

SaldosDetalladosTxt<<"C\t"<<"P\t"<<"U\t"<<"Cod.Art\t"<<"Cantidad"<<endl;

if(listaVacia(listaC)==true){

cout<<"LISTA CAlle VACIA"<<endl;}

else{//RECORRO LA LISTA DE CALLE

while(!listaVacia(listaC)&&ptrC!=NULL){

if (listaVacia(ptrC->listPiso)){

cout<<"LISta PIso Vacia"<<endl; }

else{

//RECORRO LA LISTA DE PIso

ptrPiso=primero(ptrC->listPiso);

while(!listaVacia(ptrC->listPiso)&& ptrPiso!=NULL){

//RECORRO LA PILA DE ARTICULOS

if(pilaVacia(ptrPiso->pilUb)){

cout<<"PILA ART VACIA"<<endl; }

else{

while(!pilaVacia(ptrPiso->pilUb)){

//hago el pop a la auxiliar

artAux=pop(ptrPiso->pilUb);

push(pilAux,artAux); }

while(!pilaVacia(pilAux)){

artAux=pop(pilAux);

SaldosDetalladosTxt<<getC(artAux)<<"\t"<<getP(artAux)<<"\t"<<getU(artAux)<<"\t"<<getDatoArt(artAux).codart<<"\t"<<getCantArtUb(artAux)<<endl;

push(ptrPiso->pilUb,artAux); } }

ptrPiso=siguiente(ptrC->listPiso,ptrPiso); } }

ptrC=siguiente(listaC,ptrC); }}}

void crearPesoAcumuladoTxt(ListaPesoAcum &listaPesoAcum){

PtrNodoListaPA ptrPA=primero(listaPesoAcum);

ofstream PesoAcumuladoTxt("PesoAcumulado.txt");

float pesoacumulado=0;

if(listaVacia(listaPesoAcum)==true){

cout<<"LISTA peso acumulado VACIA"<<endl;}

else{

while(!listaVacia(listaPesoAcum)&&ptrPA!=NULL){

pesoacumulado=ptrPA->datoPesoA.cant+pesoacumulado;

PesoAcumuladoTxt<<ptrPA->datoPesoA.codArt<<"\t"<<ptrPA->datoPesoA.cant<<"\t"<<pesoacumulado<<endl;

ptrPA=siguiente(listaPesoAcum,ptrPA); }}

PesoAcumuladoTxt.close();}

void crarFaltanteTxt(ListaFaltante &listaFaltante){

PtrNodoFaltante ptrF=primero(listaFaltante);

ofstream FaltanteTxt("Faltante.txt");

if(listaVacia(listaFaltante)==true){

cout<<"LISTA Ventas VACIA"<<endl;}

else{

while(!listaVacia(listaFaltante)&&ptrF!=NULL){

FaltanteTxt<< ptrF->datoFaltante.codsucursal<< "\t"<<ptrF->datoFaltante.codArt<<"\t"<<ptrF->datoFaltante.cantEnv<<"\t"<<ptrF->datoFaltante.cantTot<<"\t"<<ptrF->datoFaltante.porc <<"%"<<endl;

ptrF=siguiente(listaFaltante,ptrF); }}

FaltanteTxt.close();}

void crearVentaTxt(ListaVentas &listaVentas){

PtrNodoListaVentas ptr=primero(listaVentas);

ofstream VentasTxt("Ventas.txt");

if(listaVacia(listaVentas)==true){

cout<<"LISTA Ventas VACIA"<<endl;}

else{

while(!listaVacia(listaVentas)&&ptr!=NULL){

VentasTxt<<ptr->datoVenta.codSucursal<<"\t"<<ptr->datoVenta.montoTotal <<endl;

ptr=siguiente(listaVentas,ptr);}}

VentasTxt.close();}

* **ArtDeposito.cpp**

#include "ArtDeposito.h"

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

void crearArtDep(ArtDeposito &artDep){

artDep.codArt=0;

//artDep.pisoDep=NULL;}

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

void crearArtDep(ArtDeposito &artDep,int codart,PtrNodoPisoDep pisoDep){

artDep.codArt=codart;

artDep.pisoDep=pisoDep;}

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

void destruirArtDep(ArtDeposito &artDep){}

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

void setCodArtDep(ArtDeposito &artDep,int codart){artDep.codArt=codart;}

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

void setPisoDep(ArtDeposito &artDep,PtrNodoPisoDep pisoDep){

artDep.pisoDep=pisoDep;}

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

int getCodArtDep(ArtDeposito &artDep){

return artDep.codArt;}

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

PtrNodoPisoDep getPisoDep(ArtDeposito &artDep){

return artDep.pisoDep;}

* **ARTICULO.cpp**

#include "Articulo.h"

#include <stdio.h>

#include <string.h>

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

void constructorArt(Articulo &articulo){

articulo.codart=0;

articulo.precio=0;

strcpy(articulo.descripcion,"-");}

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

void constructorArt(Articulo &articulo,int codart,float precio, char d[]){

articulo.codart=codart;

articulo.precio=precio;

strcpy(articulo.descripcion,d);}

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

void setCodArt(Articulo &articulo,int codart){

articulo.codart=codart;}

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

void setPrecio(Articulo &articulo,float precio){

articulo.precio=precio;}

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

void setDescripcion(Articulo &articulo,char d[]){

strcpy(articulo.descripcion,d);}

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

int getCodArt(Articulo &articulo){

return articulo.codart;}

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

float getPrecio(Articulo &articulo){

return articulo.precio;}

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

char\* getDescripcion(Articulo &articulo){

char\* c;

return c=articulo.descripcion;}

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

void destructorArt(Articulo &articulo){}

* **ArtUbicacion.cpp**

#include "ArtUbicacion.h"

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

void constructorArtUb(ArtUbicacion &artUbicacion){

Articulo artAux;

constructorArt(artAux);

artUbicacion.datoArt=artAux;

artUbicacion.c=0;

artUbicacion.p=0;

artUbicacion.u=0;

artUbicacion.cantArtUb=0;}

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

void constructorArtUb(ArtUbicacion &artUbicacion,int c,int p,int u,float cantArtUb){

Articulo artAux;

constructorArt(artAux);

artUbicacion.datoArt=artAux;

artUbicacion.c=c;

artUbicacion.p=p;

artUbicacion.u=u;

artUbicacion.cantArtUb=cantArtUb;}

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

void constructorArtUb(ArtUbicacion &artUbicacion,Art &datoArtaux,int c,int p,int u,float cantArtUb){

artUbicacion.datoArt=datoArtaux;

artUbicacion.c=c;

artUbicacion.p=p;

artUbicacion.u=u;

artUbicacion.cantArtUb=cantArtUb;}

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

void setC(ArtUbicacion &artUbicacion,int c){

artUbicacion.c=c;}

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

void setP(ArtUbicacion &artUbicacion,int p){

artUbicacion.p=p;}

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

void setU(ArtUbicacion &artUbicacion,int u){

artUbicacion.u=u;}

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

void setDatoArt(ArtUbicacion &artUbicacion,Art &datoArt){

artUbicacion.datoArt=datoArt;}

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

void setCantArtUb(ArtUbicacion &artUbicacion,float cantArtUb){

artUbicacion.cantArtUb=cantArtUb;}

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

int getC(ArtUbicacion &artUbicacion){

return artUbicacion.c;}

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

int getP(ArtUbicacion &artUbicacion){

return artUbicacion.p;}

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

int getU(ArtUbicacion &artUbicacion){

return artUbicacion.u;}

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

Art getDatoArt(ArtUbicacion &artUbicacion){

return artUbicacion.datoArt;}

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

float getCantArtUb(ArtUbicacion &artUbicacion){

return artUbicacion.cantArtUb;}

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

void destructorArtUb(ArtUbicacion &artUbicacion){

destructorArt(artUbicacion.datoArt);}

* **Camion.cpp**

#include "Camion.h"

#include <iostream>

using namespace std;

//creamos el constructor del camion.

void crearCamion(Camion &camion){

camion.id\_camion=1;

camion.carga\_total=0;

camion.porcentaje=0;}

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

void crearCamion(Camion &camion,int idCamion,float Ct,float porc){

camion.id\_camion=idCamion;

camion.carga\_total=Ct;

camion.porcentaje=porc;}

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

//obtenemos el ID del camion

int getId\_Camion(Camion &camion){

return camion.id\_camion;}

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

//obtenemos la carga del camion

float getCarga\_T(Camion &camion){

return camion.carga\_total; }

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

//obtenemos el porcentaje del camion

float getPorcentaje(Camion &camion){

return camion.porcentaje; }

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

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

//modificamos el id del camion

void setId\_Camion(Camion &camion,int idCamion){

camion.id\_camion=idCamion;}

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

//modificamos la carga del camion

void setCarga\_T(Camion &camion,float cT){

camion.carga\_total=cT; }

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

//modificamos el porcentaje

void setPorcentaje(Camion &camion,float porc){

camion.porcentaje=porc;}

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

void destruirCamion(Camion &camion){}

* **Configuracion.cpp**

#include "Configuracion.h"

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

void constructorConfig(Config &config){

config.configP=0;

config.configU=0;

config.configKC=0;}

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

void constructorConfig(Config &config,int p,int u, float kc){

config.configP=p;

config.configU=u;

config.configKC=kc;}

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

void setConfigP(Config &config,int p){

config.configP=p;}

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

void setConfigU(Config &config,int u){

config.configU=u;}

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

void setConfigKC(Config &config,float kc){

config.configKC=kc;}

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

int getConfigP(Config &config){

return config.configP;}

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

int getConfigU(Config &config){

return config.configU;}

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

float getConfigKC(Config &config){

return config.configKC;}

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

void destructorConfig(Config &config){}

* **Faltantes.cpp**

#include "Faltantes.h"

void crearFaltante(Faltantes &faltantes){

faltantes.codArt=0;

faltantes.codsucursal=0;

faltantes.cantTot=0;

faltantes.cantEnv=0;

faltantes.porc=0;}

void crearFaltante(Faltantes &faltantes,int codsuc,int codart,float cantot,float cantenv,int porc){

faltantes.codArt=codart;

faltantes.codsucursal=codsuc;

faltantes.cantTot=cantot;

faltantes.cantEnv=cantenv;

faltantes.porc=porc;}

void destructorFaltantes(Faltantes &faltantes){}

int getCodArt(Faltantes &faltantes){

return faltantes.codArt;}

float getCantTot(Faltantes &faltantes){

return faltantes.cantTot;}

float getCantEnv(Faltantes &faltantes){

return faltantes.cantEnv;}

int getPorc(Faltantes &faltantes){

return faltantes.porc;}

void setCodArt(Faltantes &faltantes,int codart){

faltantes.codArt=codart;}

void setCantTot(Faltantes &faltantes,float cantot){

faltantes.cantTot=cantot;}

void setCantEnv(Faltantes &faltantes,float cantenv){

faltantes.cantEnv=cantenv;}

void setPorc(Faltantes &faltantes,int porc){

faltantes.porc=porc;}

int getCodSuc(Faltantes &faltantes){

return faltantes.codsucursal;}

void setCodSuc(Faltantes &faltantes,int codsuc){

faltantes.codsucursal=codsuc;}

* **LDeposito.cpp**

#include "LDeposito.h"

void crearListaArtDeposito(ListaArtDeposito &listaArtDep){

listaArtDep.primerArtDeposito=finAD();}

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

bool listaVacia(ListaArtDeposito &listaArtDep){

return (primero(listaArtDep)==finAD());}

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

PtrNodoListaDeposito finAD(){

return NULL;}

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

PtrNodoListaDeposito primero(ListaArtDeposito &listaArtDep){

return listaArtDep.primerArtDeposito;}

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

PtrNodoListaDeposito siguiente(ListaArtDeposito &listaArtDep,PtrNodoListaDeposito ptrNodoListaDep){

if ((! listaVacia(listaArtDep)) && (ptrNodoListaDep != finAD()))

return ptrNodoListaDep->sgteAD;

else{ return finAD();}

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

PtrNodoListaDeposito anterior(ListaArtDeposito &listaArtDep,PtrNodoListaDeposito ptrNodoListaDep){

PtrNodoListaDeposito ptrPrevio = finAD();

PtrNodoListaDeposito ptrCursor = primero(listaArtDep);

while (( ptrCursor != finAD()) && (ptrCursor != ptrNodoListaDep)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaArtDep,ptrCursor); }

return ptrPrevio;}

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

PtrNodoListaDeposito ultimo(ListaArtDeposito &listaArtDep){

return anterior(listaArtDep,finAD());}

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

PtrNodoListaDeposito crearNodoLista(ArtDep datoArtDep){

PtrNodoListaDeposito ptrAux = new NodoListaDeposito;

ptrAux->datoArtDep = datoArtDep;

ptrAux->sgteAD = finAD();

return ptrAux;}

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

PtrNodoListaDeposito adicionarPrincipio(ListaArtDeposito &listaArtDep,ArtDep datoArtDep){

/\* crea el nodo \*/

PtrNodoListaDeposito ptrNuevoNodo = crearNodoLista(datoArtDep);

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

ptrNuevoNodo->sgteAD = listaArtDep.primerArtDeposito;

listaArtDep.primerArtDeposito = ptrNuevoNodo;

return ptrNuevoNodo;}

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

PtrNodoListaDeposito adicionarDespues(ListaArtDeposito &listaArtDep,ArtDep datoArtDep, PtrNodoListaDeposito ptrNodoListaDep){

PtrNodoListaDeposito ptrNuevoNodo = finAD();

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

if (listaVacia(listaArtDep))

ptrNuevoNodo = adicionarPrincipio(listaArtDep,datoArtDep);

else {

if (ptrNodoListaDep != finAD()) {

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

ptrNuevoNodo = crearNodoLista(datoArtDep);

ptrNuevoNodo->sgteAD = ptrNodoListaDep->sgteAD;

ptrNodoListaDep->sgteAD = ptrNuevoNodo; } }

return ptrNuevoNodo;}

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

PtrNodoListaDeposito adicionarFinal(ListaArtDeposito &listaArtDep, ArtDep datoArtDep){

return adicionarDespues(listaArtDep,datoArtDep,ultimo(listaArtDep));}

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

void colocarDato(ListaArtDeposito &listaArtDep,ArtDep &datoArtDep, PtrNodoListaDeposito ptrNodoListaDep){

if ( (! listaVacia(listaArtDep)) && (ptrNodoListaDep != finAD()))

ptrNodoListaDep->datoArtDep = datoArtDep;}

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

void obtenerDato(ListaArtDeposito &listaArtDep,ArtDep &datoArtDep, PtrNodoListaDeposito ptrNodoListaDep){

if ((! listaVacia(listaArtDep)) && (ptrNodoListaDep != finAD()))

datoArtDep = ptrNodoListaDep->datoArtDep;}

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

void eliminarNodo(ListaArtDeposito &listaArtDep,PtrNodoListaDeposito ptrNodoListaDep){

PtrNodoListaDeposito ptrPrevio;

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

if ((! listaVacia(listaArtDep)) && (ptrNodoListaDep != finAD())) {

if (ptrNodoListaDep == primero(listaArtDep))

listaArtDep.primerArtDeposito = siguiente(listaArtDep,primero(listaArtDep));

else {

ptrPrevio = anterior( listaArtDep , ptrNodoListaDep );

ptrPrevio->sgteAD= ptrNodoListaDep->sgteAD; }

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

destruirArtDep(ptrNodoListaDep->datoArtDep);

delete ptrNodoListaDep; }}

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

void eliminarNodoPrimero(ListaArtDeposito &listaArtDep){

if (! listaVacia(listaArtDep))

eliminarNodo(listaArtDep,primero(listaArtDep));}

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

void eliminarNodoUltimo(ListaArtDeposito &listaArtDep){

if (! listaVacia(listaArtDep))

eliminarNodo(listaArtDep,ultimo(listaArtDep));}

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

void eliminarLista(ListaArtDeposito &listaArtDep){

while (! listaVacia(listaArtDep))

eliminarNodo(listaArtDep,primero(listaArtDep));}

* **ListaArticulo.cpp**

#include "ListaArticulo.h"

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

void crearListaArt(ListaArt &listaA){

listaA.primeroListaArt=finA();}

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

bool listaVacia(ListaArt &listaA){

return (primero(listaA)==finA());}

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

PrtNodoListaArt finA(){

return NULL;}

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

PrtNodoListaArt primero(ListaArt &listaA){

return listaA.primeroListaArt;}

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

PrtNodoListaArt siguiente(ListaArt &listaA,PrtNodoListaArt ptrNodoListaArt){

if ((! listaVacia(listaA)) && (ptrNodoListaArt != finA()))

return ptrNodoListaArt->sgteArt;

else{ return finA(); }

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

PrtNodoListaArt anterior(ListaArt &listaA,PrtNodoListaArt ptrNodoListaArt){

PrtNodoListaArt ptrPrevio = finA();

PrtNodoListaArt ptrCursor = primero(listaA);

while (( ptrCursor != finA()) && (ptrCursor != ptrNodoListaArt)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaA,ptrCursor); }

return ptrPrevio;}

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

PrtNodoListaArt ultimo(ListaArt &listaA){

return anterior(listaA,finA());}

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

PrtNodoListaArt crearNodoLista(LArt datoArticulo){

PrtNodoListaArt ptrAux = new NodoListaArt;

ptrAux->datoLArt = datoArticulo;

ptrAux->sgteArt = finA();

return ptrAux;}

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

PrtNodoListaArt adicionarPrincipio(ListaArt &listaA,LArt datoArticulo){

/\* crea el nodo \*/

PrtNodoListaArt ptrNuevoNodo = crearNodoLista(datoArticulo);

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

ptrNuevoNodo->sgteArt = listaA.primeroListaArt;

listaA.primeroListaArt = ptrNuevoNodo;

return ptrNuevoNodo;

}

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

PrtNodoListaArt adicionarDespues(ListaArt &listaA,LArt datoArticulo, PrtNodoListaArt ptrNodoListaArt){

PrtNodoListaArt ptrNuevoNodo = finA();

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

if (listaVacia(listaA))

ptrNuevoNodo = adicionarPrincipio(listaA,datoArticulo);

else {

if (ptrNodoListaArt != finA()) {

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

ptrNuevoNodo = crearNodoLista(datoArticulo);

ptrNuevoNodo->sgteArt = ptrNodoListaArt->sgteArt;

ptrNodoListaArt->sgteArt = ptrNuevoNodo; } }

return ptrNuevoNodo;}

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

PrtNodoListaArt adicionarFinal(ListaArt &listaA, LArt datoArticulo){

return adicionarDespues(listaA,datoArticulo,ultimo(listaA));}

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

void colocarDato(ListaArt &listaA,LArt &datoArticulo,PrtNodoListaArt ptrNodoListaArt){

if ( (! listaVacia(listaA)) && (ptrNodoListaArt != finA()))

ptrNodoListaArt->datoLArt = datoArticulo;}

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

void obtenerDato(ListaArt &listaA, LArt &datoArticulo,PrtNodoListaArt ptrNodoListaArt){

if ((! listaVacia(listaA)) && (ptrNodoListaArt != finA()))

datoArticulo = ptrNodoListaArt->datoLArt;}

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

void eliminarNodo(ListaArt &listaA, PrtNodoListaArt ptrNodoListaArt){

PrtNodoListaArt ptrPrevio;

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

if ((! listaVacia(listaA)) && (ptrNodoListaArt != finA())) {

if (ptrNodoListaArt == primero(listaA))

listaA.primeroListaArt = siguiente(listaA,primero(listaA));

else {

ptrPrevio = anterior( listaA , ptrNodoListaArt );

ptrPrevio->sgteArt= ptrNodoListaArt->sgteArt; }

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

destructorArt(ptrNodoListaArt->datoLArt);

delete ptrNodoListaArt; }}

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

void eliminarNodoPrimero(ListaArt &listaA){

if (! listaVacia(listaA))

eliminarNodo(listaA,primero(listaA));}

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

void eliminarNodoUltimo(ListaArt &listaA){

if (! listaVacia(listaA))

eliminarNodo(listaA,ultimo(listaA));}

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

void eliminarLista(ListaArt &listaA){

while (! listaVacia(listaA))

eliminarNodo(listaA,primero(listaA));}

* **ListaCalle.cpp**

#include "ListaCalle.h"

void crearListaCalle(ListaCalle &listaC){

listaC.primeroCalle=finC();}

bool listaVacia(ListaCalle &listaC){

return (primero(listaC) == finC());}

PtrNodoCalle finC(){

return NULL;}

PtrNodoCalle primero(ListaCalle &listaC){

return listaC.primeroCalle;}

PtrNodoCalle siguiente(ListaCalle &listaC,PtrNodoCalle ptrNodoCalle){

/\* verifica si la lista está vacia o si ptrNodoCalle es el último \*/

if ((! listaVacia(listaC)) && (ptrNodoCalle != finC()))

return ptrNodoCalle->sgteC;

else return finC();}

// Definicion de Primitivas

PtrNodoCalle anterior(ListaCalle &listaC, PtrNodoCalle ptrNodoC){

PtrNodoCalle ptrPrevio = finC();

PtrNodoCalle ptrCursor = primero(listaC);

while (( ptrCursor != finC()) && (ptrCursor != ptrNodoC)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaC,ptrCursor); }

return ptrPrevio;}

PtrNodoCalle ultimo(ListaCalle &listaC){

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

return anterior(listaC,finC());}

PtrNodoCalle crearNodoLista(ListPiso listPiso) {

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

PtrNodoCalle ptrAux = new NodoCalle;

ptrAux->listPiso = listPiso;

ptrAux->sgteC = finC();

return ptrAux;}

PtrNodoCalle adicionarPrincipio(ListaCalle &listaC, ListPiso listPiso){

/\* crea el nodo \*/

PtrNodoCalle ptrNuevoNodo = crearNodoLista(listPiso);

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

ptrNuevoNodo->sgteC = listaC.primeroCalle;

listaC.primeroCalle = ptrNuevoNodo;

return ptrNuevoNodo;}

PtrNodoCalle adicionarDespues(ListaCalle &listaC, ListPiso listPiso, PtrNodoCalle ptrNodoC){

PtrNodoCalle ptrNuevoNodo = finC();

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

if (listaVacia(listaC))

ptrNuevoNodo = adicionarPrincipio(listaC,listPiso);

else {

if (ptrNodoC != finC()) {

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

ptrNuevoNodo = crearNodoLista(listPiso);

ptrNuevoNodo->sgteC = ptrNodoC->sgteC;

ptrNodoC->sgteC = ptrNuevoNodo; } }

return ptrNuevoNodo;}

PtrNodoCalle adicionarFinal(ListaCalle &listaC,ListPiso listPiso){

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

return adicionarDespues(listaC,listPiso,ultimo(listaC)); }

PtrNodoCalle adicionarAntes(ListaCalle &listaC, ListPiso listPiso, PtrNodoCalle ptrNodoC){

PtrNodoCalle ptrNuevoNodo = finC();

if (! listaVacia(listaC)) {

if (ptrNodoC != primero(listaC))

ptrNuevoNodo = adicionarDespues(listaC,listPiso,anterior(listaC,ptrNodoC));

else

ptrNuevoNodo = adicionarPrincipio(listaC,listPiso); }

return ptrNuevoNodo; }

void colocarDato(ListaCalle &listaC,ListPiso listPiso, PtrNodoCalle ptrNodoC){

if ( (! listaVacia(listaC)) && (ptrNodoC != finC()))

ptrNodoC->listPiso = listPiso; }

void obtenerDato(ListaCalle &listaC, ListPiso listPiso, PtrNodoCalle ptrNodoC){

if ((! listaVacia(listaC)) && (ptrNodoC != finC()))

listPiso = ptrNodoC->listPiso; }

void eliminarNodo(ListaCalle &listaC, PtrNodoCalle ptrNodoC){

PtrNodoCalle ptrPrevio;

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

if ((! listaVacia(listaC)) && (ptrNodoC != finC())) {

if (ptrNodoC == primero(listaC))

listaC.primeroCalle = siguiente(listaC,primero(listaC));

else {

ptrPrevio = anterior( listaC , ptrNodoC );

ptrPrevio->sgteC = ptrNodoC->sgteC; }

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

delete ptrNodoC; }}

void eliminarNodoPrimero(ListaCalle &listaC){

if (! listaVacia(listaC))

eliminarNodo(listaC,primero(listaC));}

void eliminarNodoUltimo(ListaCalle &listaC){

if (! listaVacia(listaC))

eliminarNodo(listaC,ultimo(listaC));}

void eliminarLista(ListaCalle &listaC){

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

while (! listaVacia(listaC))

eliminarNodo(listaC,primero(listaC));}

// Definición de Operaciones Adicionales

int longitud(ListaCalle &listaC){

PtrNodoCalle ptrCursor = primero(listaC);

int longitud = 0;

while ( ptrCursor != finC() ) {

longitud++;

ptrCursor = siguiente( listaC, ptrCursor); }

return longitud;}

* **ListaFaltantes.cpp**

#include "ListaFaltantes.h"

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

void crearListaFaltante(ListaFaltante &listaF){

listaF.primerFaltante=finF();}

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

bool listaVacia(ListaFaltante &listaF){

return (primero(listaF)==finF());}

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

PtrNodoFaltante finF(){

return NULL;}

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

PtrNodoFaltante primero(ListaFaltante &listaF){

return listaF.primerFaltante;}

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

PtrNodoFaltante siguiente(ListaFaltante &listaF,PtrNodoFaltante ptrNodoFaltante){

if ((! listaVacia(listaF)) && (ptrNodoFaltante != finF()))

return ptrNodoFaltante->sgteF;

else

return finF(); }

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

PtrNodoFaltante anterior(ListaFaltante &listaF,PtrNodoFaltante ptrNodoFaltante){

PtrNodoFaltante ptrPrevio = finF();

PtrNodoFaltante ptrCursor = primero(listaF);

while (( ptrCursor != finF()) && (ptrCursor != ptrNodoFaltante)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaF,ptrCursor); }

return ptrPrevio;}

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

PtrNodoFaltante ultimo(ListaFaltante &listaF){

return anterior(listaF,finF());}

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

PtrNodoFaltante crearNodoLista(Faltante datoFaltante){

PtrNodoFaltante ptrAux = new NodoFaltante;

ptrAux->datoFaltante = datoFaltante;

ptrAux->sgteF = finF();

return ptrAux;}

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

PtrNodoFaltante adicionarPrincipio(ListaFaltante &listaF,Faltante datoFaltante){

/\* crea el nodo \*/

PtrNodoFaltante ptrNuevoNodo = crearNodoLista(datoFaltante);

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

ptrNuevoNodo->sgteF = listaF.primerFaltante;

listaF.primerFaltante = ptrNuevoNodo;

return ptrNuevoNodo;}

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

PtrNodoFaltante adicionarDespues(ListaFaltante &listaF,Faltante datoFaltante, PtrNodoFaltante ptrNodoFaltante){

PtrNodoFaltante ptrNuevoNodo = finF();

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

if (listaVacia(listaF))

ptrNuevoNodo = adicionarPrincipio(listaF,datoFaltante);

else {

if (ptrNodoFaltante != finF()) {

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

ptrNuevoNodo = crearNodoLista(datoFaltante);

ptrNuevoNodo->sgteF = ptrNodoFaltante->sgteF;

ptrNodoFaltante->sgteF = ptrNuevoNodo; } }

return ptrNuevoNodo;}

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

PtrNodoFaltante adicionarFinal(ListaFaltante &listaF,Faltante datoFaltante){

return adicionarDespues(listaF,datoFaltante,ultimo(listaF));}

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

void colocarDato(ListaFaltante &listaF,Faltante &datoFaltante, PtrNodoFaltante ptrNodoFaltante){

if ( (! listaVacia(listaF)) && (ptrNodoFaltante != finF()))

ptrNodoFaltante->datoFaltante = datoFaltante;}

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

void obtenerDato(ListaFaltante &listaF, Faltante &datoFaltante, PtrNodoFaltante ptrNodoFaltante){

if ((! listaVacia(listaF)) && (ptrNodoFaltante != finF()))

datoFaltante = ptrNodoFaltante->datoFaltante;}

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

void eliminarNodo(ListaFaltante &listaF, PtrNodoFaltante ptrNodoFaltante){

PtrNodoFaltante ptrPrevio;

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

if ((! listaVacia(listaF)) && (ptrNodoFaltante != finF())) {

if (ptrNodoFaltante == primero(listaF))

listaF.primerFaltante= siguiente(listaF,primero(listaF));

else {

ptrPrevio = anterior( listaF , ptrNodoFaltante );

ptrPrevio->sgteF= ptrNodoFaltante->sgteF; }

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

destructorFaltantes(ptrNodoFaltante->datoFaltante);

delete ptrNodoFaltante; }}

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

void eliminarNodoPrimero(ListaFaltante &listaF){

if (! listaVacia(listaF))

eliminarNodo(listaF,primero(listaF));}

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

void eliminarNodoUltimo(ListaFaltante &listaF){

if (! listaVacia(listaF))

eliminarNodo(listaF,ultimo(listaF));}

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

void eliminarLista(ListaFaltante &listaF){

while (! listaVacia(listaF))

eliminarNodo(listaF,primero(listaF));}

* **ListaPesoAcumulado.cpp**

#include "ListaPesoAcumulado.h"

//ListaPesoAcum &listaPA,PesoA datoPesoA, PtrNodoListaPA ptrNodoListaPA

#include "ListaArticulo.h"

void crearListaPesoAcum(ListaPesoAcum &listaPA){

listaPA.primerPesoAcum=finPA();}

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

bool listaVacia(ListaPesoAcum &listaPA){

return (primero(listaPA)==finPA());}

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

PtrNodoListaPA finPA(){

return NULL;}

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

PtrNodoListaPA primero(ListaPesoAcum &listaPA){

return listaPA.primerPesoAcum;}

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

PtrNodoListaPA siguiente(ListaPesoAcum &listaPA,PtrNodoListaPA ptrNodoListaPA){

if ((! listaVacia(listaPA)) && (ptrNodoListaPA != finPA()))

return ptrNodoListaPA->sgtePA;

else return finPA(); }

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

PtrNodoListaPA anterior(ListaPesoAcum &listaPA,PtrNodoListaPA ptrNodoListaPA){

PtrNodoListaPA ptrPrevio = finPA();

PtrNodoListaPA ptrCursor = primero(listaPA);

while (( ptrCursor != finPA()) && (ptrCursor != ptrNodoListaPA)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaPA,ptrCursor); }

return ptrPrevio;}

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

PtrNodoListaPA ultimo(ListaPesoAcum &listaPA){

return anterior(listaPA,finPA());}

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

PtrNodoListaPA crearNodoLista(PesoA datoPesoA){

PtrNodoListaPA ptrAux = new NodoListaPAcum;

ptrAux->datoPesoA = datoPesoA;

ptrAux->sgtePA = finPA();

return ptrAux;}

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

PtrNodoListaPA adicionarPrincipio(ListaPesoAcum &listaPA,PesoA datoPesoA){

/\* crea el nodo \*/

PtrNodoListaPA ptrNuevoNodo = crearNodoLista(datoPesoA);

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

ptrNuevoNodo->sgtePA = listaPA.primerPesoAcum;

listaPA.primerPesoAcum = ptrNuevoNodo;

return ptrNuevoNodo;}

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

PtrNodoListaPA adicionarDespues(ListaPesoAcum &listaPA,PesoA datoPesoA, PtrNodoListaPA ptrNodoListaPA){

PtrNodoListaPA ptrNuevoNodo = finPA();

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

if (listaVacia(listaPA))

ptrNuevoNodo = adicionarPrincipio(listaPA,datoPesoA);

else {

if (ptrNodoListaPA != finPA()) {

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

ptrNuevoNodo = crearNodoLista(datoPesoA);

ptrNuevoNodo->sgtePA = ptrNodoListaPA->sgtePA;

ptrNodoListaPA->sgtePA = ptrNuevoNodo; } }

return ptrNuevoNodo;}

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

PtrNodoListaPA adicionarFinal(ListaPesoAcum &listaPA, PesoA datoPesoA){

return adicionarDespues(listaPA,datoPesoA,ultimo(listaPA));}

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

void colocarDato(ListaPesoAcum &listaPA,PesoA &datoPesoA,PtrNodoListaPA ptrNodoListaPA){

if ( (! listaVacia(listaPA)) && (ptrNodoListaPA != finPA()))

ptrNodoListaPA->datoPesoA = datoPesoA;}

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

void obtenerDato(ListaPesoAcum &listaPA, PesoA &datoPesoA,PtrNodoListaPA ptrNodoListaPA){

if ((! listaVacia(listaPA)) && (ptrNodoListaPA != finPA()))

datoPesoA = ptrNodoListaPA->datoPesoA;}

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

void eliminarNodo(ListaPesoAcum &listaPA, PtrNodoListaPA ptrNodoListaPA){

PtrNodoListaPA ptrPrevio;

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

if ((! listaVacia(listaPA)) && (ptrNodoListaPA != finPA())) {

if (ptrNodoListaPA == primero(listaPA))

listaPA.primerPesoAcum = siguiente(listaPA,primero(listaPA));

else {

ptrPrevio = anterior( listaPA , ptrNodoListaPA );

ptrPrevio->sgtePA= ptrNodoListaPA->sgtePA; }

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

destruirPesoAcum(ptrNodoListaPA->datoPesoA);

delete ptrNodoListaPA; }}

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

void eliminarNodoPrimero(ListaPesoAcum &listaPA){

if (! listaVacia(listaPA))

eliminarNodo(listaPA,primero(listaPA));}

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

void eliminarNodoUltimo(ListaPesoAcum &listaPA){

if (! listaVacia(listaPA))

eliminarNodo(listaPA,ultimo(listaPA));}

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

void eliminarLista(ListaPesoAcum &listaPA){

while (! listaVacia(listaPA))

eliminarNodo(listaPA,primero(listaPA));}

* **ListaPiso.cpp**

#include "ListaPiso.h"

void crearListaPiso(ListaPiso &listaP){

listaP.primeroPiso=finP();}

bool listaVacia(ListaPiso &listaP){

return (primero(listaP) == finP());}

PtrNodoPiso finP(){

return NULL;}

PtrNodoPiso primero(ListaPiso &listaP){

return listaP.primeroPiso;}

PtrNodoPiso siguiente(ListaPiso &listaP, PtrNodoPiso ptrNodoP){

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

if ((! listaVacia(listaP)) && (ptrNodoP != finP()))

return ptrNodoP->sgteP;

else return finP();}

PtrNodoPiso anterior(ListaPiso &listaP, PtrNodoPiso ptrNodoP){

PtrNodoPiso ptrPrevio = finP();

PtrNodoPiso ptrCursor = primero(listaP);

while (( ptrCursor != finP()) && (ptrCursor != ptrNodoP)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaP,ptrCursor); }

return ptrPrevio;}

PtrNodoPiso ultimo(ListaPiso &listaP){

return anterior(listaP,finP());}

PtrNodoPiso crearNodoLista(PilUb pilUb){

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

PtrNodoPiso ptrAux = new NodoPiso;

ptrAux->pilUb = pilUb;

ptrAux->sgteP = finP();

return ptrAux;}

PtrNodoPiso adicionarPrincipio(ListaPiso &listaP, PilUb pilUb){

/\* crea el nodo \*/

PtrNodoPiso ptrNuevoNodo = crearNodoLista(pilUb);

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

ptrNuevoNodo->sgteP = listaP.primeroPiso;

listaP.primeroPiso = ptrNuevoNodo;

return ptrNuevoNodo;}

PtrNodoPiso adicionarDespues(ListaPiso &listaP, PilUb pilUb, PtrNodoPiso ptrNodoP){

PtrNodoPiso ptrNuevoNodo = finP();

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

if (listaVacia(listaP))

ptrNuevoNodo = adicionarPrincipio(listaP,pilUb);

else { if (ptrNodoP != finP()) {

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

ptrNuevoNodo = crearNodoLista(pilUb);

ptrNuevoNodo->sgteP = ptrNodoP->sgteP;

ptrNodoP->sgteP = ptrNuevoNodo; } }

return ptrNuevoNodo;}

PtrNodoPiso adicionarFinal(ListaPiso &listaP, PilUb pilUb){

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

return adicionarDespues(listaP,pilUb,ultimo(listaP));}

PtrNodoPiso adicionarAntes(ListaPiso &listaP, PilUb pilUb, PtrNodoPiso ptrNodoP){

PtrNodoPiso ptrNuevoNodo = finP();

if (! listaVacia(listaP)) {

if (ptrNodoP != primero(listaP))

ptrNuevoNodo = adicionarDespues(listaP,pilUb,anterior(listaP,ptrNodoP));

else

ptrNuevoNodo = adicionarPrincipio(listaP,pilUb); }

return ptrNuevoNodo;}

void colocarDato(ListaPiso &listaP, PilUb &pilUb, PtrNodoPiso ptrNodoP){

if ( (! listaVacia(listaP)) && (ptrNodoP != finP()))

ptrNodoP->pilUb = pilUb;}

void obtenerDato(ListaPiso &listaP, PilUb &pilUb, PtrNodoPiso ptrNodoP){

if ((! listaVacia(listaP)) && (ptrNodoP != finP()))

pilUb = ptrNodoP->pilUb;}

void eliminarNodo(ListaPiso &listaP, PtrNodoPiso ptrNodoP){

PtrNodoPiso ptrPrevio;

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

if ((! listaVacia(listaP)) && (ptrNodoP != finP())) {

if (ptrNodoP == primero(listaP))

listaP.primeroPiso = siguiente(listaP,primero(listaP));

else {

ptrPrevio = anterior( listaP , ptrNodoP );

ptrPrevio->sgteP = ptrNodoP->sgteP; }

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

delete ptrNodoP; }}

void eliminarNodoPrimero(ListaPiso &listaP){

if (! listaVacia(listaP))

eliminarNodo(listaP,primero(listaP));}

void eliminarNodoUltimo(ListaPiso &listaP){

if (! listaVacia(listaP))

eliminarNodo(listaP,ultimo(listaP));}

void eliminarLista(ListaPiso &listaP){

while (! listaVacia(listaP))

eliminarNodo(listaP,primero(listaP));}

* **ListaVenta.cpp**

#include "ListaVentas.h"

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

void crearListaVenta(ListaVentas &listaV){

listaV.primeroListaVentas=finV();}

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

bool listaVacia(ListaVentas &listaV){

return (primero(listaV)==finV());}

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

PtrNodoListaVentas finV(){

return NULL;}

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

PtrNodoListaVentas primero(ListaVentas &listaV){

return listaV.primeroListaVentas;}

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

PtrNodoListaVentas siguiente(ListaVentas &listaV,PtrNodoListaVentas ptrNodoListaVent){

if ((! listaVacia(listaV)) && (ptrNodoListaVent != finV()))

return ptrNodoListaVent->sgteV;

else return finV(); }

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

PtrNodoListaVentas anterior(ListaVentas &listaV,PtrNodoListaVentas ptrNodoListaVent){

PtrNodoListaVentas ptrPrevio = finV();

PtrNodoListaVentas ptrCursor = primero(listaV);

while (( ptrCursor != finV()) && (ptrCursor != ptrNodoListaVent)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaV,ptrCursor); }

return ptrPrevio;}

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

PtrNodoListaVentas ultimo(ListaVentas &listaV){

return anterior(listaV,finV());}

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

PtrNodoListaVentas crearNodoLista(Venta datoVenta){

PtrNodoListaVentas ptrAux = new NodoListaVentas;

ptrAux->datoVenta = datoVenta;

ptrAux->sgteV = finV();

return ptrAux;}

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

PtrNodoListaVentas adicionarPrincipio(ListaVentas &listaV,Venta datoVenta){

/\* crea el nodo \*/

PtrNodoListaVentas ptrNuevoNodo = crearNodoLista(datoVenta);

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

ptrNuevoNodo->sgteV = listaV.primeroListaVentas;

listaV.primeroListaVentas = ptrNuevoNodo;

return ptrNuevoNodo;}

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

PtrNodoListaVentas adicionarDespues(ListaVentas &listaV,Venta datoVenta, PtrNodoListaVentas ptrNodoListaVent){

PtrNodoListaVentas ptrNuevoNodo = finV();

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

if (listaVacia(listaV))

ptrNuevoNodo = adicionarPrincipio(listaV,datoVenta);

else {

if (ptrNodoListaVent != finV()) {

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

ptrNuevoNodo = crearNodoLista(datoVenta);

ptrNuevoNodo->sgteV = ptrNodoListaVent->sgteV;

ptrNodoListaVent->sgteV = ptrNuevoNodo; } }

return ptrNuevoNodo;}

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

PtrNodoListaVentas adicionarFinal(ListaVentas &listaV,Venta datoVenta){

return adicionarDespues(listaV,datoVenta,ultimo(listaV));}

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

void colocarDato(ListaVentas &listaV,Venta &datoVenta,PtrNodoListaVentas ptrNodoListaVent){

if ( (! listaVacia(listaV)) && (ptrNodoListaVent != finV()))

ptrNodoListaVent->datoVenta= datoVenta;}

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

void obtenerDato(ListaVentas &listaV, Venta &datoVenta,PtrNodoListaVentas ptrNodoListaVent){

if ((! listaVacia(listaV)) && (ptrNodoListaVent != finV()))

datoVenta = ptrNodoListaVent->datoVenta;}

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

void eliminarNodo(ListaVentas &listaV, PtrNodoListaVentas ptrNodoListaVent){

PtrNodoListaVentas ptrPrevio;

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

if ((! listaVacia(listaV)) && (ptrNodoListaVent != finV())) {

if (ptrNodoListaVent == primero(listaV))

listaV.primeroListaVentas = siguiente(listaV,primero(listaV));

else {

ptrPrevio = anterior( listaV , ptrNodoListaVent );

ptrPrevio->sgteV= ptrNodoListaVent->sgteV; }

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

destruirVentas(ptrNodoListaVent->datoVenta);

delete ptrNodoListaVent; }}

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

void eliminarNodoPrimero(ListaVentas &listaV){

if (! listaVacia(listaV))

eliminarNodo(listaV,primero(listaV));}

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

void eliminarNodoUltimo(ListaVentas &listaV){

if (! listaVacia(listaV))

eliminarNodo(listaV,ultimo(listaV));}

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

void eliminarLista(ListaVentas &listaV){

while (! listaVacia(listaV))

eliminarNodo(listaV,primero(listaV));}

* **PesoAcumulado.cpp**

#include "PesoAcumulado.h"

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

void crearPesoAcum(PesoAcum &pesoAcum){

pesoAcum.codArt=0;

pesoAcum.cant=0;}

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

void crearPesoAcum(PesoAcum &pesoAcum,int codart,float cant){

pesoAcum.codArt=codart;

pesoAcum.cant=cant;}

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

void destruirPesoAcum(PesoAcum &pesoAcum){}

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

void setCodArt(PesoAcum &pesoAcum,int codart){

pesoAcum.codArt=codart;}

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

void setCant(PesoAcum &pesoAcum,float cant){

pesoAcum.cant=cant;}

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

int getCodArt(PesoAcum &pesoAcum){

return pesoAcum.codArt;}

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

float getCant(PesoAcum &pesoAcum){

return pesoAcum.cant;}

* **PilaArtUb.cpp**

#include "PilaArtUb.h"

void crearPila(PilaArtUb &pilaArtUb){

pilaArtUb.top=finU();}

void destruir(PilaArtUb &pilaArtUb){

while(!pilaVacia(pilaArtUb)){

pop(pilaArtUb); }}

bool pilaVacia(PilaArtUb &pilaArtUb){

return pilaArtUb.top==finU();}

void push(PilaArtUb &pilaArtUb,ArtUb datoArtUb){

PtrNodoPilaArtUb aux=new NodoPilaArtUb;

aux->datoArtUb=datoArtUb;

aux->sgteU=pilaArtUb.top;

pilaArtUb.top=aux;}

ArtUb pop(PilaArtUb &pilaArtUb){

PtrNodoPilaArtUb aux=pilaArtUb.top->sgteU;//podria ser cualquier ipo de de dato

ArtUb datoArtUb=pilaArtUb.top->datoArtUb;

delete pilaArtUb.top;

pilaArtUb.top=aux;

return datoArtUb;}

PtrNodoPilaArtUb finU(){

return NULL;}

PtrNodoPilaArtUb top(PilaArtUb &pilaArtUb){

return pilaArtUb.top;}

* **PilaCamion.cpp**

#include "PilaCamion.h"

void crearPila(PilaCamion &pilaCamion){

pilaCamion.top=finCa();}

void destruir(PilaCamion &pilaCamion){

while(!pilaVacia(pilaCamion)){

pop(pilaCamion); }}

bool pilaVacia(PilaCamion &pilaCamion){

return pilaCamion.top==finCa();}

void push(PilaCamion &pilaCamion,Cam datoCamion){

PtrNodoPilaCamion aux=new NodoPilaCamion;

aux->datoCamion=datoCamion;

aux->sgteCa=pilaCamion.top;

pilaCamion.top=aux;}

Cam pop(PilaCamion &pilaCamion){

PtrNodoPilaCamion aux=pilaCamion.top->sgteCa;//podria ser cualquier ipo de de dato

Cam datoCamion=pilaCamion.top->datoCamion;

delete pilaCamion.top;

pilaCamion.top=aux;

return datoCamion;}

PtrNodoPilaCamion finCa(){

return NULL;}

PtrNodoPilaCamion top(PilaCamion &pilaCamion){

return pilaCamion.top;}

* **Reposicion.cpp**

#include "Reposicion.h"

#include <iostream>

#include <string.h>

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

void crearReposicion(Reposicion &reposicion){

reposicion.cod\_articulo=0;

reposicion.cantidad=0;

strcpy(reposicion.hora,"");}

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

void crearReposicion(Reposicion &reposicion,int CodArt,int Cant,char hora[]){

reposicion.cod\_articulo=CodArt;

reposicion.cantidad=Cant;

strcpy(reposicion.hora,hora);}

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

void destruirReposicion(Reposicion &reposicion){}

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

int getCodArt(Reposicion &reposicion){

return reposicion.cod\_articulo;}

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

int getCant(Reposicion &reposicion){

return reposicion.cantidad;}

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

char\* getHora(Reposicion &reposicion){

char\* ch;

ch=reposicion.hora;

return ch;}

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

void setCodArt(Reposicion &reposicion,int codart){

reposicion.cod\_articulo=codart;}

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

void setCant(Reposicion &reposicion,int cant){

reposicion.cantidad=cant;}

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

void setHora(Reposicion &reposicion,char hora[]){

strcpy(reposicion.hora,hora);}

* **Solicitud.cpp**

#include "Solicitud.h"

#include <iostream>

#include <string.h>

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

void crearSolicitud(Solicitud &solicitud){

solicitud.cod\_suc=0;

solicitud.cod\_articulo=0;

solicitud.cantidad=0;

strcpy(solicitud.hora,"");}

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

void crearSolicitud(Solicitud &solicitud,int CodSuc,int CodArt,int cant,char hora[]){

solicitud.cod\_suc=CodSuc;

solicitud.cod\_articulo=CodArt;

solicitud.cantidad=cant;

strcpy(solicitud.hora,hora);}

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

void destruirSolicitud(Solicitud &solicitud){}

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

int getCodSuc(Solicitud &solicitud){

return solicitud.cod\_suc;}

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

int getCodArt(Solicitud &solicitud){

return solicitud.cod\_articulo;}

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

int getCant(Solicitud &solicitud){

return solicitud.cantidad;}

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

char\* getHora(Solicitud &solicitud){

char \* ch;

ch=solicitud.hora;

return ch;}

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

void setCodSuc(Solicitud &solicitud,int CodSuc){

solicitud.cod\_suc=CodSuc;}

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

void setCodArt(Solicitud &solicitud,int CodArt){

solicitud.cod\_articulo=CodArt;}

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

void setCantidad(Solicitud &solicitud,int Cant){

solicitud.cantidad=Cant;}

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

void setHora(Solicitud &solicitud,char h[]){

strcpy(solicitud.hora,h);}

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

* **Ventas.cpp**

#include "Ventas.h"

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

void crearVentas(Ventas &ventas){

ventas.codSucursal=0;

ventas.montoTotal=0;}

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

void crearVentas(Ventas &ventas,int codSuc,float montTot){

ventas.codSucursal=codSuc;

ventas.montoTotal=montTot;}

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

void destruirVentas(Ventas &ventas){}

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

void setCodSuc(Ventas &ventas,int codsuc){

ventas.codSucursal=codsuc;}

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

void setMonTot(Ventas &ventas,float monTot){

ventas.montoTotal=monTot;}

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

int getCodSuc(Ventas &ventas){

return ventas.codSucursal;}

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

float getMonTot(Ventas &ventas){

return ventas.montoTotal;}

ARCHIVOS DE PRUEBA

* Configuracion:

|  |  |
| --- | --- |
| 2x2 | 4000 kg camión |

* Solicitudes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Solicitudes | | | | | |
| op | codSucursal | codArticulo | cantidad | hora |  |
| **11** | 1 | 1 | 1500 | 09:00:00 | 1;1;1500;09:00:00 |
| **12** | 1 | 1 | 2000 | 09:10:15 | 1;1;2000;09:10:15 |
| **13** | 2 | 1 | 1000 | 09:20:21 | 2;1;1000;09:20:21 |
| **15** | 3 | 3 | 500 | 09:30:11 | 3;3;500;09:30:11 |
| **16** | 2 | 1 | 2500 | 09:40:40 | 2;1;2500;09:40:40 |
| **17** | 3 | 1 | 1500 | 09:45:45 | 3;1;1500;09:45:45 |
| **18** | 4 | 1 | 2000 | 09:50:51 | 4;1;2000;09:50:51 |
| **19** | 1 | 4 | 1000 | 10:00:00 | 1;4;1000;10:00:00 |
| **21** | 2 | 11 | 300 | 10:10:01 | 2;11;300;10:10:01 |
| **22** | 3 | 1 | 3500 | 10:11:12 | 3;1;3500;10:11:12 |
| **25** | 4 | 6 | 1500 | 10:25:12 | 4;6;1500;10:25:12 |
| **26** | 5 | 3 | 500 | 10:30:16 | 5;3;500;10:30:16 |
| **27** | 6 | 2 | 2500 | 10:40:46 | 6;2;2500;10:40:46 |
| **28** | 5 | 6 | 4000 | 10:45:45 | 5;6;4000;10:45:45 |
| **29** | 4 | 9 | 400 | 10:50:14 | 4;9;400;10:50:14 |
| **30** | 3 | 6 | 1100 | 10:55:42 | 3;6;1100;10:55:42 |
| **32** | 2 | 6 | 3000 | 11:10:11 | 2;6;3000;11:10:11 |

* Ingresos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ingresos | | | | |
| op | codArticulo | cantidad | hora |  |
| **1** | 1 | 10000 | 08:01:37 | 1;10000;08:01:37 |
| **2** | 2 | 10000 | 08:05:00 | 2;10000;08:05:00 |
| **3** | 3 | 10000 | 08:06:10 | 3;10000;08:06:10 |
| **4** | 4 | 10000 | 08:12:12 | 4;10000;08:12:12 |
| **5** | 5 | 10000 | 08:13:33 | 5;10000;08:13:33 |
| **6** | 6 | 5000 | 08:14:50 | 6;5000;08:14:50 |
| **7** | 7 | 5000 | 08:20:12 | 7;5000;08:20:12 |
| **8** | 8 | 5000 | 08:21:22 | 8;5000;08:21:22 |
| **9** | 9 | 5000 | 08:24:31 | 9;5000;08:24:31 |
| **10** | 10 | 5000 | 08:30:00 | 10;5000;08:30:00 |
| **14** | 2 | 1000 | 09:25:30 | 2;1000;09:25:30 |
| **20** | 1 | 3000 | 10:05:03 | 1;3000;10:05:03 |
| **23** | 11 | 3500 | 10:15:23 | 11;3500;10:15:23 |
| **24** | 1 | 5000 | 10:20:21 | 1;5000;10:20:21 |
| **31** | 6 | 2500 | 11:01:14 | 6;2500;11:01:14 |
| **33** | 8 | 1000 | 12:00:00 | 8;1000;12:00:00 |

* Camión

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Camión |  |  |  |  |  |
| Operación | nro camión | kilos | kilos acum | |  |
| **11** | 1 | 1500 | 1500 |  |  |
| **12** | 1 | 2000 | 3500 |  |  |
| **13** | 2 | 1000 | 1000 |  |  |
| **15** | 2 | 500 | 1500 |  |  |
| **16** | 2 | 2500 | 4000 |  |  |
| **17** | 3 | 1500 | 1500 |  |  |
| **18** | 3 | 1500 | 3000 |  |  |
| **19** | 3 | 1000 | 4000 |  |  |
| 21 | 4 | 0 | 0 | faltante 300, cod 11 | |
| **22** | 4 | 3000 | 3000 | faltante 500, cod 1 | |
| **25** | 5 | 1500 | 1500 |  |  |
| **26** | 5 | 500 | 2000 |  |  |
| **27** | 6 | 2500 | 2500 |  |  |
| **28** | 7 | 3500 | 3500 | faltante 500 cod 6 | |
| **29** | 7 | 400 | 3900 |  |  |
| 30 |  |  |  | faltante cod6, 1100 | |
| 32 | 8 | 2500 | 2500 | faltante cod6, 500 | |

* Saldos Detallados

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | C | P | U | codArt | cant |
| 33 | 1 | 1 | 1 | 11 | 3500 |
|  | 1 | 1 | 2 | 2 | 11000 |
|  | 1 | 2 | 1 | 3 | 9000 |
|  | 1 | 2 | 2 | 4 | 9000 |
|  | 2 | 1 | 1 | 5 | 10000 |
|  | 2 | 1 | 2 | -1 | 0 |
|  | 2 | 2 | 1 | 7 | 5000 |
|  | 2 | 2 | 2 | 8 | 6000 |
|  | 3 | 1 | 1 | 9 | 4600 |
|  | 3 | 1 | 2 | 10 | 5000 |
|  | 3 | 2 | 1 | 1 | 5000 |
|  | 3 | 2 | 2 | -1 |  |

* Faltantes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | codSucursal | codArticulo | cantidadOriginal | cantidadCumplida | % cumplimiento |
| 18 | 4 | 1 | 2000 | 1500 | 0,75 |
| 21 | 2 | 11 | 300 | 0 | 0 |
| 22 | 3 | 1 | 3500 | 3000 | 0,857143 |
| 28 | 5 | 6 | 4000 | 3500 | 0,875 |
| 30 | 3 | 6 | 1100 | 0 | 0 |
| 32 | 2 | 6 | 3000 | 2500 | 0,833333 |