**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 en La lista maestra de articulas ------------

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; //Queda mas lindo mostrar que comienza este proceso

cout<<"Cod. Articulo: "<< getCodArt(reposicion) <<endl;

cout<<"Cantidad: "<< getCant(reposicion) << endl;

int codArti=0;

/\*\*\*\*\*\*\*\*\*\*ACA HAY Q CARGAR SI LA PILA ESTA VACIA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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);

PtrNodoPisoDep pisoDep=primero(piso1);

setPisoDep(artDep1,pisoDep);

setCodArtDep(artDep1, getCodArt(reposicion));

adicionarFinal(listaArtDep,artDep1);

cout<<"Primer articulo guardado exitosamente"<<endl; }

else{ cout<<"\tEl 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;

int longUb=0; //POSTERIORMENTE SON UTILIZADOS DE CONTADORES DE CALLES Y DE PISOS

PrtNodoListaArt ptrArt= primero(listaArt);

codArti=reposicion.cod\_articulo;/\*\*cambiamos esto,e deimo un valor\*\*/

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)){

//NO TENGO LA MAS MINIMA IDEA DE PORQUE NO LO TOMA, LA LOGICA QUE USE CREO ESTA BIEN PERO NO SE PORQUE NO LO TOMA

//longUb = getConfigU(top(ptrP->pilUb)->datoArtUb);

longUb=ptrP->pilUb.top->datoArtUb.u; }

if (longUb < getConfigU(configuracion)&& !aRegist) {

//MIENTRAS LA LONGITUD ES MENOR A LO DETERMINADO EN CONFIGURACION

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 CON EXITO"<<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 n°: "<<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;}