/\* TDA ListaEquipo

\* Implementación Simplemente Enlazada

\* Archivo : Lista.cpp

\* Versión : 1.1

\*/

#include "ListaEquipos.h"

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

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

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

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

/\* Funciones Adicionales \*/

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

/\*

pre : ninguna.

post: compara ambos dato1 y dato2, devuelve

mayor si dato1 es mayor que dato2,

igual si dato1 es igual a dato2,

menor si dato1 es menor que dato2.

dato1 : dato a comparar.

dato2 : dato a comparar.

return resultado de comparar dato1 respecto de dato2.

\*/

ResultadoComparacion compararDato(equipo dato1, equipo dato2) {

if (getID(dato1) > getID(dato2)) {

return MAYOR;

}

else if (getID(dato1) < getID(dato2)) {

return MENOR;

}

else {

return IGUAL;

}

}

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

/\* Implementación de Primitivas \*/

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

void crearListaEquipo(ListaEquipo &lista) {

lista.primero = finEquipo();

}

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

bool listaVaciaEquipo(ListaEquipo &lista) {

return (primeroEquipo(lista) == finEquipo());

}

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

PtrNodoListaEquipo finEquipo() {

return NULL;

}

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

PtrNodoListaEquipo primeroEquipo(ListaEquipo &lista) {

return lista.primero;

}

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

PtrNodoListaEquipo siguienteEquipo(ListaEquipo &lista, PtrNodoListaEquipo ptrNodo) {

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

if ((! listaVaciaEquipo(lista)) && (ptrNodo != finEquipo()))

return ptrNodo->sgte;

else

return finEquipo();

}

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

PtrNodoListaEquipo anteriorEquipo(ListaEquipo &lista, PtrNodoListaEquipo ptrNodo) {

PtrNodoListaEquipo ptrPrevio = finEquipo();

PtrNodoListaEquipo ptrCursor = primeroEquipo(lista);

while (( ptrCursor != finEquipo()) && (ptrCursor != ptrNodo)) {

ptrPrevio = ptrCursor;

ptrCursor = siguienteEquipo(lista,ptrCursor);

}

return ptrPrevio;

}

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

PtrNodoListaEquipo ultimoEquipo(ListaEquipo &lista) {

/\* el último nodo de la lista es el anterior al finEquipo() \*/

return anteriorEquipo(lista,finEquipo());

}

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

PtrNodoListaEquipo crearNodoLista(equipo dato) {

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

PtrNodoListaEquipo ptrAux = new NodoLista;

ptrAux->dato = dato;

ptrAux->sgte = finEquipo();

return ptrAux;

}

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

PtrNodoListaEquipo adicionarPrincipio(ListaEquipo &lista, equipo dato) {

/\* crea el nodo \*/

PtrNodoListaEquipo ptrNuevoNodo = crearNodoLista(dato);

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

ptrNuevoNodo->sgte = lista.primero;

lista.primero = ptrNuevoNodo;

return ptrNuevoNodo;

}

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

PtrNodoListaEquipo adicionarDespues(ListaEquipo &lista, equipo dato, PtrNodoListaEquipo ptrNodo) {

PtrNodoListaEquipo ptrNuevoNodo = finEquipo();

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

if (listaVaciaEquipo(lista))

ptrNuevoNodo = adicionarPrincipio(lista,dato);

else {

if (ptrNodo != finEquipo()) {

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

ptrNuevoNodo = crearNodoLista(dato);

ptrNuevoNodo->sgte = ptrNodo->sgte;

ptrNodo->sgte = ptrNuevoNodo;

}

}

return ptrNuevoNodo;

}

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

PtrNodoListaEquipo adicionarFinal(ListaEquipo &lista, equipo dato) {

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

return adicionarDespues(lista,dato,ultimoEquipo(lista));

}

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

PtrNodoListaEquipo adicionarAntes(ListaEquipo &lista, equipo dato, PtrNodoListaEquipo ptrNodo) {

PtrNodoListaEquipo ptrNuevoNodo = finEquipo();

if (! listaVaciaEquipo(lista)) {

if (ptrNodo != primeroEquipo(lista))

ptrNuevoNodo = adicionarDespues(lista,dato,anteriorEquipo(lista,ptrNodo));

else

ptrNuevoNodo = adicionarPrincipio(lista,dato);

}

return ptrNuevoNodo;

}

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

void colocarDato(ListaEquipo &lista, equipo &dato, PtrNodoListaEquipo ptrNodo) {

if ( (! listaVaciaEquipo(lista)) && (ptrNodo != finEquipo()))

ptrNodo->dato = dato;

}

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

void obtenerDato(ListaEquipo &lista, equipo &dato, PtrNodoListaEquipo ptrNodo) {

if ((! listaVaciaEquipo(lista)) && (ptrNodo != finEquipo()))

dato = ptrNodo->dato;

}

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

void eliminarNodoEquipo(ListaEquipo &lista, PtrNodoListaEquipo ptrNodo) {

PtrNodoListaEquipo ptrPrevio;

/\* verifica que la lista no esté vacia y que nodo no sea finEquipo\*/

if ((! listaVaciaEquipo(lista)) && (ptrNodo != finEquipo())) {

if (ptrNodo == primeroEquipo(lista))

lista.primero = siguienteEquipo(lista,primeroEquipo(lista));

else {

ptrPrevio = anteriorEquipo( lista , ptrNodo );

ptrPrevio->sgte = ptrNodo->sgte;

}

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

delete ptrNodo;

}

}

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

void eliminarNodoPrimeroEquipo(ListaEquipo &lista) {

if (! listaVaciaEquipo(lista))

eliminarNodoEquipo(lista,primeroEquipo(lista));

}

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

void eliminarNodoUltimoEquipo(ListaEquipo &lista) {

if (! listaVaciaEquipo(lista))

eliminarNodoEquipo(lista,ultimoEquipo(lista));

}

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

void eliminarListaEquipo(ListaEquipo &lista) {

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

while (! listaVaciaEquipo(lista))

eliminarNodoEquipo(lista,primeroEquipo(lista));

}

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

PtrNodoListaEquipo localizarDato(ListaEquipo &lista, equipo dato) {

bool encontrado = false;

equipo datoCursor;

PtrNodoListaEquipo ptrCursor = primeroEquipo(lista);

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

encontrar el nodo buscado \*/

while ((ptrCursor != finEquipo()) && (! encontrado)) {

/\* obtiene el dato del nodo y lo compara \*/

obtenerDato(lista,datoCursor,ptrCursor);

if (compararDato(datoCursor,dato) == IGUAL)

encontrado = true;

else

ptrCursor = siguienteEquipo(lista,ptrCursor);

}

/\* si no lo encontró devuelve finEquipo \*/

if (! encontrado)

ptrCursor = finEquipo();

return ptrCursor;

}

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

void eliminarDato(ListaEquipo &lista, equipo dato) {

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

PtrNodoListaEquipo ptrNodo = localizarDato(lista,dato);

if (ptrNodo != finEquipo())

eliminarNodoEquipo(lista,ptrNodo);

}

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

PtrNodoListaEquipo insertarDato(ListaEquipo &lista, equipo dato) {

PtrNodoListaEquipo ptrPrevio = primeroEquipo(lista);

PtrNodoListaEquipo ptrCursor = primeroEquipo(lista);

PtrNodoListaEquipo ptrNuevoNodo;

equipo datoCursor;

bool ubicado = false;

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

while ((ptrCursor != finEquipo()) && (! ubicado)) {

obtenerDato(lista,datoCursor,ptrCursor);

if (compararDato(datoCursor,dato) == MAYOR)

ubicado = true;

else {

ptrPrevio = ptrCursor;

ptrCursor = siguienteEquipo(lista,ptrCursor);

}

}

if (ptrCursor == primeroEquipo(lista))

ptrNuevoNodo = adicionarPrincipio(lista,dato);

else

ptrNuevoNodo = adicionarDespues(lista,dato,ptrPrevio);

return ptrNuevoNodo;

}

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

void reordenarEquipo(ListaEquipo &lista) {

ListaEquipo temp = lista;

PtrNodoListaEquipo ptrCursor = primeroEquipo(temp);

crearListaEquipo(lista);

while ( ptrCursor != finEquipo() ) {

equipo dato;

obtenerDato( temp, dato, ptrCursor);

insertarDato( lista, dato );

eliminarNodoEquipo( temp, ptrCursor );

ptrCursor = primeroEquipo(temp);

}

eliminarListaEquipo( temp );

}

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

int longitudEquipo(ListaEquipo &lista){

PtrNodoListaEquipo ptrCursor = primeroEquipo(lista);

int longitud = 0;

while ( ptrCursor != finEquipo() ) {

longitud++;

ptrCursor = siguienteEquipo( lista, ptrCursor);

}

return longitud;

}

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