/\* TDA ListaJugador

\* Implementación Simplemente Enlazada

\* Archivo : ListaJugador.cpp

\* Versión : 1.1

\*/

#include "ListaJugador.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.

\*/

ResultadoComparacionJugador compararDatoJugador(Jugador dato1, Jugador dato2) {

if (getId(dato1) > getId(dato2)) {

return MAYORJugador;

}

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

return MENORJugador;

}

else {

return IGUALJugador;

}

}

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

/\* Implementación de Primitivas \*/

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

void crearListaJugador(ListaJugador &lista) {

lista.primeroListaJugador = finListaJugador();

}

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

bool listaVaciaJugador(ListaJugador &lista) {

return (primeroListaJugador(lista) == finListaJugador());

}

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

PtrNodoListaJugador finListaJugador() {

return NULL;

}

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

PtrNodoListaJugador primeroListaJugador(ListaJugador &lista) {

return lista.primeroListaJugador;

}

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

PtrNodoListaJugador siguienteListaJugador(ListaJugador &lista, PtrNodoListaJugador ptrNodo) {

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

if ((! listaVaciaJugador(lista)) && (ptrNodo != finListaJugador()))

return ptrNodo->sgte;

else

return finListaJugador();

}

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

PtrNodoListaJugador anteriorListaJugador(ListaJugador &lista, PtrNodoListaJugador ptrNodo) {

PtrNodoListaJugador ptrPrevio = finListaJugador();

PtrNodoListaJugador ptrCursor = primeroListaJugador(lista);

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

ptrPrevio = ptrCursor;

ptrCursor = siguienteListaJugador(lista,ptrCursor);

}

return ptrPrevio;

}

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

PtrNodoListaJugador ultimoListaJugador(ListaJugador &lista) {

/\* el último nodo de la lista es el anteriorListaJugador al finListaJugador() \*/

return anteriorListaJugador(lista,finListaJugador());

}

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

PtrNodoListaJugador crearNodoListaJugador(Jugador dato) {

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

PtrNodoListaJugador ptrAux = new NodoListaJugador;

ptrAux->dato = dato;

ptrAux->sgte = finListaJugador();

return ptrAux;

}

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

PtrNodoListaJugador adicionarPrincipioListaJugador(ListaJugador &lista, Jugador dato) {

/\* crea el nodo \*/

PtrNodoListaJugador ptrNuevoNodo = crearNodoListaJugador(dato);

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

ptrNuevoNodo->sgte = lista.primeroListaJugador;

lista.primeroListaJugador = ptrNuevoNodo;

return ptrNuevoNodo;

}

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

PtrNodoListaJugador adicionarDespuesListaJugador(ListaJugador &lista, Jugador dato, PtrNodoListaJugador ptrNodo) {

PtrNodoListaJugador ptrNuevoNodo = finListaJugador();

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

if (listaVaciaJugador(lista))

ptrNuevoNodo = adicionarPrincipioListaJugador(lista,dato);

else {

if (ptrNodo != finListaJugador()) {

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

ptrNuevoNodo = crearNodoListaJugador(dato);

ptrNuevoNodo->sgte = ptrNodo->sgte;

ptrNodo->sgte = ptrNuevoNodo;

}

}

return ptrNuevoNodo;

}

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

PtrNodoListaJugador adicionarFinalListaJugador(ListaJugador &lista, Jugador dato) {

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

return adicionarDespuesListaJugador(lista,dato,ultimoListaJugador(lista));

}

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

PtrNodoListaJugador adicionarAntesListaJugador(ListaJugador &lista, Jugador dato, PtrNodoListaJugador ptrNodo) {

PtrNodoListaJugador ptrNuevoNodo = finListaJugador();

if (! listaVaciaJugador(lista)) {

if (ptrNodo != primeroListaJugador(lista))

ptrNuevoNodo = adicionarDespuesListaJugador(lista,dato,anteriorListaJugador(lista,ptrNodo));

else

ptrNuevoNodo = adicionarPrincipioListaJugador(lista,dato);

}

return ptrNuevoNodo;

}

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

void colocarDatoListaJugador(ListaJugador &lista, Jugador &dato, PtrNodoListaJugador ptrNodo) {

if ( (! listaVaciaJugador(lista)) && (ptrNodo != finListaJugador()))

ptrNodo->dato = dato;

}

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

void obtenerDatoListaJugador(ListaJugador &lista, Jugador &dato, PtrNodoListaJugador ptrNodo) {

if ((! listaVaciaJugador(lista)) && (ptrNodo != finListaJugador()))

dato = ptrNodo->dato;

}

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

void eliminarNodoListaJugador(ListaJugador &lista, PtrNodoListaJugador ptrNodo) {

PtrNodoListaJugador ptrPrevio;

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

if ((! listaVaciaJugador(lista)) && (ptrNodo != finListaJugador())) {

if (ptrNodo == primeroListaJugador(lista))

lista.primeroListaJugador = siguienteListaJugador(lista,primeroListaJugador(lista));

else {

ptrPrevio = anteriorListaJugador( lista , ptrNodo );

ptrPrevio->sgte = ptrNodo->sgte;

}

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

delete ptrNodo;

}

}

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

void eliminarNodoPrimeroListaJugador(ListaJugador &lista) {

if (! listaVaciaJugador(lista))

eliminarNodoListaJugador(lista,primeroListaJugador(lista));

}

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

void eliminarNodoUltimoListaJugador(ListaJugador &lista) {

if (! listaVaciaJugador(lista))

eliminarNodoListaJugador(lista,ultimoListaJugador(lista));

}

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

void eliminarListaJugador(ListaJugador &lista) {

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

while (! listaVaciaJugador(lista))

eliminarNodoListaJugador(lista,primeroListaJugador(lista));

}

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

PtrNodoListaJugador localizarDatoListaJugador(ListaJugador &lista, Jugador dato) {

bool encontrado = false;

Jugador datoCursor;

PtrNodoListaJugador ptrCursor = primeroListaJugador(lista);

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

encontrar el nodo buscado \*/

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

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

obtenerDatoListaJugador(lista,datoCursor,ptrCursor);

if (compararDatoJugador(datoCursor,dato) == IGUALJugador)

encontrado = true;

else

ptrCursor = siguienteListaJugador(lista,ptrCursor);

}

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

if (! encontrado)

ptrCursor = finListaJugador();

return ptrCursor;

}

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

void eliminarDatoListaJugador(ListaJugador &lista, Jugador dato) {

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

PtrNodoListaJugador ptrNodo = localizarDatoListaJugador(lista,dato);

if (ptrNodo != finListaJugador())

eliminarNodoListaJugador(lista,ptrNodo);

}

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

PtrNodoListaJugador insertarDatoListaJugador(ListaJugador &lista, Jugador dato) {

PtrNodoListaJugador ptrPrevio = primeroListaJugador(lista);

PtrNodoListaJugador ptrCursor = primeroListaJugador(lista);

PtrNodoListaJugador ptrNuevoNodo;

Jugador datoCursor;

bool ubicado = false;

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

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

obtenerDatoListaJugador(lista,datoCursor,ptrCursor);

if (compararDatoJugador(datoCursor,dato) == MAYORJugador)

ubicado = true;

else {

ptrPrevio = ptrCursor;

ptrCursor = siguienteListaJugador(lista,ptrCursor);

}

}

if (ptrCursor == primeroListaJugador(lista))

ptrNuevoNodo = adicionarPrincipioListaJugador(lista,dato);

else

ptrNuevoNodo = adicionarDespuesListaJugador(lista,dato,ptrPrevio);

return ptrNuevoNodo;

}

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

void reordenarListaJugador(ListaJugador &lista) {

ListaJugador temp = lista;

PtrNodoListaJugador ptrCursor = primeroListaJugador(temp);

crearListaJugador(lista);

while ( ptrCursor != finListaJugador() ) {

Jugador dato;

obtenerDatoListaJugador( temp, dato, ptrCursor);

insertarDatoListaJugador( lista, dato );

eliminarNodoListaJugador( temp, ptrCursor );

ptrCursor = primeroListaJugador(temp);

}

eliminarListaJugador( temp );

}

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

int longitudListaJugador(ListaJugador &lista){

PtrNodoListaJugador ptrCursor = primeroListaJugador(lista);

int longitudListaJugador = 0;

while ( ptrCursor != finListaJugador() ) {

longitudListaJugador++;

ptrCursor = siguienteListaJugador( lista, ptrCursor);

}

return longitudListaJugador;

}

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