/\* TDA Lista

\* Implementación Simplemente Enlazada

\* Archivo : ListaPuntajes.cpp

\* Versión : 1.1

\*/

#include "ListaPuntajes.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.

puntaje1 : dato a comparar.

puntaje2 : dato a comparar.

return resultado de comparar dato1 respecto de dato2.

\*/

ResultadoComparacionPuntajes compararDatoPuntajes(Puntajes dato1, Puntajes dato2) {

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

return MAYORPUNTAJE;

}

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

return MENORPUNTAJE;

}

else {

return IGUALPUNTAJE;

}

}

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

/\* Implementación de Primitivas \*/

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

void crearListaPuntajes(ListaPuntajes &lista) {

lista.primero = finPuntajes();

}

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

bool listaVaciaPuntajes(ListaPuntajes &lista) {

return (primero(lista) == finPuntajes());

}

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

PtrNodoListaPuntajes finPuntajes() {

return NULL;

}

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

PtrNodoListaPuntajes primero(ListaPuntajes &lista) {

return lista.primero;

}

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

PtrNodoListaPuntajes siguiente(ListaPuntajes &lista, PtrNodoListaPuntajes ptrNodo) {

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

if ((! listaVaciaPuntajes(lista)) && (ptrNodo != finPuntajes()))

return ptrNodo->sgte;

else

return finPuntajes();

}

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

PtrNodoListaPuntajes anterior(ListaPuntajes &lista, PtrNodoListaPuntajes ptrNodo) {

PtrNodoListaPuntajes ptrPrevio = finPuntajes();

PtrNodoListaPuntajes ptrCursor = primero(lista);

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

ptrPrevio = ptrCursor;

ptrCursor = siguiente(lista,ptrCursor);

}

return ptrPrevio;

}

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

PtrNodoListaPuntajes ultimo(ListaPuntajes &lista) {

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

return anterior(lista,finPuntajes());

}

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

PtrNodoListaPuntajes crearNodoLista(Puntajes dato) {

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

PtrNodoListaPuntajes ptrAux = new NodoListaPuntajes();

ptrAux->dato = dato;

ptrAux->sgte = finPuntajes();

return ptrAux;

}

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

PtrNodoListaPuntajes adicionarPrincipio(ListaPuntajes &lista, Puntajes puntajes) {

/\* crea el nodo \*/

PtrNodoListaPuntajes ptrNuevoNodo = crearNodoLista(puntajes);

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

ptrNuevoNodo->sgte = lista.primero;

lista.primero = ptrNuevoNodo;

return ptrNuevoNodo;

}

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

PtrNodoListaPuntajes adicionarDespues(ListaPuntajes &lista, Puntajes puntajes, PtrNodoListaPuntajes ptrNodo) {

PtrNodoListaPuntajes ptrNuevoNodo = finPuntajes();

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

if (listaVaciaPuntajes(lista))

ptrNuevoNodo = adicionarPrincipio(lista,puntajes);

else {

if (ptrNodo != finPuntajes()) {

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

ptrNuevoNodo = crearNodoLista(puntajes);

ptrNuevoNodo->sgte = ptrNodo->sgte;

ptrNodo->sgte = ptrNuevoNodo;

}

}

return ptrNuevoNodo;

}

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

PtrNodoListaPuntajes adicionarFinal(ListaPuntajes &lista, Puntajes puntajes) {

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

return adicionarDespues(lista,puntajes,ultimo(lista));

}

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

PtrNodoListaPuntajes adicionarAntes(ListaPuntajes &lista, Puntajes puntajes, PtrNodoListaPuntajes ptrNodo) {

PtrNodoListaPuntajes ptrNuevoNodo = finPuntajes();

if (! listaVaciaPuntajes(lista)) {

if (ptrNodo != primero(lista))

ptrNuevoNodo = adicionarDespues(lista,puntajes,anterior(lista,ptrNodo));

else

ptrNuevoNodo = adicionarPrincipio(lista,puntajes);

}

return ptrNuevoNodo;

}

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

void colocarDatoPuntajes(ListaPuntajes &lista, Puntajes &puntajes, PtrNodoListaPuntajes ptrNodo) {

if ( (! listaVaciaPuntajes(lista)) && (ptrNodo != finPuntajes()))

ptrNodo->dato = puntajes;

}

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

void obtenerDatoPuntajes(ListaPuntajes &lista, Puntajes &puntajes, PtrNodoListaPuntajes ptrNodo) {

if ((! listaVaciaPuntajes(lista)) && (ptrNodo != finPuntajes()))

puntajes = ptrNodo->dato;

}

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

void eliminarNodoPuntajes(ListaPuntajes &lista, PtrNodoListaPuntajes ptrNodo) {

PtrNodoListaPuntajes ptrPrevio;

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

if ((! listaVaciaPuntajes(lista)) && (ptrNodo != finPuntajes())) {

if (ptrNodo == primero(lista))

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

else {

ptrPrevio = anterior( lista , ptrNodo );

ptrPrevio->sgte = ptrNodo->sgte;

}

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

delete ptrNodo;

}

}

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

void eliminarNodoPrimeroPuntajes(ListaPuntajes &lista) {

if (! listaVaciaPuntajes(lista))

eliminarNodoPuntajes(lista,primero(lista));

}

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

void eliminarNodoUltimoPuntajes(ListaPuntajes &lista) {

if (! listaVaciaPuntajes(lista))

eliminarNodoPuntajes(lista,ultimo(lista));

}

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

void eliminarListaPuntajes(ListaPuntajes &lista) {

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

while (! listaVaciaPuntajes(lista))

eliminarNodoPuntajes(lista,primero(lista));

}

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

PtrNodoListaPuntajes localizarDatoPuntajes(ListaPuntajes &lista, Puntajes puntajes) {

bool encontrado = false;

Puntajes datoCursor;

PtrNodoListaPuntajes ptrCursor = primero(lista);

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

encontrar el nodo buscado \*/

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

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

obtenerDatoPuntajes(lista,datoCursor,ptrCursor);

if (compararDatoPuntajes(datoCursor,puntajes) == IGUALPUNTAJE)

encontrado = true;

else

ptrCursor = siguiente(lista,ptrCursor);

}

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

if (! encontrado)

ptrCursor = finPuntajes();

return ptrCursor;

}

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

void eliminarDatoPuntajes(ListaPuntajes &lista, Puntajes puntajes) {

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

PtrNodoListaPuntajes ptrNodo = localizarDatoPuntajes(lista,puntajes);

if (ptrNodo != finPuntajes())

eliminarNodoPuntajes(lista,ptrNodo);

}

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

PtrNodoListaPuntajes insertarDato(ListaPuntajes &lista, Puntajes puntajes) {

PtrNodoListaPuntajes ptrPrevio = primero(lista);

PtrNodoListaPuntajes ptrCursor = primero(lista);

PtrNodoListaPuntajes ptrNuevoNodo;

Puntajes datoCursor;

bool ubicado = false;

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

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

obtenerDatoPuntajes(lista,datoCursor,ptrCursor);

if (compararDatoPuntajes(datoCursor,puntajes) == MAYORPUNTAJE)

ubicado = true;

else {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(lista,ptrCursor);

}

}

if (ptrCursor == primero(lista))

ptrNuevoNodo = adicionarPrincipio(lista,puntajes);

else

ptrNuevoNodo = adicionarDespues(lista,puntajes,ptrPrevio);

return ptrNuevoNodo;

}

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

void reordenarPuntajes(ListaPuntajes &lista) {

ListaPuntajes temp = lista;

PtrNodoListaPuntajes ptrCursor = primero(temp);

crearListaPuntajes(lista);

while ( ptrCursor != finPuntajes() ) {

Puntajes puntajes;

obtenerDatoPuntajes( temp, puntajes, ptrCursor);

insertarDato( lista, puntajes );

eliminarNodoPuntajes( temp, ptrCursor );

ptrCursor = primero(temp);

}

eliminarListaPuntajes( temp );

}

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

int longitudPuntajes(ListaPuntajes &lista){

PtrNodoListaPuntajes ptrCursor = primero(lista);

int longitud = 0;

while ( ptrCursor != finPuntajes() ) {

longitud++;

ptrCursor = siguiente( lista, ptrCursor);

}

return longitud;

}

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