/\* TDA Lista

\* Implementación Simplemente Enlazada

\* Archivo : Lista.cpp

\* Versión : 1.1

\*/

#include "ListaErrores.h"

#include <fstream>

using namespace std;

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

/\* 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.

\*/

ResultadoComparacionError compararDato(DatoError dato1, DatoError dato2) {

if (getCodigoError(dato1) > getCodigoError(dato2)) {

return MAYORERROR;

}

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

return MENORERROR;

}

else {

return IGUALERROR;

}

}

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

/\* Implementación de Primitivas \*/

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

void crearListaError(ListaError &listaerror) {

listaerror.primero = fin();

}

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

bool listaVaciaError(ListaError &listaerror) {

return (primero(listaerror) == fin());

}

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

PtrNodoListaError fin() {

return NULL;

}

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

PtrNodoListaError primero(ListaError &listaerror) {

return listaerror.primero;

}

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

PtrNodoListaError siguiente(ListaError &listaerror, PtrNodoListaError ptrNodo) {

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

if ((! listaVaciaError(listaerror)) && (ptrNodo != fin()))

return ptrNodo->sgte;

else

return fin();

}

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

PtrNodoListaError anterior(ListaError &listaerror, PtrNodoListaError ptrNodo) {

PtrNodoListaError ptrPrevio = fin();

PtrNodoListaError ptrCursor = primero(listaerror);

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

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaerror,ptrCursor);

}

return ptrPrevio;

}

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

PtrNodoListaError ultimo(ListaError &listaerror) {

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

return anterior(listaerror,fin());

}

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

PtrNodoListaError crearNodoLista(DatoError datoerror) {

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

PtrNodoListaError ptrAux = new NodoListaError;

ptrAux->datoerror = datoerror;

ptrAux->sgte = fin();

return ptrAux;

}

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

PtrNodoListaError adicionarPrincipio(ListaError &listaerror, DatoError datoerror) {

/\* crea el nodo \*/

PtrNodoListaError ptrNuevoNodo = crearNodoLista(datoerror);

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

ptrNuevoNodo->sgte = listaerror.primero;

listaerror.primero = ptrNuevoNodo;

return ptrNuevoNodo;

}

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

PtrNodoListaError adicionarDespues(ListaError &listaerror, DatoError datoerror, PtrNodoListaError ptrNodo) {

PtrNodoListaError ptrNuevoNodo = fin();

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

if (listaVaciaError(listaerror))

ptrNuevoNodo = adicionarPrincipio(listaerror,datoerror);

else {

if (ptrNodo != fin()) {

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

ptrNuevoNodo = crearNodoLista(datoerror);

ptrNuevoNodo->sgte = ptrNodo->sgte;

ptrNodo->sgte = ptrNuevoNodo;

}

}

return ptrNuevoNodo;

}

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

PtrNodoListaError adicionarFinal(ListaError &listaerror, DatoError datoerror) {

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

return adicionarDespues(listaerror,datoerror,ultimo(listaerror));

}

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

PtrNodoListaError adicionarAntes(ListaError &listaerror, DatoError datoerror, PtrNodoListaError ptrNodo) {

PtrNodoListaError ptrNuevoNodo = fin();

if (! listaVaciaError(listaerror)) {

if (ptrNodo != primero(listaerror))

ptrNuevoNodo = adicionarDespues(listaerror,datoerror,anterior(listaerror,ptrNodo));

else

ptrNuevoNodo = adicionarPrincipio(listaerror,datoerror);

}

return ptrNuevoNodo;

}

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

void colocarDatoError(ListaError &listaerror, DatoError &datoerror, PtrNodoListaError ptrNodo) {

if ( (! listaVaciaError(listaerror)) && (ptrNodo != fin()))

ptrNodo->datoerror = datoerror;

}

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

void obtenerDatoError(ListaError &listaerror, DatoError &datoerror, PtrNodoListaError ptrNodo) {

if ((! listaVaciaError(listaerror)) && (ptrNodo != fin()))

datoerror = ptrNodo->datoerror;

}

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

void eliminarNodoError(ListaError &listaerror, PtrNodoListaError ptrNodo) {

PtrNodoListaError ptrPrevio;

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

if ((! listaVaciaError(listaerror)) && (ptrNodo != fin())) {

if (ptrNodo == primero(listaerror))

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

else {

ptrPrevio = anterior( listaerror , ptrNodo );

ptrPrevio->sgte = ptrNodo->sgte;

}

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

delete ptrNodo;

}

}

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

void eliminarNodoPrimeroError(ListaError &listaerror) {

if (! listaVaciaError(listaerror))

eliminarNodoError(listaerror,primero(listaerror));

}

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

void eliminarNodoUltimo(ListaError &listaerror) {

if (! listaVaciaError(listaerror))

eliminarNodoError(listaerror,ultimo(listaerror));

}

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

void eliminarListaError(ListaError &listaerror) {

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

while (! listaVaciaError(listaerror))

eliminarNodoError(listaerror,primero(listaerror));

}

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

PtrNodoListaError localizarDato(ListaError &listaerror, DatoError datoerror) {

bool encontrado = false;

DatoError datoCursor;

PtrNodoListaError ptrCursor = primero(listaerror);

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

encontrar el nodo buscado \*/

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

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

obtenerDatoError(listaerror,datoCursor,ptrCursor);

if (compararDato(datoCursor,datoerror) == IGUALERROR)

encontrado = true;

else

ptrCursor = siguiente(listaerror,ptrCursor);

}

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

if (! encontrado)

ptrCursor = fin();

return ptrCursor;

}

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

void eliminarDatoError(ListaError &listaerror, DatoError datoerror) {

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

PtrNodoListaError ptrNodo = localizarDato(listaerror,datoerror);

if (ptrNodo != fin())

eliminarNodoError(listaerror,ptrNodo);

}

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

PtrNodoListaError insertarDato(ListaError &listaerror, DatoError datoerror) {

PtrNodoListaError ptrPrevio = primero(listaerror);

PtrNodoListaError ptrCursor = primero(listaerror);

PtrNodoListaError ptrNuevoNodo;

DatoError datoCursor;

bool ubicado = false;

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

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

obtenerDatoError(listaerror,datoCursor,ptrCursor);

if (compararDato(datoCursor,datoerror) == MAYORERROR)

ubicado = true;

else {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(listaerror,ptrCursor);

}

}

if (ptrCursor == primero(listaerror))

ptrNuevoNodo = adicionarPrincipio(listaerror,datoerror);

else

ptrNuevoNodo = adicionarDespues(listaerror,datoerror,ptrPrevio);

return ptrNuevoNodo;

}

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

void reordenarError(ListaError &listaerror) {

ListaError temp = listaerror;

PtrNodoListaError ptrCursor = primero(temp);

crearListaError(listaerror);

while ( ptrCursor != fin() ) {

DatoError datoerror;

obtenerDatoError( temp, datoerror, ptrCursor);

insertarDato( listaerror, datoerror );

eliminarNodoError( temp, ptrCursor );

ptrCursor = primero(temp);

}

eliminarListaError( temp );

}

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

int longitudError(ListaError &listaerror){

PtrNodoListaError ptrCursor = primero(listaerror);

int longitud = 0;

while ( ptrCursor != fin() ) {

longitud++;

ptrCursor = siguiente( listaerror, ptrCursor);

}

return longitud;

}

void GuardarDatos(Error &error,ListaError &listaerror)

{

int cont=1;

int cantlineas=0;

fstream fich("errores.txt",ios::app);

PtrNodoListaError cursor;

cursor=primero(listaerror);

cantlineas=cantLineas();

while(cursor!=fin())

{

obtenerDatoError(listaerror,error,cursor);

fich<<cantlineas+1<<":"<<getDescripcion(error)<<"."<<endl<<flush;

cursor=siguiente(listaerror,cursor);

cantlineas=cantlineas+1;

}

fich.close();

}

int cantLineas()

{

ifstream arch("errores.txt", ifstream::in);

int lin=0;

while(arch.good()) if(arch.get()=='\n') lin++;

return lin;

}

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