```
#ifndef LISTA_H
#define LISTA_H

#include "Nodo/Nodo.h"

#define ELEMENTO_NO_ENCONTRADO -1

enum COMPARACION {
   MENOR = -1,
   IGUAL,
   MAYOR
};

struct Lista {
   Nodo* inicio;
   int iTamanio_Lista;
};

//-----------------------------------------------Constructor-----------------------------------------------
/*
   PRE: La lista no debe haber sido creada.
   POST: La lista queda creada y el tamanio de la lista queda seteado en 0.
*/
Lista* crearLista();

//-----------------------------------------------Getter-----------------------------------------------------
/*
   PRE: La lista debe haber sido creada.
   POST: Devuelve el dato contenido en el iTamanio_Lista de la lista.
*/
```

```
int getCantidadDeElementosEnLaLista(Lista*);

//----------------------------------------Funciones de la lista----------------------------------------
/*
   PRE: La lista debe haber sido creada.
   POST: La lista queda vacia y de tamanio 0.
*/
void vaciarLista(Lista*&);

/*
   PRE: La lista debe haber sido creada.
   POST: Indica si la lista tiene o no elementos que la conpongan.
*/
bool listaEstaVacia(Lista*);

/*
   PRE: La lista debe haber sido creada.
   POST: Busca un elemento en la lista y me indica en que posicion se encuetra y -1 en caso de no existir el dato en la lista.
*/
int posicionElementoEnLaLista(Lista*, ELEMENTO);

/*
   PRE: La lista debe haber sido creada.
   POST: Agrego al inicio de la lista 1 elemento.
*/
void insertarElementoAlInicioDeLaLista(Lista*&, ELEMENTO);

/*
   PRE: La lista debe haber sido creada.
   POST: Agrego 1 elemento a la lista en la posicion indicada si es que existe.
```

```
*/
void insertarElementoALaLista(Lista*&, int iPosicion, ELEMENTO);

/*
   PRE: La lista debe haber sido creada.
   POST: Agrego 1 elemento al final de la lista.
*/
void insertarElementoAlFinalDeLaLista(Lista*&, ELEMENTO);

/*
   PRE: La lista debe haber sido creada.
   POST: Almaceno en una variable el ELEMENTO que poseo en el inicio de la lista.
*/
void obtenerElementoInicialDeLaLista(Lista*, ELEMENTO&);

/*
   PRE: La lista debe haber sido creada.
   POST: Almaceno en una variable el ELEMENTO que poseo en una posicion indicada de la lista si es que existe.
*/
void obtenerElementoDeLaLista(Lista* &, int iPosicion, ELEMENTO&);

/*
   PRE: La lista debe haber sido creada.
   POST: Almaceno en una variable el ELEMENTO que poseo al final de la lista.
*/
void obtenerElementoFinalDeLaLista(Lista* &, ELEMENTO&);

/*
   PRE: La lista debe haber sido creada.
```

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo en el inicio de la lista.

***/**

void eliminarElementoInicialDeLaLista(Lista*&, ELEMENTO&);

/*

PRE: La lista debe haber sido creada.

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo en una posicion indicada de la lista si es que existe.

***/**

void eliminarElementoDeLaLista(Lista*&, int iPosicion, ELEMENTO&);

/*

PRE: La lista debe haber sido creada.

POST: Remuevo y almaceno en una variable el ELEMENTO que poseo al final de la lista.

***/**

void eliminarElementoFinalDeLaLista(Lista*&, ELEMENTO&);

/*

PRE: La lista debe haber sido creada y debe existir una funcion que muestre 1 tipo de dato en particular.

POST: Se muestra por consola los datos de todos los elementos que hay en la lista.

***/**

void mostrarElementosDeLaLista(Lista*, void mostrarDatos(ELEMENTO));

/*

PRE: La lista debe haber sido creada.

POST: Invierto los elementos de la lista si es que ambas posiciones existen en la lista.

***/**

void invertirElementos(Lista*, int iPosicion1, int iPosicion2);

**/*

PRE: Debe existir el enum COMPARACION.

POST: Indica si los datos deben ser invertidos o no de posiciones en base al criterio.

*/**

bool ascendente(int);

**/*

PRE: Debe existir el enum COMPARACION.

POST: Indica si los datos deben ser invertidos o no de posiciones en base al criterio.

*/**

bool descendente(int);

**/*

PRE: La lista debe haber sido creada y debe exister la funcion que se encargue de comparar los datos.

POST: Ordeno la lista en forma ascendente o descendente segun se indique en el tercer parametro.

*/**

void reordenarLista(Lista*, int comparar(ELEMENTO elemento1, ELEMENTO elemento2), bool criterio(int));

**/*

PRE: La lista debe haber sido creada, debe existir el dato que deseo buscar y, además, debe existir una funcion que se encargue de comparar si 2 elementos son o no iguales

* preferentemente se lo utiliza para comparar TDA si deseo comparar en base a 1 TD que lo forma (atributo), en caso de buscar elementos en una lista de TD o TDA utilizar mejor la funcion posicionElementoEnLaLista.

POST: Indico la posicion de donde se encuentra el elemento en la lista o ELEMENTO_NO_ENCONTRADO(-1) si es que en la lista no existe.

*/**

int buscarElementoEnLaLista(Lista*, ELEMENTO dato_Buscado, bool comparar(ELEMENTO dato_Buscado, ELEMENTO elemento));

```
//----------------------------------------------Destructores----------------------------------------------
/*
   PRE: La lista debe haber sido creada.
   POST: La lista es eliminada pero sin eliminar los datos que la formaban.
*/
void destruirLista(Lista*);

/*
   PRE: La lista debe haber sido creada y debe existir la funcion que elimine un tipo de dato en particular.
   POST: La lista es eliminada y los datos que la formaban tambien son destruidos.
*/
void destruirListaYDatos(Lista*, void eliminarDatos(ELEMENTO));

#endif // !LISTA_H
```