```
#include <iostream>

#include "lista.h"

//----------------------------------------------Constructor----------------------------------------------

Lista* crearLista() {
    Lista* l = new Lista();

    l->inicio = NULL;
    l->iTamanio_Lista = 0;

    return l;
}

//-------------------------------------------------------------Getter-------------------------------------
-----------------

int getCantidadDeElementosEnLaLista(Lista* l) {
    return l->iTamanio_Lista;
}

//----------------------------------------------Funciones de la lista-----------------------------------

void vaciarLista(Lista*& l) {
    vaciarLista(l->inicio);
    l->iTamanio_Lista = 0;
}

bool listaEstaVacia(Lista* l) {
    return listaVacia(l->inicio);
```

```
}

int posicionElementoEnLaLista(Lista* l, ELEMENTO dato) {
   return buscarElemento(l->inicio, dato);
}

/*
   PRE: La lista debe haber sido creada.
   POST: Indica si la posicion indicada existe o no en la lista.
 */
bool existePosicion(Lista* l, int iPosicion) {
   return ((iPosicion >= 0) && (iPosicion < (l->iTamanio_Lista))) ? true : false;
}

void insertarElementoAlInicioDeLaLista(Lista*& l, ELEMENTO dato) {
   insertarAlInicio(l->inicio, dato);
   l->iTamanio_Lista++;
}

void insertarElementoALaLista(Lista*& l, int iPosicion, ELEMENTO dato) {
   if (existePosicion(l, iPosicion)) {
      insertarElementoEnPosicion(l->inicio, iPosicion, dato);
      l->iTamanio_Lista++;
   } else
      std::cout << "\nNo se puede cargar el dato en la posicion solicitada\n";
}

void insertarElementoAlFinalDeLaLista(Lista*& l, ELEMENTO dato) {
   insertarElementoEnPosicion(l->inicio, l->iTamanio_Lista, dato);
   l->iTamanio_Lista++;
```

```
}

void obtenerElementoInicialDeLaLista(Lista* l, ELEMENTO &dato) {
   obtenerElemento(l->inicio, 0, dato);
}

void obtenerElementoDeLaLista(Lista* &l, int iPosicion, ELEMENTO &dato) {
   if (existePosicion(l, iPosicion) && !listaEstaVacia(l)) {
      obtenerElemento(l->inicio, iPosicion, dato);
   } else {
      std::cout << "\nNo hay datos...\n";
      dato = NULL;
   }
}

void obtenerElementoFinalDeLaLista(Lista* &l, ELEMENTO &dato) {
   if (existePosicion(l, (l->iTamanio_Lista) - 1) && !listaEstaVacia(l)) {
      obtenerElemento(l->inicio, (l->iTamanio_Lista) - 1, dato);
   } else {
      std::cout << "\nNo hay datos...\n";
      dato = NULL;
   }
}

void eliminarElementoInicialDeLaLista(Lista*& l, ELEMENTO &dato) {
   if (!listaEstaVacia(l)) {
      quitarElementoDelInicio(l->inicio, dato);
      l->iTamanio_Lista--;
   } else {
      std::cout << "\nNo hay datos...\n";
```

```
        dato = NULL;
    }
}

void eliminarElementoDeLaLista(Lista*& l, int iPosicion, ELEMENTO &dato) {
    if (existePosicion(l, iPosicion) && !listaEstaVacia(l)) {
        quitarElementoDePosicion(l->inicio, iPosicion, dato);
        l->iTamanio_Lista--;
    } else {
        std::cout << "\nNo hay datos...\n";
        dato = NULL;
    }
}

void eliminarElementoFinalDeLaLista(Lista*& l, ELEMENTO &dato) {
    if (existePosicion(l, (l->iTamanio_Lista) - 1) && !listaEstaVacia(l)) {
        quitarElementoDePosicion(l->inicio, (l->iTamanio_Lista) - 1, dato);
        l->iTamanio_Lista--;
    } else {
        std::cout << "\nNo hay datos...\n";
        dato = NULL;
    }
}

void mostrarElementosDeLaLista(Lista* l, void mostrarDatos(ELEMENTO)) {
    ELEMENTO temp;

    for (int i = 0; i < getCantidadDeElementosEnLaLista(l); i++) {
        obtenerElementoDeLaLista(l, i, temp);
        mostrarDatos(temp);
```

```
    }
}

void invertirElementos(Lista* lista, int iPosicion1, int iPosicion2) {
    if (iPosicion1 != iPosicion2 && existePosicion(lista, iPosicion1) &&
existePosicion(lista, iPosicion2)) {
        int iAux;

        if (iPosicion1 > iPosicion2) {
            iAux = iPosicion1;
            iPosicion1 = iPosicion2;
            iPosicion2 = iAux;
        }

        ELEMENTO elemento1, elemento2;

        obtenerElementoDeLaLista(lista, iPosicion1, elemento1);
        obtenerElementoDeLaLista(lista, iPosicion2, elemento2);

        insertarElementoALaLista(lista, iPosicion2, elemento1);
        insertarElementoALaLista(lista, iPosicion1, elemento2);

        eliminarElementoDeLaLista(lista, iPosicion1 + 1, elemento1);
        eliminarElementoDeLaLista(lista, iPosicion2 + 1, elemento2);
    }
}

bool ascendente(int iComparacion) {
    return (iComparacion == MAYOR) ? true : false;
}
```

```
bool descendente(int iComparacion) {
   return (iComparacion == MENOR) ? true : false;
}

void reordenarLista(Lista* lista, int comparar(ELEMENTO elemento1, ELEMENTO
elemento2), bool criterio(int)) {
   int iTamanio = getCantidadDeElementosEnLaLista(lista);
   ELEMENTO elemento1, elemento2;

   for (int i = 0; i < iTamanio; i++) {

      for (int j = (i + 1); j < iTamanio; j++) {

         obtenerElementoDeLaLista(lista, i, elemento1);
         obtenerElementoDeLaLista(lista, j, elemento2);

         if (criterio(comparar(elemento1, elemento2)))
            invertirElementos(lista, i, j);

      }
   }
}

int buscarElementoEnLaLista(Lista* lista, ELEMENTO dato_Buscado, bool
comparar(ELEMENTO dato_Buscado, ELEMENTO elemento)) {
   int iContador = 0, iPosicion = ELEMENTO_NO_ENCONTRADO;
   ELEMENTO aux;

   while ((iContador < lista->iTamanio_Lista) && (iPosicion ==
ELEMENTO_NO_ENCONTRADO)) {
      obtenerElementoDeLaLista(lista, iContador, aux);
```

```
        if (comparar(dato_Buscado, aux))
            iPosicion = iContador;

        iContador++;
    }

    return iPosicion;
}

//-----------------------------------------------Destructores-----------------------------------------------

void destruirLista(Lista* l){
    vaciarLista(l);
    delete l;
}

void destruirListaYDatos(Lista* l, void eliminarDatos(ELEMENTO)) {
    ELEMENTO temp;

    while (getCantidadDeElementosEnLaLista(l) != 0) {
        eliminarElementoInicialDeLaLista(l, temp);
        eliminarDatos(temp);
    }

    delete l;
}
```