```c
#ifndef LISTA_H
#define LISTA_H

#include <stdbool.h>
#include <stddef.h>


/*
*****************************************************
**********
 *                DEFINICION DE LOS TIPOS DE DATOS
 *
 *****************************************************
 **********/

/* La cola está planteada como una cola de punteros
genéricos. */

typedef struct lista lista_t; //Esto antes estaba
definido por separado, si no funciona revisar

typedef struct lista_iter lista_iter_t;
/*
*****************************************************
**********
 *                    PRIMITIVAS DE LA LISTA
 *
 *****************************************************
 **********/

 // Crea una lista.
 // Post: devuelve una nueva lista vacía.
 lista_t *lista_crear(void);

 // Devuelve verdadero si la lista no tiene elementos,
 false en caso contrario.
 // Pre: la lista fue creada.
 bool lista_esta_vacia(const lista_t *lista);

 // Agrega un nuevo elemento al comienzo de la lista.
 Devuelve falso en caso de
```

```c
// error.
// Pre: la lista fue creada.
// Post: se agregó un nuevo elemento al comienzo de la lista.
bool lista_insertar_primero(lista_t *lista, void *dato);

// Agrega un nuevo elemento al final de la lista. Devuelve falso en caso de
// error.
// Pre: la lista fue creada.
// Post: se agregó un nuevo elemento al final de la lista.
bool lista_insertar_ultimo(lista_t *lista, void *dato);

// Devuelve el primer elemento de la lista y elimina el nodo. Si la lista
// está vacía, devuelve NULL.
// Pre: la lista fue creada.
// Post: se devolvió el valor del primer elemento de la lista y se eliminó el
// primer nodo de la lista
void* lista_borrar_primero(lista_t *lista);

// Obtiene el valor del primer elemento de la lista. Si la lista tiene
// elementos, se devuelve el valor del primero, si está vacía devuelve NULL.
// Pre: la lista fue creada.
// Post: se devolvió el primer elemento de la lista, cuando no está vacía.
void* lista_ver_primero(const lista_t *lista);

// Obtiene el valor del último elemento de la lista. Si la lista tiene
// elementos, se devuelve el valor del último, si está vacía devuelve NULL.
// Pre: la lista fue creada.
// Post: se devolvió el último elemento de la lista, cuando no está vacía.
void* lista_ver_ultimo(const lista_t* lista);
```

```c
void* lista_ver_ultimo(const lista_t* lista);

// Devuelve el largo de la lista
// Pre: la lista fue creada.
// Post: se devolvió el largo de la lista
size_t lista_largo(const lista_t *lista);

// Destruye la lista. Si se recibe la función destruir_dato por parámetro,
// para cada uno de los elementos de la lista llama a destruir_dato.
// Pre: la lista fue creada. destruir_dato es una función capaz de destruir
// los datos de la lista, o NULL en caso de que no se la utilice.
// Post: se eliminaron todos los elementos de la lista y la lista.
void lista_destruir(lista_t *lista, void destruir_dato(void *));

/*
*****************************************************************
 *                 PRIMITIVAS DEL ITERADOR EXTERNO
 *
 *****************************************************************/
 // Crea un iterador de lista externo y lo devuelve por puntero.
 // Pre: la lista ya fue creada.
 // Post: devuelve un iterador externo apuntando al primer elemento de la lista.
lista_iter_t *lista_iter_crear(lista_t *lista);

// Avanza de nodo en la lista, devuelve false en caso de llegar al final y que
// no existan mas nodos en la lista.
// Pre: El iterador ya fue creado.
// Post: Avanza una posicion de nodo y devuelve true en caso de ser exitoso.
bool lista_iter_avanzar(lista_iter_t *iter);

```

```c
// Devuelve el dato en el que este el iterador
// Pre: El iterador ya fue creado y apunta a un nodo de la lista.
// Post: Devuele el dato en el que esta posicionado el iterador.
void *lista_iter_ver_actual(const lista_iter_t *iter);

// Devuelve true en caso de que el iterador se encuentre al final
// Pre: El iterador ya fue creado y apunta a un nodo de la lista.
// Post: Devuelve true si se encuentra al final de la lista.
bool lista_iter_al_final(const lista_iter_t *iter);

// Destruye el iterador
// Pre: El iterador ya fue creado.
// Post: Destruye el iterador
void lista_iter_destruir(lista_iter_t *iter);

// Inserta en la posicion que este apuntando el iterador.
// Pre: El iterador ya fue creado.
// Post: Inserta un nuevo nodo en la posicion que este apuntando el iterador
// con el dato pasado por parametro. En caso de ser en la primero o ultima po
// sicion se actualizan los valores de primero y ultimo. Se actualiza largo de
// la lista.
bool lista_iter_insertar(lista_iter_t *iter, void *dato);

// Borra el nodo en la posicion que este apuntando el iterador y devuelve el
// dato contenido.
// Pre: El iterador ya fue creado.
// Post: Borro el nodo de la lista a la cual apuntaba el iterador y devuelvo el
// valor contenido. En caso de ser en la primero o ultima posicion se
```

```c
// actualizan los valores de primero y ultimo. Se actualiza largo de la lista.
void *lista_iter_borrar(lista_iter_t *iter);

/*
 *                  PRIMITIVAS DEL ITERADOR INTERNO
 *
 *************/
// Itera sobre la lista y aplica la funcion pasada por parametro
// Pre: La lista fue creada, la funcion funciona correctamente y es distinto
// de NULL.
// Post: Se ejecuta la funcion pasada por parametro para cada uno de los
// nodos de la lista.
void lista_iterar(lista_t *lista, bool visitar(void *dato, void *extra), void *extra);

/*
 *                      PRUEBAS UNITARIAS
 *
 ***********/

// Realiza pruebas sobre la implementación del alumno.
//
// Para la implementación de las pruebas se debe emplear la función
// print_test(), como se ha visto en TPs anteriores.
void pruebas_lista_alumno(void);

#endif // LISTA_H
```