```c
// Alumno: Daniel Bizari
// Padrón: 100445
// Correctora: Camila Dvorkin

#include "lista.h"
#include <stdlib.h>
#include <stdio.h>

// Definición del struct lista y nodo.

typedef struct nodo nodo_t;

struct nodo{
    void* dato;
    nodo_t* siguiente;
};

struct lista{
    nodo_t* primero;
    nodo_t* ultimo;
    size_t largo;
};

struct lista_iter{
    lista_t* lista;
    nodo_t* ant;
    nodo_t* act;
};

typedef void (*destructor_t)(void*);

/*
*********************************************************
**********
 *                    PRIMITIVAS DE NODO
 *
 *********************************************************
 **********/

nodo_t* nodo_crear(void* valor){
    nodo_t* aux = malloc(sizeof(nodo_t));
```

```
38
39      if(!aux) return NULL;
40
41      aux->dato = valor;
42      aux->siguiente = NULL;
43      return aux;
44  }
45
46  void nodo_destruir(nodo_t* nodo){
47      free(nodo);
48  }
49  /*
    ************************************************************
    **********
50  *                       PRIMITIVAS DE LA LISTA
51  *
    ************************************************************
    **********/
52  lista_t* lista_crear(void){
53      lista_t* aux = malloc(sizeof(lista_t));
54
55      if(!aux) return NULL;
56
57      aux->primero = NULL;
58      aux->ultimo = NULL;
59      aux->largo = 0;
60      return aux;
61  }
62
63  void lista_destruir(lista_t *lista, destructor_t
    destructor){
64      nodo_t* aux,*aux2;
65
66      if(lista_esta_vacia(lista) == true){
67          free(lista);
68          return;
69      }
70      for(aux = lista->primero; aux != NULL; aux = aux2){
71          aux2 = aux->siguiente;
72          if(destructor != NULL)
73              destructor(aux->dato);
74
```

```
74
75          nodo_destruir(aux);
76          lista->largo--;
77      }
78      free(lista);
79  }
80
81  bool lista_esta_vacia(const lista_t *lista){
82      return (lista->primero == NULL);
83  }
84
85  bool lista_insertar_primero(lista_t *lista, void
    *dato){
86      nodo_t * aux_nuevo,* aux_prim = lista->primero;
87      bool is_empty;
88
89      if((aux_nuevo = nodo_crear(dato)) == NULL) return
        false;
90      is_empty = lista_esta_vacia(lista) == true ? true
        : false;
91      lista->primero = aux_nuevo;
92      if(is_empty)
93          lista->ultimo = aux_nuevo;
94      else
95          lista->primero->siguiente = aux_prim;
96
97      lista->largo++;
98      return true;
99  }
100
101 bool lista_insertar_ultimo(lista_t *lista, void *dato){
102     nodo_t * aux_nuevo;
103
104     if((aux_nuevo = nodo_crear(dato)) == NULL) return
        false;
105     if(lista_esta_vacia(lista) == true)
106         lista->primero = aux_nuevo;
107     else
108         lista->ultimo->siguiente = aux_nuevo;
109
110     lista->ultimo = aux_nuevo;
111     lista->largo++;
```

```
111     lista->largo--;
112     return true;
113 }
114 
115 void* lista_ver_primero(const lista_t *lista){
116     return lista_esta_vacia(lista) == true ? NULL :
    •     lista->primero->dato;
117 }
118 
119 void* lista_ver_ultimo(const lista_t* lista){
120     return lista_esta_vacia(lista) == true ? NULL :
    •     lista->ultimo->dato;
121 }
122 
123 size_t lista_largo(const lista_t *lista){
124     return lista->largo;
125 }
126 
127 void* lista_borrar_primero(lista_t *lista){
128     if(lista_esta_vacia(lista) == true) return NULL;
129     void*  dato = lista_ver_primero(lista); //Apunto
    •     al dato para luego devolverlo
130     nodo_t* aux = lista->primero; //Apunto al prmer
    •     nodo
131 
132     lista->primero = lista->primero->siguiente; //
    •     Reacomodar lista
133     nodo_destruir(aux); //Destruir nodo
134     lista->largo--;
135     return dato;
136 }
137 
138 /*
    • ********************************************************
    • **********
139  *                 PRIMITIVAS DE ITERADOR EXTERNO
140  *
    •  ******************************************************
    •  ***********/
141 
142 lista_iter_t *lista_iter_crear(lista_t *lista){
143     lista iter t* aux = malloc(sizeof(lista iter t));
```

```
144
145     if(aux == NULL) return NULL;
146     aux->lista = lista;
147     aux->ant = NULL;
148     aux->act = lista->primero;
149     return aux;
150 }
151
152 bool lista_iter_avanzar(lista_iter_t *iter){
153     if(iter->act == NULL) return false; //Ya llego al
    •   final de la lista;
154     iter->ant = iter->act;
155     iter->act = iter->act->siguiente;
156     return true;
157 }
158
159 void *lista_iter_ver_actual(const lista_iter_t *iter){
160     return lista_iter_al_final(iter) ? NULL : iter-
    •   >act->dato;
161 }
162
163 bool lista_iter_al_final(const lista_iter_t *iter){
164     return (iter->act == NULL);
165 }
166
167 void lista_iter_destruir(lista_iter_t *iter){
168     free(iter);
169 }
170
171 bool lista_iter_insertar(lista_iter_t *iter, void
    •   *dato){
172     nodo_t* aux_nuevo;
173
174     if((aux_nuevo = nodo_crear(dato)) == NULL) return
    •   false;
175
176     if(iter->ant == NULL){    //insertar al principio
177         aux_nuevo->siguiente = iter->act;
178         iter->lista->primero = aux_nuevo;
179     }else{ // resto de los casos
180         iter->ant->siguiente = aux_nuevo;
```

```
181         iter->ant->siguiente->siguiente = iter->act;
182     }
183     iter->act = aux_nuevo;
184     if(iter->act->siguiente == NULL) //Si es el ultimo
    •   nodo actualizar lista
185         iter->lista->ultimo = aux_nuevo;
186 
187     iter->lista->largo++;
188     return true;
189 }
190 
191 void *lista_iter_borrar(lista_iter_t *iter){
192     void* dato = lista_iter_ver_actual(iter);
193     nodo_t* aux;
194 
195     if(iter->act == NULL) return NULL;
196     if(iter->ant == NULL)    //eliminar primer nodo
197         iter->lista->primero = iter->act->siguiente;
198     else
199         iter->ant->siguiente = iter->act->siguiente;
200 
201     aux = iter->act;
202     iter->act = iter->act->siguiente;
203     nodo_destruir(aux);
204     if(iter->act == NULL) //Si es el ultimo nodo
    •   actualizar lista
205         iter->lista->ultimo = iter->ant;
206 
207     iter->lista->largo--;
208     return dato;
209 }
210 /*
    •   **********************************************************
    •   **********
211  *                  PRIMITIVAS DE ITERADOR INTERNO
212  *
    •   **********************************************************
    •   **********/
213 void lista_iterar(lista_t *lista, bool visitar(void
    •   *dato, void *extra), void *extra){
214     nodo_t* aux = lista->primero;
```

```
215
216     while(aux != NULL){
217         if(visitar(aux->dato,extra) == false) break;
218         aux = aux->siguiente;
219     }
220 }
221
```