```c
#include "lista.h"
#include "pila.h"
#include "testing.h"
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>

#define MAX 10000
/*
*****************************************************
***********
 *                    PRUEBAS UNITARIAS ALUMNO
 *
 *****************************************************
 ***********/
void destruir_dato(void * dato){
    free(dato);
}
void destruir_pila(void * pila){
    pila_destruir((pila_t*)pila);
}
void prueba_lista_vacia(){
    lista_t* lista = lista_crear();

    print_test("Creacion de lista", lista != NULL);
    print_test("Desenlistar lista vacia",
    lista_borrar_primero(lista) == NULL);
    print_test("Ver primer elemento de lista vacia",
    lista_ver_primero(lista) == NULL);
    print_test("Ver ultimo elemento de lista vacia",
    lista_ver_ultimo(lista) == NULL);
    print_test("Lista esta vacia",
    lista_esta_vacia(lista) == true);
    print_test("Largo de lista vacia",
    !lista_largo(lista));
    lista_destruir(lista,NULL);
    print_test("Destruir lista vacia",true);
    return;
}

void prueba_un_elemento(){
```

```c
    lista_t* lista = lista_crear();
    int num=10;
    char letra='a';
    long num_long;
    char palabra[10]="Test";

    //Pruebas con int
    print_test("Insertar int al principio",
    lista_insertar_primero(lista,&num) == true);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Ver primero", lista_ver_primero(lista)
    == &num);
    print_test("Ver ultimo", lista_ver_ultimo(lista)
    == &num);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Eliminar int de la lista",
    lista_borrar_primero(lista) == &num);
    print_test("Insertar int a lo ultimo",
    lista_insertar_ultimo(lista,&num) == true);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Ver primero", lista_ver_primero(lista)
    == &num);
    print_test("Ver ultimo", lista_ver_ultimo(lista)
    == &num);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Eliminar int de la lista",
    lista_borrar_primero(lista) == &num);
    //Pruebas con char
    print_test("Insertar char al principio",
    lista_insertar_primero(lista,&letra) == true);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Ver primero", lista_ver_primero(lista)
    == &letra);
    print_test("Ver ultimo", lista_ver_ultimo(lista)
    == &letra);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Eliminar char de la lista",
    lista_borrar_primero(lista) == &letra);
    print_test("Insertar char a lo ultimo",
```

```
59      print_test("Insertar char a lo ultimo",
 •      lista_insertar_ultimo(lista,&letra) == true);
60      print_test("Ver largo", lista_largo(lista) == 1);
61      print_test("Ver primero", lista_ver_primero(lista)
 •      == &letra);
62      print_test("Ver ultimo", lista_ver_ultimo(lista)
 •      == &letra);
63      print_test("lista esta vacia?",
 •      lista_esta_vacia(lista) == false);
64      print_test("Eliminar char de la lista",
 •      lista_borrar_primero(lista) == &letra);
65      //Pruebas con NULL
66      print_test("Insertar NULL al principio",
 •      lista_insertar_primero(lista,NULL) == true);
67      print_test("Ver largo", lista_largo(lista) == 1);
68      print_test("Ver primero", lista_ver_primero(lista)
 •      == NULL);
69      print_test("Ver ultimo", lista_ver_ultimo(lista)
 •      == NULL);
70      print_test("lista esta vacia?",
 •      lista_esta_vacia(lista) == false);
71      print_test("Eliminar NULL de la lista",
 •      lista_borrar_primero(lista) == NULL);
72      print_test("Insertar NULL a lo ultimo",
 •      lista_insertar_ultimo(lista,NULL) == true);
73      print_test("Ver largo", lista_largo(lista) == 1);
74      print_test("Ver primero", lista_ver_primero(lista)
 •      == NULL);
75      print_test("Ver ultimo", lista_ver_ultimo(lista)
 •      == NULL);
76      print_test("lista esta vacia?",
 •      lista_esta_vacia(lista) == false);
77      print_test("Eliminar NULL de la lista",
 •      lista_borrar_primero(lista) == NULL);
78      //Pruebas con long
79      print_test("Insertar long al principio",
 •      lista_insertar_primero(lista,&num_long) == true);
80      print_test("Ver largo", lista_largo(lista) == 1);
81      print_test("Ver primero", lista_ver_primero(lista)
 •      == &num_long);
82      print_test("Ver ultimo", lista_ver_ultimo(lista)
 •      == &num_long);
```

```
    == &num_long);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Eliminar long de la lista",
    lista_borrar_primero(lista) == &num_long);
    print_test("Insertar long a lo ultimo",
    lista_insertar_ultimo(lista,&num_long) == true);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Ver primero", lista_ver_primero(lista)
    == &num_long);
    print_test("Ver ultimo", lista_ver_ultimo(lista)
    == &num_long);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Eliminar long de la lista",
    lista_borrar_primero(lista) == &num_long);
    //Pruebas con string
    print_test("Insertar string al principio",
    lista_insertar_primero(lista,palabra) == true);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Ver primero", lista_ver_primero(lista)
    == palabra);
    print_test("Ver ultimo", lista_ver_ultimo(lista)
    == palabra);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Eliminar string de la lista",
    lista_borrar_primero(lista) == palabra);
    print_test("Insertar string a lo ultimo",
    lista_insertar_ultimo(lista,palabra) == true);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Ver primero", lista_ver_primero(lista)
    == palabra);
    print_test("Ver ultimo", lista_ver_ultimo(lista)
    == palabra);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Eliminar string de la lista",
    lista_borrar_primero(lista) == palabra);
    //Destruir lista con pila
    pila_t* pila = pila_crear();

```

```
107     if(!pila){
108         lista_destruir(lista,NULL);
109         return;
110     }
111     if((pila_apilar(pila,&num)) == false){
112         lista_destruir(lista,NULL);
113         return;
114     }
115     print_test("Insertar pila al principio",
  •     lista_insertar_primero(lista,pila) == true);
116     print_test("Ver largo", lista_largo(lista) == 1);
117     print_test("Ver primero", lista_ver_primero(lista)
  •     == pila);
118     print_test("Ver ultimo", lista_ver_ultimo(lista)
  •     == pila);
119     print_test("lista esta vacia?",
  •     lista_esta_vacia(lista) == false);
120     print_test("Eliminar pila de la lista",
  •     lista_borrar_primero(lista) == pila);
121     print_test("Insertar pila a lo ultimo",
  •     lista_insertar_ultimo(lista,pila) == true);
122     print_test("Ver largo", lista_largo(lista) == 1);
123     print_test("Ver primero", lista_ver_primero(lista)
  •     == pila);
124     print_test("Ver ultimo", lista_ver_ultimo(lista)
  •     == pila);
125     print_test("lista esta vacia?",
  •     lista_esta_vacia(lista) == false);
126
127     lista_destruir(lista,destruir_pila);
128     print_test("Destruir lista con una pila",true);
129     return;
130 }
131
132 void prueba_varios_elementos(){
133     lista_t* lista = lista_crear();
134     int num=10;
135     char letra='a';
136     long num_long;
137     char palabra[10]="Test";
138     float
```

```
    random[15]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14};
    bool status;
    //Agregar al principio varios datos de distinto
    tipo
    print_test("Ver largo", lista_largo(lista) == 0);
    print_test("Agregar al principio int",
    lista_insertar_primero(lista,&num) == true);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Agregar al principio char",
    lista_insertar_primero(lista,&letra) == true);
    print_test("Ver largo", lista_largo(lista) == 2);
    print_test("Agregar al principio NULL",
    lista_insertar_primero(lista,NULL) == true);
    print_test("Ver largo", lista_largo(lista) == 3);
    print_test("Agregar al principio long",
    lista_insertar_primero(lista,&num_long) == true);
    print_test("Ver largo", lista_largo(lista) == 4);
    print_test("Agregar al principio string",
    lista_insertar_primero(lista,palabra) == true);
    print_test("Ver largo", lista_largo(lista) == 5);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Borrar primero: string",
    lista_borrar_primero(lista) == palabra);
    print_test("Ver largo", lista_largo(lista) == 4);
    print_test("Borrar primero: long",
    lista_borrar_primero(lista) == &num_long);
    print_test("Ver largo", lista_largo(lista) == 3);
    print_test("Borrar primero: NULL",
    lista_borrar_primero(lista) == NULL);
    print_test("Ver largo", lista_largo(lista) == 2);
    print_test("Borrar primero: char",
    lista_borrar_primero(lista) == &letra);
    print_test("Ver largo", lista_largo(lista) == 1);
    print_test("Borrar primero: int",
    lista_borrar_primero(lista) == &num);
    print_test("Ver largo", lista_largo(lista) == 0);
    print_test("lista esta vacia?",
    lista_esta_vacia(lista) == true);
    //Agregar al final varios datos de distinto tipo
    print_test("Ver largo", lista_largo(lista) == 0);
```

```
166     print_test("Agregar al final int",
  •     lista_insertar_ultimo(lista,&num) == true);
167     print_test("Ver largo", lista_largo(lista) == 1);
168     print_test("Agregar al final char",
  •     lista_insertar_ultimo(lista,&letra) == true);
169     print_test("Ver largo", lista_largo(lista) == 2);
170     print_test("Agregar al final NULL",
  •     lista_insertar_ultimo(lista,NULL) == true);
171     print_test("Ver largo", lista_largo(lista) == 3);
172     print_test("Agregar al final long",
  •     lista_insertar_ultimo(lista,&num_long) == true);
173     print_test("Ver largo", lista_largo(lista) == 4);
174     print_test("Agregar al final string",
  •     lista_insertar_ultimo(lista,palabra) == true);
175     print_test("Ver largo", lista_largo(lista) == 5);
176     print_test("lista esta vacia?",
  •     lista_esta_vacia(lista) == false);
177     print_test("Borrar primero: int",
  •     lista_borrar_primero(lista) == &num);
178     print_test("Ver largo", lista_largo(lista) == 4);
179     print_test("Borrar primero: char",
  •     lista_borrar_primero(lista) == &letra);
180     print_test("Ver largo", lista_largo(lista) == 3);
181     print_test("Borrar primero: NULL",
  •     lista_borrar_primero(lista) == NULL);
182     print_test("Ver largo", lista_largo(lista) == 2);
183     print_test("Borrar primero: long",
  •     lista_borrar_primero(lista) == &num_long);
184     print_test("Ver largo", lista_largo(lista) == 1);
185     print_test("Borrar primero: string",
  •     lista_borrar_primero(lista) == palabra);
186     print_test("Ver largo", lista_largo(lista) == 0);
187     print_test("lista esta vacia?",
  •     lista_esta_vacia(lista) == true);
188     // Prueba agregar un vector al final
189     status = true;
190     for(int i = 0; i < 15; i++){
191         if((lista_insertar_ultimo(lista,&random[i]))
  •         == false){
192             status = false;
193             break;
```

```
194         }
195         if(lista_largo(lista) != (i+1)){
196             status = false;
197             break;
198         }
199     }
200     print_test("Agregar al final 15 floats de un
    •   vector", status == true);
201     print_test("Ver primero", lista_ver_primero(lista)
    •   == &random[0]);
202     status = true;
203     for(int i = 0; i < 15; i++){
204         if(lista_borrar_primero(lista) != &random[i]){
205             status=false;
206             break;
207         }
208         if(lista_largo(lista) != (14 - i)){
209             status = false;
210             break;
211         }
212     }
213     print_test("Borrar elementos corroborando orden",
    •   status == true);
214     // Prueba agregar un vector al principio
215     status = true;
216     for(int i = 0; i < 15; i++){
217         if((lista_insertar_primero(lista,&random[i]))
    •       == false){
218             status = false;
219             break;
220         }
221         if(lista_largo(lista) != (i+1)){
222             status = false;
223             break;
224         }
225     }
226     print_test("Agregar al principio 15 floats de un
    •   vector", status == true);
227     print_test("Ver primero", lista_ver_primero(lista)
    •   == &random[14]);
228     status = true;
229     for(int i = 0; i < 7; i++){
```

```
    for(int i = 0; i < 7; i++){
        if(lista_borrar_primero(lista) != &random[14 -
        i]){
            status=false;
            break;
        }
        if(lista_largo(lista) != (14 - i)){
            status = false;
            break;
        }
    }
    print_test("Ver primero", lista_ver_primero(lista)
    == &random[7]);

    lista_destruir(lista,NULL);
    print_test("Destruir lista con 7 elementos",true);
    return;
}

void prueba_volumen(){
    lista_t* lista = lista_crear();
    int vec[MAX];
    int * vec_pointers[MAX];
    bool status=true;
    int * aux;

    for(int i = 0; i < MAX; i++){
        vec[i]=i+1;
        if(lista_insertar_ultimo(lista,&vec[i]) ==
        false){
            status=false;
            break;
        }
    }

    print_test("Agregar 10000 int a lo ultimo",status);
    print_test("lista con elementos esta vacia?",
    lista_esta_vacia(lista) == false);
    print_test("Corroborar largo de la
    lista",lista_largo(lista) == 10000);
    print_test("Corrobar que primer elemento sea
    1",*(int *)lista ver primero(lista) == 1);
```

```
265
266     status=true;
267     for(int i = 0; i < MAX; i++){
268         if(*(int *)lista_borrar_primero(lista) !=
    •     vec[i]){
269             status = false;
270             break;
271         }
272         if(lista_largo(lista) != (MAX-1-i)){ //
    •     Verificar largo de la lista
273             status = false;
274             break;
275         }
276     }
277     print_test("Eliminar 10000 int y verificar que los
    •     elementos de la lista esten en el orden
    •     correspondiente",status);
278     print_test("Eliminar primero (luego de haberla
    •     vaciado)", lista_borrar_primero(lista) == NULL);
279     print_test("Ver primer elemento de lista vacia",
    •     lista_ver_primero(lista) == NULL);
280     print_test("lista esta vacia",
    •     lista_esta_vacia(lista) == true);
281     print_test("Corroborar largo de la lista",
    •     !lista_largo(lista));
282
283     for(int i = 0; i < MAX; i++){
284         if((vec_pointers[i] = malloc(sizeof(int))) ==
    •     NULL){
285             lista_destruir(lista,NULL);
286             return;
287         }
288         *vec_pointers[i] = i * 2;
289     }
290     status = true;
291     for(int i = 0; i < MAX; i++){
292         if((i % 2) == 0){
293             if(lista_insertar_primero(lista,NULL) ==
    •         false){
294                 status=false;
295                 break;
```

```
296             }
297         }else{
298             if(lista_insertar_primero(lista,vec_pointer
  •         s[i]) == false){
299                 status=false;
300                 break;
301             }
302         }
303     }
304     print_test("Enlistar NULL e int de forma
  •     intercalada",status);
305     print_test("Corrobar que primer elemento sea
  •     NULL",lista_ver_primero(lista)==vec_pointers[MAX-
  •     1]);
306     print_test("Corroborar largo de la
  •     lista",lista_largo(lista) == 10000);
307 
308     for(int i = 0; i < 1000; i++){
309         aux =(int*)lista_borrar_primero(lista);
310 
311         if((i % 2) != 0){
312             if(aux != NULL){
313                 status=false;
314                 break;
315             }
316         }else{
317             if(aux != vec_pointers[MAX-1-i]){
318                 status=false;
319                 break;
320             }
321         }
322         if(lista_largo(lista) != (MAX-1-i)){ //
  •         Verificar largo de la lista
323             status = false;
324             break;
325         }
326         free(vec_pointers[MAX-1-i]);
327     }
328 
329     print_test("Eliminar y verificar que los elementos
  •     de la lista esten en el orden
```

```
    correspondiente",status);

    lista_destruir(lista,destruir_dato);
    for(int i = 1000; i < MAX; i++) //Libero memoria de los elementos que no se cargaron en la lista
        if((i % 2) != 0)
            free(vec_pointers[MAX-1-i]);

    print_test("Destruir lista con 9000 elementos",true);
    return;
}

void prueba_iterador_lista_vacia(){
    lista_t*     lista = lista_crear();
    lista_iter_t* iter = lista_iter_crear(lista);

    print_test("Creacion de Iterador", iter != NULL);
    print_test("Avanzar Iterador en lista vacia", !lista_iter_avanzar(iter));
    print_test("Ver actual en lista vacia", lista_iter_ver_actual(iter) == NULL);
    print_test("Iterador esta al final?", lista_iter_al_final(iter));
    print_test("Lista esta vacia", lista_esta_vacia(lista) == true);
    print_test("Largo de lista vacia", !lista_largo(lista));
    lista_iter_destruir(iter);
    lista_destruir(lista,NULL);
    print_test("Destruir lista e iterador vacio",true);
}

void prueba_iterador_un_elemento(){
    int num=10;
    lista_t* lista = lista_crear();
    lista_iter_t* iter = lista_iter_crear(lista);
    pila_t* pila = pila_crear();

    if(!pila && !lista && !iter){
        return;
    }
```

```
363     }
364     if((pila_apilar(pila,&num)) == false){
365         lista_destruir(lista,NULL);
366         return;
367     }
368     print_test("Ver largo", lista_largo(lista) == 0);
369     print_test("Insertar con iterador",
  •     lista_iter_insertar(iter,pila) == true);
370     print_test("Ver largo", lista_largo(lista) == 1);
371     print_test("Ver actual con iterador",
  •     (pila_t*)lista_iter_ver_actual(iter) == pila);
372     print_test("lista esta vacia?",
  •     lista_esta_vacia(lista) == false);
373     print_test("Ver ultimo", lista_ver_ultimo(lista)
  •     == pila);
374     print_test("Insertar pila a lo ultimo",
  •     (pila_t*)lista_iter_borrar(iter) == pila);
375     print_test("Ver largo", lista_largo(lista) == 0);
376     print_test("Ver actual",
  •     lista_iter_ver_actual(iter) == NULL);
377     print_test("Ver ultimo", lista_ver_ultimo(lista)
  •     == NULL);
378     print_test("lista esta vacia?",
  •     lista_esta_vacia(lista) == true);
379
380     print_test("Insertar con iterador",
  •     lista_iter_insertar(iter,pila) == true);
381     print_test("Ver largo", lista_largo(lista) == 1);
382     print_test("Avanzar iterador",
  •     lista_iter_avanzar(iter) == true);
383     print_test("Ver actual con iterador",
  •     (pila_t*)lista_iter_ver_actual(iter) == NULL);
384     print_test("Avanzar iterador",
  •     lista_iter_avanzar(iter) == false);
385     print_test("Ver actual con iterador",
  •     (pila_t*)lista_iter_ver_actual(iter) == NULL);
386
387     lista_iter_destruir(iter);
388     lista_destruir(lista,destruir_pila);
389     print_test("Destruir lista con una pila",true);
390     return;
391 }
```

```
391 }
392
393 void prueba_iterador_varios_elementos(){
394     lista_t* lista = lista_crear();
395     lista_iter_t* iter = lista_iter_crear(lista);
396     int num=10;
397     char letra='a';
398     long num_long;
399     char palabra[10]="Test";
400     //float
    •    random[15]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14};
401     //bool status;
402
403     if(!lista && !iter){
404         return;
405     }
406     //Insercion en distintas posiciones
407     print_test("Ver largo", lista_largo(lista) == 0);
408     print_test("Agregar int con iterador",
    •    lista_iter_insertar(iter,&num) == true);
409     print_test("Ver actual",
    •    lista_iter_ver_actual(iter) == &num);
410     print_test("Ver ultimo", lista_ver_ultimo(lista)
    •    == &num);
411     print_test("Ver largo", lista_largo(lista) == 1);
412     print_test("Agregar char con inter",
    •    lista_iter_insertar(iter,&letra) == true);
413     print_test("Ver actual",
    •    lista_iter_ver_actual(iter) == &letra);
414     print_test("Ver ultimo", lista_ver_ultimo(lista)
    •    == &num);
415     print_test("Ver largo", lista_largo(lista) == 2);
416     print_test("Agregar NULL",
    •    lista_iter_insertar(iter,NULL) == true);
417     print_test("Ver actual",
    •    lista_iter_ver_actual(iter) == NULL);
418     print_test("Ver ultimo", lista_ver_ultimo(lista)
    •    == &num);
419     print_test("Avanzar iterador",
    •    lista_iter_avanzar(iter) == true);
420     print_test("Ver largo", lista_largo(lista) == 3);
421     print test("Agregar long",
```

```
    lista_iter_insertar(iter,&num_long) == true);
422 print_test("Final de iterador",
    lista_iter_al_final(iter) == false);
423 print_test("Ver actual",
    lista_iter_ver_actual(iter) == &num_long);
424 print_test("Ver ultimo", lista_ver_ultimo(lista)
    == &num);
425 print_test("Ver primero", lista_ver_primero(lista)
    == NULL);
426 print_test("Avanzar iterador",
    lista_iter_avanzar(iter) == true);
427 print_test("Ver largo", lista_largo(lista) == 4);
428 print_test("Avanzar iterador",
    lista_iter_avanzar(iter) == true);
429 print_test("Avanzar iterador",
    lista_iter_avanzar(iter) == true);
430 print_test("Agregar string",
    lista_iter_insertar(iter,palabra) == true);
431 print_test("Ver largo", lista_largo(lista) == 5);
432 print_test("Ver actual",
    lista_iter_ver_actual(iter) == palabra);
433 print_test("Ver ultimo", lista_ver_ultimo(lista)
    == palabra);
434 print_test("Ver primero", lista_ver_primero(lista)
    == NULL);
435 print_test("Avanzar iterador",
    lista_iter_avanzar(iter) == true);
436 print_test("Ver actual",
    lista_iter_ver_actual(iter) == NULL);
437 print_test("Avanzar iterador",
    lista_iter_avanzar(iter) == false);
438 print_test("Final de iterador",
    lista_iter_al_final(iter) == true);
439 lista_iter_destruir(iter);
440
441 // Borrado en distintas posiciones
442 iter = lista_iter_crear(lista);
443
444 print_test("Eliminar primero con iter",
    lista_iter_borrar(iter) == NULL);
445 print_test("Ver largo", lista_largo(lista) == 4);
```

```
446     print_test("Ver actual",
  •     lista_iter_ver_actual(iter) == &num_long);
447     print_test("Ver ultimo", lista_ver_ultimo(lista)
  •     == palabra);
448     print_test("Ver primero", lista_ver_primero(lista)
  •     == &num_long);
449     print_test("Final de iterador",
  •     lista_iter_al_final(iter) == false);
450     for (size_t i = 0; i < 3; i++) {
451         lista_iter_avanzar(iter);
452     }
453     print_test("Borrar con iter",
  •     lista_iter_borrar(iter) == palabra);
454     print_test("Ver largo", lista_largo(lista) == 3);
455     print_test("Ver primero", lista_ver_primero(lista)
  •     == &num_long);
456     print_test("Ver ultimo", lista_ver_ultimo(lista)
  •     == &num);
457     print_test("Agregar string",
  •     lista_iter_insertar(iter,palabra) == true);
458     print_test("Agregar string",
  •     lista_iter_insertar(iter,palabra) == true);
459     lista_iter_destruir(iter);
460
461     iter = lista_iter_crear(lista);
462     lista_iter_avanzar(iter);
463     print_test("Borrar con iter",
  •     lista_iter_borrar(iter) == &letra);
464     print_test("Ver largo", lista_largo(lista) == 4);
465     print_test("Borrar con iter",
  •     lista_iter_borrar(iter) == &num);
466     print_test("Borrar con iter",
  •     lista_iter_borrar(iter) == palabra);
467     print_test("Borrar con iter",
  •     lista_iter_borrar(iter) == palabra);
468     print_test("Ver largo", lista_largo(lista) == 1);
469     print_test("Ver primero", lista_ver_primero(lista)
  •     == &num_long);
470     print_test("Ver ultimo", lista_ver_ultimo(lista)
  •     == &num_long);
471
```

```
472     lista_iter_destruir(iter);
473     lista_destruir(lista,NULL);
474     print_test("Destruir lista e iterador",true);
475     return;
476 }
477 
478 void prueba_iterador_volumen(){
479     lista_t* lista = lista_crear();
480     lista_iter_t* iter = lista_iter_crear(lista);
481     int vec[MAX];
482     bool status=true;
483 
484     for(int i = 0; i < MAX; i++){
485         vec[i]=i+1;
486         if(lista_iter_insertar(iter,&vec[i]) == false){
487             status=false;
488             break;
489         }
490     }
491 
492     print_test("Agregar 10000 int a lo ultimo",status);
493     print_test("lista con elementos esta vacia?",
  •     lista_esta_vacia(lista) == false);
494     print_test("Corroborar largo de la
  •     lista",lista_largo(lista) == 10000);
495     print_test("Corrobar que primer elemento sea
  •     10000",*(int *)lista_iter_ver_actual(iter) ==
  •     10000);
496 
497     status=true;
498     for(int i = 0; i < MAX; i++){
499         if(*(int *)lista_iter_borrar(iter) != vec[MAX-
  •         1-i]){
500             status = false;
501             break;
502         }
503         if(lista_largo(lista) != (MAX-1-i)){ //
  •         Verificar largo de la lista
504             status = false;
505             break;
506         }
```

```
507     }
508     print_test("Eliminar 10000 int y verificar que los
  •     elementos de la lista esten en el orden
  •     correspondiente",status);
509     print_test("Eliminar primero (luego de haberla
  •     vaciado)", lista_iter_borrar(iter) == NULL);
510     print_test("Ver primer elemento de lista vacia",
  •     lista_iter_ver_actual(iter) == NULL);
511     print_test("lista esta vacia",
  •     lista_esta_vacia(lista) == true);
512     print_test("Corroborar largo de la lista",
  •     !lista_largo(lista));
513
514     lista_iter_destruir(iter);
515     lista_destruir(lista,NULL);
516 }
517
518 bool imprimir(void *elemento, void *extra)
519 {
520     int *contador = extra;
521     printf("%d. %s\n", ++(*contador), (char*)
  •     elemento);
522
523     return true; // seguir iterando
524 }
525
526 void test() {
527     lista_t *super = lista_crear();
528
529     lista_insertar_ultimo(super, "leche");
530     lista_insertar_ultimo(super, "huevos");
531     lista_insertar_ultimo(super, "pan");
532     lista_insertar_ultimo(super, "mermelada");
533
534     int num_items = 0;
535     lista_iterar(super, imprimir, &num_items);
536     printf("Tengo que comprar %d ítems\n", num_items);
537
538     lista_destruir(super,NULL);
539 }
540 void pruebas_lista_alumno() {
541     //Pruebas lista
```

```
    //Pruebas lista
    prueba_lista_vacia();
    prueba_un_elemento();
    prueba_varios_elementos();
    prueba_volumen();
    //Pruebas Iterador externo
    prueba_iterador_lista_vacia();
    prueba_iterador_un_elemento();
    prueba_iterador_varios_elementos();
    prueba_iterador_volumen();
    //Pruebas Iterador interno
    test();
}

```