#include "lista.h"

#include <stdbool.h>

#include <stdlib.h>

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DEFINICION DE LOS TIPOS DE DATOS

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Nodo tiene un puntero generico. \*/

struct nodo{

void \* dato;

struct nodo\* siguiente;

};

typedef struct nodo nodo\_t;

/\* Lista compuesta de nodos. \*/

struct lista{

nodo\_t\* primero;

nodo\_t\* ultimo;

size\_t largo;

};

typedef struct lista lista\_t;

/\* Iterador externo de la lista. \*/

struct lista\_iter{

lista\_t\* lista;

nodo\_t\* anterior;

nodo\_t\* actual;

};

typedef struct lista\_iter lista\_iter\_t;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DEFINICION DE FUNCIONES AUXILIARES

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Se encarga de pedir memoria para un nodo\_t e inicializar las

// referencia del nodo a NULL. Similar a calloc.Si algo falla

// devuelve NULL.

// Post: nodo incializado a NULL.

nodo\_t\* crear\_nodo(){

nodo\_t\* nodo = malloc(sizeof(nodo\_t));

if(!nodo){

return NULL;

}

nodo->dato = NULL;

nodo->siguiente = NULL;

return nodo;

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PRIMITIVAS DE LA LISTA

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Crea una lista y la incializa.

// Post: Devuelve una lista vacia.

lista\_t \*lista\_crear(void){

lista\_t\* lista = malloc(sizeof(lista\_t));

if (!lista){

return NULL;

}

lista->primero = NULL;

lista->ultimo = NULL;

lista->largo = 0;

return lista;

}

// Devuelve verdadero o falso, según si la lista tiene o no elementos.

// Pre: la lista fue creada.

bool lista\_esta\_vacia(const lista\_t \*lista){

return lista->largo==0;

}

// Inserta un elemento al comienzo de la lista. Devuelve un bool

// si la operacion fue realizada con exito o no.

// Pre: la lista fue creada.

// Post: la lista tiene un elemento mas.

bool lista\_insertar\_primero(lista\_t \*lista, void \*dato){

nodo\_t\*nodo = crear\_nodo();

if (!nodo){

return false;

}

nodo->dato = dato;

if (lista\_esta\_vacia(lista)){

lista->ultimo = nodo;

}

nodo->siguiente = lista->primero;

lista->primero = nodo;

lista->largo++;

return true;

}

// Inserta un elemento al final de la lista. Devuelve un bool

// si la operacion fue realizada con exito o no.

// Pre: la lista fue creada.

// Post: la lista tiene un elemento mas.

bool lista\_insertar\_ultimo(lista\_t \*lista, void \*dato){

nodo\_t\* nodo = crear\_nodo();

if (!nodo){

return false;

}

nodo->dato = dato;

if (lista\_esta\_vacia(lista)){

lista->primero = nodo;

}

else{

lista->ultimo->siguiente = nodo;

}

lista->ultimo = nodo;

lista->largo++;

return true;

}

// Elimina el primer elemento de la lista y lo devuelve.

// Si la lista estaba vacia devuelve NULL.

// Pre: la lista fue creada.

// Post: la lista tiene un elemento menos.

void \*lista\_borrar\_primero(lista\_t \*lista){

if (lista\_esta\_vacia(lista)){

return NULL;

}

nodo\_t\* nodo = lista->primero;

void\* dato = nodo->dato;

if (lista\_largo(lista)==1){

lista->primero = NULL;

lista->ultimo = NULL;

}

else{

lista->primero = nodo->siguiente;

}

lista->largo--;

free(nodo);

return dato;

}

// Devuelve el primer elemento de la lista sin elimarlo.

// Pre: la lista fue creada.

void \*lista\_ver\_primero(const lista\_t \*lista){

return lista\_esta\_vacia(lista) ? NULL: lista->primero->dato;

}

// Devuelve el ultimo elemento de la lista sin elimarlo.

// Pre: la lista fue creada.

void \*lista\_ver\_ultimo(const lista\_t\* lista){

return lista\_esta\_vacia(lista) ? NULL: lista->ultimo->dato;

}

// Devuelve la cantidad de elementos que tiene la lista.

// Pre: la lista fue creada.

size\_t lista\_largo(const lista\_t \*lista){

return lista->largo;

}

// 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.

void lista\_destruir(lista\_t \*lista, void destruir\_dato(void \*)){

void\* dato=NULL;

while (!lista\_esta\_vacia(lista)){

dato = lista\_borrar\_primero(lista);

if (destruir\_dato){

destruir\_dato(dato);

}

}

free(lista);

}

// Iterador interno de la lista. Recibe una funcion la cual se aplica

// a todos los elementos de la funcion.

// Pre: la lista fue creada. La funcion visitar fue definida.

void lista\_iterar(lista\_t \*lista, bool visitar(void \*dato, void \*extra), void \*extra){

nodo\_t\* actual = lista->primero;

while(actual && visitar(actual->dato, extra) ){

actual = actual->siguiente;

}

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PRIMITIVAS DEL ITERADOR EXTERNO

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Crea un iterador de la lista pasada por parametro. El iterador esta

// posicionado en el primer elemento de la lista. Si ocurrio algun

// problema al crear el iterador devuelve NULL.

// Pre: la lista fue creada.

lista\_iter\_t \*lista\_iter\_crear(lista\_t \*lista){

if (!lista){

return NULL;

}

lista\_iter\_t\* iter= malloc(sizeof(lista\_iter\_t));

if (!iter){

return NULL;

}

iter->lista = lista;

iter->actual = lista->primero;

iter->anterior = NULL;

return iter;

}

// Devuelve true si se avanzo al siguiente elemento de la lista.

// False si se llego al final.

// Pre: el iterador fue creada.

bool lista\_iter\_avanzar(lista\_iter\_t \*iter){

if(!iter->actual){

return false;

}

iter->anterior = iter->actual;

iter->actual = iter->actual->siguiente;

return true;

}

// Devuelve el elemento de la lista al cual apunta el iterador sin

// eliminarlo. En caso de que la lista este vacia devuelve NULL.

// Pre: el iterador fue creada.

void \*lista\_iter\_ver\_actual(const lista\_iter\_t \*iter){

if(iter->actual){

return iter->actual->dato;

}

return NULL;

}

// Indica mediante un booleano si el iterador se encuentra al final

// de la lista

// Pre: el iterador fue creada.

bool lista\_iter\_al\_final(const lista\_iter\_t \*iter){

return iter->actual == NULL;

}

// Destruye el iterador. No afecta la lista.

// Pre: el iterador fue creada.

void lista\_iter\_destruir(lista\_iter\_t \*iter){

free(iter);

}

// Inserta el dato previo al elemento al cual hace referencia el

// iterador. Devuelve un booleano segun si la operacion se realizo

// correctamente.

// Pre: el iterador fue creada.

bool lista\_iter\_insertar(lista\_iter\_t \*iter, void \*dato){

if(lista\_esta\_vacia(iter->lista)||!iter->anterior){

if (!lista\_insertar\_primero(iter->lista,dato)) return false;

iter->actual = iter->lista->primero;

return true;

}

if (lista\_iter\_al\_final(iter)){

if (!lista\_insertar\_ultimo(iter->lista,dato)) return false;

iter->actual = iter->lista->ultimo;

return true;

}

nodo\_t\* nodo = crear\_nodo();

if (!nodo){

return false;

}

nodo->dato = dato;

nodo->siguiente = iter->actual;

iter->anterior->siguiente=nodo;

iter->actual=nodo;

iter->lista->largo++;

return true;

}

// Elimina el elemento al cual hace referencia el iterador. El iterador

// hara referencia al elemento siguiente del eliminado. Si hacia

// referencia al ultimo elemento ahora estara al final.

// Pre: el iterador fue creada.

void \*lista\_iter\_borrar(lista\_iter\_t \*iter){

if(!iter->actual){

return NULL;

}

iter->actual=iter->actual->siguiente;

if(!iter->anterior){

return lista\_borrar\_primero(iter->lista);

}

nodo\_t\* nodo = iter->anterior->siguiente;

void \* dato = nodo->dato;

iter->anterior->siguiente = iter->actual;

if(iter->actual == iter->lista->ultimo){

iter->lista->ultimo = iter ->anterior;

}

free(nodo);

iter->lista->largo-=1;

return dato;

}

#include "testing.h"

#include "lista.h"

#include "pila.h"

#include <stdio.h>

#include <stddef.h>

#include <stdlib.h>

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PRUEBAS PARA LA COLA

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Realiza pruebas sobre la implementación de la cola del alumno.

//

// Las pruebas deben emplazarse en el archivo ‘pruebas\_alumno.c’, y

// solamente pueden emplear la interfaz pública tal y como aparece en cola.h

// (esto es, las pruebas no pueden acceder a los miembros del struct cola).

//

// Para la implementación de las pruebas se debe emplear la función

// print\_test(), como se ha visto en TPs anteriores.

void imprimir\_iter\_externo(lista\_t \*lista){

lista\_iter\_t \*iter = lista\_iter\_crear(lista);

int num\_items = 0;

while (!lista\_iter\_al\_final(iter)){

int \*elemento = lista\_iter\_ver\_actual(iter);

++num\_items;

printf("%d ",\*elemento);

lista\_iter\_avanzar(iter);

}

printf("La lista tiene %d ítems\n", num\_items);

lista\_iter\_destruir(iter);

}

bool imprimir\_un\_entero(void \*elemento, void \*extra){

// Sabemos que ‘extra’ es un entero, por tanto le podemos hacer un cast.

int \*contador = extra;

(\*contador)++ ;

printf("%d ", \*(int\*)elemento);

return true; // seguir iterando

}

void imprimir\_iter\_interno(lista\_t \*lista){

int num\_items = 0;

lista\_iterar(lista, imprimir\_un\_entero, &num\_items);

printf("La lista tiene %d ítems\n", num\_items);

}

void prueba\_imprimir(lista\_t\* lista){

printf("\nImprimimos usando ambos iteradores.");

printf("Iterador interno.\n");

imprimir\_iter\_interno(lista);

printf("Iterador externo.\n");

imprimir\_iter\_externo(lista);

}

// Realiza pruebas con la lista vacia y no le agrega elementos.

void pruebas\_vacia(lista\_t\* lista){

printf("\nHacemos las pruebas si la lisat esta vacia.\n");

print\_test("Verificamos si la lista esta vacia ",lista\_esta\_vacia(lista));

print\_test("Intentamos borrar un elemento de la lista",lista\_borrar\_primero(lista)==NULL);

print\_test("Vemos el primer elemento(NULL) ",lista\_ver\_primero(lista)==NULL);

print\_test("Vemos el ultimo elemento(NULL) ",lista\_ver\_ultimo(lista)==NULL);

print\_test("El largo debe ser cero ",lista\_largo(lista)==0);

}

// Verifica el funciomiento de ciertas primitivas con la lista llena.

void pruebas\_llena(lista\_t\* lista){

printf("\nHacemos las pruebas si la lisat esta llena.\n");

print\_test("Verificamos si la lista esta llena ",lista\_esta\_vacia(lista)==false);

print\_test("Vemos el primer elemento(NULL) ",\*(int\*)lista\_ver\_primero(lista)==10);

print\_test("Vemos el ultimo elemento(NULL) ",\*(int\*)lista\_ver\_ultimo(lista)==10);

print\_test("Intentamos borrar un elemento de la lista",\*(int\*)lista\_borrar\_primero(lista)==10);

print\_test("El largo debe ser 19 ",lista\_largo(lista)==19);

}

// Prueba las primitivas relacionadas a ingresar elementos.

void pruebas\_cargar\_elementos(lista\_t\* lista,int\* numeros){

printf("\nCargamos un conjunto de numero de tal manera que\n");

printf("la lista queda de tal manera \n");

printf("[10,9,8,7,6,5,4,3,2,1,1,2,3,4,5,6,7,8,9,10]\n");

for (int i = 0; i<10 ; i++){

lista\_insertar\_ultimo(lista,&numeros[i]);

}

for (int i = 0; i<10 ; i++){

lista\_insertar\_primero(lista,&numeros[i]);

}

}

// Prueba las primitivas relacionadas a eliminar elementos.

void pruebas\_vaciar\_elementos(lista\_t\* lista){

printf("\nVaciamos la lista\n");

for (int i = 9; i>0 ; i--){

print\_test("Se elmino el correcto ",\*(int\*)lista\_borrar\_primero(lista)==i);

}

for (int i = 0; i<10 ; i++){

print\_test("Se elmino el correcto ",\*(int\*)lista\_borrar\_primero(lista)==i+1);

}

}

// Prueba las primitivas del iterador externo

void pruebas\_iterador\_llena(lista\_t\* lista, int\* dato){

printf("\nProbramos las pruebas del iterador\n\n");

lista\_iter\_t\* iter = lista\_iter\_crear(lista);

print\_test("El largo debe ser 19 ",lista\_largo(lista)==19);

print\_test("No estamos en el final ",lista\_iter\_al\_final(iter)==false);

print\_test("Insertamos un elemento al principio ",lista\_iter\_insertar(iter,dato));

print\_test("Lo borramos ",\*(int\*)lista\_iter\_borrar(iter)==\*dato);

lista\_iter\_avanzar(iter);

lista\_iter\_avanzar(iter);

print\_test("Insertamos un elemento ",lista\_iter\_insertar(iter,dato));

lista\_iter\_avanzar(iter);

print\_test("Lo borramos ",\*(int\*)lista\_iter\_borrar(iter)==\*dato);

for(int i=0;i<15;i++){

lista\_iter\_avanzar(iter);

}

print\_test("Insertamos un elemento al final ",lista\_iter\_insertar(iter,dato));

lista\_iter\_avanzar(iter);

print\_test("Lo borramos",\*(int\*)lista\_iter\_borrar(iter)==\*dato);

print\_test("Ultimo elemento es el 10",\*(int\*)lista\_ver\_ultimo(lista)==10);

lista\_iter\_destruir(iter);

}

void pruebas\_cargar\_pilas(lista\_t\* lista){

for (int i=0;i<15;i++){

pila\_t\* pila = pila\_crear();

lista\_insertar\_primero(lista,pila);

}

}

void wrraper(void\* dato){

pila\_t\* pila = dato;

pila\_destruir(pila);

}

void pruebas\_lista\_alumno(void){

int \*numeros = malloc(sizeof(int)\*10);

for (int i = 0; i<10 ; i++){

numeros[i]=i+1;

}

lista\_t\* lista = lista\_crear();

pruebas\_vacia(lista);

lista\_destruir(lista,NULL);

lista = lista\_crear();

pruebas\_cargar\_elementos(lista,numeros);

prueba\_imprimir(lista);

pruebas\_llena(lista);

pruebas\_iterador\_llena(lista,&numeros[4]);

prueba\_imprimir(lista);

pruebas\_vaciar\_elementos(lista);

pruebas\_vacia(lista);

pruebas\_cargar\_pilas(lista);

printf("Destruimos la lista\n");

lista\_destruir(lista,wrraper);

free(numeros);

printf("Se termino correctamente el programa\n");

}