75.41 Algortimos y Programación II Curso 4
TDA Lista

Simplemente enlazada

1 de octubre de 2019

1. Enunciado

Se pide implementar una Lista simplemente enlazada. Para ello se brindan las firmas de las funciones públicas a implementar y se deja a criterio del alumno la creación de las funciones privadas del TDA para el correcto funcionamiento de la Lista cumpliendo con las buenas prácticas de programación.

Adicionalmente se pide la creación de un iterador interno y uno externo para la lista.

2. lista.h

```c
#ifndef __LISTA_H__
#define __LISTA_H__

#include <stdbool.h>

typedef struct lista lista_t;
typedef struct lista_iterador lista_iterador_t;

/*
 * Crea la lista reservando la memoria necesaria.
 * Devuelve un puntero a la lista creada o NULL en caso de error.
 */
lista_t* lista_crear();

/*
 * Inserta un elemento al final de la lista.
 * Devuelve 0 si pudo insertar o -1 si no pudo.
 */
int lista_insertar(lista_t* lista, void* elemento);

/*
 * Inserta un elemento en la posicion indicada, donde 0 es insertar
 * como primer elemento y 1 es insertar luego del primer elemento.
 * En caso de no existir la posicion indicada, lo inserta al final.
 * Devuelve 0 si pudo insertar o -1 si no pudo.
 */
int lista_insertar_en_posicion(lista_t* lista, void* elemento, size_t posicion);

/*
 * Quita de la lista el elemento que se encuentra en la ultima posición.
 * Devuelve 0 si pudo eliminar o -1 si no pudo.
 */
int lista_borrar(lista_t* lista);

/*
 * Quita de la lista el elemento que se encuentra en la posición
 * indicada, donde 0 es el primer elemento.
 * En caso de no existir esa posición se intentará borrar el último
 * elemento.
 * Devuelve 0 si pudo eliminar o -1 si no pudo.
 */
int lista_borrar_de_posicion(lista_t* lista, size_t posicion);

/*
 * Devuelve el elemento en la posicion indicada, donde 0 es el primer
```

```
 * elemento.
 * Si no existe dicha posicion devuelve NULL.
 */
void* lista_elemento_en_posicion(lista_t* lista, size_t posicion);

/*
 * Devuelve el último elemento de la lista o NULL si la lista se
 * encuentra vacía.
 */
void* lista_ultimo(lista_t* lista);

/*
 * Devuelve true si la lista está vacía o false en caso contrario.
 */
bool lista_vacia(lista_t* lista);

/*
 * Devuelve la cantidad de elementos almacenados en la lista.
 */
size_t lista_elementos(lista_t* lista);

/*
 * Libera la memoria reservada por la lista.
 */
void lista_destruir(lista_t* lista);


/***** Iterador interno. *****/

/*
* Recorre la lista e invoca la funcion con cada
* elemento de la misma.
*/
void lista_con_cada_elemento(lista_t* lista, void (*funcion)(void*));

/*****************************/


/***** Iterador externo. *****/
/*
 * Crea un iterador para una lista. El iterador creado es válido desde
 * el momento de su creación hasta que no haya mas elementos por
 * recorrer o se modifique la lista iterada (agregando o quitando
 * elementos de la lista).
 * Devuelve el puntero al iterador creado o NULL en caso de error.
 */
lista_iterador_t* lista_iterador_crear(lista_t* lista);

/*
 * Devuelve true si hay mas elementos sobre los cuales iterar o false
 * si no hay mas.
 */
bool lista_iterador_tiene_siguiente(lista_iterador_t* iterador);

/*
 * Avanza el iterador al proximo elemento y lo devuelve.
 * En caso de error devuelve NULL.
 */
void* lista_iterador_siguiente(lista_iterador_t* iterador);

/*
 * Libera la memoria reservada por el iterador.
 */
void lista_iterador_destruir(lista_iterador_t* iterador);

/*****************************/

#endif /* __LISTA_H__ */
```

3. Compilación y Ejecución

El TDA entregado deberá compilar y pasar las pruebas dispuestas por la cátedra sin errores, adicionalmente estas pruebas deberán ser ejecutadas sin pérdida de memoria.

Compilación:

```
gcc *.c -o lista_se -g -std=c99 -Wall -Wconversion -Wtype-limits -pedantic -Werror -O0
```

Ejecución:

```
valgrind --leak-check=full --track-origins=yes --show-reachable=yes ./lista_se
```

4. Minipruebas

Se les brindará un lote de minipruebas, las cuales recomendamos fuertemente sean ampliadas ya que no son exhaustivas y no prueban los casos borde, solo son un ejemplo de como agregar, eliminar, obtener elementos de la lista y qué debería verse en la terminal en el **caso feliz**.

Minipruebas:

```c
#include "lista.h"
#include <stdio.h>

void mostrar_elemento(void* elemento){
        if(elemento)
                printf("%c ", *(char*)elemento);
}

int main(){

        lista_t* lista = lista_crear();

        char a='a', b='b', c='c', d='d', w='w';

        lista_insertar(lista, &a);
        lista_insertar(lista, &c);
        lista_insertar_en_posicion(lista, &d, 100);
        lista_insertar_en_posicion(lista, &b, 1);
        lista_insertar_en_posicion(lista, &w, 3);

        lista_borrar_de_posicion(lista, 3);

        printf("Elementos en la lista: ");
        for(size_t i=0;i<lista_elementos(lista);i++)
                printf("%c ", *(char*)lista_elemento_en_posicion(lista, i));

        printf("\n");

        lista_iterador_t* it = lista_iterador_crear(lista);
        while(lista_iterador_tiene_siguiente(it))
                printf("%c ", *(char*)lista_iterador_siguiente(it));
        printf("\n");

        lista_iterador_destruir(it);

        printf("Imprimo la lista usando el iterador interno: ");
        lista_con_cada_elemento(lista, mostrar_elemento);
        printf("\n");

        lista_destruir(lista);

        return 0;
}
```

La salida por pantalla luego de correrlas con valgrind debería ser:

```
==10126== Memcheck, a memory error detector
==10126== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10126== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==10126== Command: ./lista_se
==10126==
Elementos en la lista: a b c d
a b c d
Imprimo la lista usando el iterador interno: a b c d
==10126==
==10126== HEAP SUMMARY:
==10126==     in use at exit: 0 bytes in 0 blocks
==10126==   total heap usage: 8 allocs, 8 frees, 1,128 bytes allocated
==10126==
==10126== All heap blocks were freed -- no leaks are possible
```

```
==10126==
==10126== For counts of detected and suppressed errors, rerun with: -v
==10126== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

5. Entrega

La entrega deberá contar con todos los archivos necesarios para compilar y ejecutar correctamente el TDA.

Dichos archivos deberán formar parte de un único archivo **.zip** el cual será entregado a través de la plataforma de corrección automática **Kwyjibo**.

El archivo comprimido deberá contar, además del TDA con:

- El archivo con las pruebas agregadas para comprobar el correcto funcionamiento del TDA. Es importante notar que, en este TDA es impescindible realizar pruebas de caja negra ya que el usuario no conoce la estructura interna de la lista, solo puede comunicarse con ella a través de las funciones expuestas.
- Un **Readme.txt** donde se deberá explicar qué es lo entregado, como compilarlo (línea de compilación), como ejecutarlo (línea de ejecución) y todo lo que crea necesario aclarar.
- El enunciado.