75.41 Algortimos y Programación II Curso 4
TDA Lista

Simplemente enlazada

7 de abril 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.

2. lista_se_.h

```c
#ifndef __LISTA_SE_H__
#define __LISTA_SE_H__

#include <stdbool.h>
#include <stdlib.h>

typedef struct nodo {
        void* elemento;
        struct nodo* siguiente;
} nodo_t;

typedef struct lista_se {
        nodo_t* inicio;
        size_t  cantidad;
} lista_se_t;

/*
 * Crea la estructura de la lista, inicializando
 * los nodos inicio y actual en NULL y reservando la memoria
 * necesaria para la estructura.
 */
lista_se_t* crear_lista();

/*
 * Inserta un elemento luego al final de la lista, reservando la memoria necesaria para este nodo.
 * Devuelve 0 si pudo insertar o -1 si no pudo.
 */
int insertar(lista_se_t* lista, void* elemento);

/*
 * Inserta un elemento en la posicion indicada, reservando la memoria necesaria para este nodo.
 * En caso de no existir la posicion indicada, lo inserta al final.
 * Devuelve 0 si pudo insertar o -1 si no pudo.
 */
int insertar_en_posicion(lista_se_t* lista, void* elemento, int indice);

/*
 * Borra el elemento que se encuentra en la ultima posición liberando la memoria reservada para el.
 * Devuelve 0 si pudo eliminar o -1 si no pudo.
 */
int borrar(lista_se_t* lista);

/*
 * Borra el elemento que se encuentra en la posición indicada, liberando la memoria reservada para el.
```

```
 * En caso de no existir esa posición se intentará borrar el último elemento.
 * Devuelve 0 si pudo eliminar o -1 si no pudo.
 */
int borrar_de_posicion(lista_se_t* lista, int indice);

/*
 * Devuelve el elemento en la posicion indice.
 * Si no existe dicha posicion devuelve NULL.
 */
void* elemento_en_posicion(lista_se_t* lista, int indice);

/*
 * Devuelve el último elemento de la lista.
 * Si no existe dicha posicion devuelve NULL.
 */
void* ultimo(lista_se_t* lista);

/*
 * Devuelve true si la lista está vacía o false si no lo está.
 */
bool vacia(lista_se_t* lista);

/*
 * Devuelve la cantidad de elementos en una lista.
 */
size_t elementos(lista_se_t* lista);

/*
 * Libera la memoria reservada por los nodos presentes en la lista y luego la memoria
    reservada por la estructura.
 * Devuelve 0 si pudo destruirla o -1 si no pudo.
 */
int destruir_lista(lista_se_t* lista);

#endif /* __LISTA_SE_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:

```
#include "lista_se.h"
#include <stdio.h>

int main(){
        lista_se_t* lista = crear_lista();

        char elemento_1 = 'A';
        char elemento_2 = 'l';
        char elemento_3 = 'g';
        char elemento_4 = 'o';
        char elemento_5 = '2';

        for (int i = 0; i < 3; i++) {
                insertar(lista, &elemento_1);
                insertar(lista, &elemento_2);
                insertar(lista, &elemento_3);
                insertar(lista, &elemento_4);
```

```
                insertar(lista, &elemento_5);
        }

        insertar_en_posicion(lista, &elemento_1, 0);
        insertar_en_posicion(lista, &elemento_2, 2);
        insertar_en_posicion(lista, &elemento_3, 4);
        insertar_en_posicion(lista, &elemento_4, 6);
        insertar_en_posicion(lista, &elemento_5, 8);

        for (int i = 0; i < 10; i+=2) {
                printf("%c\n", *(char*)elemento_en_posicion(lista, i));
        }

        for (int i = 0; i < 10; i+=2) {
                borrar_de_posicion(lista, i);
        }

        destruir_lista(lista);
        return 0;
}
```

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

```
==18753== Memcheck, a memory error detector
==18753== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18753== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==18753== Command: ./lista_se
==18753==
A
l
g
o
2
==18753==
==18753== HEAP SUMMARY:
==18753==     in use at exit: 0 bytes in 0 blocks
==18753==   total heap usage: 22 allocs, 22 frees, 1,360 bytes allocated
==18753==
==18753== All heap blocks were freed -- no leaks are possible
==18753==
==18753== For counts of detected and suppressed errors, rerun with: -v
==18753== 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.
- 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.