75.41 Algortimos y Programación II Curso 4
TDA Hash

Abierto o Cerrado

27 de octubre de 2019

1. Enunciado

Se pide implementar un Hash. Para ello se brindan las firmas de las funciones públicas a implementar y se deja a criterio del alumno tanto la creación de las funciones privadas del TDA para el correcto funcionamiento del Hash cumpliendo con las buenas prácticas de programación como la elección de que sea Abierto o Cerrado para la resolución de colisiones.

Adicionalmente se pide la creación de un iterador para el recorrido del Hash.

2. hash.h

```c
#ifndef __HASH_H__
#define __HASH_H__

#include <stdbool.h>
#include <stddef.h>

typedef struct hash hash_t;
typedef void (*hash_destruir_dato_t)(void*);

/*
 * Crea el hash reservando la memoria necesaria para el.
 * Destruir_elemento es un destructor que se utilizará para liberar
 * los elementos que se eliminen del hash.
 * Capacidad indica la capacidad minima inicial con la que se crea el hash.
 * Devuelve un puntero al hash creado o NULL en caso de no poder crearlo.
 */
hash_t* hash_crear(hash_destruir_dato_t destruir_elemento, size_t capacidad);

/*
 * Inserta un elemento reservando la memoria necesaria para el mismo.
 * Devuelve 0 si pudo guardarlo o -1 si no pudo.
 */
int hash_insertar(hash_t* hash, const char* clave, void* elemento);

/*
 * Quita un elemento del hash e invoca la funcion destructora
 * pasandole dicho elemento.
 * Devuelve 0 si pudo eliminar el elemento o -1 si no pudo.
 */
int hash_quitar(hash_t* hash, const char* clave);

/*
 * Devuelve un elemento del hash con la clave dada o NULL si dicho
 * elemento no existe.
 */
void* hash_obtener(hash_t* hash, const char* clave);

/*
 * Devuelve true si el hash contiene un elemento almacenado con la
 * clave dada o false en caso contrario.
 */
bool hash_contiene(hash_t* hash, const char* clave);

/*
```

```
 * Devuelve la cantidad de elementos almacenados en el hash.
 */
size_t hash_cantidad(hash_t* hash);

/*
 * Destruye el hash liberando la memoria reservada y asegurandose de
 * invocar la funcion destructora con cada elemento almacenado en el
 * hash.
 */
void hash_destruir(hash_t* hash);

#endif /* __HASH_H__ */
```

3. hash_iterador.h

```
#ifndef _HASH_ITERADOR_H_
#define _HASH_ITERADOR_H_

#include <stdbool.h>
#include "hash.h"

/* Iterador externo para el HASH */
typedef struct hash_iter hash_iterador_t;

/*
 * Crea un iterador de claves para el hash reservando la memoria
 * necesaria para el mismo. El iterador creado es válido desde su
 * creación hasta que se modifique la tabla de hash (insertando o
 * removiendo elementos);
 *
 * Devuelve el puntero al iterador creado o NULL en caso de error.
 */
hash_iterador_t* hash_iterador_crear(hash_t* hash);

/*
 * Devuelve la próxima clave almacenada en el hash y avanza el iterador.
 * Devuelve la clave o NULL si no habia mas.
 */
void* hash_iterador_siguiente(hash_iterador_t* iterador);

/*
 * Devuelve true si quedan claves por recorrer o false en caso
 * contrario.
 */
bool hash_iterador_tiene_siguiente(hash_iterador_t* iterador);

/*
 * Destruye el iterador del hash.
 */
void hash_iterador_destruir(hash_iterador_t* iterador);

#endif /* _HASH_ITERADOR_H_ */
```

4. 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 hash -g -std=c99 -Wall -Wconversion -Wtype-limits -pedantic -Werror -O0
```

Ejecución:

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

5. Minipruebas

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

Minipruebas:

```c
#include "hash.h"
#include "hash_iterador.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Necesario porque strdup es POSIX pero no C99
extern char* strdup(const char*);

void destruir_string(void* elemento){
	if(elemento){
		printf("(Destructor) Libero el vehiculo: %s\n", (char*)elemento);
		free(elemento);
	}
}

void guardar_vehiculo(hash_t* garage, const char* patente, const char* descripcion){
	int retorno = hash_insertar(garage, patente, strdup(descripcion));
	printf("Guardando vehiculo patente %s (%s): ", patente, descripcion);
	printf("%s\n", retorno==0?"OK":"ERROR");
}

void quitar_vehiculo(hash_t* garage, const char* patente){
	int retorno = hash_quitar(garage, patente);
	printf("Retirando vehiculo patente %s: ", patente);
	printf("%s\n", retorno==0?"OK":"ERROR");
}

void verificar_vehiculo(hash_t* garage, const char* patente, bool deberia_existir){
	printf("Verifico el vehiculo patente %s: ", patente);
	bool retorno = hash_contiene(garage, patente);
	printf("%s\n", (retorno==deberia_existir)?"OK":"ERROR");
}

int main(){
	hash_t* garage = hash_crear(destruir_string, 5);

	printf("Agrego autos al garage\n");

	guardar_vehiculo(garage, "AC123BD", "Auto de Mariano");
	guardar_vehiculo(garage, "OPQ976", "Auto de Lucas");
	guardar_vehiculo(garage, "A421ACB", "Moto de Manu");
	guardar_vehiculo(garage, "AA442CD", "Auto de Guido");
	guardar_vehiculo(garage, "AC152AD", "Auto de Agustina");
	guardar_vehiculo(garage, "DZE443", "Auto de Jonathan");
	guardar_vehiculo(garage, "AA436BA", "Auto de Gonzalo");
	guardar_vehiculo(garage, "QDM443", "Auto de Daniela");
	guardar_vehiculo(garage, "BD123AC", "Auto de Pablo");
	guardar_vehiculo(garage, "CD442AA", "Auto de Micaela");
	guardar_vehiculo(garage, "PQO697", "Auto de Juan");
	guardar_vehiculo(garage, "DZE443", "Auto de Jonathan otra vez");
	guardar_vehiculo(garage, "AC152AD", "Auto de Agustina otra vez");

	verificar_vehiculo(garage, "QDM443", true);
	verificar_vehiculo(garage, "PQO697", true);

	quitar_vehiculo(garage, "QDM443");
	quitar_vehiculo(garage, "PQO697");

	verificar_vehiculo(garage, "QDM443", false);
	verificar_vehiculo(garage, "PQO697", false);

	hash_iterador_t* iter = hash_iterador_crear(garage);
	size_t listados = 0;

	while(hash_iterador_tiene_siguiente(iter)){
		const char* clave = hash_iterador_siguiente(iter);
		if(clave){
			listados++;
```

```
                        printf("Patente: %s -- Vehiculo: %s\n", clave, (char*)hash_obtener(garage, clave));
                }
        }

        printf("Cantidad de autos guardados: %zu. Cantidad de autos listados: %zu -- %s\n", hash_cantidad(garage), listados, (hash_cantidad(garage)==listados)?"OK":"ERROR");

        hash_iterador_destruir(iter);
        hash_destruir(garage);

        return 0;
}
```

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

```
==10525== Memcheck, a memory error detector
==10525== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10525== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==10525== Command: ./hash
==10525==
Agrego autos al garage
Guardando vehiculo patente AC123BD (Auto de Mariano): OK
Guardando vehiculo patente OPQ976 (Auto de Lucas): OK
Guardando vehiculo patente A421ACB (Bici de Manu): OK
Guardando vehiculo patente AA442CD (Auto de Guido): OK
Guardando vehiculo patente AC152AD (Auto de Agustina): OK
Guardando vehiculo patente DZE443 (Auto de Jonathan): OK
Guardando vehiculo patente AA436BA (Auto de Gonzalo): OK
Guardando vehiculo patente QDM443 (Auto de Daniela): OK
Guardando vehiculo patente BD123AC (Auto de Pablo): OK
Guardando vehiculo patente CD442AA (Auto de Micaela): OK
Guardando vehiculo patente PQO697 (Auto de Juan): OK
(Destructor) Libero el vehiculo: Auto de Jonathan
Guardando vehiculo patente DZE443 (Auto de Jonathan otra vez): OK
(Destructor) Libero el vehiculo: Auto de Agustina
Guardando vehiculo patente AC152AD (Auto de Agustina otra vez): OK
Verifico el vehiculo patente QDM443: OK
Verifico el vehiculo patente PQO697: OK
(Destructor) Libero el vehiculo: Auto de Daniela
Retirando vehiculo patente QDM443: OK
(Destructor) Libero el vehiculo: Auto de Juan
Retirando vehiculo patente PQO697: OK
Verifico el vehiculo patente QDM443: OK
Verifico el vehiculo patente PQO697: OK
Patente: AC123BD -- Vehiculo: Auto de Mariano
Patente: BD123AC -- Vehiculo: Auto de Pablo
Patente: AA442CD -- Vehiculo: Auto de Guido
Patente: AA436BA -- Vehiculo: Auto de Gonzalo
Patente: OPQ976 -- Vehiculo: Auto de Lucas
Patente: DZE443 -- Vehiculo: Auto de Jonathan otra vez
Patente: AC152AD -- Vehiculo: Auto de Agustina otra vez
Patente: CD442AA -- Vehiculo: Auto de Micaela
Patente: A421ACB -- Vehiculo: Bici de Manu
Cantidad de autos guardados: 9. Cantidad de autos listados: 9 -- OK
(Destructor) Libero el vehiculo: Auto de Mariano
(Destructor) Libero el vehiculo: Auto de Pablo
(Destructor) Libero el vehiculo: Auto de Guido
(Destructor) Libero el vehiculo: Auto de Gonzalo
(Destructor) Libero el vehiculo: Auto de Lucas
(Destructor) Libero el vehiculo: Auto de Jonathan otra vez
(Destructor) Libero el vehiculo: Auto de Agustina otra vez
(Destructor) Libero el vehiculo: Auto de Micaela
(Destructor) Libero el vehiculo: Bici de Manu
==10525==
==10525== HEAP SUMMARY:
==10525==     in use at exit: 0 bytes in 0 blocks
==10525==   total heap usage: 50 allocs, 50 frees, 1,782 bytes allocated
==10525==
==10525== All heap blocks were freed -- no leaks are possible
==10525==
==10525== For counts of detected and suppressed errors, rerun with: -v
==10525== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

6. 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.