75.41 Algortimos y Programación II Curso 4
TDA Hash

Abierto

10 de julio de 2020

1. Enunciado

Se pide implementar un TDA 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. El Hash a implementar debe ser abierto y cumplir con los requisitos detallados en el *.h* .

Adicionalmente se pide la creación de un TDA iterador externo para el recorrido de las claves almacenadas en el Hash.

2. hash.h

```c
#ifndef __HASH_H__
#define __HASH_H__

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

typedef struct hash hash_t;

/*
 * Destructor de los datos almacenados en el hash.  Cada vez que un
 * elemento abandone el hash, debe invocarse al destructor pasandole
 * como parámetro dicho elemento.
 */
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 inicial con la que se crea el hash. La capacidad inicial
 * no puede ser menor a 3. Si se solicita una capacidad menor, el hash
 * se creará con una capacidad de 3.
 *
 * 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 en el hash asociado a la clave dada.
 *
 * Nota para los alumnos: Recordar que si insertar un elemento provoca
 * que el factor de carga exceda cierto umbral, se debe ajustar el
 * tamaño de la tabla para evitar futuras colisiones.
 *
 * 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.
 */
```

```c
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 (o en caso de error).
 */
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 (o en caso de error).
 */
bool hash_contiene(hash_t* hash, const char* clave);

/*
 * Devuelve la cantidad de elementos almacenados en el hash o 0 en
 * caso de error.
 */
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);


/*
 * Recorre cada una de las claves almacenadas en la tabla de hash e
 * invoca a la función funcion, pasandole como parámetros el hash, la
 * clave en cuestión y el puntero auxiliar.
 *
 * Mientras que queden mas claves o la funcion retorne false, la
 * iteración continúa. Cuando no quedan mas claves o la función
 * devuelve true, la iteración se corta y la función principal
 * retorna.
 *
 * Devuelve la cantidad de claves totales iteradas (la cantidad de
 * veces que fue invocada la función) o 0 en caso de error.
 *
 */
size_t hash_con_cada_clave(hash_t* hash, bool (*funcion)(hash_t* hash, const char* clave, void* aux)
    , void* aux);

#endif /* __HASH_H__ */
```

3. hash_iterador.h

```c
#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.
 */
const char* hash_iterador_siguiente(hash_iterador_t* iterador);

/*
 * Devuelve true si quedan claves por recorrer o false en caso
```

```
 * contrario o de error.
 */
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>

//strdup no lo podemos usar porque es POSIX pero no es C99
char* duplicar_string(const char* s){
  if(!s)
    return NULL;

  char* p = malloc(strlen(s)+1);
  strcpy(p,s);
  return p;
}

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

bool mostrar_patente(hash_t* hash, const char* clave, void* aux){
  if(!clave)
    return true;

  aux=aux;
  hash=hash;

  printf("Patente en el hash: %s\n", clave);

  return false;
}

void guardar_vehiculo(hash_t* garage, const char* patente, const char* descripcion){
  int retorno = hash_insertar(garage, patente, duplicar_string(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){
```

```c
  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, 3);

  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\n",
    hash_cantidad(garage), listados, (hash_cantidad(garage)==listados)?"OK":"ERROR");

  hash_iterador_destruir(iter);

  size_t impresas = hash_con_cada_clave(garage, mostrar_patente, NULL);
  printf("Se mostraron %zu patentes con el iterador interno\n\n", impresas);

  hash_destruir(garage);

  return 0;
}
```

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

```
==21415== Memcheck, a memory error detector
==21415== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==21415== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==21415== Command: ./hash_minipruebas
==21415==
Agrego autos al garage
Guardando vehiculo patente AC123BD (Auto de Mariano): OK
Guardando vehiculo patente OPQ976 (Auto de Lucas): OK
Guardando vehiculo patente A421ACB (Moto 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: A421ACB -- Vehiculo: Moto de Manu
Patente: OPQ976 -- Vehiculo: Auto de Lucas
Patente: DZE443 -- Vehiculo: Auto de Jonathan otra vez
Patente: AA442CD -- Vehiculo: Auto de Guido
Patente: AC152AD -- Vehiculo: Auto de Agustina otra vez
Patente: AA436BA -- Vehiculo: Auto de Gonzalo
Patente: BD123AC -- Vehiculo: Auto de Pablo
Patente: CD442AA -- Vehiculo: Auto de Micaela
Patente: AC123BD -- Vehiculo: Auto de Mariano
Cantidad de autos guardados: 9. Cantidad de autos listados: 9 -- OK

Patente en el hash: A421ACB
Patente en el hash: OPQ976
Patente en el hash: DZE443
Patente en el hash: AA442CD
Patente en el hash: AC152AD
Patente en el hash: AA436BA
Patente en el hash: BD123AC
Patente en el hash: CD442AA
Patente en el hash: AC123B
Se mostraron 9 patentes con el iterador interno

(Destructor) Libero el vehiculo: Moto de Manu
(Destructor) Libero el vehiculo: Auto de Lucas
(Destructor) Libero el vehiculo: Auto de Jonathan otra vez
(Destructor) Libero el vehiculo: Auto de Guido
(Destructor) Libero el vehiculo: Auto de Agustina otra vez
(Destructor) Libero el vehiculo: Auto de Gonzalo
(Destructor) Libero el vehiculo: Auto de Pablo
(Destructor) Libero el vehiculo: Auto de Micaela
(Destructor) Libero el vehiculo: Auto de Mariano
==21415==
==21415== HEAP SUMMARY:
==21415==     in use at exit: 0 bytes in 0 blocks
==21415==   total heap usage: 98 allocs, 98 frees, 2,665 bytes allocated
==21415==
==21415== All heap blocks were freed -- no leaks are possible
==21415==
==21415== For counts of detected and suppressed errors, rerun with: -v
==21415== 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.

    - Una explicación de los diferentes tipos de hashes que conoce y cómo se resuelven las colisiones.
    - Qué entiende por rehash y por qué es importante.
- El enunciado.