**Apellidos: Cardozo - Allende Cantidad de hojas: 2**

**Padrones: 98750 - 98559 Corrector Jorge Collinet**

#ifndef HASH\_H

#define HASH\_H

#include <stdbool.h>

#include <stddef.h>

// Los structs deben llamarse "hash" y "hash\_iter".

struct hash;

struct hash\_iter;

typedef struct hash hash\_t;

typedef struct hash\_iter hash\_iter\_t;

// tipo de función para destruir dato

typedef void (\*hash\_destruir\_dato\_t)(void \*);

/\* Crea el hash

\*/

hash\_t \*hash\_crear(hash\_destruir\_dato\_t destruir\_dato);

/\* Guarda un elemento en el hash, si la clave ya se encuentra en la

\* estructura, la reemplaza. De no poder guardarlo devuelve false.

\* Pre: La estructura hash fue inicializada

\* Post: Se almacenó el par (clave, dato)

\*/

bool hash\_guardar(hash\_t \*hash, const char \*clave, void \*dato);

/\* Borra un elemento del hash y devuelve el dato asociado. Devuelve

\* NULL si el dato no estaba.

\* Pre: La estructura hash fue inicializada

\* Post: El elemento fue borrado de la estructura y se lo devolvió,

\* en el caso de que estuviera guardado.

\*/

void \*hash\_borrar(hash\_t \*hash, const char \*clave);

/\* Obtiene el valor de un elemento del hash, si la clave no se encuentra

\* devuelve NULL.

\* Pre: La estructura hash fue inicializada

\*/

void \*hash\_obtener(const hash\_t \*hash, const char \*clave);

/\* Determina si clave pertenece o no al hash.

\* Pre: La estructura hash fue inicializada

\*/

bool hash\_pertenece(const hash\_t \*hash, const char \*clave);

/\* Devuelve la cantidad de elementos del hash.

\* Pre: La estructura hash fue inicializada

\*/

size\_t hash\_cantidad(const hash\_t \*hash);

/\* Destruye la estructura liberando la memoria pedida y llamando a la función

\* destruir para cada par (clave, dato).

\* Pre: La estructura hash fue inicializada

\* Post: La estructura hash fue destruida

\*/

void hash\_destruir(hash\_t \*hash);

/\* Iterador del hash \*/

// Crea iterador

hash\_iter\_t \*hash\_iter\_crear(const hash\_t \*hash);

// Avanza iterador

bool hash\_iter\_avanzar(hash\_iter\_t \*iter);

// Devuelve clave actual, esa clave no se puede modificar ni liberar.

const char \*hash\_iter\_ver\_actual(const hash\_iter\_t \*iter);

// Comprueba si terminó la iteración

bool hash\_iter\_al\_final(const hash\_iter\_t \*iter);

// Destruye iterador

void hash\_iter\_destruir(hash\_iter\_t\* iter);

#endif // HASH\_H