Задача 7. Хеш-таблица+

Источник:	повышенной сложности I
Имя входного файла:	---
Имя выходного файла:	---
Ограничение по времени:	разумное
Ограничение по памяти:	разумное

В этой задаче нужно реализовать “map” (отображение) с помощью хеш-таблицы. Задача продолжает предыдущую с некоторыми усложнениями.

Отличия этой задачи от предыдущей:

1. Пользователь **не** задаёт количество ячеек в хеш-таблице. Реализация должна сама следить за степенью заполненности таблицы и самостоятельно увеличивать таблицу при необходимости.
2. Нужно дополнительно реализовать операцию удаления заданного ключа из таблицы.
3. Функция сравнения и хеш-функция принимают контекст — указатель на произвольные дополнительные данные пользователя.
4. Функция `HM_Destroy` заменена на функцию `HM_Clear`, которая также освобождает память, но с таблицей после этого можно продолжать работать — таблица просто становится пустой.

Вы должны оформить вашу хеш-таблицу как динамическую библиотеку. В хедере `hashmap.h` должна быть объявлена структура `HashMap`, а также следующие типы и функции:

```
//pointer to key or value (untyped)
typedef const void *cpvoid;

//returns 1 if and only if two keys pointed by [a] and [b] are equal
//returns 0 otherwise
typedef int (*EqualFunc)(void *context, cpvoid a, cpvoid b);
//returns 32-bit hash of a key pointed by [key]
typedef uint32_t (*HashFunc)(void *context, cpvoid key);

//creates and returns new hash table with:
// [ef] -- function which compares keys for equality
// [hf] -- function which produces a hash for a key
// [context] -- a pointer which is passed to HashFunc and EqualFunc
HashMap HM_Init(EqualFunc ef, HashFunc hf, void *context);
//frees memory of hash map [self] and resets it to empty state
//it is allowed to use any operations on the table afterwards
void HM_Clear(HashMap *self);

//returns value corresponding to the specified key [key] in hash map [self]
//if [key] is not present in the map, then returns NULL
cpvoid HM_Get(const HashMap *self, cpvoid key);
//sets value [value] for the key [key] in hash map [self]
//if [self] already has some value for [key], it is overwritten
void HM_Set(HashMap *self, cpvoid key, cpvoid value);
//removes key [key] from the table [self]
//returns the value corresponding to key [key] before removal
//if key [key] is not present, then nothing is done, and NULL is returned
cpvoid HM_Remove(HashMap *self, cpvoid key);
```

Пример кода, который должен собираться вместе с библиотекой импорта и работать:

```
#include "hashmap.h"
#include <assert.h>
#include <string.h>

#define INTOF(ptr) (*(int*)(ptr))
int IntModEqualFunc(void *context, cpvoid a, cpvoid b) {
    return (INTOF(a) - INTOF(b)) % INTOF(context) == 0;
}
uint32_t IntModHashFunc(void *context, cpvoid key) {
    int t = INTOF(key) % INTOF(context);
    if (t < 0)
        t += INTOF(context);
    return t * 0xDEADBEEF;
}

int main() {
    int MOD = 1000;
    HashMap h = HM_Init(IntModEqualFunc, IntModHashFunc, &MOD);
    int data[] = {13, 174, 1013, -987, 0, 1};
    HM_Set(&h, &data[0], "hello");
    HM_Set(&h, &data[1], "world");
    assert(strcmp(HM_Get(&h, &data[2]), "hello") == 0);
    const char *old = HM_Remove(&h, &data[3]);
    assert(old && strcmp(old, "hello") == 0);
    assert(HM_Get(&h, &data[0]) == 0);
    old = HM_Remove(&h, &data[2]);
    assert(!old);
    HM_Set(&h, &data[4], "zero");
    assert(strcmp(HM_Get(&h, &data[4]), "zero") == 0);
    assert(strcmp(HM_Get(&h, &data[1]), "world") == 0);
    HM_Set(&h, &data[5], "one");
    assert(HM_Get(&h, &data[0]) == 0);
    HM_Clear(&h);   //note: frees memory
    assert(HM_Get(&h, &data[5]) == 0);
    assert(HM_Remove(&h, &data[3]) == 0);
    HM_Set(&h, &data[1], "newtable");
    HM_Set(&h, &data[1], "newtableX");
    assert(strcmp(HM_Get(&h, &data[1]), "newtableX") == 0);
}
```

Как и в примере к предыдущей задаче, здесь ключом является остаток от деления на 1000. Однако в этот раз модуль 1000 не зашит в код, а лежит в локальной переменной. Указатель на эту переменную передаётся как `context` при инициализации хеш-таблицы, которая затем передаёт его в функции сравнения и вычисления хеша.