COMP2200 과제 4

본 과제는 컴퓨터에서 해야 하는 과제입니다. 코드 작성이 끝났다면 실습 1에서 만들었던 깃 저장소에 커밋 및 푸시를 하고 슬랙을 통해 자동으로 채점을 받으세요.

해시 테이블(또는 해시 맵)은 아주 유용한 데이터 구조 중 하나로 프로그래밍을 하다 보면 사용할 일이 정말 많습니다. 배열이나 연결 리스트와는 달리 해시 테이블은 검색, 삽입, 삭제에 드는 시간이 모두 O(1)이죠. 따라서 방대한 양의 데이터를 빨리 검색해야 한다면 해시 테이블에 그 데이터를 저장하는 게 옳은 경우가 대부분입니다. 나중에 알고리듬 과목에서 배울 테지만 알고리듬의 실행시간을 최적화하는 기법들(예: 메모이제이션, memoization)은 해시 테이블을 자주 이용합니다. 왜냐면 알고리듬을 최적화하려면 수행해야 하는 연산 수를 줄여야 하는데 이러는 방법 중 하나가 기존의 상태를 메모리에 저장하고 그로부터 다음 상태를 도출해 내는 것이기 때문이죠. 해시 테이블은 O(1)의 검색 시간을 가지고 있기에 이를 사용하는 것이 연결 리스트나 배열을 사용하는 것보다 훨씬 매력적인 방법입니다.

최신 언어들은 대부분 해시 테이블을 자체적으로 지원합니다. 다음이 그 예이죠.

* C#의 Dictionary
* Java의 HashMap
* C++의 unordered\_map
* JavaScript의 JSON도 기본적으로 해시 테이블처럼 작동합니다.

불행히도 고대 언어인 C는 해시 테이블을 자체적으로 지원하지 않습니다. 그러면 이 실습에서 뭘 해야 하는지 알겠죠? 직접 만들어 보는 겁니다! 준비되었나요?

1. 프로젝트를 준비한다

1. Assignment4 폴더로 이동합니다.
2. 이 폴더에 node.h 헤더 파일을 추가합니다.
3. 헤더 파일 안에 다음의 내용을 추가합니다.

#ifndef NODE\_H  
#define NODE\_H

typedef struct node {  
 char\* key;  
 int value;  
 struct node\* next;  
} node\_t;

#endif */\* NODE\_H \*/*

1. hashmap.h 파일을 폴더에 추가한 뒤, 다음의 내용을 그 헤더 파일에 추가해 주세요.

#ifndef HASHMAP\_H  
#define HASHMAP\_H

#include "node.h"

#define TRUE (1)  
#define FALSE (0)

typedef struct hashmap {  
 size\_t (\*hash\_func)(const char\* key);  
 node\_t\*\* plist;  
 size\_t length;  
} hashmap\_t;

hashmap\_t\* init\_hashmap\_malloc(size\_t length, size\_t (\*p\_hash\_func)(const char\* key));

int add\_key(hashmap\_t\* hashmap, const char\* key, const int value);

int get\_value(const hashmap\_t\* hashmap, const char\* key);

int update\_value(hashmap\_t\* hashmap, const char\* key, int value);

int remove\_key(hashmap\_t\* hashmap, const char\* key);

void destroy(hashmap\_t\* hashmap);

#endif */\* HASHMAP\_H \*/*

1. hashmap.c 파일을 폴더에 추가합니다.
2. 헤더 파일에 선언되어 있는 모든 함수를 위 .c 파일 안에 추가합니다.

2. hashmap.h 안에서 선언된 함수들 구현하기

전반적인 규칙

* 현재 hashmap.h 안에 선언되어 있는 모든 함수를 구현해야 합니다.
* 이미 헤더 파일에 들어있는 전역 함수들의 함수 시그니처를 변경하는 것을 금합니다.
* 필요에 따라 함수를 추가해도 됩니다. 이미 헤더 파일에 제공된 것들을 변경하거나 지울 수 없을 뿐입니다.
* 이 과제에서 전역(global) 변수와 정적(static) 변수의 **사용을 금합니다**. 사용하면 빌드봇이 감점을 하니 시도조차 하지 마세요. :)
* hashmap.h 안에 선언된 함수들이 받는 포인터형 매개변수에 NULL이 들어오지 않는다고 가정하셔도 좋습니다.
* key에 대응하는 값(value)은 언제나 0보다 같거나 크다고 가정하셔도 좋습니다

2.1 init\_hashmap\_malloc() 함수를 구현한다

* init\_hashmap\_malloc() 함수는 다음의 인자를 받습니다.
  + hashmap->plist 배열의 길이: size\_t length
  + 해시 함수의 함수 포인터: size\_t (\*p\_hash\_func)(const char\* key)
* 이 함수는 동적으로 할당된 hashmap\_t 구조체를 반환합니다.
* hashmap->plist의 모든 요소는 NULL로 초기화돼야 합니다.

*/\* hash\_function 구현이 어딘가에 있다고 가정... \*/*  
hashmap\_t\* hashmap = init\_hashmap\_malloc(100, hash\_function);

2.2 add\_key() 함수를 구현한다

* add\_key() 함수는 다음의 인자를 받습니다.
  + 해시 맵 포인터: hashmap\_t\* hashmap
  + 키(key)로 사용할 문자열: const char\* key
  + key에 대응하는 값(value): const int value
* 지정된 키와 값을 hashmap에 추가합니다.
* 이 키/값 쌍을 hashmap에 성공적으로 추가했다면 TRUE(1)를, 아니라면 FALSE(0)를 반환합니다.

*/\* hash\_function 구현이 어딘가에 있다고 가정... \*/*  
hashmap\_t\* hashmap = init\_hashmap\_malloc(100, hash\_function);

add\_key(hashmap, "key1", 10); */\* TRUE \*/*  
add\_key(hashmap, "key1", 13); */\* FALSE \*/*  
add\_key(hashmap, "key1", 10); */\* FALSE \*/*

2.3 get\_value() 함수를 구현한다

* get\_value() 함수는 다음의 인자를 받습니다.
  + 해시 맵 포인터: const hashmap\_t\* hashmap
  + 키로 사용할 문자열: const char\* key
* key에 대응하는 값을 반환합니다.
* hashmap이 지정된 키에 대응하는 요소를 가지고 있지 않다면 -1을 반환합니다.

*/\* hash\_function 구현이 어딘가에 있다고 가정... \*/*  
hashmap\_t\* hashmap = init\_hashmap\_malloc(100, hash\_function);

add\_key(hashmap, "key1", 21);

get\_value(hashmap, "key1"); */\* 21 \*/*

2.4 update\_value() 함수를 구현한다

* update\_value() 함수는 다음의 인자를 받습니다.
  + 해시 맵 포인터: hashmap\_t\* hashmap
  + 키로 사용할 문자열: const char\* key
  + key에 대응하는 값: const int value
* 지정된 키의 값을 업데이트합니다.
* 지정된 키에 대응하는 값을 성공적으로 업데이트했다면 TRUE(1)를, 아니라면 FALSE(0)를 반환합니다.

*/\* hash\_function 구현이 어딘가에 있다고 가정... \*/*  
hashmap\_t\* hashmap = init\_hashmap\_malloc(100, hash\_function);

update\_value(hashmap, "key1", 12); */\* FALSE \*/*

get\_value(hashmap, "key1"); */\* -1 \*/*

add\_key(hashmap, "key1", 10);

update\_value(hashmap, "key1", 12); */\* TRUE \*/*

get\_value(hashmap, "key1"); */\* 12 \*/*

2.5 remove\_key() 함수를 구현한다

* remove\_key() 함수는 다음의 인자를 받습니다.
  + 해쉬 맵 포인터: hashmap\_t\* hashmap
  + 제거할 요소의 키: const char\* key
* hashmap에서 key에 대응하는 값을 찾아 제거합니다.
* 이 키/값 쌍을 hashmap에서 성공적으로 제거했다면 TRUE(1)를 아니라면 FALSE(0)를 반환합니다.

*/\* hash\_function 구현이 어딘가에 있다고 가정... \*/*  
hashmap\_t\* hashmap = init\_hashmap\_malloc(100, hash\_function);

remove\_key(hashmap, "key1"); */\* FALSE \*/*

add\_key(hashmap, "key1", 98);

remove\_key(hashmap, "key1"); */\* TRUE \*/*

2.6 destroy() 함수를 구현한다

* destroy() 함수는 다음의 인자를 받습니다.
  + 해시 맵 포인터: hashmap\_t\* hashmap
* hashmap이 사용 중인 모든 동적 메모리 할당을 해제합니다.

*/\* hash\_function 구현이 어딘가에 있다고 가정... \*/*  
hashmap\_t\* hashmap = init\_hashmap\_malloc(100, hash\_function);

*/\* hashmap을 사용해서 이런저런 일을 함 \*/*

destroy(hashmap);

3. 본인 컴퓨터에서 테스트하는 법

* main.c 파일을 아래의 예처럼 고쳐주세요. 컴파일을 할 때는 이전 실습에서 사용했던 것과 같은 옵션을 사용합니다.

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

#define DEFAULT\_ARRAY\_LENGTH (20)

static size\_t hash\_function(const char\* key);

int main(void)  
{  
 size\_t i = 0;  
 hashmap\_t\* hashmap = NULL;

hashmap = init\_hashmap\_malloc(DEFAULT\_ARRAY\_LENGTH, hash\_function);

for (i = 0; i < 100; i++) {  
 char key[100];  
 int value = (int)i;  
 int c;  
 int dummy = 512;

sprintf(key, "key%u", i);

assert(add\_key(hashmap, key, value) == TRUE);

c = get\_value(hashmap, key);  
 assert(c == value);

assert(add\_key(hashmap, key, dummy) == FALSE);

c = get\_value(hashmap, key);  
 assert(c == value);  
 }

for (i = 0; i < 100; i++) {  
 char key[100];  
 int value = (int)(i \* i \* i);  
 int c;

sprintf(key, "key%u", i);

assert(update\_value(hashmap, key, value) == TRUE);  
 c = get\_value(hashmap, key);

assert(c == value);  
 }

for (i = 0; i < 100; i++) {  
 char key[100];  
 int c;

sprintf(key, "key%u", i);

assert(remove\_key(hashmap, key) == TRUE);  
 c = get\_value(hashmap, key);

assert(c == -1);

assert(remove\_key(hashmap, key) == FALSE);  
 }

for (i = 0; i < DEFAULT\_ARRAY\_LENGTH; i++) {  
 assert((hashmap->plist)[i] == NULL);  
 }

destroy(hashmap);

return 0;  
}

static size\_t hash\_function(const char\* key)  
{  
 size\_t code = 0;  
 const char\* c = key;

while (\*c != '\0') {  
 code += \*c++;  
 }

return code;  
}

* 위에 해시 함수의 예를 하나 제공해 드린 게 보이시죠? 이런 해시 함수는 해시 충돌을 많이 일으키기 때문에 보통 잘 쓰지 않죠. 하지만 훌륭한 해시 함수에 대한 이야기는 그 자체로 방대한 주제라 여기서는 자세히 다루지 않습니다. :)

4. 커밋, 푸시 그리고 빌드 요청

이건 어떻게 하는지 이제 다 아시죠? :)

*출처: <*[*https://pocu.academy/ko/MyPOCU/COMP2200/202009/Assignment4*](https://pocu.academy/ko/MyPOCU/COMP2200/202009/Assignment4)*>*

5. 피드백

\* Dangling Pointer를 유의하자.

\* 문자열 포인터에 담긴 문자열은 얕은 복사를 해주자.