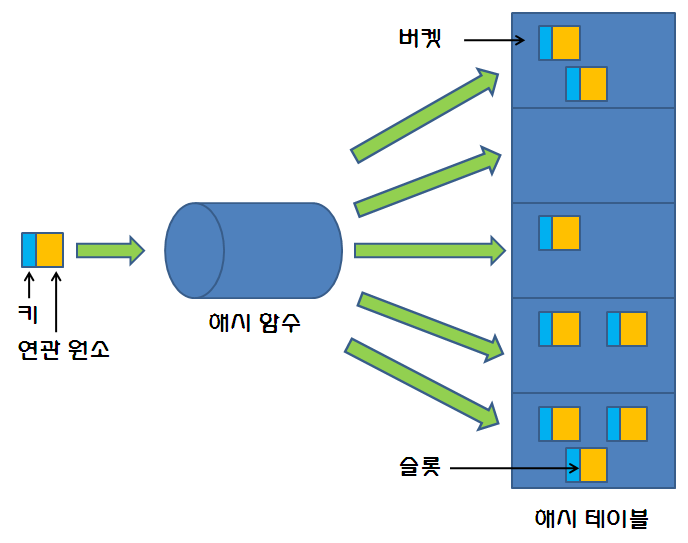
**———————————————————————————————————————————————————**

[1] 기초 문제

———————————————————————————————————————————————————

1. **다음 그림에서 찾을 수 있는 해시의 용어들에 어떤 것이 있는지 나열하시오.**



1. **해시 용어와 관련된 문제다. 빈 칸에 들어가는 단어를 맞춰보도록 한다.**
2. ① 해시 함수 ② 키 ③ 해시 테이블 ④ 버켓
3. ① 키 ② 사전 ③ 해시 테이블 ④ 버켓
4. ① 해시 테이블 ② 버켓 ③ 사전 쌍
5. ① 버켓 ② 슬롯 ③ 배열 ④ 리스트
6. **해시를 사용하고 있는 곳을 세 가지씩만 나열해 보자.**
7. 매장 전시, 편지 분류, 집 주소, 도서 분류, 주민등록번호
8. 맞춤법 검사, 전화 번호, 방문 웹 페이지, 파일 인증, 안전한 해시, 인증 번호

|  |
| --- |
| 해답  정답으로 간주할 수 있는 것들이 너무 많아서 이번 장에서 설명한 것만 실었습니다. |

1. **좋은 해시의 전제 조건은 좋은 해시 함수다. 아래 해시 함수들을 간단하게 설명해 보자.**

|  |
| --- |
| 해답  자세한 설명이 2장의 심화 학습에 있습니다. 설명이 너무 길어 답안은 생략했습니다. |

1. **해시를 구현하는 대표적인 두 가지 방법으로 개방 주소법과 체인법이 있다. 빈 칸에 들어가는 내용을 단답형으로 채워보자.**

|  |  |  |
| --- | --- | --- |
|  | 개방 주소법 | 체인법 |
| 메모리 구성 | 배열 | 리스트 |
| 오버플로우 | 빈 슬롯 검색 | 노드 추가 |
| 방법 | 선형, 이차, 재해싱, 임의 | 없다 |
| 성능 | 나쁘다 | 좋다 |
| 충돌 | 있다 | 없다 |

|  |
| --- |
| 해답  표에 대한 완전한 설명은 1장의 해시의 종류에 있습니다. 설명이 너무 길어 기본적인 답안만 제공합니다. |

1. **본문에 나왔던 코드에 대해 설명을 하는 곳이다. 해시를 배우려는 친구에게 설명한다는 생각으로 차분하게 기술해 보자.**

|  |
| --- |
| 해답  자세한 설명이 2장의 심화 학습에 있습니다. 설명이 너무 길어 답안은 생략했습니다. |

**———————————————————————————————————————————————————**

[2] 기본 실습 문제

———————————————————————————————————————————————————

1. **해시에서 나올 수 있는 코드들이다. 아무리 봐도 해시에는 어려운 코드가 없다. 직접 구현해 보자.**
2. unsigned getHashCode(const char\* s)

{

unsigned sum = 0;

while(\*s)

sum += \*s++;

return sum%BUCKET\_COUNT;

}

1. void insertKey(const char\* s)

{

BUCKET\* pb = g\_hash+getHashCode(s);

strcpy(pb->bucket[pb->count], s);

pb->count++;

}

1. **본문에서 나오진 않았지만 나올 수도 있었던 간단한 함수들이다. 구현해 보자.**
2. 앞에서는 항상 이 함수를 구현했었는데, 해시에서는 남겨 두었었습니다. 모든 버켓이 비었을 때, 해시가 비었다라고 얘기합니다.

int isEmpty(void)

{

int i;

for(i = 0; i < BUCKET\_COUNT; i++)

{

if(g\_hash[i].count != 0)

return 0;

}

return 1;

}

1. 정렬되지 않았으므로, 전체 키에 대해 반복합니다. 비어있는 해시에 대한 동작은 에러이므로 앞에서 만든 함수로 검사합니다. max 초기값으로 문자열이 가질 수 있는 최소값인 null 문자열을 주었습니다. 첫 번째 문자열을 가리키도록 설정하면 좋은데, 생각보다 첫 번째 문자열을 가리킨다는 것이 귀찮고 애매합니다.

const char\* getMaxKey(void)

{

if(!isEmpty())

{

int i, j;

char\* max = "";

for(i = 0; i < BUCKET\_COUNT; i++)

{

for(j = 0; j < g\_hash[i].count; j++)

{

if(strcmp(max, g\_hash[i].bucket[j]) < 0)

max = g\_hash[i].bucket[j];

}

}

return max;

}

return NULL;

}

1. **저자는 여러분이 복잡한 코드까지 직접 구현하기를 바라지는 않는다. 대신 비어있는 곳의 코드를 채울 정도는 되어야 한다고 생각한다. 빈 칸에 들어가는 코드는 무엇인가?**
   1. ① g\_hash+getHashCode(s);

② if(strcmp(pb->bucket[i], s) == 0)

③ return 0;

* 1. ① malloc(WORD\_SIZE \* pb->size \* 2);

② free(pb->bucket);

③ pb->bucket = big;

**———————————————————————————————————————————————————**

[3] 응용 실습 문제

———————————————————————————————————————————————————

1. 본문과 똑같이 다중 파일 프로젝트로 처리했습니다. 코드의 핵심은 insertKey 함수에 나오는 동적 확장입니다. 본문에서는 구현할 필요가 없었던 동적 할당을 제거하는 clear 함수가 추가됐습니다.

**// HashUnlimited.h**

void init(void);

void clear(void);

void insertKey(unsigned int key);

int deleteKey(unsigned int key);

unsigned getHashCode(unsigned int key);

void printHash(void);

void printBucket(unsigned index);

**// HashUnlimited.c**

#pragma warning(disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "Hash.h"

typedef struct \_BUCKET

{

int used, size; // used 사용 중인 개수 size 할당 크기

unsigned\* bucket; // 동적 배열

}

BUCKET;

static int g\_bucket; // 해시 테이블 크기

static BUCKET\* g\_hash; // 동적 배열

void init(void)

{

int i;

g\_bucket = rand()%6 + 5; // 5에서 10

g\_hash = calloc(g\_bucket, sizeof(BUCKET)); // 최초 할당

for(i = 0; i < g\_bucket; i++)

{

g\_hash[i].size = 1;

g\_hash[i].bucket = malloc(sizeof(int)\*1); // 버켓별 할당

}

}

void clear(void)

{

int i;

for(i = 0; i < g\_bucket; i++)

free(g\_hash[i].bucket);

free(g\_hash);

}

unsigned getHashCode(unsigned int key) // 제대로 된 해시 함수는 아닙니다

{

int sum = 0;

key \*= 142857; // 소수를 사용해서 큰 수로 확장

do

{

sum += key / g\_bucket; // 해시 테이블에 맞춘 진법 적용

}

while(key /= g\_bucket);

return sum%g\_bucket;

}

void insertKey(unsigned int key)

{

BUCKET\* pb = g\_hash+getHashCode(key);

if(pb->used >= pb->size) // 슬롯 부족

{

unsigned\* temp = malloc(sizeof(int)\*pb->size\*2); // 확장, 복사,해제

memcpy(temp, pb->bucket, sizeof(int)\*pb->size);

free(pb->bucket);

pb->bucket = temp;

pb->size \*= 2;

}

pb->bucket[pb->used++] = key;

}

int deleteKey(unsigned int key)

{

BUCKET\* pb = g\_hash+getHashCode(key);

int i;

for(i = 0; i < pb->used; i++)

{

if(pb->bucket[i] == key)

{

pb->bucket[i] = pb->bucket[--pb->used];

return 1;

}

}

return 0;

}

void printBucket(unsigned index)

{

BUCKET\* pb = g\_hash+index;

int i;

printf("%d[%2d of %2d] : ", index, pb->used, pb->size);

for(i = 0; i < pb->used; i++)

printf("%d ", pb->bucket[i]);

printf("\n");

}

void printHash(void)

{

int i;

for(i = 0; i < g\_bucket; i++)

printBucket(i);

printf("\n");

}

**// HashUnlimitedMain.c**

#pragma warning(disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include "Hash.h"

enum { INPUT\_COUNT = 50 }; // 해시에 넣은 데이터 개수

void main(void)

{

int i, key;

int remove[INPUT\_COUNT/5] = { 0 }; // 삭제할 데이터 저장

int\* pr = remove;

init();

printf("[추가]\n");

for(i = 0; i < INPUT\_COUNT; i++)

{

insertKey(key = rand()); // 추가

printf("%5d ", key);

if(i%5 == 4)

\*pr++ = key;

if(i%8 == 7)

printf("\n");

}

printf("\n\n");

printf("[해시 원본]\n");

printHash();

printf("[삭제]\n");

for(i = 0; i < sizeof(remove)/sizeof(remove[0]); i++)

{

printf("%d[%d] ", remove[i], getHashCode(remove[i]));

deleteKey(remove[i]); // 삭제

}

printf("\n\n");

printf("[해시 결과]\n");

printHash();

clear();

}

**// 출력 결과**

[추가]

18467 6334 26500 19169 15724 11478 29358 26962

24464 5705 28145 23281 16827 9961 491 2995

11942 4827 5436 32391 14604 3902 153 292

12382 17421 18716 19718 19895 5447 21726 14771

11538 1869 19912 25667 26299 17035 9894 28703

23811 31322 30333 17673 4664 15141 7711 28253

6868 25547

[해시 원본]

0[ 6 of 8] : 2995 11942 12382 14771 19912 7711

1[ 4 of 4] : 32391 1869 26299 31322

2[ 5 of 8] : 19169 26962 28145 3902 19895

3[ 2 of 2] : 5436 292

4[ 2 of 2] : 18716 25547

5[ 4 of 4] : 4827 21726 17673 28253

6[ 7 of 8] : 26500 24464 16827 153 5447 25667 28703

7[ 9 of 16] : 18467 6334 15724 9961 14604 17421 30333 4664 15141

8[ 3 of 4] : 29358 23281 9894

9[ 8 of 8] : 11478 5705 491 19718 11538 17035 23811 6868

[삭제]

15724[7] 5705[9] 491[9] 32391[1] 12382[0] 5447[6] 19912[0] 28703[6] 4664[7] 25547[4]

[해시 결과]

0[ 4 of 8] : 2995 11942 7711 14771

1[ 3 of 4] : 31322 1869 26299

2[ 5 of 8] : 19169 26962 28145 3902 19895

3[ 2 of 2] : 5436 292

4[ 1 of 2] : 18716

5[ 4 of 4] : 4827 21726 17673 28253

6[ 5 of 8] : 26500 24464 16827 153 25667

7[ 7 of 16] : 18467 6334 15141 9961 14604 17421 30333

8[ 3 of 4] : 29358 23281 9894

9[ 6 of 8] : 11478 6868 23811 19718 11538 17035

1. 본문과 똑같이 다중 파일 프로젝트로 처리했습니다. 코드의 핵심은 구조체 배열을 다루는 부분과 해시 함수입니다. 특히 두 개의 멤버를 조합해서 어떻게 키를 구성하는지가 중요합니다.

**// HashStruct.h**

typedef struct \_INFO

{

char name[32];

int age;

char addr[256];

}

INFO;

void init(void);

void insertData(const INFO\* pInfo);

int deleteData(const INFO\* pInfo);

unsigned getHashCode(const INFO\* pInfo);

void printHash(void);

void printBucket(unsigned index);

**// HashStruct.c**

#pragma warning(disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "Hash.h"

enum { BUCKET\_COUNT = 7, BUCKET\_SIZE = 512 };

typedef struct \_BUCKET

{

int count;

INFO bucket[BUCKET\_SIZE];

}

BUCKET;

static BUCKET g\_hash[BUCKET\_COUNT];

void init(void)

{

int i;

for(i = 0; i < BUCKET\_COUNT; i++)

g\_hash[i].count = 0;

}

unsigned getHashCode(const INFO\* pInfo) // 해시 함수. 매개 변수가 구조체

{

unsigned sum = 0;

const char\* key = pInfo->name; // 이름 키

while(\*key)

sum += \*key++;

key = pInfo->addr; // 주소 키

while(\*key)

sum += \*key++;

return sum%BUCKET\_COUNT;

}

void insertData(const INFO\* pInfo)

{

BUCKET\* pb = g\_hash+getHashCode(pInfo);

pb->bucket[pb->count++] = \*pInfo;

}

int deleteData(const INFO\* pInfo)

{

BUCKET\* pb = g\_hash+getHashCode(pInfo);

int i;

for(i = 0; i < pb->count; i++)

{

if( strcmp(pb->bucket[i].name, pInfo->name) == 0 && // 이름과 주소가 모두 일치

strcmp(pb->bucket[i].addr, pInfo->addr) == 0)

{

pb->bucket[i] = pb->bucket[--pb->count];

return 1;

}

}

return 0;

}

void printBucket(unsigned index)

{

BUCKET\* pb = g\_hash+index;

int i;

printf("%d[%2d] : ", index, pb->count);

for(i = 0; i < pb->count; i++)

printf("[%s %d %s] ", pb->bucket[i].name, pb->bucket[i].age, pb->bucket[i].addr);

printf("\n");

}

void printHash(void)

{

int i;

for(i = 0; i < BUCKET\_COUNT; i++)

printBucket(i);

printf("\n");

}

**// HashStructMain.c**

#pragma warning(disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "Hash.h"

enum { INPUT\_COUNT = 20 };

void main(void)

{

// 이름과 주소는 각각 10개씩

const char\* names[] = { "kim", "lee", "cho", "han", "cha", "seo", "min", "nam", "woo", "jin" };

const char\* addrs[] = { "seoul", "cheju", "pusan", "ilsan", "dokdo", "yeosu", "mokpo", "taean", "daegu", "suwon" };

INFO info;

INFO remove[INPUT\_COUNT/5] = { 0 }; // 삭제할 데이터

INFO\* pr = remove;

int i;

init();

printf("[추가]\n");

for(i = 0; i < INPUT\_COUNT; i++)

{

strcpy(info.name, names[rand()%10]);

strcpy(info.addr, addrs[rand()%10]);

info.age = rand()%90 + 10;

insertData(&info); // 추가

printf("[%s %d %s] ", info.name, info.age, info.addr);

if(i%5 == 4)

\*pr++ = info, printf("\n");

}

printf("\n");

printf("[해시 원본]\n");

printHash();

printf("[삭제]\n");

for(i = 0; i < sizeof(remove)/sizeof(remove[0]); i++)

{

printf("[%d : %s %d %s] ", getHashCode(remove+i), remove[i].name, remove[i].age, remove[i].addr);

deleteData(remove+i); // 삭제

}

printf("\n\n");

printf("[해시 결과]\n");

printHash();

}

**// 출력 결과**

[추가]

[lee 44 taean] [kim 74 suwon] [woo 62 daegu] [cha 75 yeosu] [lee 71 taean]

[lee 72 yeosu] [nam 91 mokpo] [cha 73 pusan] [cho 61 pusan] [min 15 daegu]

[nam 21 mokpo] [woo 32 suwon] [nam 35 suwon] [cha 61 ilsan] [cho 43 ilsan]

[cha 71 cheju] [han 87 daegu] [cha 97 pusan] [nam 93 suwon] [lee 68 suwon]

[해시 원본]

0[ 2] : [lee 72 yeosu] [lee 68 suwon]

1[ 1] : [cha 71 cheju]

2[ 3] : [min 15 daegu] [cha 61 ilsan] [cho 43 ilsan]

3[ 2] : [woo 32 suwon] [han 87 daegu]

4[ 5] : [kim 74 suwon] [cha 75 yeosu] [cha 73 pusan] [cho 61 pusan] [cha 97 pusan]

5[ 5] : [lee 44 taean] [woo 62 daegu] [lee 71 taean] [nam 91 mokpo] [nam 21 mokpo]

6[ 2] : [nam 35 suwon] [nam 93 suwon]

[삭제]

[5 : lee 71 taean] [2 : min 15 daegu] [2 : cho 43 ilsan] [0 : lee 68 suwon]

[해시 결과]

0[ 1] : [lee 68 suwon]

1[ 1] : [cha 71 cheju]

2[ 1] : [cha 61 ilsan]

3[ 2] : [woo 32 suwon] [han 87 daegu]

4[ 5] : [kim 74 suwon] [cha 75 yeosu] [cha 73 pusan] [cho 61 pusan] [cha 97 pusan]

5[ 4] : [nam 21 mokpo] [woo 62 daegu] [lee 71 taean] [nam 91 mokpo]

6[ 2] : [nam 35 suwon] [nam 93 suwon]