실습11- Hashing

실습11 Hash Table을 이용한 자료검색

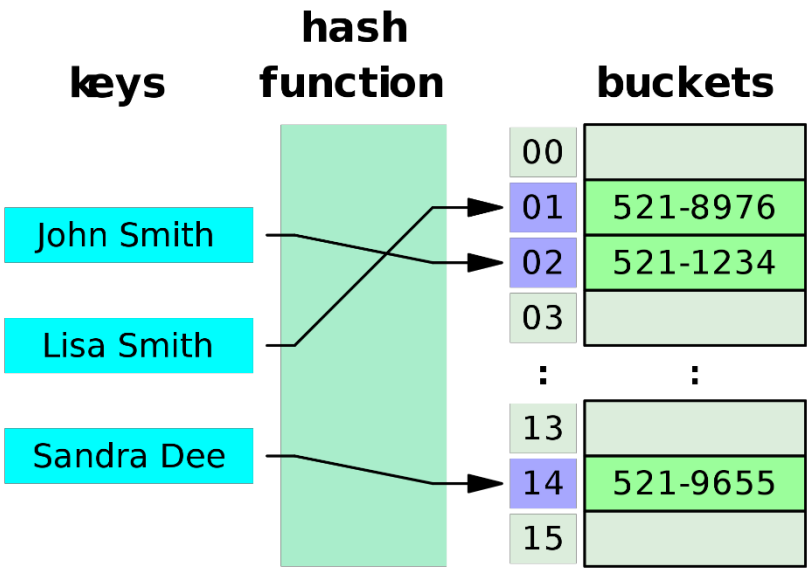
The Index is like this.

INDEX

실습 목적

- **실습 목적**
  - Hash의 특성을 이해한다.
  - Hash를 활용하여 자료를 저장하고, 저장된 자료를 검색한다.

- **해시(Hash)**
  - 키 값에 직접 산술적인 연산을 적용하여(해시 함수), 항목이 저장되어 있는 테이블의 주소를 계산하여 항목에 접근
  - 값의 연산에 의해 직접 접근이 가능한 구조를 해시 테이블(hash table)이라 부르며, 해시테이블을 이용한 탐색을 해싱(hashing)



실습 목적

- **실습 목적**
  - Hash의 특성을 이해한다.
  - Hash를 활용하여 자료를 저장하고, 저장된 자료를 검색한다.

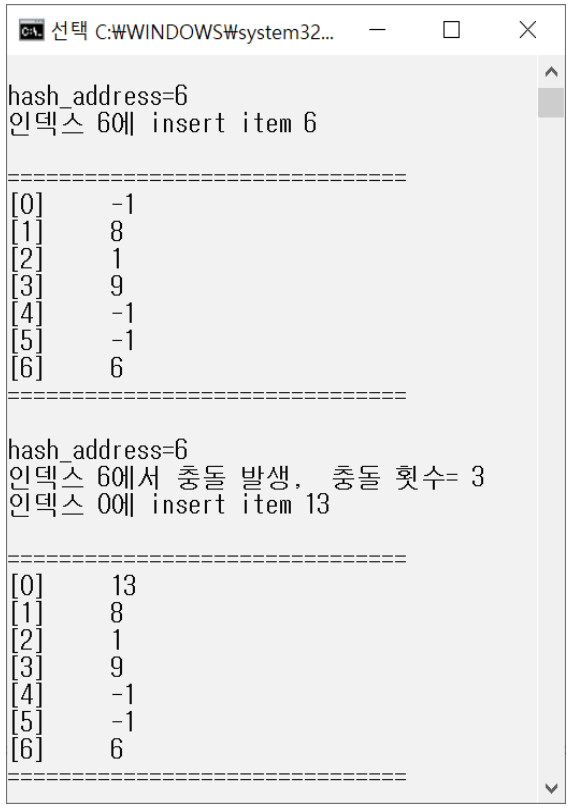
- **충돌 해결책**
  - 선형 조사법(linear probing)
    - 충돌이 ht[k]에서 충돌이 발생했다면 ht[k+1]이 비어 있는지를 조사, 만약 비어있지 않다면 ht[k+2]를 살펴본다.
    - 비어있는 공간이 나올 때까지 계속하는 조사하는 방법, 테이블의 끝에 도달하게 되면 다시 테이블의 처음부터 조사
    - 조사를 시작했던 곳으로 다시 되돌아오게 되면 테이블이 가득 찬 것임
    - 조사되는 위치 h(k), h(k)+1, h(k)+2,...
    - 아래는 크기가 7인 해시테이블에 h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13, 7, 5, 2을 삽입할 때에의 선형 조사법에 의한 충돌 처리 예시

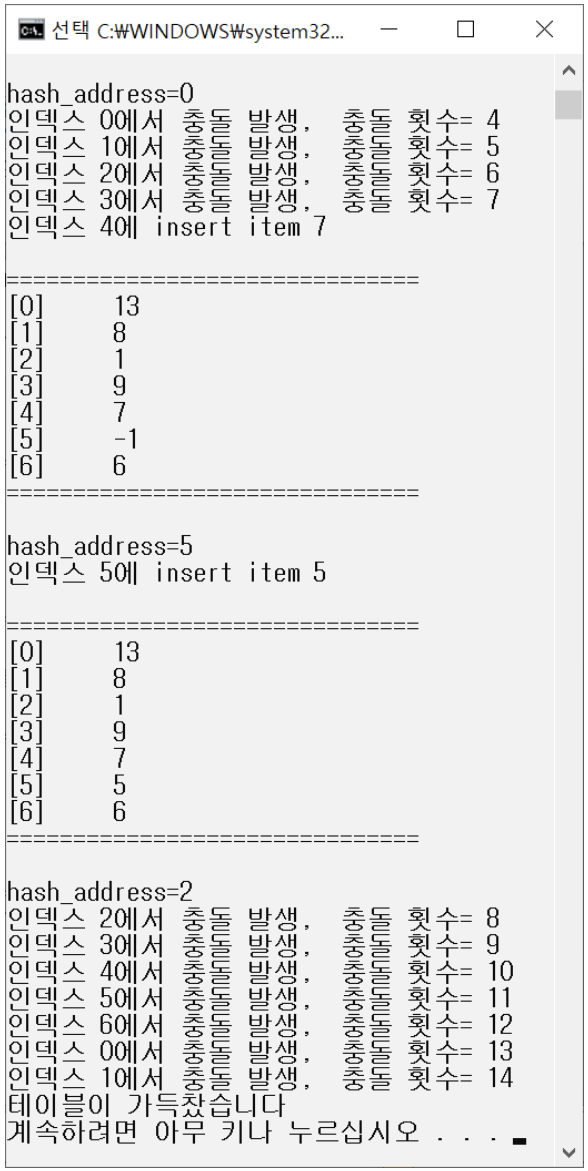
1단계 (8) : ·$h(8) = 8 \bmod 7 = 1$(저장)
2단계 (1) : ·$h(1) = 1 \bmod 7 = 1$(충돌발생)
·$(h(1)+1) \bmod 7 = 2$(저장)
3단계 (9) : ·$h(9) = 9 \bmod 7 = 2$(충돌발생)
·$(h(9)+1) \bmod 7 = 3$(저장)
4단계 (6) : ·$h(6) = 6 \bmod 7 = 6$(저장)
5단계 (13) :·$h(13) = 13 \bmod 7 = 6$(충돌 발생)
6단계 (7) :·$h(7) = 7 \bmod 7 = 0$(충돌 발생)
·$(h(7)+1) \bmod 7 = 1$(충돌발생)
·$(h(7)+2) \bmod 7 = 2$(충돌발생)
·$(h(7)+3) \bmod 7 = 3$(충돌발생)
·$(h(7)+4) \bmod 7 = 4$(저장)

7단계 (5) : ·$h(5) = 5 \bmod 7 = 5$(저장)
8단계 (2) : ·$h(2) = 2 \bmod 7 = 2$ (충돌발생)
· $(h(2)+1) \bmod 7 = 3$ (충돌발생)
· $(h(2)+2) \bmod 7 = 4$ (충돌발생)
· $(h(2)+3) \bmod 7 = 5$ (충돌발생)
· $(h(2)+4) \bmod 7 = 6$ (충돌발생)
· $(h(2)+5) \bmod 7 = 0$ (충돌발생)
· $(h(2)+6) \bmod 7 = 1$ (충돌발생)
· $(h(2)+7) \bmod 7 = 2$ (테이블 포화, 종료)

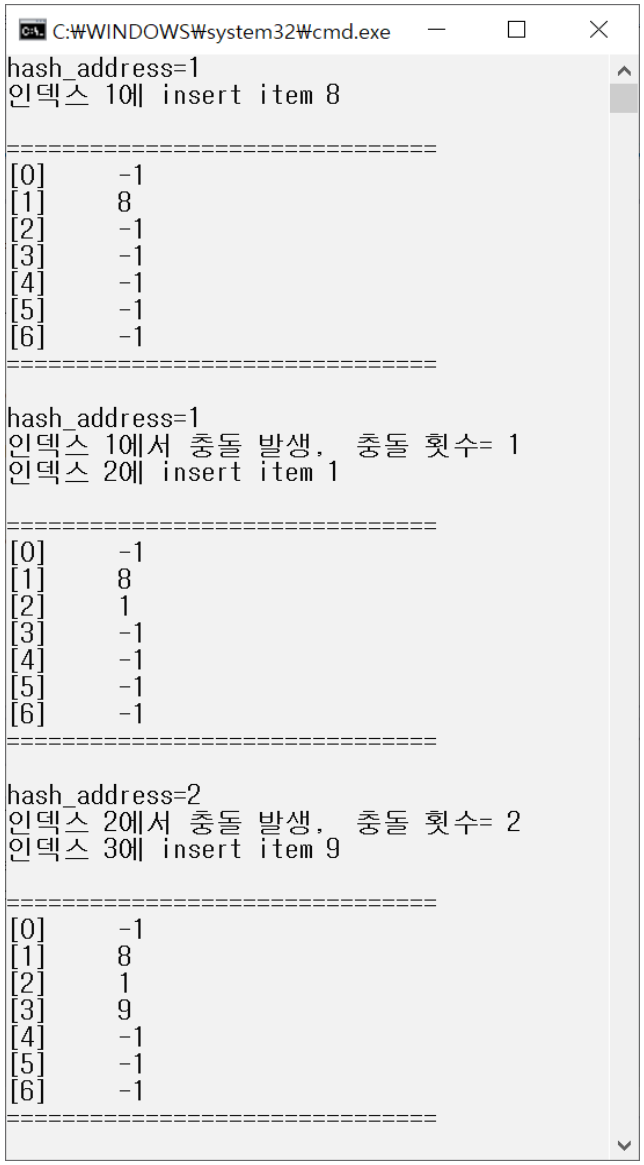
[0]	13
[1]	8
[2]	1
[3]	9
[4]	7
[5]	5
[6]	6

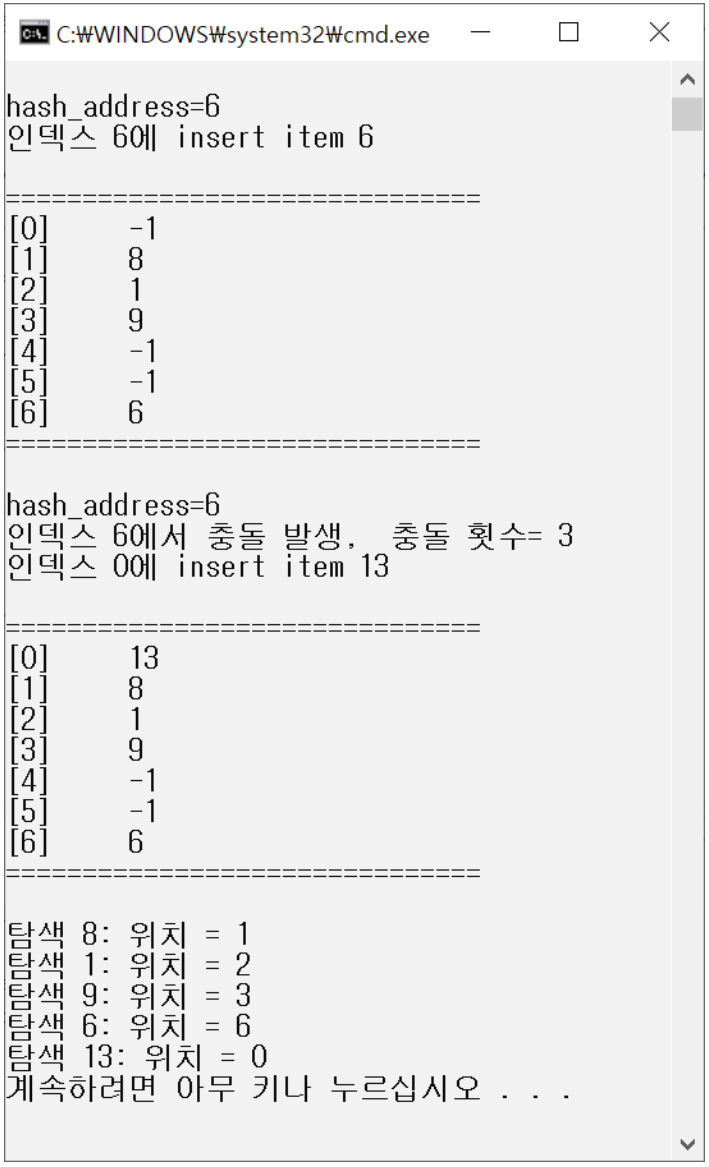
크기가 7인 해시테이블에 h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13, 7, 5, 2를 삽입할 때에의 선형 조사법에 의한 충돌 처리 출력 예시





크기가 7인 해시테이블에 h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13 을 삽입할 때에의 선형 조사법에 의한 충돌 처리 출력 예시





[0]	13
[1]	8
[2]	1
[3]	9
[4]	-1
[5]	-1
[6]	6

실습 목적

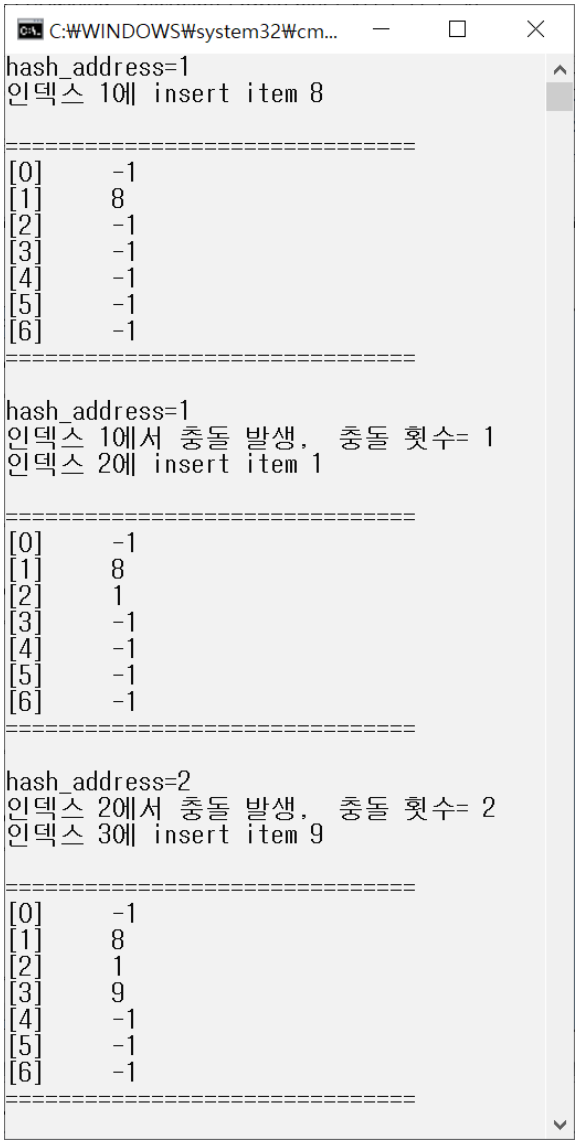
- **실습 목적**
  - Hash의 특성을 이해한다.
  - Hash를 활용하여 자료를 저장하고, 저장된 자료를 검색한다.

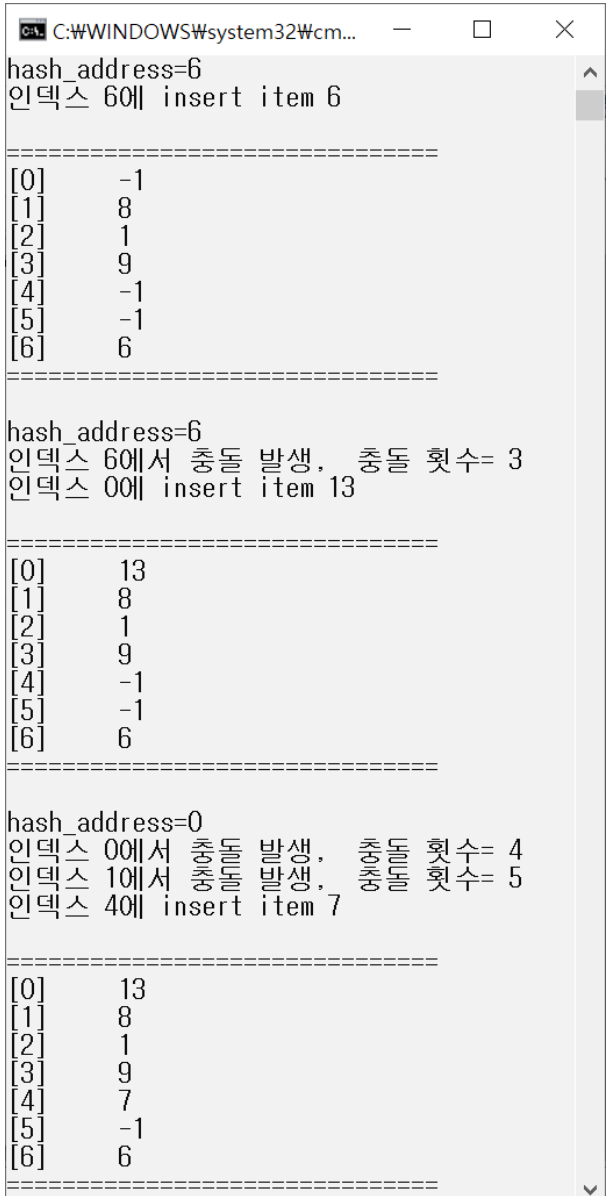
- **충돌 해결책**
  - 이차 조사법(quadratic probing)
    - 선형 조사법과 유사하지만, 다음 조사할 위치를 다음 식에 의하여 결정한다.
    - (h(k)+i*i) mod M
    - 조사되는 위치 h(k), h(k)+1, h(k)+4,…
    - 선형 조사법에서의 문제점인 군집과 결합을 크게 완화
      - h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13, 7, 5, 1 을 삽입할 때에 이차 조사법에 의한 충돌 처리 예시

1단계 (8) : ·$h(8) = 8 \bmod 7 = 1$(저장)
2단계 (1) : ·$h(1) = 1 \bmod 7 = 1$(충돌발생)
·$(h(1)+1*1) \bmod 7 = 2$(저장)
3단계 (9) : ·$h(9) = 9 \bmod 7 = 2$(충돌발생)
·$(h(9)+1*1) \bmod 7 = 3$(저장)
4단계 (6) : ·$h(6) = 6 \bmod 7 = 6$(저장)
5단계 (13) :·$h(13) = 13 \bmod 7 = 6$(충돌 발생)
6단계 (7) :·$h(7) = 7 \bmod 7 = 0$(충돌 발생)
·$(h(7)+1*1) \bmod 7 = 1$(충돌발생)
·$(h(7)+2*2) \bmod 7 = 4$(저장)
7단계 (5) : ·$h(5) = 5 \bmod 7 = 5$(저장)
8단계 (1) : ·$h(1) = 1 \bmod 7 = 1$ (충돌 발생)
·$(h(1)+1*1) \bmod 7 = 2$(탐색키 중복, 종료)

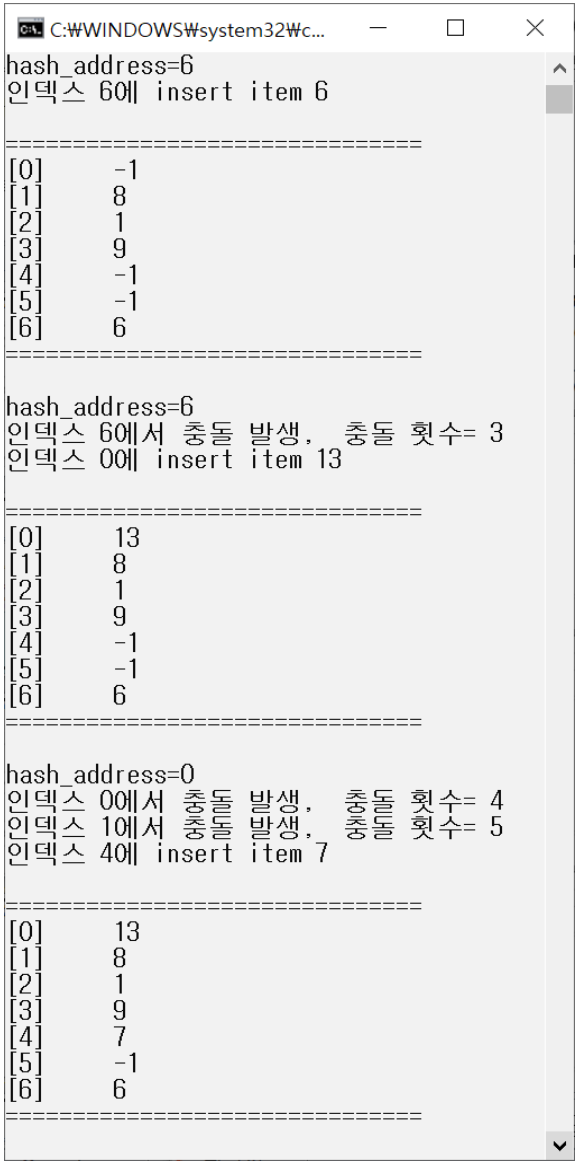
[0]	13
[1]	8
[2]	1
[3]	9
[4]	7
[5]	5
[6]	6

크기가 7인 해시테이블에 h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13, 7, 5, 1을 삽입할 때에 이차 조사법에 의한 충돌 처리 출력 예시





크기가 7인 해시테이블에 h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13, 7, 5을 삽입할 때에의 이차 조사법에 의한 충돌 처리 출력 예시

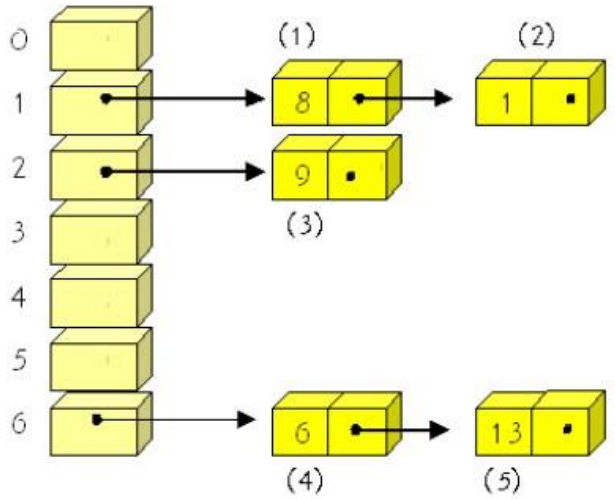


실습 목적

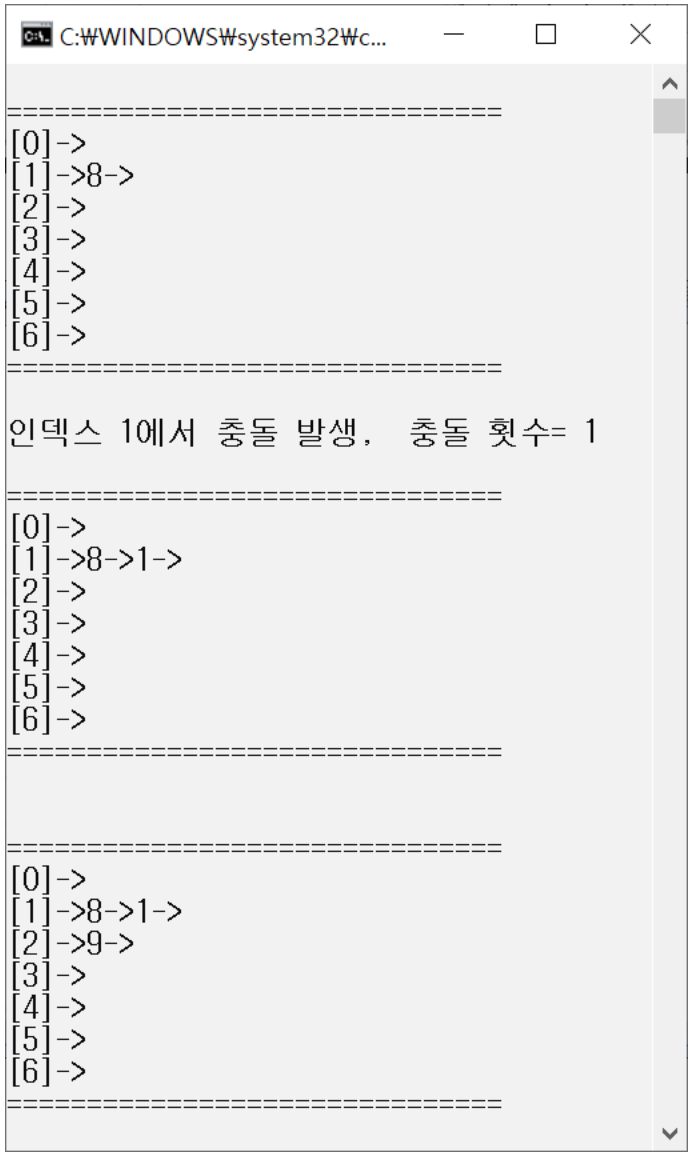
- **실습 목적**
  - Hash의 특성을 이해한다.
  - Hash를 활용하여 자료를 저장하고, 저장된 자료를 검색한다.

- **충돌 해결책**
  - 체이닝
    - 해시테이블의 하나의 위치가 여러 개의 항목을 저장하는 구조
    - 각 버켓에 고정된 슬롯을 할당하는 것이 아니라 각 버켓에, 삽입과 삭제가 용이한 연결 리스트를 할당
    - 버켓 내에서는 원하는 항목을 찾을 때는 연결 리스트를 순차 탐색
    - 아래는 크기가 7인 해시테이블에 h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13 을 삽입할 때에의 체이닝에 의한 충돌 처리 예시

1단계 (8) : h(8) = 8 mod 7 = 1(저장)
2단계 (1) : h(1) = 1 mod 7 = 1(충돌발생->새로운 노드 생성 저장)
3단계 (9) : h(9) = 9 mod 7 = 2(저장)
4단계 (6) : h(6) = 6 mod 7 = 6(저장)
5단계 (13) : h(13) = 13 mod 7 = 6(충돌 발생->새로운 노드 생성 저장)



크기가 7인 해시테이블에 h(k)=k mod 7의 해시 함수를 이용하여 8, 1, 9, 6, 13 을 삽입할 때에의 체이닝에 의한 충돌 처리 출력 예시



실습 문제

- **실습 프로그램 설명**
  - Hash Table에 h(k) 해시 함수를 이용하여 8, 1, 9, 6, 13 등의 값을 삽입할 때에의 충돌 처리 및 KEY 검색
    - 배열에 저장된 값이 key이고, 해시 함수를 이용하여 계산한 결과값이 hash_value인 Hash Table을 생성
    - key 값으로 해시 함수를 이용하여 탐색 위치를 조회
    - Hash Address은 h(k)=k % (Hash Table의 크기)
    - 선형 조사법, 이차 조사법, 체이닝을 이용하여 충돌 처리

- **구조체 struct list**

```
struct list
{
    element item;
    struct list  *link;
};
```

- 체이닝을 위한 struct list 구조체

element

```
typedef struct
{
    int key;
} element;
```

- element 구조체의 멤버를 key로 함

실습 문제

- **실습 프로그램 설명**
  - Hash Table에 h(k) 해시 함수를 이용하여 8, 1, 9, 6, 13 등의 값을 삽입할 때에의 충돌 처리 및 KEY 검색
    - 배열에 저장된 값이 key이고, 해시 함수를 이용하여 계산한 결과값이 hash_value인 Hash Table을 생성
    - key 값으로 해시 함수를 이용하여 탐색 위치를 조회
    - Hash Address은 h(k)=k % (Hash Table의 크기)
    - 선형 조사법, 이차 조사법, 체이닝을 이용하여 충돌 처리

- **함수 목록**
  - **int hash_function(int key)**
    - Hash Address를 계산하여 반환
  - **void init_table(element ht[])**
    - Hash Table을 초기화
  - **void hash_lp_add(element item, element ht[])**
  - **void hash_qp_add(element item, element ht[])**
  - **void hash_chain_add(element item, struct list *ht[])**
    - Item의 key에 따라 Hash Table에 삽입
  - **void hash_lp_search(element item, element ht[]);**
  - **void hash_qp_search(element item, element ht[]);**
  - **void hash_chain_search(element item, struct list *ht[])**
    - Item의 key 값으로 Hash Table을 조회
  - **void hash_lp_print(element ht[]);**
  - **void hash_qp_print(element ht[]);**
  - **void hash_chain_print(struct list *ht[])**
    - Hash Table의 모든 데이터를 출력

실습 문제

- **실습 문제 : 해시 테이블의 구성과 탐색**
  - 해시 테이블을 구성하는 함수와 탐색하는 함수를 이해한다.
  - 자료를 해시 테이블에 저장하고 활용해 본다.

- **구현 함수 설명**

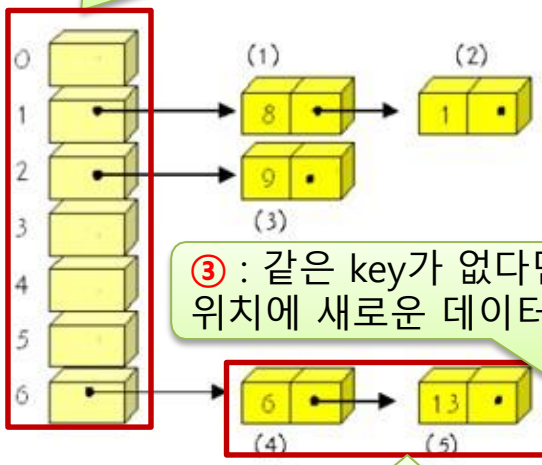
```
// 체인법을 이용하여 테이블에 키를 삽입
void hash_chain_add(element item, struct list *ht[])
{
①   int hash_value = hash_function(item.key);
    struct list *ptr;
    struct list *node_before = NULL, *node = ht[hash_value];

②

③

}
```

① : key에 맞는 Hash address를 계산하고, Hash address에 맞는 리스트의 첫 주소를 저장



③ : 같은 key가 없다면, 마지막 위치에 새로운 데이터를 추가

② : 리스트를 순차적으로 검색해 가며 같은 key가 이미 저장되어 있는지 검사

실습 문제

- **실습 문제 : 해시 테이블의 구성과 탐색**
  - 해시 테이블을 구성하는 함수와 탐색하는 함수를 이해한다.
  - 자료를 해시 테이블에 저장하고 활용해 본다.

- **구현 함수 설명**

```
// 체인법을 이용하여 테이블에 저장된 키를 탐색
void hash_chain_search(element item, struct list *ht[])
{
    struct list *node;
    int search_count = 0; // 탐색 횟수
    int hash_value = hash_function(item.key);
    for (node = ht[hash_value]; node; node = node->link) {
        search_count++;
        if (equal(item.key, node->item.key)) {
            printf("탐색 성공: key=%d, 탐색 횟수=%d\n", item.key, search_count);
            return;
        }
    }
    printf("키를 찾지 못했음\n");
}
```

① : key에 맞는 Hash address 계산

② : key에 맞는 Hash addres의 리스트를 순차적으로 방문

③ : 탐색하는 key와 저장된 item의 key가 같으면 해당 item의 key를 출력

실습 문제

- **실습 문제 : 해시 테이블의 구성과 탐색**
  - 해시 테이블을 구성하는 함수와 탐색하는 함수를 이해한다.
  - 자료를 해시 테이블에 저장하고 활용해 본다.

- **메인 함수 설명**
  - 주어진 입력의 data배열(input) 원소를 해시 함수를 이용하여 Hash Table에 삽입, 선형 조사법, 이차 조사법, 체인법으로 충돌 처리
  - Hash Table에 저장된 자료를 선형 조사법, 이차 조사법, 체인법을 이용하여 Hash Table에서 탐색한다.

```
// 해싱 테이블을 사용한 예제
int main(void)
{
    int data[SIZE] = { 8, 1, 9, 6, 13 };
    element e;

    for (int i = 0; i < SIZE; i++) {
        e.key = data[i];
        hash_chain_add(e, hash_table);
        hash_chain_print(hash_table);
    }
    for (int i = 0; i < SIZE; i++) {
        e.key = data[i];
        hash_chain_search(e, hash_table);
    }
    return 0;
}
```

Thank You

Q&A