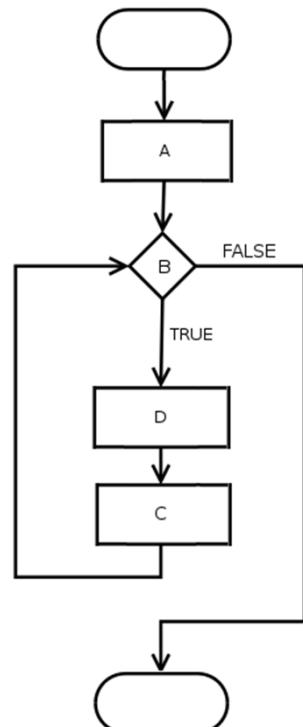


# 자료구조 및 알고리즘

```
for(A;B;C)  
D;
```



## 해시 테이블 (Hash Table)

Seo, Doo-Ok

Clickseo.com  
[clickseo@gmail.com](mailto:clickseo@gmail.com)



# 목 차



백문이불여일타(百聞而不如一打)

- 해시 테이블의 이해

- 충돌 해결



# 해시 테이블의 이해



백문이불여일타(百聞而不如一打)

- 해시 테이블의 이해

- 해시 함수

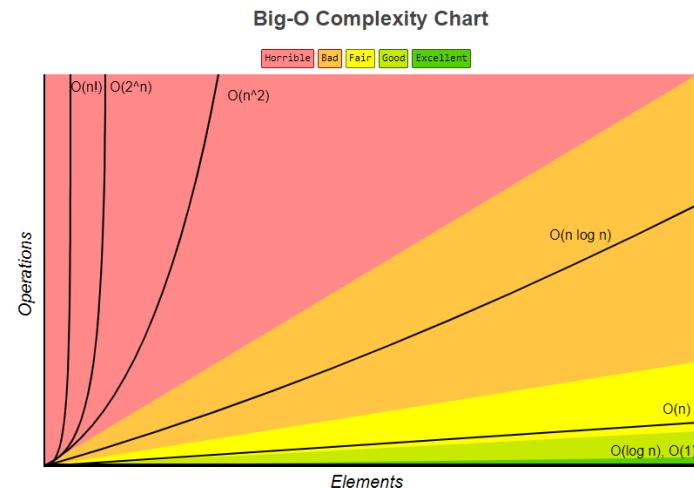
- 충돌 해결



# 검색 트리와 해시 테이블

## ● 자료 구조: 시간 복잡도

- 배열 또는 연결 리스트:  $O(N)$ , 평균  $\Theta(N)$
- 이진 검색 트리: 검색, 삽입, 삭제 시 평균  $\Theta(\log N)$ , 최악의 경우  $O(N)$
- 균형 이진 검색 트리
  - 검색, 삽입, 삭제 시 최악의 경우  $O(\log N)$
  - AVL 트리, RB 트리
- 균형 다진 검색 트리
  - 검색, 삽입, 삭제 시 최악의 경우  $O(\log N)$
  - 2-3 트리, 2-3-4 트리, B-트리
- 해시 테이블
  - 검색, 삽입, 삭제 시 평균  $O(1)$





# 해시 테이블의 이해

## 해시 테이블



입력 : 25, 13, 16, 15, 7

0	13
1	
2	15
3	16
4	
5	
6	
7	7
8	
9	
10	
11	
12	25

그림 12-1 해시 테이블 예

# 해시 테이블 (1/2)

## ● 해시 테이블(Hash Table)

### ○ 키를 변환하여 배열의 인덱스로 사용한다.

- **O(1)** : 아주 빠른 검색, 삽입, 삭제 작업을 제공한다.
  - 자신의 키값에 의해 위치가 결정된다.
  - 다른 키값과의 상대적인 크기에 의해 위치가 결정되지 않는다.
- 키를 배열의 인덱스로 그대로 사용하면 메모리 낭비가 심해질 수 있다.

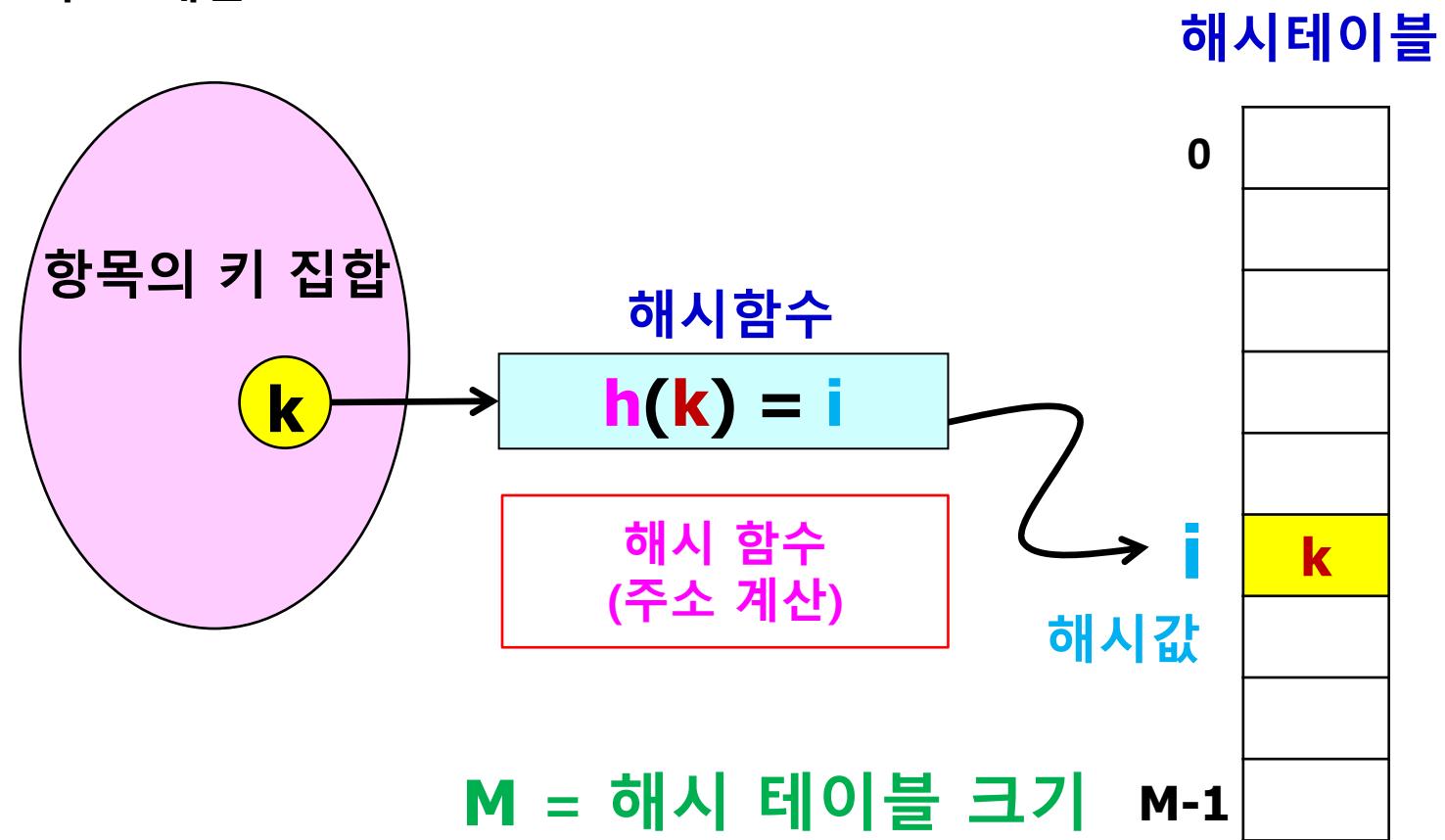
### ○ 해싱(Hashing)

- 키를 간단한 함수를 사용해 변환한 값을 배열의 인덱스로 이용하여 항목을 저장하는 것이다.
  - **해시 함수**(Hash Function): 해싱에 사용되는 함수
  - **해시 값**(Hash value) 또는 **해시주소**: 해시함수가 계산한 값
  - **해시 테이블**(Hash Table): 항목이 해시 값에 따라 저장되는 배열

# 해시 테이블 (2/2)

- 해시 테이블: 주소 계산

- 주소 계산





# 해시 테이블의 이해

## 해시 함수



# 해시 함수 (1/3)

---

## ● 해시 함수(Hash Functions)

### ○ 가장 이상적인 해시 함수

- 키들을 균등하게(Uniformly) 해시 테이블의 인덱스로 변환하는 함수
- 균등하게 변환 한다는 것은 키들을 해시 테이블에 랜덤하게 흩어지도록 저장하는 것을 뜻한다.
- 해시 함수는 키들을 균등하게 해시 테이블의 인덱스로 변환하기 위해 의미가 부여되어 있는 키를 간단한 계산을 통해 ‘뒤죽박죽’ 만든 후 해시 테이블의 크기에 맞도록 해시 값을 계산한다.
- 아무리 균등한 결과를 보장하는 해시 함수라도 함수 계산 자체에 긴 시간이 소요된다면 해싱의 장점인 연산의 신속성을 상실하므로 그 가치를 잃어버린다.

### ○ 일반적으로 키들은 부여된 의미나 특성을 가지므로 키의 가장 앞 부분 또는 뒤의 몇 자리 등을 취하여 해시 값으로 사용하는 방식의 해시함수는 많은 **충돌**을 야기시킨다.

# 해시 함수 (2/3)

---

## ● 해시 함수

### ○ 장난감 수준의 함수

- 자릿수를 선택:  $h(001364825) = 35$
- 접기(Folding)
  - $h(001364825) = 001 + 364 + 825 = 1190$
  - 해시테이블의 크기가 3이라면 1190에서 3자리 수만을 해시 값으로 이용한다.

### ○ 나누기 방법(Division Method)에 의한 함수

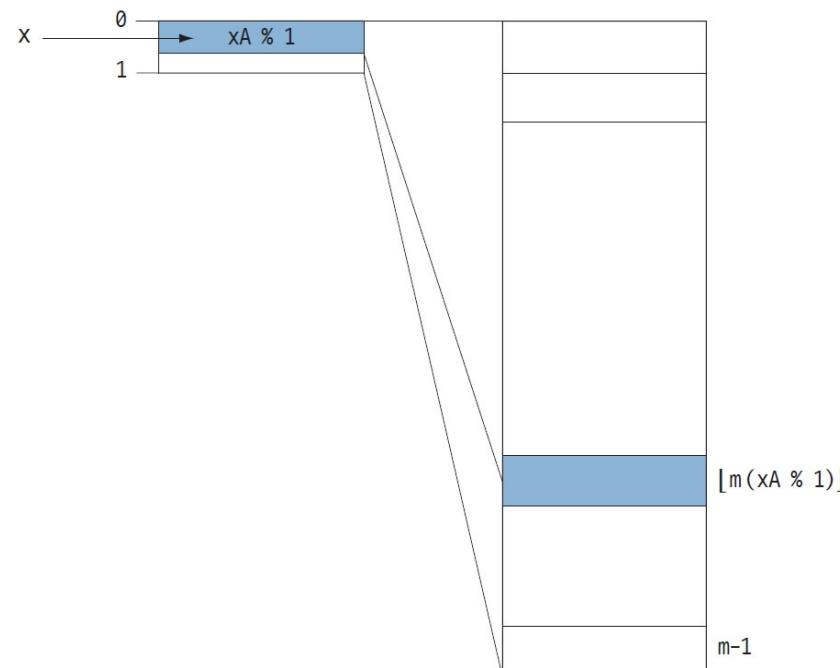
- $h(key) = key \% m \quad \leftarrow m : \text{해시 테이블 사이즈}, \% : \text{나머지 연산}$ 
  - $m$ 은 소수를 권장
    - » 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, ...

# 해시 함수 (3/3)

## ● 해시 함수

### ○ 곱하기 방법(Multiplication Method)에 의한 함수

- $h(key) = (keyA \bmod 1) * m$ 
  - $A : (0, 1)$  범위의 상수
  - $m$  이 굳이 소수일 필요 없음. 보통  $2^p$  ( $p$ 는 양의 정수).



# 충돌 해결



백문이불여일타(百聞而不如一打)

- 해시 테이블의 이해

- **충돌 해결**

- 개방 주소 방식

- 폐쇄 주소 방식



# 충돌 해결

## ● 충돌 해결(Collision Resolution)

### ○ 충돌 해결

$$h(224) = 224 \% 101 = 22$$

table[22]가  
이미 점유됨

충돌(Collision) :

어떤 깃값으로 도출된 주소에 이미 다른 키가 자리함

table[ ]

0	
⋮	
22	123
⋮	
100	

### 충돌 해결

- 일련의 해시 함수를 생성한다.
- $h_0(x)(=h(x)), h_1(x), h_2(x), h_3(x), \dots$
- 해시 테이블에서 핵심적인 부분

해시 함수:  $h(x) = x \% 101$       100



## 충돌 해결

개방형 주소 방식



# 개방형 주소 방식

## ● 개방형 주소 방식(Open Addressing Methods)

- 해시테이블 전체를 열린 공간으로 가정하고 충돌된 키를 일정한 방식에 따라서 찾아낸 empty 원소에 저장한다.
  - 선형 조사(Linear Probing)
  - 이차원 조사(Quadratic Probing)
  - 무작위 조사(Random Probing)
  - 이중 해싱(Double Hashing)

### 개방 주소 방법(Open Addressing)

: 주어진 배열 안에서 해결

선형 조사(Linear Probing)

$$h_i(x) = (h_0(x) + ai + b) \% m$$

이차원 조사(Quadratic Probing)

$$h_i(x) = (h_0(x) + ai^2 + bi + c) \% m$$

이중 해싱(Double Hashing)

$$h_i(x) = (h_0(x) + i \cdot f(x)) \% m$$

$f(x)$ : 보조 해시 함수

$m$  : 해시 테이블 크기,  $\%$  : 나머지 연산

# 개방형 주소 방식: 선형 조사 (1/10)

## ● 선형 조사(Linear probing)

- 선형 조사는 충돌이 일어난 원소에서부터 순차적으로 검색하여, 처음 발견한 empty 원소에 충돌이 일어난 키를 저장한다.
  - $h(key) = i$  : 해시테이블  $a[i], a[i+1], a[i+2], \dots, a[i+j]$  를 차례로 검색하여 처음으로 찾아낸 empty 원소에 key를 저장한다.
    - 해시테이블은 1차원 리스트이므로,  $i + j$  가  $m$  이 되면  $a[0]$  을 검색한다.
  - $(h(key) + j) \% m, j = 0, 1, 2, 3, \dots$

### 선형 조사(Linear Probing)

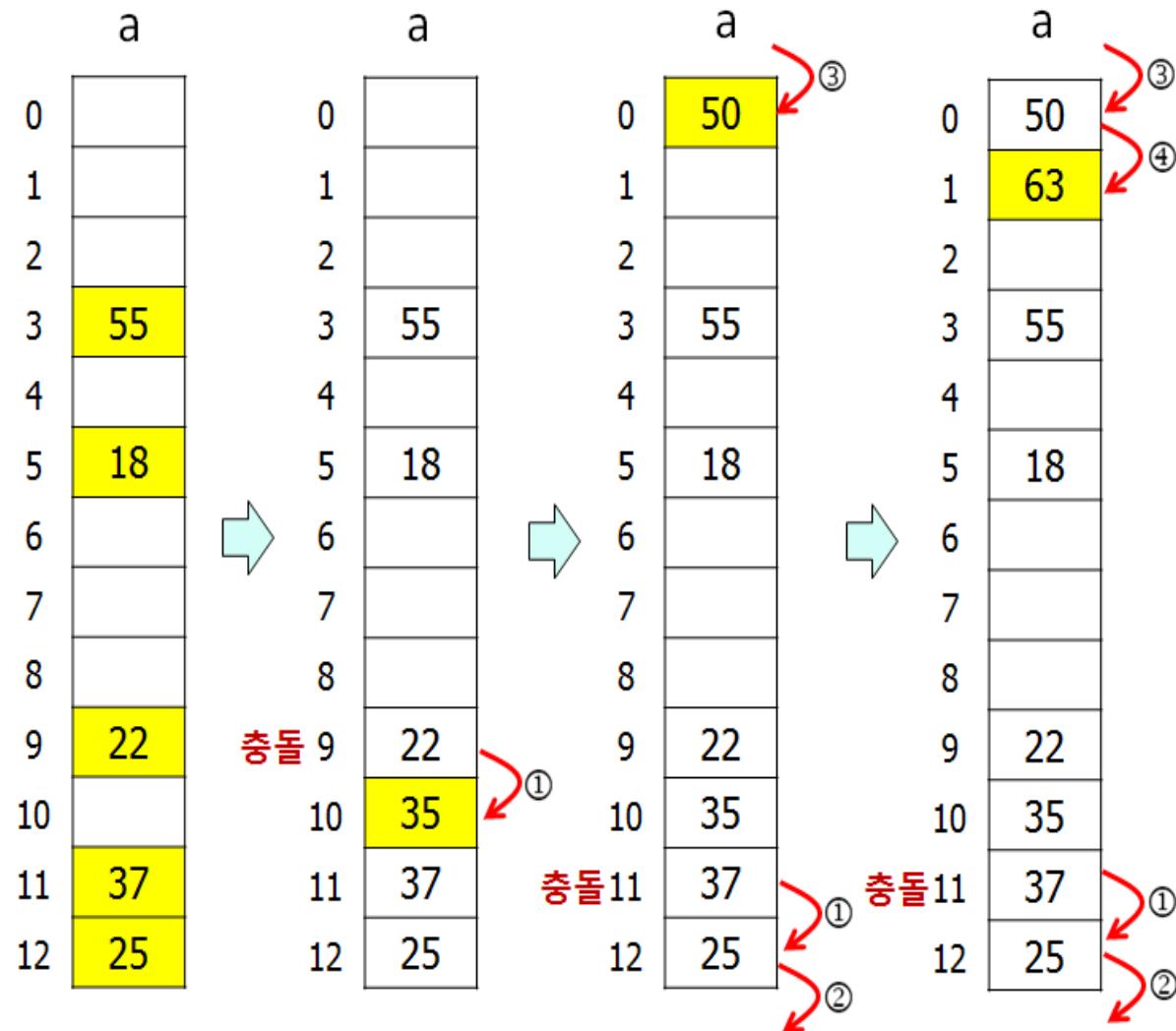
$$h_i(x) = (h_0(x) + i) \% m$$

1차 군집(Primary Clustering)에 취약

# 개방형 주소 방식: 선형 조사 (2/10)

## ● 선형 조사: 선형 조사 방식의 키 저장 과정

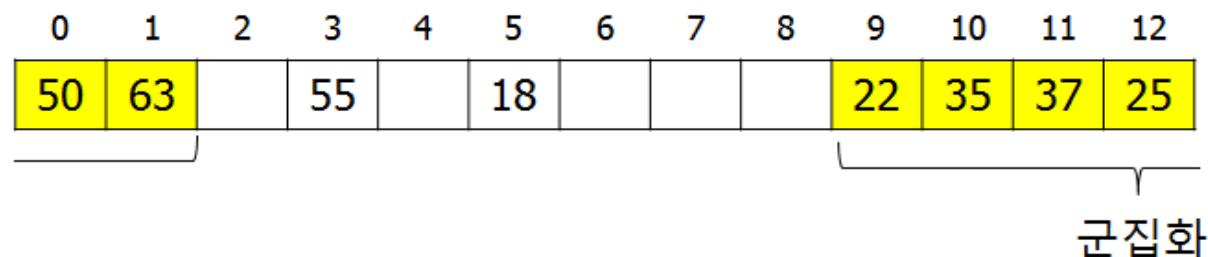
key	$h(key) = key \% 13$
25	12
37	11
18	5
55	3
22	9
35	9
50	11
63	11



# 개방형 주소 방식: 선형 조사 (3/10)

## ● 선형 조사: 1차 군집화

- 선형 조사는 순차 검색으로 empty 원소를 찾아 충돌된 키를 저장하므로 해시 테이블의 키들이 빈틈없이 뭉쳐지는 현상이 발생한다[**1차 군집화(Primary Clustering)**].
  - 이러한 군집화는 검색, 삽입, 삭제 연산 시 군집 된 키들을 순차적으로 방문해야 하는 문제점을 야기한다.



- 군집화는 해시 테이블에 empty 원소 수가 적을수록 더 심화되며 해시 성능을 극단적으로 저하시킨다.

# 개방형 주소 방식: 선형 조사 (4/10)

## ● 선형 조사

선형 조사(Linear Probing)

$$h_i(x) = (h_0(x) + i) \% m$$

1차 군집(Primary Clustering)에 취약

i 번째 해시 함수의 예

$$h_i(x) = (h_0(x) + i) \% 101$$

table[ ]
0
:
22
123
23
224
24
25
22
729
:
100

삽입 순서: 123, 24, 224, 22, 729, ...

$h_0(123)=h_0(224)=h_0(22)=h_0(729) = 22$

i + 1

i + 2       $h_0(24) = 24$

i + 3

i + 4

# 개방형 주소 방식: 선형 조사 (5/10)

## ● 선형 조사: 알고리즘

### ○ 검색, 삽입, 삭제

#### 알고리즘 12-2 개방 주소 방법

```
◀ table[]: 해시 테이블
insert(x):
    i ← 0
    repeat
        j ← hi(x)
        if (table[j] = null or table[slot] = DELETE)
            table[j] ← x
            return j
        else i++
    until (i = m)
    error "테이블 오버플로우"
```

#### 알고리즘 12-2 개방 주소 방법

```
search(x):
    i ← 0
    repeat
        j ← hi(x)
        if (table[j] = x) return j
        else i++
    until (table[j] = null or i = m)
    return NOT_FOUND ◀ search failed!
```

#### 알고리즘 12-2 개방 주소 방법

```
delete(x):
    i ← 0
    repeat
        j ← hi(x)
        if (table[j] = x)
            ① table[j] ← DELETED;
            break
        else i++
    until (table[j] = null or i = m)
```

# 개방형 주소 방식: 선형 조사 (6/10)

## ● 선형 조사: 삭제

### ○ 삭제 시 주의할 점

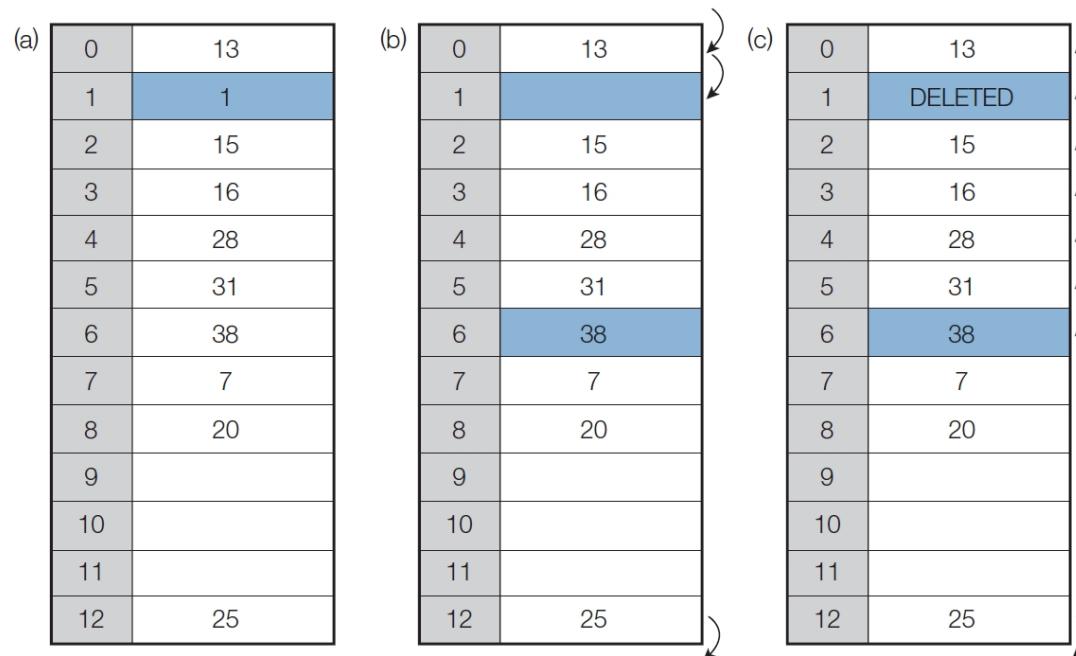


그림 12-5 해시 테이블에서 자료가 삭제될 경우의 처리법

삭제된 자리를 **null** 값으로 채우면  
검색 시 존재하는 원소를 없다고 대답할 수 있다.



지운 자리에 표식을 해둔다  
예: 상수 **DELETED** 할당

# 개방형 주소 방식: 선형 조사 (7/10)

## ● 선형 조사: 삽입

### ○ 삽입의 예

입력 : 25, 13, 16, 15, 7, 28, 31, 20, 1, 38

0	13
1	
2	15
3	16
4	28
5	
6	
7	7
8	
9	
10	
11	
12	25

(a) 키 25, 13, 16, 15, 7, 28 삽입

0	13
1	
2	15
3	16
4	28
5	31
6	
7	7
8	20
9	
10	
11	
12	25

(b) 키 31, 20 삽입

0	13
1	1
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

(c) 키 1, 38 삽입

그림 12-6 선형 탐색의 예

# 개방형 주소 방식: 선형 조사 (8/10)

## ● 선형 조사: 삽입

### ○ 1차 군집화

0	
1	
2	15
3	16
4	28
5	31
6	44
7	
8	
9	
10	
11	37
12	

(a)  $a = 1, b = 0$ 인 경우

그림 12-7 1차 군집의 예

0	
1	
2	15
3	
4	17
5	
6	28
7	33
8	43
9	
10	23
11	37
12	

(b)  $a = 2, b = 0$ 인 경우

선형 조사(Linear Probing)

$$h_i(x) = (h_0(x) + ai + b) \% m$$

이차원 조사(Quadratic Probing)

$$h_i(x) = (h_0(x) + ai^2 + bi + c) \% m$$

이중 해싱(Double Hashing)

$$h_i(x) = (h_0(x) + i \cdot f(x)) \% m$$

$f(x)$ : 보조 해시 함수

# 개방형 주소 방식: 선형 조사 (9/10)

## ● 선형 조사: 삽입

### ○ 1차 군집화와 2차 군집화

0	
1	
2	15
3	16
4	28
5	31
6	44
7	29
8	
9	
10	
11	37
12	



0	
1	
2	15
3	28
4	
5	54
6	41
7	
8	21
9	
10	
11	67
12	

그림 12-9 2차 군집의 예

$$h(x) = x \% 13$$

$$f(x) = 1 + (x \% 11)$$

$$h_i(x) = (h(x) + i \cdot f(x)) \% 13$$

0	
1	
2	15
3	
4	67
5	
6	19
7	
8	
9	28
10	
11	41
12	

그림 12-10 2차 군집에서 해방된 예

$$h(15) = h(28) = h(41) = h(67) = 2$$

$$h_1(67) = 4$$

$$h_1(28) = 9$$

$$h_1(41) = 11$$

그림 12-8 1차 군집을 빨리 벗어나는 예

# 개방형 주소 방식: 선형 조사 (10/10)

## ● 선형 조사

### ○ 이차원 조사

이차원 조사(Quadratic Probing)

$$h_i(x) = (h_0(x) + i^2) \% m$$

2차 군집(Secondary Clustering)에 취약

i 번째 해시 함수의 예

$$h_i(x) = (h_0(x) + i^2) \% 101$$



table[ ]	
22	123
23	224
	24
26	22
	:
31	729
	:

$i = 123 \bmod 101 = 22$

$i + 1^2$

$i + 2^2$

$i + 3^2$

# 개방형 주소 방식: 이차 조사 (1/2)

## ● 이차 조사(Quadratic probing)

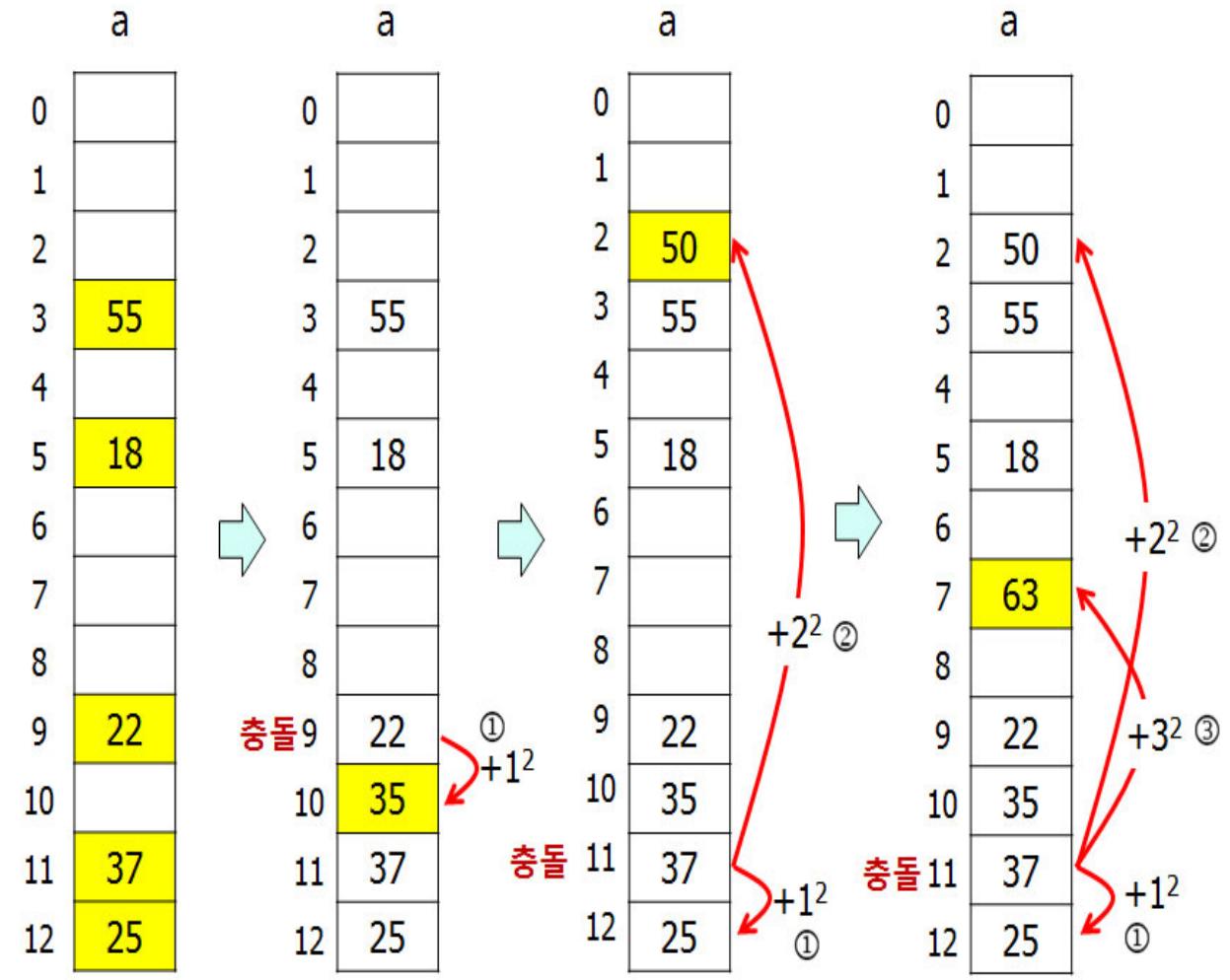
### ○ 선형 조사와 근본적으로 동일한 충돌해결 방법

- 충돌 후 배열  $a$ 에서 ...
- $(h(key) + j^2) \% M, j = 0, 1, 2, 3, \dots$
- 으로 선형 조사보다 더 멀리 떨어진 곳에서 empty 원소를 찾는다.
- 이차 조사는 이웃하는 빈 곳이 채워져 만들어지는 1차 군집화 문제를 해결하지만,
- 같은 해시 값을 갖는 서로 다른 키들인 동의어(Synonym)들이 똑같은 점프 시퀀스(Jump Sequence)를 따라 empty 원소를 찾아 저장하므로 결국 또 다른 형태의 군집화인 **2차 군집화(Secondary Clustering)**를 야기한다.
- 점프 크기가 제곱 만큼 씩 커지므로 배열에 empty 원소가 있는데도 empty 원소를 건너뛰어 검색에 실패하는 경우도 피할 수 없다.

# 개방형 주소 방식: 이차 조사 (2/2)

- 이차 조사: 이차 조사 방식의 키 저장 과정

key	$h(key) = key \% 13$
25	12
37	11
18	5
55	3
22	9
35	9
50	11
63	11



# 개방형 주소 방식: 무작위 조사

---

## ● 무작위 조사(Random probing)

- 선형 조사와 이차 조사의 규칙적인 점프 시퀀스와는 달리 점프 시퀀스를 무작위화 하여 empty 원소를 찾는다.
  - 무작위 조사는 의사 난수 생성기를 사용하여 다음 위치를 찾는다.
  - 무작위 조사 방식도 동의어들이 똑같은 점프 시퀀스에 따라 empty 원소를 찾아 키를 저장하게 되고, 이 때문 **3차 군집화(Tertiary Clustering)**가 발생한다.

# 개방형 주소 방식: 이중 해싱 (1/3)

## ● 이중 해싱(Double Hashing)

### ○ 이중 해싱은 2개의 해시 함수를 사용한다.

- 하나는 기본적인 해시 함수  $h(key)$ 로 키를 해시 테이블의 인덱스로 변환하고, 제2의 함수  $d(key)$ 는 충돌 발생 시 다음 위치를 위한 점프 크기를 다음의 규칙에 따라 정한다.
- $(h(key) + j \cdot d(key)) \bmod M, j = 0, 1, 2, \dots$ 
  - 제 2의 함수  $d(key)$ 는 점프 크기를 정하는 함수이므로 0을 반환 해서는 안된다.
  - 그 외의 조건으로  $d(key)$ 의 값과 해시 테이블의 크기  $M$  과 서로소(Relatively Prime) 관계일 때 좋은 성능을 보인다.
  - 하지만 해시 테이블 크기  $M$ 을 소수로 선택하면, 이 제약 조건을 만족한다.

### ○ 이중 해싱의 장점

- 빈 곳을 찾기 위한 점프 시퀀스가 일정하지 않으며(저마다 제2의 해시 함수를 갖기 때문에), 모든 군집화 현상을 발생시키지 않는다(모든 군집화 문제를 해결).
- 또한 해시 성능을 저하시키지 않는 동시에 해시테이블에 많은 키들을 저장할 수 있다는 장점을 가지고 있다.

# 개방형 주소 방식: 이중 해싱 (2/3)

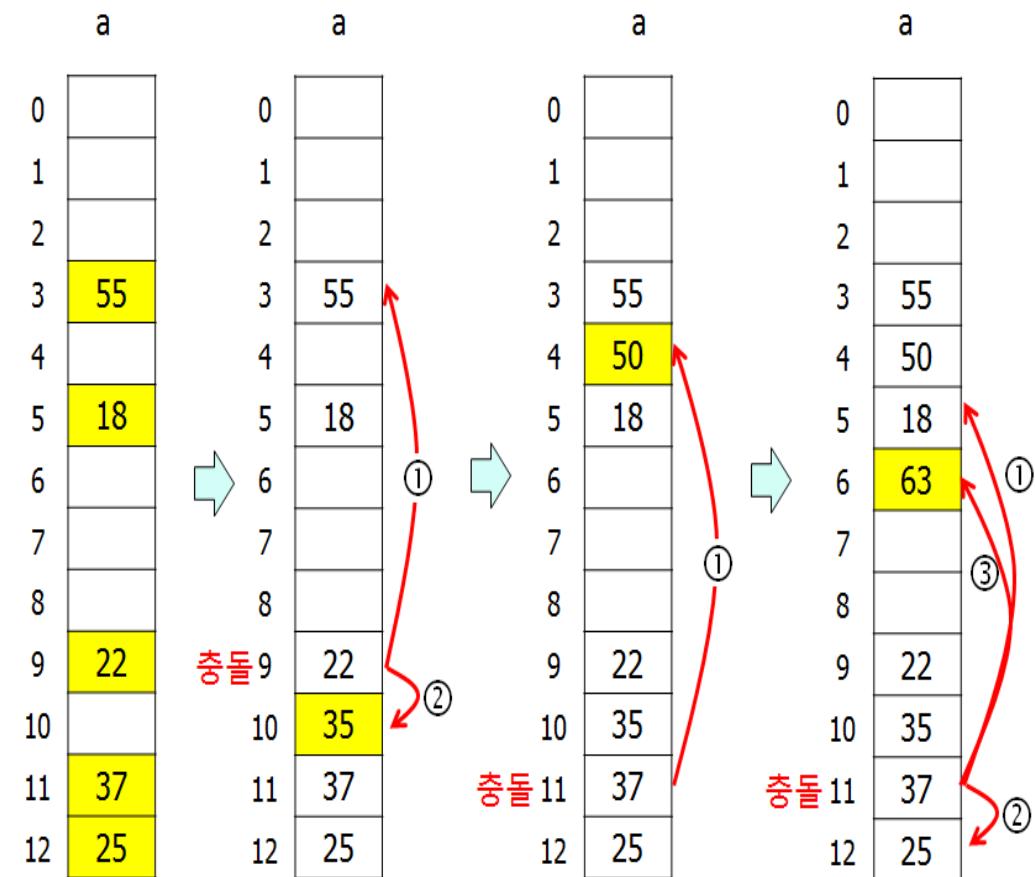
$h(key) = key \% 13$ 과  $d(key) = 7 - (key \% 7)$ 에 따라, 25, 37, 18, 55, 22, 35, 50, 63을 해시테이블에 차례로 저장하는 과정

key	h(key)	d(key)	$(h(key) + j*d(key)) \% 13$		
			j=1	j=2	j=3
25	12				
37	11				
18	5				
55	3				
22	9		①	②	
35	9	7	3	10	
50	11	6	4		③
63	11	7	5	12	6

$$h(key) = key \% 13$$

$$d(key) = 7 - (key \% 7)$$

$$(h(key) + j*d(key)) \% 13, j = 0, 1, \dots$$



# 개방형 주소 방식: 이중 해싱 (3/3)

이중 해싱(Double hashing)

$$h_i(x) = (h_0(x) + i \cdot f(x)) \% 101$$

$i$  번째 해시 함수의 예

$$h_0(x) = x \% 101$$

$$f(x) = 1 + (x \% 97)$$

table[ ]	
22	123
45	22
53	224
73	729

$h_0(123) = h_0(224) = h_0(22) = h_0(729) = 22$

$\beta(22) = 23, h_1(22) = 45$

$\beta(224) = 31, h_1(224) = 53$

$\beta(729) = 51, h_1(729) = 73$



## 충돌 해결

폐쇄 주소 방식



# 폐쇄 주소 방식 (1/2)

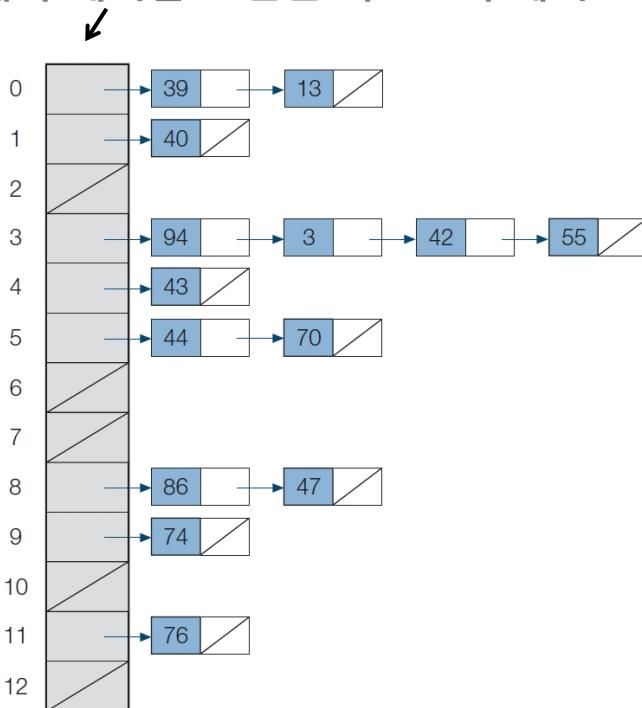
## ● 폐쇄 주소 방식(Closed Addressing Methods)

- 키에 대한 해시 값에 대응되는 곳에만 키를 저장한다.

- 충돌이 발생한 키들은 한 위치에 모여 저장한다.

- 가장 대표적인 방법: 체이닝(Chaining)

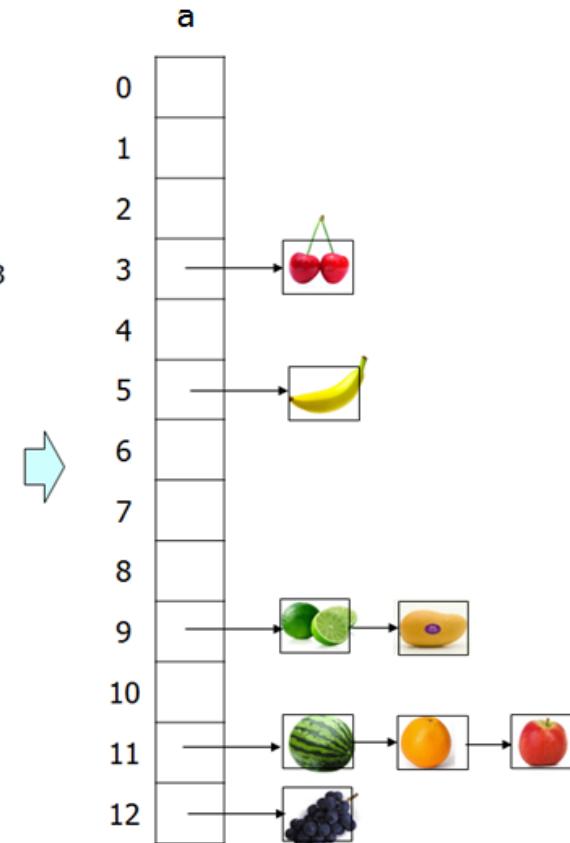
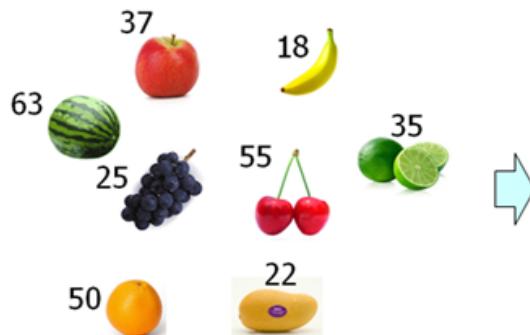
해시 테이블은 연결 리스트의 헤더 노드를 참조한다



직접 충돌이 일어나지 않은  
키들끼리는 간섭하지 않는다.

# 폐쇄 주소 방식 (2/2)

## ● 폐쇄 주소 방식(Closed Addressing Methods)





## 충돌 해결

개방형 주소 방식 구현: 선형.이차 조사



# 개방형 주소 방식 구현: 선형.이차 조사 (3/3)

## ● 개방형 주소 방식: 순차 자료 구조

```
# 클래스 설계: HashOpenAddressing
```

```
class HashOpenAddressing:
```

```
    def __init__(self, n:int):
```

```
        self.__table = [ None for i in range(n) ]
```

```
        self.__count = 0
```

```
        self.__DELETED = -54321
```

```
# 해시 함수
```

```
def __Hash(self, i:int, num:int) -> int:
```

```
    return (num + i) % len(self.__table)
```

```
    # return (num + i*i) % len(self.__table)
```

# 선형 조사

# 이차 조사

```
# 데이터 삽입.삭제.검색, 전체 출력
```

```
def insert(self, num):
```

```
def delete(self, num):
```

```
def search(self, num):
```

```
def output(self):
```



# 개방형 주소 방식 구현: 선형.이차 조사 (2/3)

## ● 개방형 주소 방식: 순차 자료 구조

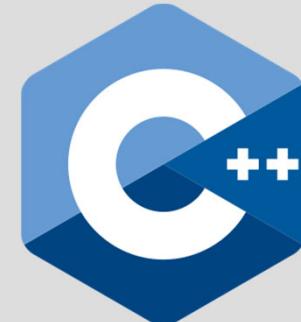
```
// #pragma once
#ifndef __HashOpenAddressing_H__
#define __HashOpenAddressing_H__

// 클래스 설계: HashOpenAddressing
template <typename T>
class HashOpenAddressing {
public:
    HashOpenAddressing(int num);
    int insert(int num);
    int remove(int num);
    int search(int num);
    void print(void);

private:
    vector<T> table_;
    int count_;
    const int DELETED = -54321;

    // 해시함수
    int Hash_(int i, int num) {
        return (num + i) % table_.size();
        // return (num + i*i) % __table.size();           // 선형 조사
    }
};

#endif
```





## 충돌 해결

폐쇄 주소 방식 구현: Chaining



# 폐쇄 주소 방식 구현(Python): Chaining (3/3)

## ● 폐쇄 주소 방식: 연결 자료 구조

```
class Chaining:  
    class SNode:  
        def __init__(self, data):  
            self.data = data  
            self.link = None  
  
    def __init__(self, n:int):  
        self.__table = [None] * n  
  
    def __Hash(self, num):  
        return num % len(self.__table)  
  
    def insert(self, num):  
    def delete(self, num):  
    def search(self, num):  
    def output(self):
```



# 폐쇄 주소 방식 구현(C++): Chaining (2/3)

## ● 폐쇄 주소 방식: 연결 자료 구조

```
// Chaining class
template<typename E>
class Chaining {
public:
    Chaining(int num);
    ~Chaining(void);
    E      insert(E num);
    E      remove(E num);
    E      search(E num);
    void   output(void);
private:
    class SNode {
public:
    E      data;
    Snode *link;
    SNode(E data) : data(data), link(nullptr) {}
};
vector<SNode*> _table;
int __Hash(int num){ // 해시 함수
    return num % _table.size();
}
};
```



# 충돌 해결: 구현



백문이불여일타(百聞而不如一打)

- 해시 테이블의 이해

- 충돌 해결: 구현

○ 개방 주소 방식

○ 폐쇄 주소 방식



# 개방형 주소 방식(Python): 순차 자료 구조 (1/5)

## ● 개방형 주소 방식: 순차 자료구조

```
# 클래스 설계: HashOpenAddressing
class HashOpenAddressing:
    def __init__(self, n:int):
        self.__table = [ None for i in range(n) ]
        self.__count = 0
        self.__DELETED = -54321

    # 해시 함수
    def __Hash(self, i:int, num:int) -> int:
        return (num + i) % len(self.__table)
        # return (num + i*i) % len(self.__table)      # 선형 조사
                                                # 이차 조사

    # 데이터 삽입.삭제.검색, 전체 출력
    def insert(self, num):
    def delete(self, num):
    def search(self, num):
    def output(self):
```



# 개방형 주소 방식(Python): 순차 자료 구조 (2/5)

## ● 개방형 주소 방식: 순차 자료구조

- 프로그램 실행 결과는 다음과 같다.

The screenshot shows a Python IDLE Shell window with the title '\*IDLE Shell 3.11.2\*'. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell area displays the following sequence of events:

```
>>> Python 3.11.2 (tags/v3.11.2:878ead, Type "help", "copyright", "credits", "license" for more information)
= RESTART: C:\Users\click\OneDrive\...
    ## 해시 테이블 ##
1) 데이터 삽입
2) 데이터 삭제
3) 데이터 검색
4) 전체 출력
5) 프로그램 종료

메뉴 선택: 2
임의의 정수 입력(종료: 0): 50
임의의 정수 입력(종료: 0): 0

    ## 해시 테이블 ##
1) 데이터 삽입
2) 데이터 삭제
3) 데이터 검색
4) 전체 출력
5) 프로그램 종료

메뉴 선택: 4
count(7): [-54321, 63, None, 55, None, 18, None, None, 22, 35, 37, 25]

    ## 해시 테이블 ##
1) 데이터 삽입
2) 데이터 삭제
3) 데이터 검색
4) 전체 출력
5) 프로그램 종료

메뉴 선택: 1
임의의 정수 입력(종료: 0): 25
임의의 정수 입력(종료: 0): 37
임의의 정수 입력(종료: 0): 18
임의의 정수 입력(종료: 0): 55
임의의 정수 입력(종료: 0): 22
임의의 정수 입력(종료: 0): 35
임의의 정수 입력(종료: 0): 50
임의의 정수 입력(종료: 0): 63
임의의 정수 입력(종료: 0): 0

    ## 해시 테이블 ##
1) 데이터 삽입
2) 데이터 삭제
3) 데이터 검색
4) 전체 출력
5) 프로그램 종료

메뉴 선택: 3
임의의 정수 입력(종료: 0): 63
검색된 데이터: 63
임의의 정수 입력(종료: 0): 50
데이터가 존재하지 않습니다.
임의의 정수 입력(종료: 0):
```

At the bottom left, a yellow box highlights the menu selection '메뉴 선택: 4' and its output 'count(8): [50, 63, None, 55, None, 18, None, None, 22, 35, 37, 25]'. The rest of the output is enclosed in green boxes.

# 개방형 주소 방식(Python): 순차 자료 구조 (3/5)

예제 9-2: 개방형 주소 방식 -- 선형 조사 HashOpenAddressing.py (1/3)

```
# 클래스 설계: HashOpenAddressing
class HashOpenAddressing:
    def __init__(self, n:int):
        self.__table = [ None for i in range(n) ]
        self.__count = 0
        self.__DELETED = -54321

    # 해시 함수
    def __Hash(self, i:int, num) -> int:
        return (num + i) % len(self.__table)           # 선형 조사
        # return (num + i*i) % len(self.__table)       # 이차 조사

    # 데이터 삽입.삭제.검색
    def insert(self, num):
    def delete(self, num):
    def search(self, num):

    # 전체 출력
    def output(self) -> None:
        print(f'count({self.__count}): {self.__table}')
```

# 개방형 주소 방식(Python): 순차 자료 구조 (4/5)

예제 9-2: 개방형 주소 방식 -- 선형 조사 HashOpenAddressing.py (2/3)

```
if __name__ == '__main__':
    import os          # system
    import sys         # exit

h = HashOpenAddressing(13)
while (True):
    os.system('cls')
    print('\n ### 해시 테이블 ###')
    print('1) 데이터 삽입')
    print('2) 데이터 삭제')
    print('3) 데이터 검색')
    print('4) 전체 출력')
    print('5) 프로그램 종료\n')
    choice = int(input('메뉴 선택: '))

    match choice:
        case 1:
            while True:
                num = int(input('임의의 정수 입력(종료: 0): '))
                if num == 0: break
                h.insert(num)
```

```
09_HashOpenAddressing.py X
(DS)_알고리즘 > 04.자료구조&알고리즘 > 02.(예제)_연습문제 > 01.(초판)_202409_연습문제 > 03

12 # 클래스 설계: HashOpenAddressing
13 class HashOpenAddressing:
14     def __init__(self, n:int):
15         self.__table = [None for i in range(n)]
16         self.__count = 0
17         self.__DELETED = -54321
18
19     # 해시 함수
20     def __Hash(self, i:int, num) -> int:
21         return (num + i) % len(self.__table)           # 선형 조사
22         # return (num + i*i) % len(self.__table)       # 이차 조사
23
24
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

### 해시 테이블 ###
1) 데이터 삽입
2) 데이터 삭제
3) 데이터 검색
4) 전체 출력
5) 프로그램 종료

메뉴 선택:
```

# 개방형 주소 방식(Python): 순차 자료 구조 (5/5)

예제 9-2: 개방형 주소 방식 -- 선형 조사 HashOpenAddressing.py (3/3)

```
case 2:
    while True:
        num = int(input('임의의 정수 입력(종료: 0): '))
        if num == 0: break
        h.delete(num)

case 3:
    while True:
        num = int(input('임의의 정수 입력(종료: 0): '))
        if num == 0: break
        searched = h.search(num)
        if searched: print(f'검색된 데이터: {searched}')
        else: print(f'데이터가 존재하지 않습니다.')
    case 4: h.output()
case 5:
    sys.exit("\n프로그램 종료!!!")
case _: print('\n잘못 선택 하셨습니다. \n')
os.system('pause')

# del h
# h.__del__
```

# 개방형 주소 방식(C++): 순차 자료 구조 (1/6)

## ● 개방형 주소 방식: 순차 자료 구조

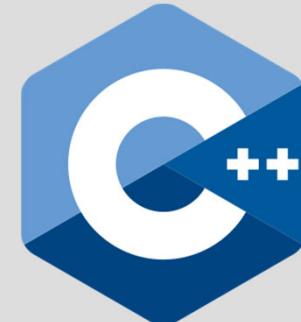
```
// #pragma once
#ifndef __HashOpenAddressing_H__
#define __HashOpenAddressing_H__

// 클래스 설계: HashOpenAddressing
template <typename T>
class HashOpenAddressing {
public:
    HashOpenAddressing(int num);
    int insert(int num);
    int remove(int num);
    int search(int num);
    void print(void);

private:
    vector<T> table_;
    int count_;
    const int DELETED = -54321;

    // 해시함수
    int Hash_(int i, int num) {
        return (num + i) % table_.size();
        // return (num + i*i) % __table.size();           // 선형 조사
    }
};

#endif
```



# 개방형 주소 방식(C++): 순차 자료 구조 (2/6)

## ● 개방형 주소 방식: 순차 자료 구조

- 프로그램 실행 결과는 다음과 같다.

The image shows two terminal windows side-by-side, both titled 'C:\Users\click\OneDrive\'.

**Left Terminal Window:**

- Output: ##### 해시 테이블 #####
- Output: 1) 데이터 삽입  
2) 데이터 삭제  
3) 데이터 검색  
4) 전체 원소 출력  
5) 프로그램 종료
- Output: 메뉴 선택: 1  
임의의 정수 입력(종료 : 0): 25  
임의의 정수 입력(종료 : 0): 37  
임의의 정수 입력(종료 : 0): 18  
임의의 정수 입력(종료 : 0): 55  
임의의 정수 입력(종료 : 0): 22  
임의의 정수 입력(종료 : 0): 35  
임의의 정수 입력(종료 : 0): 50  
임의의 정수 입력(종료 : 0): 63  
임의의 정수 입력(종료 : 0): 0  
계속하려면 아무 키나 누르십시오 . . .
- Output: 메뉴 선택: 4  
count(8): 50 63 -1 55 -1 18 -1 -1 22 35 37 25  
계속하려면 아무 키나 누르십시오 . . .

**Right Terminal Window:**

- Output: ##### 해시 테이블 #####
- Output: 1) 데이터 삽입  
2) 데이터 삭제  
3) 데이터 검색  
4) 전체 원소 출력  
5) 프로그램 종료
- Output: 메뉴 선택: 2  
임의의 정수 입력(종료 : 0): 50  
임의의 정수 입력(종료 : 0): 0  
계속하려면 아무 키나 누르십시오 . . .
- Output: 메뉴 선택: 4  
count(7): -54321 63 -1 55 -1 18 -1 -1 22 35 37 25  
계속하려면 아무 키나 누르십시오 . . . |
- Output: 메뉴 선택: 3  
임의의 정수 입력(종료 : 0): 63  
검색된 데이터: 63  
임의의 정수 입력(종료 : 0): 50  
데이터가 존재하지 않습니다.  
임의의 정수 입력(종료 : 0): 0  
계속하려면 아무 키나 누르십시오 . . .

# 개방형 주소 방식(C++): 순차 자료 구조 (3/6)

예제 9-1: 개방형 주소 방식 -- 선형 조사 HashOpenAddressing.cpp (1/4)

```
#include <iostream>
#include <vector>
using namespace std;

// #pragma once
#ifndef __HashOpenAddressing_H__
#define __HashOpenAddressing_H__

// 클래스 설계: HashOpenAddressing
template <typename T>
class HashOpenAddressing {
public:
    HashOpenAddressing(int num);
    int insert(int num);
    int remove(int num);
    int search(int num);
    void print(void);

private:
    vector<T> table_;
    int count_;
    const int DELETED = -54321;

    // 해시 함수
    int Hash_(int i, int num) {
        return (num + i) % table_.size();
        // return (num + i*i) % __table.size();           // 선형 조사
    }
};


```

# 개방형 주소 방식(C++): 순차 자료 구조 (4/6)

예제 9-1: 개방형 주소 방식 -- 선형 조사 HashOpenAddressing.cpp (2/4)

```
// 생성자
template <typename T>
HashOpenAddressing<T>::HashOpenAddressing(int num): count_(0) {
    table_ = vector<int>(num, -1);
}

// 데이터 삽입.삭제.검색
template <typename T>
void HashOpenAddressing<T>::insert(void) {}

template <typename T>
void HashOpenAddressing<T>::remove(void) {}

template <typename T>
void HashOpenAddressing<T>::search(void) {}

// 전체 원소 출력
template <typename T>
void HashOpenAddressing<T>::print(void) {
    cout << "count(" << count_ << ")";
    for (int i = 0; i < table_.size(); ++i) {
        cout.width(3);
        cout << table_[i];
    }
    cout << endl;
}

#endif
```

# 개방형 주소 방식(C++): 순차 자료 구조 (5/6)

예제 9-1: 개방형 주소 방식 -- 선형 조사 HashOpenAddressing.cpp (3/4)

```
int main(void) {
    HashOpenAddressing<int>    h(13);
    int      choice, num;

    while (true) {
        system("cls");
        cout << "\n##### 해시 테이블 ##### \n" << endl;
        cout << "1) 데이터 삽입" << endl;
        cout << "2) 데이터 삭제" << endl;
        cout << "3) 데이터 검색" << endl;
        cout << "4) 전체 원소 출력" << endl;
        cout << "5) 프로그램 종료" << endl;
        cout << "\n메뉴 선택: ";
        cin >> choice;

        switch (choice) {
            case 1:
            case 2:
            case 3:
                while (true) {
                    cout << "임의의 정수 입력(종료 : 0): ";
                    cin >> num;
                    if (num == 0)
                        break;
                }
        }
    }
}
```

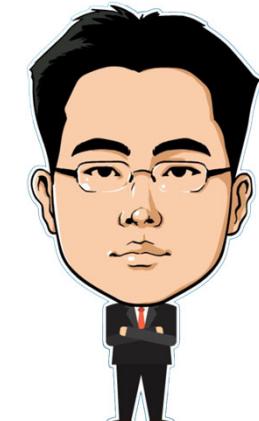
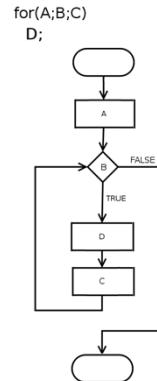
# 개방형 주소 방식(C++): 순차 자료 구조 (6/6)

예제 9-1: 개방형 주소 방식 -- 선형 조사 HashOpenAddressing.cpp (4/4)

```
// 선택한 메뉴별 실행: 삽입.삭제.검색
if (choice == 1)          h.insert(num);
else if (choice == 2)      h.remove(num);
else if (choice == 3) {
    int searched = h.search(num);
    if (searched != -1) {
        cout << "검색된 데이터: " << searched << endl;
    }
    else {
        cout << "데이터가 존재하지 않습니다." << endl;
    }
}
break;
case 4: h.print(); break;
case 5: cout << "프로그램 종료!!!\n" << endl;
         return 0;
default: cout << "메뉴를 잘못 선택하셨습니다." << endl;
}
system("pause");
}
return 0;
}
```

# 참고문헌

- [1] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛미디어, 2022.
- [2] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.
- [3] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(3판), 개정판, 한빛아카데미, 2024.
- [5] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.
- [6] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.
- [7] "SW Expert Academy", SAMSUNG, 2025 of viewing the site, <https://swexpertacademy.com/>.
- [8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2025 of viewing the site, <https://www.acmicpc.net/>.
- [9] "programmers", grepp, 2025 of viewing the site, <https://programmers.co.kr/>.
- [10] "goormlevel", goorm, 2025 of viewing the site, <https://level.goorm.io/>



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,  
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.