집합과 맵

📚 집합과 맵 정리

🔍 핵심 개념

자료구조	목적	특징
집합 (Set)	중복 제거, 원소 존재 여부 확인	`add`, `remove`, `contains`, `size`
맵 (Map)	키-값 쌍 저장, 키 기반 접근	`insert`, `find`, `erase`, `get`

📌 1. 집합 (Set)

🎯 기능

- 원소를 저장하고 중복을 제거
- 원소가 존재하는지 확인
- 원소 추가/제거

✅ C 스타일 구현 (예: `int` 타입)

```c
// set.h
#ifndef SET_H
#define SET_H
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// 집합 구조체 (단순한 배열 기반)
typedef struct {
    int* data;
    int size;
    int capacity;
} Set;

// 초기화
void set_init(Set* s, int capacity);

// 원소 추가 (중복 제거)
void set_add(Set* s, int value);

// 원소 존재 여부 확인
```

```c
bool set_contains(Set* s, int value);

// 원소 제거
void set_remove(Set* s, int value);

// 크기 반환
int set_size(Set* s);

// 정리 (메모리 해제)
void set_destroy(Set* s);

#endif
```

```c
// set.c
#include "set.h"
#include <string.h>

void set_init(Set* s, int capacity) {
    s->data = (int*)malloc(capacity * sizeof(int));
    s->size = 0;
    s->capacity = capacity;
}

void set_add(Set* s, int value) {
    if (s->size >= s->capacity) {
        // 확장 (간단한 경우는 제한)
        return;
    }
    for (int i = 0; i < s->size; i++) {
        if (s->data[i] == value) {
            return; // 중복 제거
        }
    }
    s->data[s->size] = value;
    s->size++;
}

bool set_contains(Set* s, int value) {
    for (int i = 0; i < s->size; i++) {
        if (s->data[i] == value) {
            return true;
        }
    }
    return false;
}

void set_remove(Set* s, int value) {
    for (int i = 0; i < s->size; i++) {
        if (s->data[i] == value) {
```

```
            // 위치 i 제거 (뒤의 원소를 앞으로 이동)
            for (int j = i; j < s->size - 1; j++) {
                s->data[j] = s->data[j+1];
            }
            s->size--;
            return;
        }
    }
}

int set_size(Set* s) {
    return s->size;
}

void set_destroy(Set* s) {
    free(s->data);
    s->size = 0;
    s->capacity = 0;
}
```

📌 2. 맵 (Map)

🎯 기능

- 키-값 쌍 저장 (예: `key -> value`)
- 키 기반으로 값 가져오기
- 키 존재 여부 확인

✅ C 스타일 구현 (예: `int` 키, `int` 값)

```
// map.h
#ifndef MAP_H
#define MAP_H
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// 맵 구조체 (키-값 쌍 저장)
typedef struct {
    int* keys;
    int* values;
    int size;
    int capacity;
} Map;

// 초기화
```

```c
void map_init(Map* m, int capacity);

// 키-값 추가 (중복 키는 오버라이드)
void map_put(Map* m, int key, int value);

// 키 기반 값 가져오기
int map_get(Map* m, int key);

// 키 존재 여부 확인
bool map_contains(Map* m, int key);

// 키 삭제
void map_remove(Map* m, int key);

// 크기 반환
int map_size(Map* m);

// 정리
void map_destroy(Map* m);

#endif
```

```c
// map.c
#include "map.h"
#include <string.h>

void map_init(Map* m, int capacity) {
    m->keys = (int*)malloc(capacity * sizeof(int));
    m->values = (int*)malloc(capacity * sizeof(int));
    m->size = 0;
    m->capacity = capacity;
}

void map_put(Map* m, int key, int value) {
    for (int i = 0; i < m->size; i++) {
        if (m->keys[i] == key) {
            m->values[i] = value;
            return;
        }
    }
    if (m->size >= m->capacity) {
        // 확장 (간단히 제한)
        return;
    }
    m->keys[m->size] = key;
    m->values[m->size] = value;
    m->size++;
}
```

```c
int map_get(Map* m, int key) {
    for (int i = 0; i < m->size; i++) {
        if (m->keys[i] == key) {
            return m->values[i];
        }
    }
    return -1; // 키 없음
}

bool map_contains(Map* m, int key) {
    for (int i = 0; i < m->size; i++) {
        if (m->keys[i] == key) {
            return true;
        }
    }
    return false;
}

void map_remove(Map* m, int key) {
    for (int i = 0; i < m->size; i++) {
        if (m->keys[i] == key) {
            // 위치 i 제거
            for (int j = i; j < m->size - 1; j++) {
                m->keys[j] = m->keys[j+1];
                m->values[j] = m->values[j+1];
            }
            m->size--;
            return;
        }
    }
}

int map_size(Map* m) {
    return m->size;
}

void map_destroy(Map* m) {
    free(m->keys);
    free(m->values);
    m->size = 0;
    m->capacity = 0;
}
```

📌 3. 예제: 집합과 맵 사용 예

```c
// main.c
#include "set.h"
#include "map.h"
#include <stdio.h>

int main() {
    Set s;
    Map m;

    // 집합 초기화
    set_init(&s, 10);
    set_add(&s, 1);
    set_add(&s, 2);
    set_add(&s, 3);
    set_add(&s, 2); // 중복 무시

    printf("Set size: %d\n", set_size(&s));
    printf("Contains 2? %s\n", set_contains(&s, 2) ? "Yes" : "No");
    set_remove(&s, 2);
    printf("After remove 2: size = %d\n", set_size(&s));

    // 맵 초기화
    map_init(&m, 10);
    map_put(&m, 1, 100);
    map_put(&m, 2, 200);
    map_put(&m, 3, 300);

    printf("Map: key 1 -> %d\n", map_get(&m, 1));
    printf("Map: key 2 -> %d\n", map_get(&m, 2));
    printf("Map size: %d\n", map_size(&m));

    // 맵에 없는 키
    printf("Map: key 4 -> %d\n", map_get(&m, 4)); // -1

    // 정리
    set_destroy(&s);
    map_destroy(&m);

    return 0;
}
```

✅ 출력 예:

```
Set size: 3
Contains 2? Yes
After remove 2: size = 2
Map: key 1 -> 100
Map: key 2 -> 200
```

```
Map size: 3
Map: key 4 -> -1
```

📊 비교 요약

기능	집합 (Set)	맵 (Map)
기능	원소 존재 여부, 중복 제거	키-값 저장, 키 기반 접근
시간 복잡도 (검색)	O(n) (최악)	O(n) (최악)
효율성	중복 제거 가능	값에 접근 가능
응용	유니크 값 처리, 집합 연산	키 기반 데이터 저장

📌 핵심 팁

- **C 스타일은 메모리 관리 책임이 있음** → `malloc/free` 사용
- **중복 제거는 for-반복으로 처리** (효율성은 제한)
- **키-값 구조는 배열 기반으로 구현** (실제 프로젝트에서는 해시 테이블이 더 효율)
- **확장성**: `capacity`를 증가시켜 더 많은 원소 저장 가능

📚 학습 포인트

1. **집합 (Set)** – 중복 제거, 원소 존재 확인
2. **맵 (Map)** – 키-값 저장, 값 접근
3. **예제 코드 작성** – 실제 문제에 적용
4. **효율성 개선** – 해시 테이블, 트리 구조로 확장

✅ C++ 스타일

✅ 1. 집합 (std::set)

```cpp
// set_example.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <set>
#include <iostream>
```

```cpp
using namespace std;

int main() {
    set<int> s;

    // 원소 추가 (중복 제거 자동)
    s.insert(1);
    s.insert(2);
    s.insert(3);
    s.insert(2); // 중복 무시됨

    // 원소 존재 여부 확인
    if (s.find(2) != s.end()) {
        printf("Contains 2\n");
    }

    // 크기 확인
    printf("Set size: %d\n", s.size());

    // 원소 제거
    s.erase(2);

    printf("After remove 2: size = %d\n", s.size());

    return 0;
}
```

✅ 출력 예:

```
Contains 2
Set size: 3
After remove 2: size = 2
```

✅ 2. 맵 (std::map)

```cpp
// map_example.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <map>
#include <iostream>

using namespace std;

int main() {
```

```
    map<int, int> m;

    // 키-값 추가 (중복 키는 오버라이드)
    m[1] = 100;
    m[2] = 200;
    m[3] = 300;

    // 키 기반 값 가져오기
    if (m.find(1) != m.end()) {
        printf("Map: key 1 -> %d\n", m[1]);
    }

    printf("Map: key 2 -> %d\n", m[2]);
    printf("Map size: %d\n", m.size());

    // 없는 키 접근
    printf("Map: key 4 -> %d\n", m.find(4) != m.end() ? m[4] : -1);

    // 원소 제거
    m.erase(2);

    printf("After remove 2: size = %d\n", m.size());

    return 0;
}
```

✅ 출력 예:

```
Map: key 1 -> 100
Map: key 2 -> 200
Map size: 3
Map: key 4 -> -1
After remove 2: size = 2
```

📊 비교 요약 (STL 버전)

기능	`std::set`	`std::map`
기능	중복 제거, 순서 유지	키-값 저장, 키 기반 접근
시간 복잡도 (검색/삽입)	O(log n)	O(log n)
효율성	자동 중복 제거	키 기반 접근 가능
응용	유니크 값 처리	키 기반 데이터 저장