

STL - 8주 2일

Wulong

String을 key로 사용하기

map<key, value>에서 map의 key로 사용할 자료형은 정렬 가능하여야 합니다. map은 key를 다른 key와 비교하여 저장할 원소의 순서를 정합니다. key로 사용할 자료형이 POD타입이거나 표준에서 이미 정의한 class라면 따로 프로그래머가 신경 쓸 것은 없습니다.

```
map<int, string> test;
```

위와 같은 test 객체를 만들었다면 map은 어떻게 순서를 정하는 것일까요? map<int, string>은 map<int, string, less<int>>와 같습니다. 디폴트 타입으로 숨겨진 세 번째 인자가 바로 어떻게 정렬할 것인가를 정해주는 호출가능 타입입니다. map test의 key 타입은 int이고 디폴트 정렬 타입인 less<int>는 표준에서 정해놓은 function object 중 하나입니다.

<https://en.cppreference.com/w/cpp/utility/functional>를 보면 모든 표준 function object를 볼 수 있습니다.

그럼 사용자가 만든 자료형을 key로 사용하려면 어떤 방법이 있겠습니까? 제목과 같이 String을 key로 사용하려면 어떻게 해야 할까요. 위의 설명을 보면 less<String>이 가능하도록 하면 되겠습니다. 아니면 세 번째 인자에 해당하는 호출가능타입을 직접 제공해도 될 것 같군요. 그런데 기본 오름차순 정렬을 한다면 이것보다 더 편리한 방법도 있습니다. 바로 String이 < 연산자를 제공하도록 하는 것입니다. 자 정리하겠습니다.

- less<String>을 정의한다.
- 호출가능타입을 직접 정의한다.
- String에 < 연산자를 오버로딩한다.

내가 알고 있는 것은 이 3가지 방법입니다. 모두 코딩해 보겠습니다.

먼저 간단한 예제 프로그램을 만들어 봅시다. 영화와 감독을 하나로 묶어 map에 저장합니다. map<String, String>을 사용하겠습니다.

```
// 코드 8-5: 영화와 감독
```

```
#include <iostream>
#include <map>
#include "String.h"
using namespace std;
```

```

int main( )
{
    map<String, String> movies;

    movies.insert( make_pair( "터미네이터2", "제임스 카메론" ) );
    movies.insert( movies.begin( ), pair<String, String>( "인셉션", "크리스토퍼 놀란" ) );

    movies.emplace( "기생충", "봉준호" );

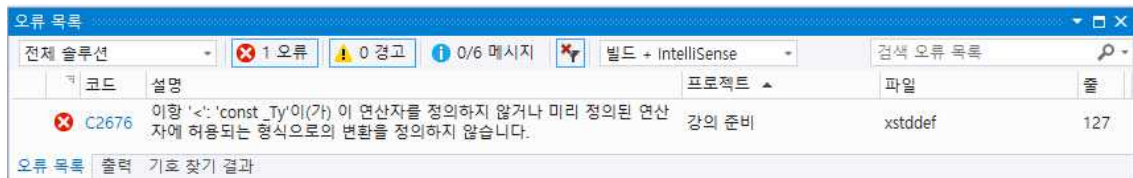
    movies["어벤져스:엔드게임"] = "안소니 루소, 조 루소";
    movies["크리스마스의 악몽"] = "팀 버튼";

    movies.insert_or_assign( "인생은 아름다워", "로베르토 베니니" );
    movies.try_emplace( movies.end( ), "쇼생크 탈출", "프랭크 다라본트" );

    for ( auto [제목, 감독] : movies )
        cout << 제목 << " - " << 감독 << endl;
}

```

서로 다른 방식으로 movies에 원소를 추가하였습니다. 이 프로그램에서 문법적으로 틀린 곳은 없습니다. 그러나 컴파일하면 다음과 같은 오류가 납니다.



이 오류는 템플릿을 사용한 오류 내용 중에서는 아주 상세하며 친절하고 이해하기 쉬운 오류에 속합니다. 프로그램을 하려면 오류 메시지와 친해져야 합니다. 오류의 내용을 잘 읽어보고 무슨 뜻인지 이해해보려고 노력해야 합니다. 오류 코드 C2676을 왼쪽버튼 클릭하면 이 오류가 왜 발생했는지 설명을 볼 수 있습니다.

```
map<String,String> movies;
```

이 문장은 아래와 같습니다.

```
map<String,String, less<String>> movies;
```

컴파일러는 less<String>이 어딘가에 정의되어 있는 자료형일거라 생각하고 찾아봅니다. 그런데 String은 내가 만든 자료형이고 나는 less<String>을 만든 적이 없으니 위와 같은 오류 메시지가 나오는 것입니다.

오류 내용을 조금 더 파고 들어갈 수도 있습니다. xstddef.h의 127번째 줄로 갑니다.

```
// Visual C++의 xstddef.h에서 가져옴

// STRUCT TEMPLATE less
template <class _Ty = void>
struct less {
    _CXX17_DEPRECATED_ADAPTOR_TYPEDEFS typedef _Ty first_argument_type;
    _CXX17_DEPRECATED_ADAPTOR_TYPEDEFS typedef _Ty second_argument_type;
    _CXX17_DEPRECATED_ADAPTOR_TYPEDEFS typedef bool result_type;

    constexpr bool operator()(const _Ty& _Left, const _Ty& _Right) const {
        return _Left < _Right;
    }
};
```

코드를 살펴보니 클래스 템플릿 **less**에서 함수 호출연산자 ()를 정의했네요. 이 연산자 안에서 < 연산자를 사용하여 **_Left**와 **_Right**를 비교하고 있습니다. 어려운 부분은 없습니다. 뭘 해야 문제가 해결될 것인지도 알 수 있습니다. 이런 코드를 보면 설명하다가도 강조하고 넘어가지 않을 수 없습니다. 변수 이름 앞에 _ 사용하지 말라는 것을 말이죠. 프로그램하면서 변수이름에 _를 붙이지 마세요. 그럼 문제를 해결해 보겠습니다.

less<String>을 정의

```
// 생략

template <>
struct less< String >
{
    bool operator()( const String& a, const String& b ) const {
        return a.getString( ) < b.getString( );
    }
};

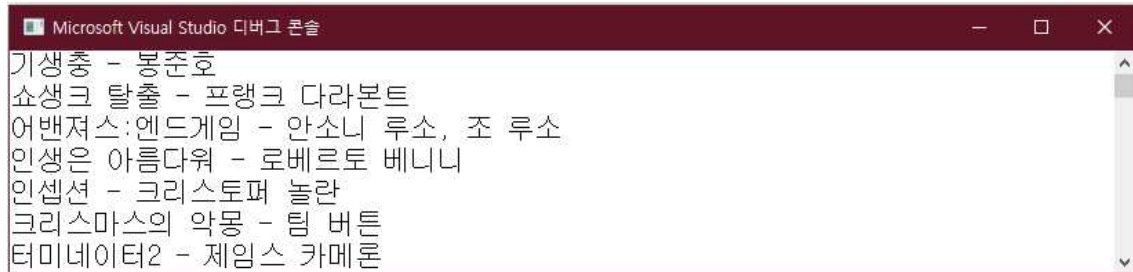
int main( )
{
    map<String, String> movies;

// 이하 생략
```

컴파일러에게 less<String>이 무엇인지 알려주는 방법입니다. 위의 템플릿은 표준의 struct less를 String 타입에 대해서 특수화한 것입니다. 이 템플릿을 다른 곳에 두지 말고 main() 위에 코딩하면 되겠습니다. 내가 만든 String 클래스에서 문제를 간단하게 하려고 getString() 멤버를 제공한 것은 아쉬움이 있습니다. 표준 string으로 변환하지 않도록 코딩했다면 마음이

불편하진 않았을 텐데 그냥 편하게 넘어갔습니다.

프로그램을 실행해 보겠습니다.



호출가능타입을 직접 정의

`map<String, String, 호출가능타입>`과 같이 `String`을 비교할 수 있는 호출가능타입을 프로그램머가 직접 제공할 수 있습니다. default인 `less<String>`을 대신하여 사용할 수 있어야 합니다. 그러므로 이것의 형식은 정해져 있습니다. 두 개의 `String`을 인자로 받아 `bool` 값을 되돌려 줘야 하는 것으로 말입니다. `bool` 값은 `true`와 `false` 두 가지 밖에 없으며 이와 같이 어떤 조건을 판단하여 `bool`을 리턴하는 함수를 **predicate**이라고 부릅니다.

```
function< bool( const String&, const String&) >
```

`function` 템플릿을 사용하면 편리합니다. `<functional>` 헤더에 정의되어 있습니다. 함께 붙여 제대로 프로그램해 보겠습니다.

```
// 생략

bool StringCmp( const String& a, const String& b ) {
    return a.getString( ) < b.getString( );
};

int main( )
{
    map<String, String, function<bool(const String&, const String&) >>
        movies( StringCmp );

    movies.insert( make_pair( "터미네이터2", "제임스 카메론" ) );

// 이하 생략
```

이렇게 비교함수를 직접 만들어 주려면 map을 생성할 때 조금 복잡해집니다. 호출가능타입의 자료형을 템플릿의 인자로 적어줘야 하고 movies 객체를 생성할 때도 비교함수인 StringCmp를 인자로 넘겨야합니다. 복잡하지만 이렇게 써야만 하는 경우가 있습니다. map의 원소를 정렬할 때 디폴트 < 연산자가 아닌 다른 방식으로 정렬하고 싶다면 반드시 이런 식으로 만드는 수밖에 없습니다. 사전식 비교를 바꿔 key의 길이를 기준으로 오름차순 정렬을 하면 다음과 같은 결과가 출력됩니다.



String에 < 연산자를 오버로딩

map<String, String> movies; 문장이 컴파일되면 less<String> 템플릿 클래스가 작성되면서 < 연산자를 호출하는 코드가 생성됨을 앞에서 알아보았습니다. class String이 < 연산자를 정의하면 movies는 원소들의 순서를 정할 수 있습니다.

```
// string.h에 추가
```

```
// 2020. 5. 4 추가
```

```
bool operator<( const String& rhs ) const;
```

```
// string.h에 추가
```

```
// 2020. 5. 4 추가
```

```
bool String::operator<( const String& rhs ) const {
    return getString( ) < rhs.getString( );
}
```

이렇게 map의 key 타입이 < 연산자를 제공하게 되면 코드 8-5는 수정 없이 그대로 실행됩니다.

multimap

바로 앞의 실행 창을 보고 “뭔가 이상한데?”라는 생각이 들었죠? 모두 여섯 개의 영화가 있었는데 결과는 4개밖에 없으니 말이죠. 왜 그런지 설명할 수 있나요?

movies의 key를 비교할 때 길이 기준으로 바뀌 그런 것이죠. 길이가 6인 “인셉션”이 먼저 들어가 있는 상태에서 길이가 같은 “기생충”을 넣으려고 하니 동일한 key가 있다고 판단하여 원소 추가가 거부 되어버린 것입니다.

multimap은 동일 key를 허용합니다. String에 < 연산자도 다 만든 상태이니 map을 multimap으로 바꾸기만 하면 전부 출력되어 나올 겁니다.

```
int main( )
{
    multimap<String, String> movies;

    movies.insert( make_pair( "터미네이터2", "제임스 카메론" ) );
    movies.insert( movies.begin( ), pair<String, String>( "인셉션", "크리스토퍼 놀란" ) );

    movies.emplace( "기생충", "봉준호" );
```

왜 코드를 여기까지만 보이게 했을까요? 그건 이 코드 밑에 있는 멤버함수들이 multimap에 서는 지원되지 않기 때문입니다. 중복 key를 써도 되니 다음 문장을 생각해 보세요.

```
movies["크리스마스의 악몽"] = "팀 버튼";
```

“크리스마스의 악몽”이라는 key를 갖는 원소를 생성하거나 바꾸려는 동작이니 이런 문장은 무의미할 뿐입니다. 밑에 들어갈 코드를 전부 emplace로 바꾸겠습니다.

```
movies.emplace("어벤져스:엔드게임", "안소니 루소, 조 루소" );
movies.emplace( "크리스마스의 악몽", "팀 버튼" );

movies.emplace( "인생은 아름다워", "로베르토 베니니" );
movies.emplace( "쇼생크 탈출", "프랭크 다라본트" );

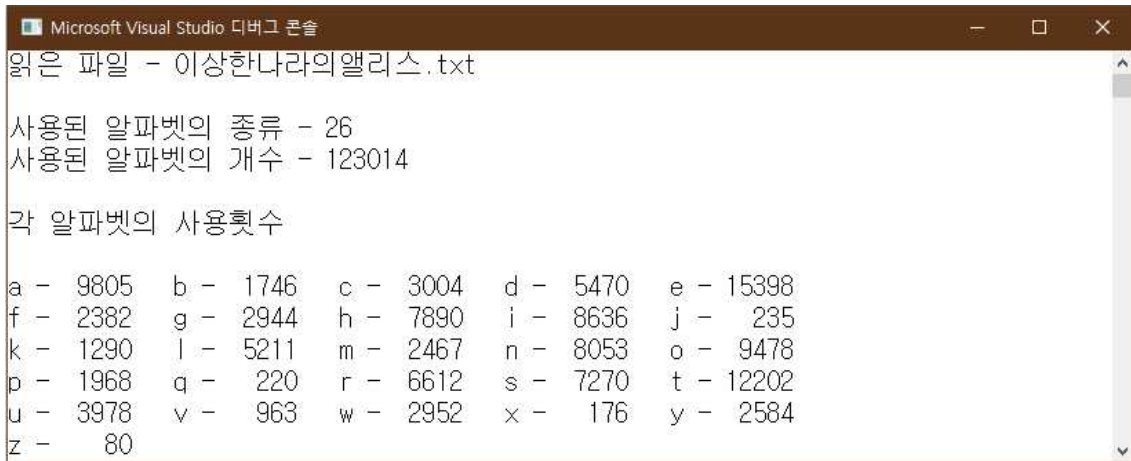
for ( auto [제목, 감독] : movies )
    cout << 제목 << " - " << 감독 << endl;
}
```

이 정도면 map/multimap을 사용하는데 아무 문제가 없을 것입니다. 재미있는 실습 문제를 풀어보며 마치겠습니다.

[실습] "이상한나라의앨리스.txt"에 사용된 알파벳 문자의 사용횟수를 알아보자. 알파벳에서 가장 많이 사용되는 것은 'e'라고 한다. 과연 그런지 사용된 알파벳의 횟수를 다음 실행화면과 같이 나오도록 코딩하라. (20분)

(주의) 알파벳인 경우만 개수를 센다.

(주의) 대문자와 소문자를 구분하지 않는다.



```
Microsoft Visual Studio 디버그 콘솔
읽은 파일 - 이상한나라의앨리스.txt

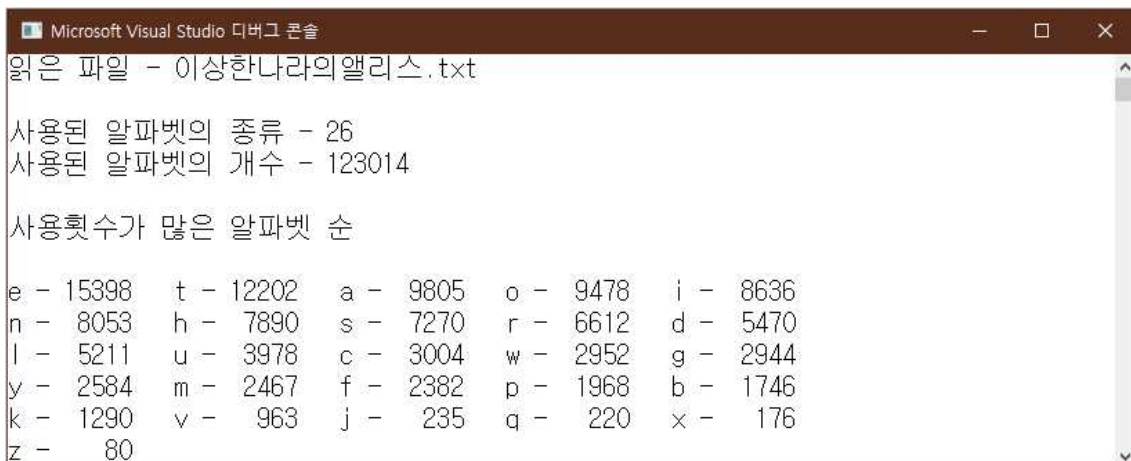
사용된 알파벳의 종류 - 26
사용된 알파벳의 개수 - 123014

각 알파벳의 사용횟수
a - 9805   b - 1746   c - 3004   d - 5470   e - 15398
f - 2382   g - 2944   h - 7890   i - 8636   j - 235
k - 1290   l - 5211   m - 2467   n - 8053   o - 9478
p - 1968   q - 220    r - 6612   s - 7270   t - 12202
u - 3978   v - 963    w - 2952   x - 176    y - 2584
z - 80
```

듣던 대로 e가 15398번으로 가장 많이 사용되었습니다. 순서대로 출력한다면 더 좋겠습니다.

[실습] "이상한나라의앨리스.txt"에 사용된 알파벳을 사용빈도가 많은 순으로 출력하라. (10분)

(주의) 앞의 실습결과를 활용하여 해결해 보자. 실행화면과 같이 나오는 가 확인하자.



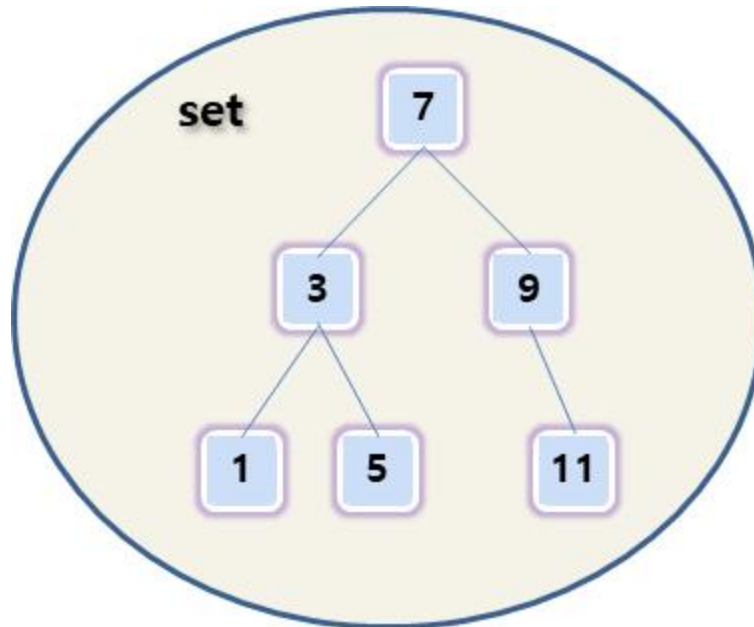
```
Microsoft Visual Studio 디버그 콘솔
읽은 파일 - 이상한나라의앨리스.txt

사용된 알파벳의 종류 - 26
사용된 알파벳의 개수 - 123014

사용횟수가 많은 알파벳 순
e - 15398   t - 12202   a - 9805   o - 9478   i - 8636
n - 8053    h - 7890   s - 7270   r - 6612   d - 5470
l - 5211    u - 3978   c - 3004   w - 2952   g - 2944
y - 2584    m - 2467   f - 2382   p - 1968   b - 1746
k - 1290    v - 963    j - 235    q - 220    x - 176
z - 80
```

set

책281~318



map을 제대로 공부했으니 set은 설명할 것도 없습니다. map은 pair<key, value>를 원소로 갖습니다. set은 key를 정렬기준으로 사용하며 이것을 그대로 저장합니다. key와 value가 같은 map이라고 생각해도 문제가 없겠습니다. 멤버함수도 map과 거의 같습니다.

표준에서는 set을 이렇게 설명합니다. 그대로 두는 것이 의미가 더 명확합니다. 한 번 보세요.

A **set** is an associative container that supports **unique keys** (contains at most one of each key value) and provides for **fast retrieval of the keys** themselves. The set class supports **bidirectional iterators**.

set은 원소를 항상 정렬상태로 저장하는 컨테이너라고 이해하면 됩니다. set<int>는 모든 int를 less<int>로 비교하여 저장할 것입니다. set<String>은 모든 원소를 less<String>으로 비교하여 순서대로 저장할 것입니다. 간단한 프로그램으로 set을 알아보겠습니다.

// 코드 8-6:로또 번호 생성기

```

#include <iostream>
#include <set>
#include <random>
using namespace std;

```



```

int main( )
{
    random_device rd;
    default_random_engine dre { rd() };
    uniform_int_distribution<> uid { 1, 45 };

    set<int> s;

    while ( s.size( ) < 6 )
        s.insert( uid( dre ) );

    cout << "이번 주 당첨번호: ";
    for ( int n : s )
        cout << n << ' ';
    cout << endl;
}

```

set이 중복을 허용하지 않는다는 점을 활용하여 [1, 45] 사이의 서로 다른 번호 6개를 저장하였습니다. 로또 번호입니다. 이 번호대로 사서 1등 되면 좋겠습니다. 실행할 때마다 다른 수가 나오도록 random_device를 사용하여 엔진을 초기화하였습니다.

```

random_device rd;
default_random_engine dre { rd() };

```

multiset

multiset은 중복 key를 허용하는 set입니다. 4주 1일 강의에서 키보드에서 입력한 단어를 정렬하는 예제가 있었습니다. multiset으로 다시 작성해 보겠습니다.

[문제] 키보드에서 입력한 단어를 오름차순으로 정렬하여 출력하라.

```

// 헤더 생략

#include <iterator>
using namespace std;

int main( )
{
    cout << "뭐든지 입력하세요. 끝내려면 Ctrl-Z를 누르세요" << endl << endl;

    multiset<string> ms { istream_iterator<string>( cin ), {} };
}

```

```

    cout << endl << "오름차순 정렬한 결과입니다" << endl << endl;
    for ( const string& s : ms )
        cout << s << endl;
}

```

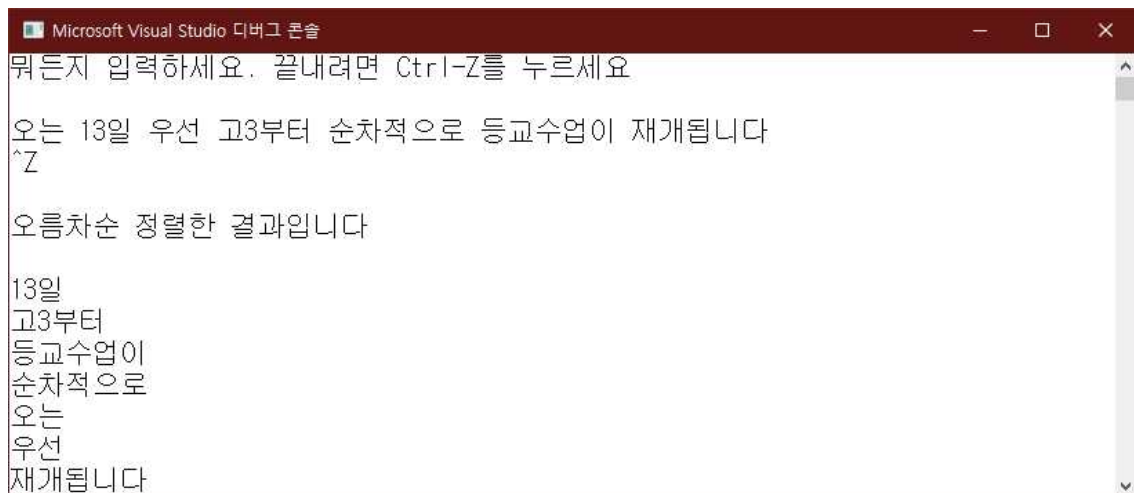
이 프로그램은 모든 기능이 이 한 줄에 들어 있습니다.

```

multiset<string> ms { istream_iterator<string>( cin ), {} };

```

array를 제외한 모든 STL 컨테이너는 반복자 범위로 초기화할 수 있습니다. 반복자 어댑터인 입력스트림반복자를 사용하여 키보드 입력의 시작과 끝을 나타낼 수 있습니다. 이런 프로그램을 이 한 줄로 해결할 수 있다니 깜짝 놀랄 일이라고 감탄할 것 없습니다. 여러분은 지금까지 공부한 것으로, 이 한 줄 속에서 어떤 일이 일어나는지 모두 이해하고 다른 사람에게 설명해 줄 수도 있습니다. 실행결과입니다. 우리는 11일 대면수업 시작합니다.



[실습] 더 해 보고 싶은 학생은 이 문제를 해 보자. (30분+)

- 3주 1일 강의에서 만든 “Dog만마리”를 set으로 읽어라. (파일이 여러분 하드에 있을 것임)

[고급] 이렇게 읽을 수 있도록 여러 군데를 고칠 수도 있다.

```

set<Dog> dogs { istream_iterator<Dog>( in ), {} }; (60분+)

```

- set의 정렬 기준은 Dog의 이름 오름차순이다.
- set의 size()를 출력하여 1'0000인지 확인하라.
- 만일 size == 1'0000이라면 어느 원소가 중복되었는지 출력하라.

다음 주 강의실에서 만나요!