STL - 8주 1일

Wulong

8주 강의 2020. 5. 4 ~ 2020. 5. 8	
지난 주 4.27 ~ 5.1	**STL 반복자** 반복자 range-based for 반복자 요구사항 구현 iterator_traits 제네릭 함수 컨테이너 어댑터
이번 주 5.4 ~ 5.8	**STL 컨테이너** map map에 원소 추가 연관 배열로서의 map String을 key로 사용하기 multimap set mutiset
다음 주 5.11 ~ 5.15	**STL 컨테이너** 강의실에서 강의예정입니다. **과제 설명** **시험**

지난 주

안녕하세요? STL 수강생 여러분!
지난 주에는 컨테이너와 알고리즘을 이어주는 반복자를 파헤쳐 봤습니다. 반복자를 이해하기 위한 핵심 내용이지만 어려워 잘 다루지 않는 것들이다 보니 바로 이해하기 쉽지 않았다는 이야기가 많았습니다. 이 강의자료만 공부하면 안 됩니다. 기본 내용을 먼저 공부하고 관련 내용 찾아보면서 따라해 보세요. 7주 1일 자료에 틀린 곳이 있어 학생들 지적을 받았습니다. 꼼꼼히 따라 해 보고 잘못된 곳을 찾아 알려준 학생들에게 감사합니다.

이 자료는 직접 코딩하면서 평소 내가 강의하는 방식으로 흐름을 따라가며 만들고 있습니다. 자료를 일단 만들고 나서 처음부터 다시 읽어나가며 맞춤법을 고치고 중요한 곳은 밑줄로 **강조**하고 전체 모양도 **시원하게** 보이도록 칸도 조정합니다. 지난 주 자료는 학생들이 질문하기 좋게 코드에 번호를 처음 붙여 봤습니다만 처음 해 본 일이라 번호가 틀리는 실수를 했습니다. 지금까지 강의자료에서 지적받은 곳이 없어 천만다행이라고 생각하고 있었지만 역시 그럴리는 없었습니다. 어디에서 또 잘못된 곳이 있을 겁니다. 의심해보고 잘못된 곳을 발견했다면 얼른 알려주세요. 틀린 곳 찾는 게임은 아니지만 class String에도 틀린 부분이 있으며 해설한 동영상 내용에서도 잘못된 설명이 있습니다. 다 맞는다고 생각하지 마세요.

알고리즘 함수 sort에서 사용가능한 반복자가 되도록 String_Iterator를 랜덤반복자로 만드는 부분은 문서로 계속 적어서 설명할 필요까지는 없어 [실습]으로 남겼습니다. 강의하다가 시간이 남고 학생들 요청이 있으면 모든 연산자를 구현하기도 했었던 생각이 납니다. 이걸 구현하면 오류 메시지를 읽으며 하나씩 해결하는 재미가 있습니다. 또 7주 2일의 제네릭 함수 구현에서도 String을 삽입반복자를 사용하여 원소를 넣을 수 있도록 프로그램할 수도 있습니다. 그런데 이걸 구현하는 것은 랜덤반복자를 만드는 것보다 조금 더 어려워 [고급]이라는 이름을 붙여 남겼습니다. 전체 흐름을 이해하길 바라는 것이고 이 부분까지 구현해 볼 필요는 없습니다만 혹시 이 부분을 해 보다가 안 되는 학생의 질문은 환영합니다.

7주 2일의 제네릭 함수에서는 알고리즘 함수가 어떤 식으로 만들어진 것인지 살펴보았습니다. 반복자가 어떻게 동작하는지 알면 알고리즘 함수를 이해하는데 큰 도움이 됩니다. C++의 모든 알고리즘 함수는 소스가 공개되어 있습니다. 물론 비주얼 C++의 소스를 제대로 이해하는 것은 그리 쉽지 않습니다. 또 표준에서 알고리즘 함수를 이렇게 만들어야 한다고 규정하지도 않습니다. 표준에서는 알고리즘이 지켜야 할 복잡도 기준만 적혀있을 뿐입니다. 그렇더라도 알고리즘 소스를 보는 것은 공부에 도움이 됩니다. 여러분이 즐겨 찾아봐야할 곳은 https://en.cppreference.com/w/ 입니다. 하루에 한 번 이상 방문해 보세요.

이번 주

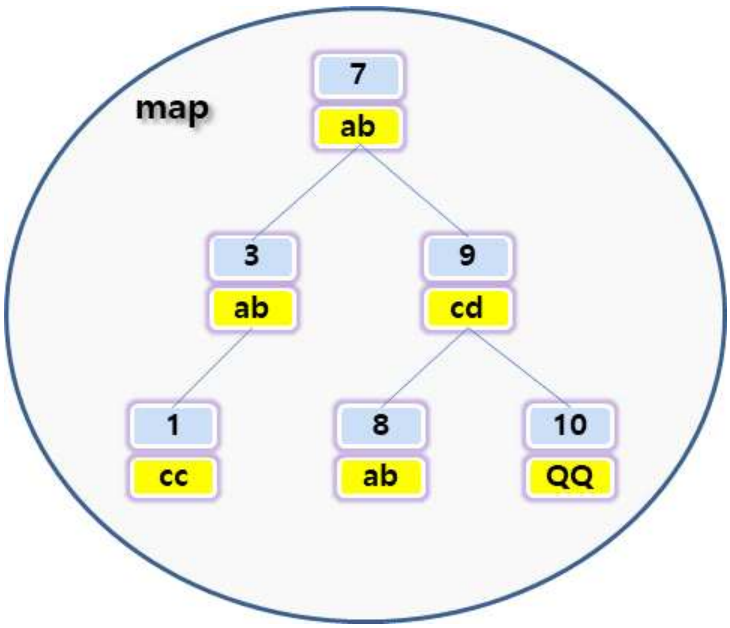
STL **연관 컨테이너(Associative container)**인 **map**과 **set**을 알아봅니다. 연관이라는 말은 저장하려고 하는 자료와 key가 서로 관련이 있다는 의미입니다. map은 **pair<key, vlaue>**를 원소로 저장하는데 key를 기준으로 정렬하여 저장합니다. set은 key를 기준으로 원소를 정렬상

태로 저장합니다만 value가 따로 있지는 않습니다.

연관 컨테이너는 찾고자 하는 원소를 빨리 찾기 위해 사용합니다. 여기서 빠르다는 것은 시퀀스 컨테이너보다 빠르다는 의미이며 연관 컨테이너에서 원소를 찾는 데 걸리는 시간 복잡도는 $O(\log_2 n)$입니다. 시퀀스 컨테이너에서는 원소를 삽입하는데 $O(1)$ 시간이 소요되는데 비하여 연관컨테이너는 원소를 삽입하는데 기본 로그시간 복잡도를 갖습니다. 그러니 연관 컨테이너는 원소가 항상 정렬된 상태가 되도록 하는 데 시간을 더 써서, 찾는데 드는 시간을 단축하는 것입니다. map의 사용법을 알아보고 String을 key로 사용해 보겠습니다. set은 map의 단순 버전이라고 생각하면 되니 map을 주로 살펴보겠습니다.

map

책 197~248



이 그림에서 숫자가 key이고 문자가 value입니다. 그림은 트리구조로 map을 보이지만 표준에서 map을 이런 식으로 구현해야 한다는 내용은 물론 없습니다. 웹의 설명을 보면 보통 이진탐색 트리의 일종인 red-black 트리로 map을 구현한다고 합니다만 나는 아무 관심이 없습니다. 그런가 할 뿐입니다.

map의 key는 유일(unique)해야 합니다. 중복된 key를 사용할 수 없습니다. 같은 key 값을 갖는 원소를 저장하고 싶다면 multimap을 사용하면 되겠습니다. map은 key를 이용해 다른 자료형인 value를 빠르게 검색할 수 있는 연관 컨테이너입니다. map은 사전을 STL로 구현한 것이라고 생각하면 되겠습니다. 요즘은 그냥 단어를 치면 뜻이 나오니 여러분이 사전을 사용

해 봤는지 모르겠습니다. 사전은 표제어를 기준으로 정렬되어 있어서 빠르게 원하는 단어의 설명을 찾아볼 수 있습니다. 이런 이유로 map<key, value>를 다른 언어에서는 dictionary로 쓰기도 하고 hash table 또는 table이라는 단어도 씁니다만 통칭 associative array라고 합니다. 꼭 알아야 할 내용을 코딩하면서 살펴보겠습니다.

map에 원소 추가

지금까지 수많은 아이돌 그룹이 활동했고 지금도 활동하고 있습니다. 아이돌 그룹의 이름을 치면 몇 명인지와 구성원들의 이름이 나오는 프로그램을 map으로 한 번 만들어 보겠습니다. 아이돌 그룹의 이름을 key로 사용하고 value는 class를 새로 만들어 사용하면 되겠습니다. 기능을 보이는 것이 목적이므로 최소한으로 작성하겠습니다. 코드를 먼저 만들고 설명합니다.

```
// 코드 8-1: map에 원소 추가

// #include 생략
#include <map>
using namespace std;

enum class Gender { Boy, Girl, Hybrid };
string Genders[] { "보이그룹", "걸그룹", "혼성그룹" };

class IdolGroup {
    string groupName;                   // 그룹 이름
    Gender gender;                      // 성별
    int year;                           // 데뷔
    int num;                            // 인원 수
    string member;                      // 멤버들

public:
    explicit IdolGroup( string gn, Gender g, int y, int n, string m  )
            : groupName { gn }, gender { g }, year { y }, num { n }, member { m } { }

    void show( ) const {
        cout << "이름:" << left << setw( 14 ) << groupName << setw( 8 )
             << Genders[static_cast<int>(gender)] << ", "  << year << "년, "
             << num << "명, 멤버 - " << member << endl;
    }
};

int main( )
{
    map<string, IdolGroup> idolGroups;

    // 대한민국의 아이돌 그룹 목록 - 위키백과에서
```

```
        idolGroups.insert( pair<string, IdolGroup>( "펄 시스터즈",
                IdolGroup( "펄 시스터즈", Gender::Girl, 1968, 2, "배인순" ) ) );

        idolGroups.insert( make_pair( "송골매",
                IdolGroup( "송골매", Gender::Boy, 1979, 6, "배철수" ) ) );

        auto koyote = make_pair( "코요태",
                IdolGroup( "코요태", Gender::Hybrid, 1998, 3, "김종민 신지 빽가" ) );
        idolGroups.insert( idolGroups.begin(), koyote );

        idolGroups.emplace(  "마마무",
                IdolGroup( "마마무", Gender::Girl, 2014, 4, "솔라 문별 휘인 화사" ) );

        idolGroups.insert_or_assign( "있지",
                IdolGroup( "있지", Gender::Girl, 2019, 5, "예지 리아 류진 채령 유리" ) );

        // 순회하며 출력
        for ( const auto& ig : idolGroups )
                ig.second.show( );
    }
```

간단하지만 그럴듯하게 만들어보려니 코드가 조금 길어졌습니다. map을 생성하려면 클래스 템플릿의 인자로 key와 value의 자료형을 넘겨야 합니다.

```
    map<string, IdolGroup> idolGroups;
```

코드 8-1에서 key는 string, value는 IdolGroup인 idolGroups라는 이름의 비어있는 map을 만들었습니다.

이 맵에 원소를 추가합니다. 서로 다른 방법으로 원소를 추가할 수 있음을 보이기 위해 다른 멤버 함수를 사용하였습니다. map은 원소를 추가하는데 걸리는 시간에 예민하지 않습니다. 어떤 것을 사용하더라도 문제될 것이 없습니다. (정렬된 데이터를 map에 읽어오는 경우에는 더 빠르게($O(1)$) 원소를 추가할 수도 있다) **map의 원소는 pair<key, value>**이므로 항상 pair 객체를 만들어 원소를 추가해야 한다는 것을 주의하면 됩니다. 물론 map에도 pair 객체를 직접 생성하는 emplace 멤버도 있습니다.

map은 원소를 추가하거나 삭제할 때 key값을 기준으로 합니다. 원소를 삭제하는 경우를 예로 들어 봅니다. 당연히 remove 알고리즘으로 삭제하면 안 됩니다. 전용함수를 사용합니다.

```
        idolGroups.erase( "트와이스" );
```

이 문장은 idolGroups에서 “트와이스”라는 key가 있나 검색하여 만일 있다면 “트와이스”와 연관된 IdolGroup 원소를 idolGroups에서 제거합니다.

프로그램을 실행시켜 보겠습니다.

그룹 이름을 key로 사용하였기 때문에 이름 기준 오름차순으로 정렬된 출력이 나옵니다. 출력문에서 idolGroups의 원소는 pair입니다. 그러니까 ig는 pair<key, value>입니다. pair의 second가 IdolGroup이니 여기에 show()를 호출할 수 있겠습니다. 혹시 지난 시간의 구조체 바인딩 기억하나요? 그럼 이 문장을 조금 더 편하게 쓸 수도 있겠습니다.

```
for ( const auto& ig : idolGroups )
        ig.second.show( );
```

위의 출력화면 잠시 감상해 보세요. 내가 좋아하는 그룹이 없어 섭섭한가요?

뭐라구요? "**있지**" 멤버 이름이 잘못되었다구요?
아이구! 죄송합니다. 찐팬을 몰라봤습니다.
찾아보니 "유리"가 아니고 "유나"네요.
반성합니다. 얼른 고치겠습니다.

위의 프로그램에서 멤버 이름을 "유나"라고 바꾸면 안 됩니다.
상황은 이렇습니다.
지금 map에 모든 데이터가 들어가 있는데 그 중에 특정 원소의 값을 변경하려고 하는 겁니다. 프로그램이 이미 실행되고 있는데 "있지"의 멤버 이름을 바꾸려는 상황인 것이죠.

map을 순회하며 출력하기 전에 제대로 된 데이터를 idolGroups에 다시 삽입해 봅니다.

```
idolGroups.insert( make_pair( "있지",
        IdolGroup( "있지", Gender::Girl, 2019, 5, "예지 리아 류진 채령 유나" ) ) );
```

이후 map을 출력해보면 데이터가 바뀌지 않습니다. 왜 그런지 알겠습니까?

map은 key를 기준으로 동작합니다. "있지"라는 이름의 key가 있기 때문에 insert 동작이 거부되어 그런 것입니다. 이런 경우에 사용할 수 있는 멤버함수가 바로 insert_or_assign입니다. 단어 뜻 그대로 동작합니다. 제대로 고치고 싶으면 위의 insert 멤버 대신 이걸 사용합니다. C++17 이전에는 이런 동작을 하려면 key를 갖는 원소가 있는지 검색해서 있다면 원소를 제거하고 새 원소를 끼워 넣어야 했습니다. 그래서 이런 함수가 표준에 추가된 것입니다.

```
idolGroups.insert_or_assign( "있지",
        IdolGroup( "있지", Gender::Girl, 2019, 5, "예지 리아 류진 채령 유나" ) );
```

이제 idolGroups의 전체 원소를 출력하면 고친 내용이 출력됩니다.

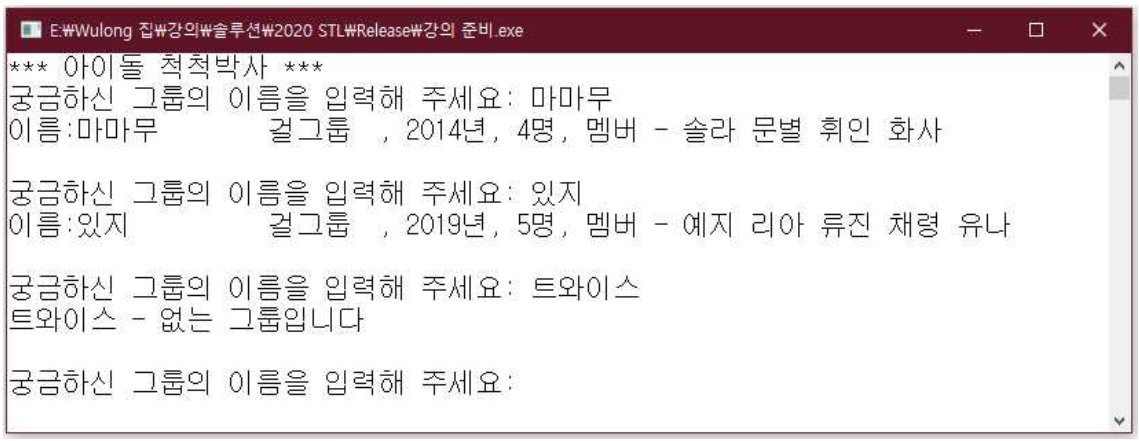
map에 원소를 추가하는 것에 익숙해졌다면 이제 map을 사용하는 진짜 목적인 빨리 찾기를 해 볼 차례입니다. map을 왜 사용한다구요? 그렇죠. 어떤 값을 빨리 찾으려고 사용합니다. idolGroups에서 그룹이름을 key로 사용하여 원하는 IdolGroup을 찾으려고 하는 것이죠. 빨리 찾으려고 원소들을 순서대로 저장하고 있는 것이거든요. 그런데 이건 집고 가야겠습니다. 지금 idolGroups의 원소는 5개뿐이라 진짜 찾기 실력을 보여줄 수 없다는 점을요. 원소의 수가 많을수록 진짜 실력이 나온다는 점을 알아주세요.

프로그램을 찾기 구조로 바꿔보겠습니다. 코드 8-1의 순회하며 출력하는 부분을 없애고 다음과 같이 무한루프로 바꿔봅니다.

```
// 코드 8-2: map에서 원소 찾기

int main()
{
    // 생략

    cout << "*** 아이돌 척척박사 ***" << endl;

    while ( true ) {
            cout << "궁금하신 그룹의 이름을 입력해 주세요: ";
            string name;
            cin >> name;

            auto p = idolGroups.find( name );
            if ( p != idolGroups.end( ) )
                    p->second.show( );
            else
                    cout << name << " - 없는 그룹입니다" << endl;

            cout << endl;
    }
}
```

알고리즘 함수 find를 사용해서 원하는 원소를 찾으려고 하면 안 됩니다. 전역 find 함수는 선형 알고리즘이며 $O(n)$ 복잡도를 갖습니다. 시간을 들여 원소들을 map에 넣은 이유를 잊으

면 안 됩니다. map의 멤버함수 find를 사용하여 원하는 원소를 찾아야 합니다. 찾고자 하는 key를 find에 넘기면 find는 반복자를 리턴합니다. 찾는 key가 없으면 end()가 리턴되며 key가 있으면 그 key와 연관된 원소를 가리키는 반복자가 리턴됩니다. map의 원소는 pair이므로 찾는 원소는 pair의 second가 됩니다. 실행결과는 아래 화면을 참고하세요. IdolGroup 자료가 많이 있다면 쓸 만한 프로그램이 될 수 도 있을 것입니다.



조금 다른 각도에서 생각해 보겠습니다.
코드 8-1에서 입력한 것과 같이 모든 원소가 map에 저장되어 있는 상태에서 시작합니다.
IdolGroup 5개가 idolGroups에 저장되어 있습니다.
map을 순회하며 출력하면 key 오름차순으로 출력됩니다.
그런데 이렇게 말고 데뷔년도 기준으로 출력되면 좋겠습니다.
아래 화면 캡쳐와 같은 식으로 말이죠. 어때요. 이렇게 할 수 있겠습니까?

지금 map idolGroups에서 이런 일이 가능합니까? 그렇지 않습니다.
map의 자료를 다른 컨테이너로 옮겨야 할 겁니다.
정렬기준 때문에 class IdolGroup도 손 봐야 할 겁니다.
그런데 말이죠!
이런 작업이 STL에서는 정말로 간단하면서 쉽습니다.
여러분도 금방 할 수 있습니다.

[실습] idolGroups의 원소들을 그림과 같이 데뷔년도 기준 오름차순으로 출력하라. (20분)

연관 배열로서의 map

map을 연관 배열이라고 부르면 말장난 같기도 합니다. 연관 배열로 검색해서 잠깐 살펴보세요. 여기에서 일부러 이런 제목을 붙인 것은 그냥 배열과 다른 점을 설명하기 위함이고 특정 문제해결에 유용한 면이 있어서 입니다. 간단한 설명과 프로그램을 통해 알아보겠습니다.

프로그램 언어에서 배열이라 하면 무엇이 떠오르나요? 같은 타입의 원소를 여러 개 저장한다는 것과 함께 메모리가 연속되어 있다는 것이 떠오를 겁니다만 한 가지가 더 있습니다. 배열은 특별한 기호를 제공합니다. 바로 **[ ]** 입니다. 이 연산자는 보통 **subscript operator** 또는 **array index operator**라고 부르며 특별한 사용법이 정해져 있습니다. 바로 이 연산자 안에 **정수**를 써서 원하는 원소에 접근하는 것입니다.

```
int a[10];
cout << a[0] << a[9] << endl;
```

코드는 그냥 한 번 써 봤습니다. 여기에서 배열의 index로 사용되는 숫자 n을 이렇게 생각해보면 어떨까요? n을 key로 해서 int라는 value에 억세스한다고요! 분명히 그렇게 생각할 수 있겠죠? 그렇다면 지금까지 알아본 map과 뭔가 비슷하지 않습니까? 여기에서 제목으로 바로 이어가 보겠습니다.

배열과 같이 map도 key를 index로 사용하여 value에 억세스할 수 있습니다. 그런데 다른 점이 있습니다. 네. 물론 메모리가 연속되어 있지 않다는 것이 다른 점이지만 이것은 지금 이야기와 아무 관련이 없습니다. 다른 점은 바로 map에서는 **어떤 자료형이라도 key로 사용할 수 있다**는 것입니다. 그리고 map은 class로 만든 자료형입니다. 연산자 오버로딩이 가능하다는 말입니다. 그럼 [ ] 연산자도 당연히 오버로딩할 수 있습니다. 그렇다면 바로 이런 표현이 가능합니다.

```
map[key] = value;
```

바로 이렇게 배열 연산자를 사용할 수 있다는 점에서 이 항목의 제목을 적은 것입니다. 처음 보면 조금 묘하게 보일 수 있습니다만 실제 사용하면 편리하며 기능이 막강합니다. 이것을 사용하여 간단한 전화번호부를 만들어 보겠습니다. 역시 코드가 이해하기 쉽습니다.

```
// code 8-3: 전화번호부

int main( )
{
    map<string, string> phoneBook;

    phoneBook[ "번호안내" ] = "114";
    // phoneBook.operator[]( "번호안내 " ) = "114";

    phoneBook[ "일기예보" ] = "131";
```

```
    phoneBook[ "교통정보" ] = "1333";
    phoneBook[ "전기고장신고" ] = "123";
    phoneBook[ "사이버테러" ] = "118";

    for ( const auto& [name, number] : phoneBook )
        cout << name << ": " << number << endl;
}
```

[ ] 연산자 사용법이 신선한가요? 이것은 그냥 사용법을 간단하게 보이는 예에 불과합니다. 앞에서 map의 **insert_or_assign** 멤버를 설명했는데, [ ] 연산자도 같은 방식으로 동작합니다. 사실은 [ ] 연산자가 먼저 나왔습니다.

[ ] 연산자를 사용한 진짜 좋은 프로그램을 만들어 보겠습니다.

파일을 읽어 어떤 단어가 몇 번 사용되었나를 출력하는 프로그램입니다. 줄이면 **단어 사용횟수를 출력**하는 프로그램 되겠습니다.

```
// 코드 8-4: 단어 사용횟수 출력

#include <iostream>
#include <map>
#include <fstream>
#include <string>
using namespace std;

int main( )
{
    cout << "읽을 파일은? ";
    string name;
    cin >> name;

    ifstream in( name );

    if ( !in ) {
        cout << name << " - 파일 열기 실패" << endl;
        return 0;
    }

    map<string, int> words;

    string str;

    while ( in >> str )
        words[ str ]++;                    // 이 문장을 함수 호출로 바꿔볼 것

    // 출력
```

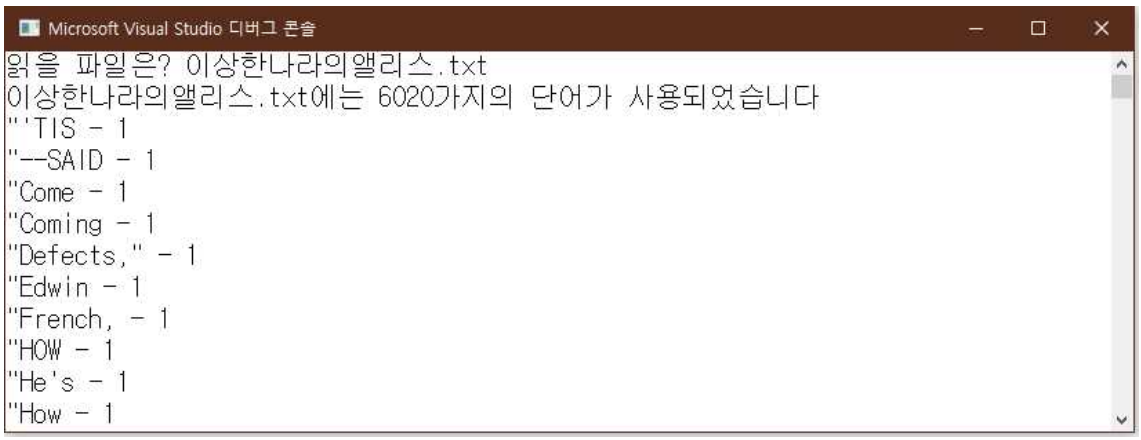
```
        for ( const auto& [단어, 출현횟수] : words )
                cout << 단어 << " - " << 출현횟수 << endl;
    }
```

while 문 안의 이 문장이 핵심입니다.

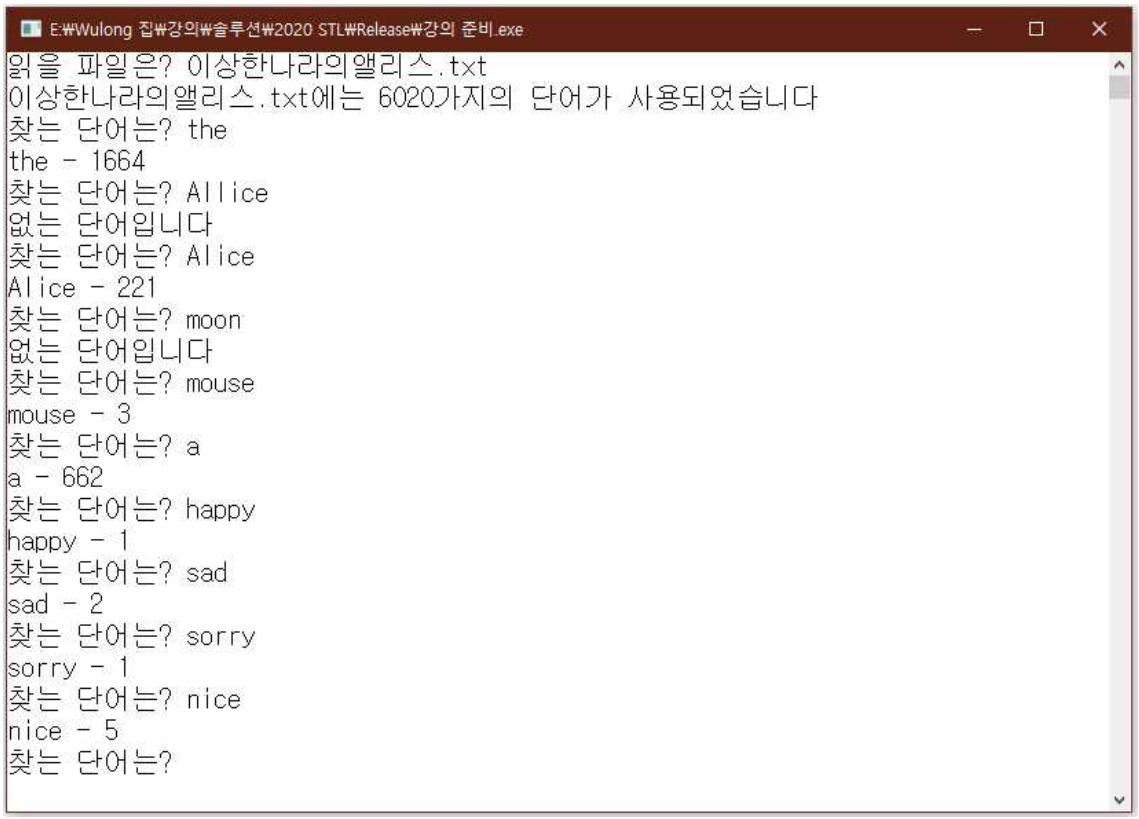
```
        words[ str ]++;                    // 이 문장을 함수 호출로 바꿔볼 것
```

key인 string을 찾으면 value인 int의 값을 증가시킵니다. 다음 화면 캡처는 "소스.cpp"에 출현하는 단어의 횟수를 출력한 창의 일부입니다.

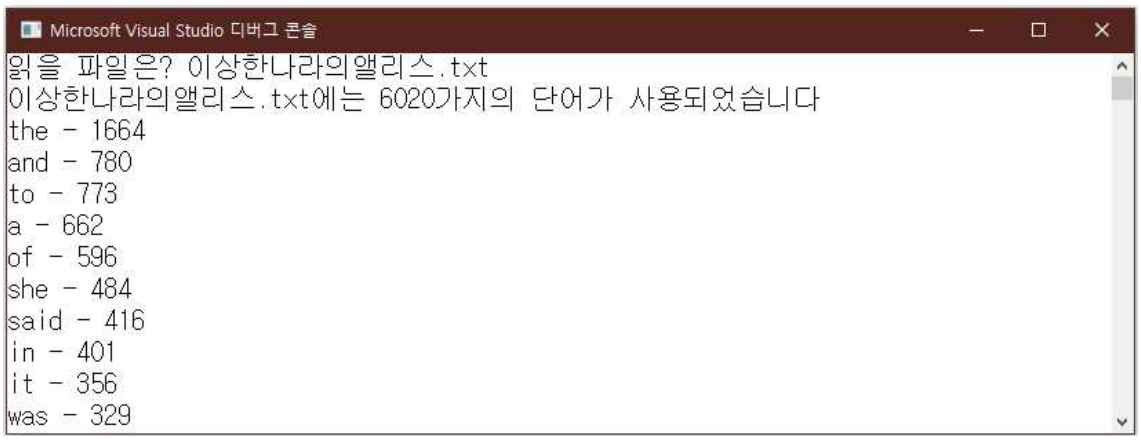
조금 더 큰 파일을 읽어 보겠습니다. 이 자료와 같이 올려놓은 이상한 나라의 앨리스 파일을 다운받아 사용해 주세요. 이 파일은 이미 저작권이 만료된 것이니 마음대로 사용해도 됩니다. 구분되는 단어의 수는 6020개입니다. 앞에서 10개를 출력해 봤습니다.



[실습] map의 찾기 실력을 활용할 수 있도록 읽은 파일에서 특정 단어가 몇 번이나 사용되었는지 확인하는 프로그램을 작성하라. 다음과 같이 실행되도록 해 보자. (10분)



[실습] 읽은 파일에서 가장 많이 사용된 단어부터 단어와 출력횟수를 모두 출력하라. 앞에서부터 10개를 출력한 출력화면 캡처를 참고하라. (20분+)



마지막 문제 제대로 해결했다고 확신하십니까? 오늘 강의는 여기까지 입니다.