



# ALOHA

## #특별 자료

알아두면 좋은 것들

# #CH.1

```
#include <bits/stdc++.h>
```



# #include <bits/stdc++.h>

백준이나 코드포스 등에서 다른 사람이 제출한 코드를 보면

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const double eps = 1e-9;
```

“#include <bits/stdc++.h>”  
을 자주 볼 수 있다.

이 헤더파일의 정체는 뭘까?

# #include <bits/stdc++.h>

bits/stdc++.h 헤더파일을 열어보면...

#include <~~~> 가 엄청 많다!

즉, 우리는 bits/stdc++.h 헤더파일만 include하면,  
저 많은 헤더파일을 **한번에** include할 수 있는 것이다!

특히, 우리가 **자주 쓰는 헤더파일들도 모두 포함되어 있다.**

```
#include <algorithm>
#include <deque>
#include <iostream>
#include <queue>
#include <set>
#include <string>
#include <vector>
```

```
1  #ifndef _GLIBCXX_NO_ASSERT 50  #include <memory>
2  #include <cassert>        51  #include <new>
3  #endif                    52  #include <numeric>
4  #include <cctype>          53  #include <ostream>
5  #include <cerrno>          54  #include <queue>
6  #include <cfloating>       55  #include <set>
7  #include <ciso646>         56  #include <sstream>
8  #include <climits>         57  #include <stack>
9  #include <clocale>         58  #include <stdexcept>
10 #include <cmath>           59  #include <streambuf>
11 #include <csignal>         60  #include <string>
12 #include <csignal>         61  #include <typeinfo>
13 #include <csignal>         62  #include <utility>
14 #include <csignal>         63  #include <valarray>
15 #include <csignal>         64  #include <vector>
16 #include <csignal>         65
17 #include <csignal>         66  #if __cplusplus >= 201103L
18 #include <csignal>         67  #include <array>
19                             68  #include <atomic>
20                             69  #include <chrono>
21 #include <complex>         70  #include <condition_variable>
22 #include <complex>         71  #include <forward_list>
23 #include <complex>         72  #include <future>
24 #include <complex>         73  #include <initializer_list>
25 #include <complex>         74  #include <mutex>
26 #include <complex>         75  #include <random>
27 #include <complex>         76  #include <ratio>
28 #include <complex>         77  #include <regex>
29 #include <complex>         78  #include <scoped_allocator>
30 #endif                     79  #include <system_error>
31                             80  #include <thread>
32 // C++                     81  #include <tuple>
33 #include <algorithm>        82  #include <typeindex>
34 #include <bitset>          83  #include <type_traits>
35 #include <complex>         84  #include <unordered_map>
36 #include <deque>           85  #include <unordered_set>
37 #include <exception>       86  #endif
38 #include <fstream>         87
39 #include <functional>
40 #include <iomanip>
41 #include <iostream>
42 #include <iostream>
43 #include <iostream>
44 #include <iostream>
45 #include <iostream>
46 #include <iostream>
47 #include <iostream>
48 #include <iostream>
49 #include <iostream>
```

# #include <bits/stdc++.h>

정리해보자!

장점

bits/stdc++.h 헤더파일을 사용하면 **표준 헤더파일을 한 번에 모두 불러올 수 있다.**  
함수가 어떤 헤더파일에 있는지 **고민할 필요가 없다.**

단점

불필요한 헤더파일을 컴파일하게 되는 경우가 많다.  
GNU C++의 **표준 라이브러리 헤더가 아니다.** (백준에는 제출해도 됩니다)

# #CH.2

solved.ac



# solved.ac

---

solved.ac는 백준 온라인 저지에 여러 가지 편의를 더한 사이트이다!

제공하는 기능으로는...

**문제마다 레벨(티어)/알고리즘 제공** (해결한 사람들의 투표로 결정)

**Class 기능 제공** (문제 추천)

**계정에 레벨 제공** (해결한 문제의 레벨이 높을 수록, 풀 문제가 많을 수록, Class가 높을 수록 아이디의 레벨이 올라감)

**라이벌 기능 제공** (사실상 친구추가 기능)

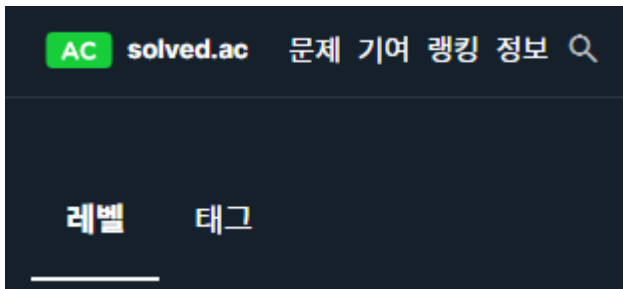
**해결한 문제 분석 기능** (각 레벨마다 해결된 문제, 각 알고리즘마다 해결된 문제)

**계정의 히스토리 제공**

# solved.ac

문제마다 레벨(티어)/알고리즘 제공 (해결한 사람들의 투표로 결정)

브론즈 -> 실버 -> 골드 -> 플래티넘 -> 다이아 -> 루비 순으로 올라가며, 숫자(1~5)가 낮을 수록 높은 레벨  
ex) 골드5 < 골드1 < 플래티넘5 <<< 루비1 (롤이랑 비슷)



상단 바에서 “문제”를 클릭하면, 레벨, 태그(알고리즘)로 문제가 분류됨



# solved.ac

## Class 기능 제공 (문제 추천)


Class는 1~10의 숫자와 +, ++ 옵션이 있다.

$1 < 1+ < 1++ < 2 < 2+ < 2++ < \dots < 9++ < 10 < 10+ < 10++$

왼쪽에서 오른쪽으로 갈 수록 Class가 높아진다.

오른쪽 그림은 Class 7을 획득하기 위한 문제들의 일부이다.

(월 풀지 모르겠으면 여길 공략한다.)



정렬 —	ID	레벨 ↓	제목	푼 사람 수	평균 시도	시프트 마음대로
#			제목		해결	평균 시도
1	3878	👤	점 분리		197	3.40
1	4008		특공대 <b>ESSENTIAL</b>		356	2.71
1	13548		수열과 쿼리 6 <b>ESSENTIAL</b>		332	2.93
2	1126		같은 탑 <b>ESSENTIAL</b>		900	4.24
2	8462		배열의 힘		239	2.94
2	10277		JuQueen		92	2.88
2	13263		나무 자르기 <b>ESSENTIAL</b>		286	2.35
2	14897		서로 다른 수와 쿼리 1		163	3.21

# solved.ac

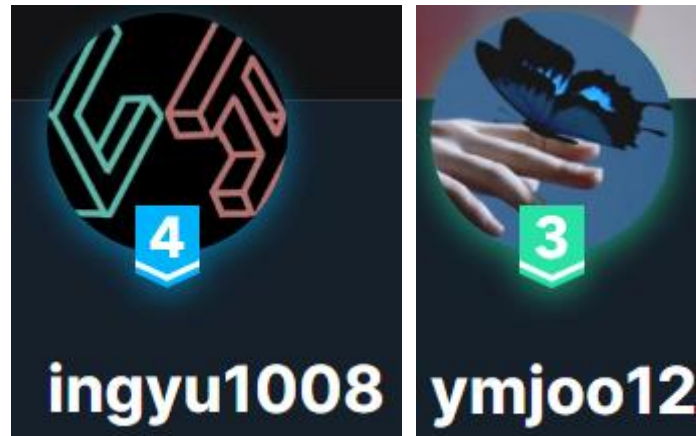
계정에 레벨 제공 (해결한 문제의 레벨이 높을 수록, 풀 문제가 많을 수록, Class가 높을 수록 아이디의 레벨이 올라감)

계정도 문제와 마찬가지로 레벨 제공

브론즈 -> 실버 -> 골드 -> 플래티넘 -> 다이아 -> 루비 순으로 올라가며, 숫자(1~5)가 낮을 수록 높은 레벨

ex) 골드5 < 골드1 < 플래티넘5 <<< 루비1

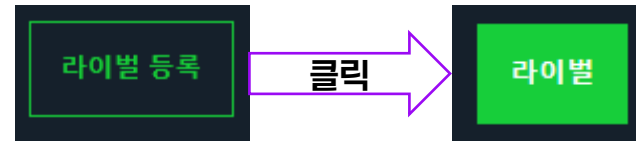
예를 들면, 왼쪽 아이디는 다이아4 레벨, 오른쪽 아이디는 플래티넘3 레벨



# solved.ac

## 라이벌 기능 제공

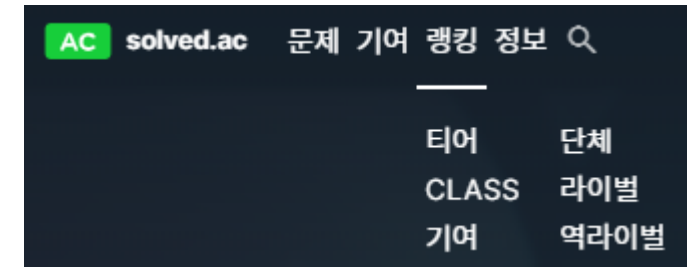
다른 사람들과 라이벌을 설정할 수 있음



라이벌로 설정하게 되면,

랭킹 -> **라이벌**에서 내가 라이벌로 둔 사람을 볼 수 있고,  
랭킹 -> **역라이벌**에서 나를 라이벌로 둔 사람을 볼 수 있다.

(ALPHA에서 친해지면 가장 먼저 solved.ac 라이벌을 신청한다. ALPHA식 친구추가)

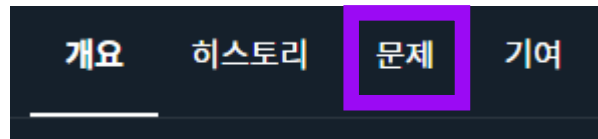


# solved.ac

**해결한 문제 분석 기능** (각 레벨마다 해결된 문제, 각 알고리즘마다 해결된 문제)

**어떤 레벨의 문제들을 주로 해결했는지, 어떤 알고리즘을 주로 해결했는지 분석하는 기능이 있다.**  
(본인 혹은 타인의 프로필에 들어가면 확인할 수 있다.)

해결한 문제는 **여기서** 볼 수 있다.

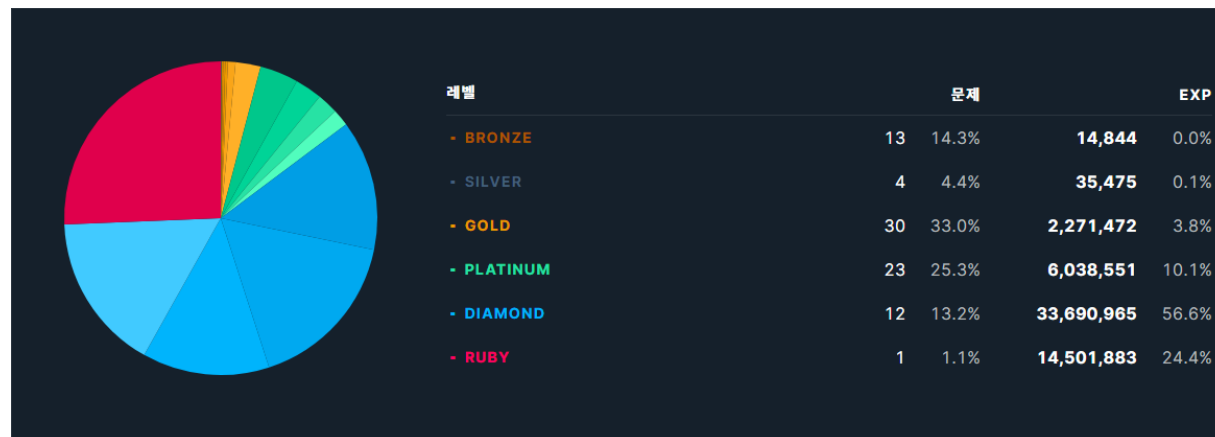


# solved.ac

**해결한 문제 분석 기능** (각 레벨마다 해결된 문제, 각 알고리즘마다 해결된 문제)

**어떤 레벨의 문제들을 주로 해결했는지, 어떤 알고리즘을 주로 해결했는지 분석하는 기능이 있다.**  
(본인 혹은 타인의 프로필에 들어가면 확인할 수 있다.)

**왼쪽의 원형 그래프는 레벨별로 얻은 경험치,  
오른쪽 표에는 해결한 문제 수와 경험치가 적혀 있다.**



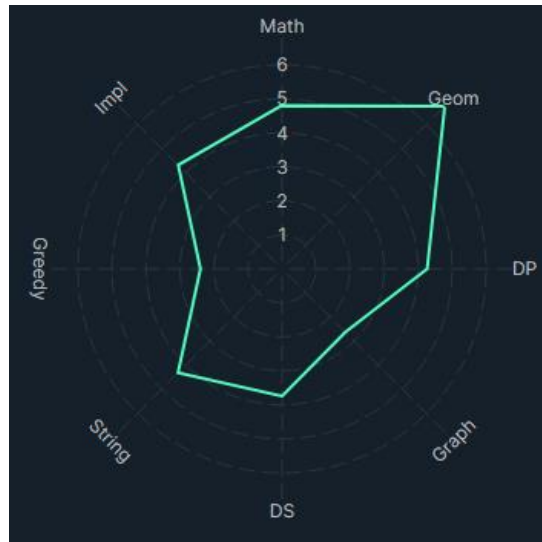
# solved.ac

**해결한 문제 분석 기능** (각 레벨마다 해결된 문제, 각 알고리즘마다 해결된 문제)

**어떤 레벨의 문제들을 주로 해결했는지, 어떤 알고리즘을 주로 해결했는지 분석하는 기능이 있다.**  
(본인 혹은 타인의 프로필에 들어가면 확인할 수 있다.)

**아래 그래프에서는 어떤 알고리즘들을 주로 해결했는지 나온다.**

Math: 수학  
Geom: 기하  
DP: 동적계획법  
Graph: 그래프  
DS: 자료구조  
String: 문자열  
Greedy: 그리디  
Impl: 구현



# solved.ac

해결한 문제 분석 기능 (각 레벨마다 해결된 문제, 각 알고리즘마다 해결된 문제)

어떤 레벨의 문제들을 주로 해결했는지, 어떤 알고리즘을 주로 해결했는지 분석하는 기능이 있다.  
(본인 혹은 타인의 프로필에 들어가면 확인할 수 있다.)

아래 표에서는 어떤 태그(알고리즘)의 문제를 얼마나 풀었는지 볼 수 있다.

태그	문제		EXP	
- 기하학	35	38.5%	46,574,538	78.3%
- 블록 꺾질	22	24.2%	37,246,370	62.6%
- 회전하는 캘리퍼스	7	7.7%	27,659,536	46.5%
- 이분 탐색	3	3.3%	18,394,020	30.9%
- 수학	27	29.7%	6,953,074	11.7%
- 임의 정밀도 / 큰 수 연산	3	3.3%	6,083,913	10.2%

(아 사람은 거하에 미쳐 있습니다!)

# solved.ac

## 계정의 히스토리 제공

시간에 따라 얻은 경험치와 랭킹을 볼 수 있다.

경험치



경험치 랭킹





# #CH.3

Github





# Github

---

깃헙은 너무 길어서 생략

# #CH.4

Visual Studio Code



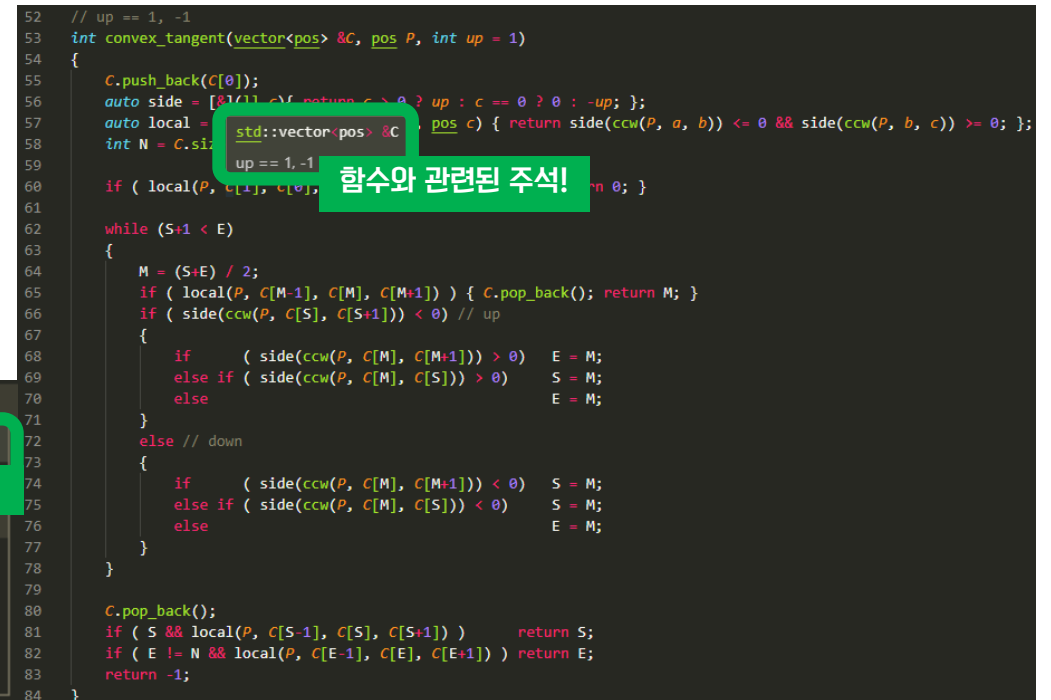
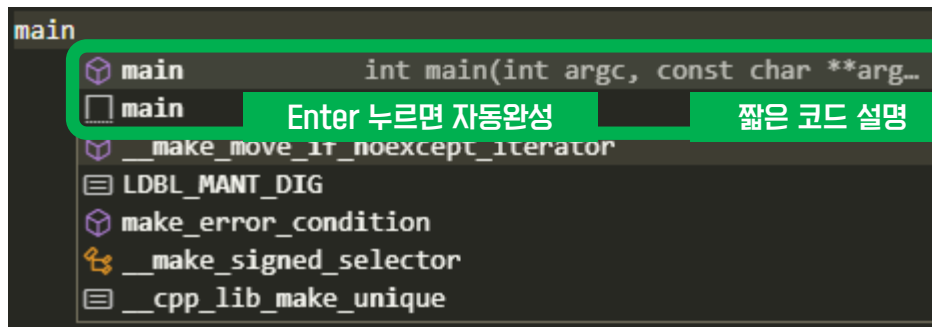
# Visual Studio Code

Visual Studio Code (이하 VSC)는 기능이 다양한 **텍스트 에디터!**

Python, C/C++, Java는 물론 html, css, Javascript 등의 다양한 언어의 **IntelliCode**를 지원한다.  
(IntelliCode: 자동완성/코드 하이라이트 등)

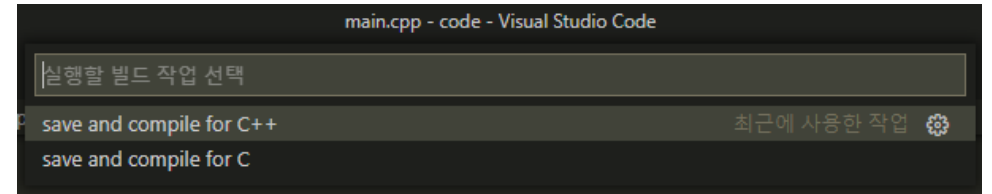
또한, 함수/클래스 등에 마우스를 가져가면,  
해당 함수와 관련된 **주석**이 나온다!

(C++의 IntelliCode)



# Visual Studio Code

또한 C++ 컴파일러나 Python 인터프리터 등을 적용하게 되면 VSC 터미널에서 바로 실행할 수 있다!



이 뿐만 아니라 프로젝트에서 아나콘다 환경을 사용하는 경우, 아나콘다 환경을 적용시키는 것도 가능하다!  
또한 WSL과의 연동도 지원하고 있다! (WSL을 설치하면, 소입설 시간에 VirtualBox를 켤 필요가 없어요! 무~야호!)

# Visual Studio Code

마지막으로, VSC의 가장 강력한 기능으로는, 막강한 단축키가 있다!  
(아래 단축키는 윈도우 기준, 자주 사용되는 단축키만)

Ctrl B	좌측 탐색기(sidebar) ON/OFF
Ctrl D	반복되는 단어 복수 선택*
Ctrl F	검색*
Ctrl G	n번째 줄로 이동
Ctrl ↑	위/아래로 화면 이동
Ctrl ⇐	단어 단위로 커서 이동
Ctrl /	해당 줄을 주석 처리
Ctrl `	터미널 열기 (따옴표 아니고 Tab 위 버튼)

Alt ↓	현재 줄과 위/아래줄 교환
Alt Ctrl ↑	위/아래에 커서 추가
Ctrl Shift ↑	위/아래로 현재 줄 복사
Ctrl Click	마우스 커서 추가

## \* 검색 옵션



Alt C	검색 시 대소문자 비교 ON/OFF
Alt R	정규표현식으로 검색 ON/OFF
Alt W	단어 단위로 검색 ON/OFF

더 자세한 단축키는 [검색](#) 혹은 [파일->기본 설정->바로 가기 키](#)를 참고해주세요~ (단축키가 너무 많아요)

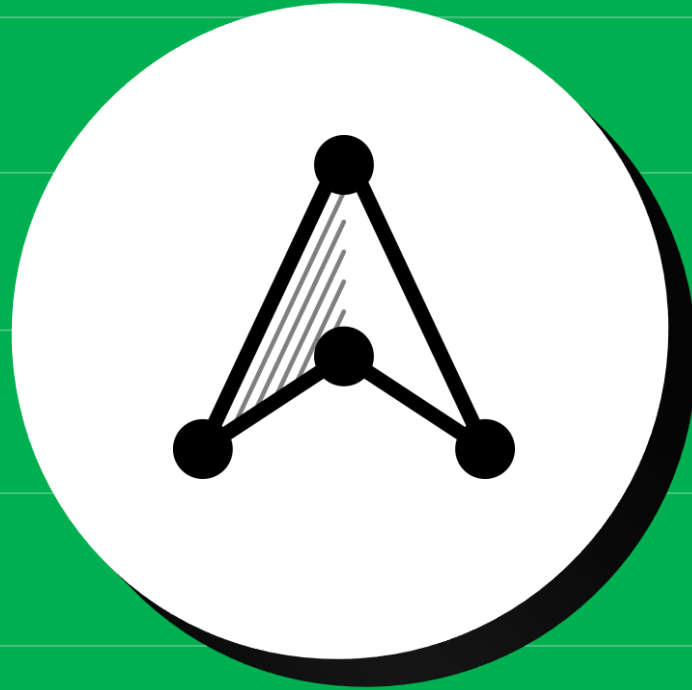


# Visual Studio Code

---

VSC에서 C++ 컴파일러 설치하기

[링크 참고](#)



다음 시간에 만나요~