```
Error: Undefined variable.
   ╷
115 │                 color: $color_accepted;
   │                        ^^^^^^^^^^^^^^^^
   ╵
  resources/user-detail-info.scss 115:28  @import
  resources/style.scss 13:9               root stylesheet
```

[튜토리얼00] 문제를 풀기 전에

시간 제한: 0.1s **메모리 제한:** 512M
Java 8: 0.4s
Python 3: 0.4s

처음 알고리즘에 입문하면서 흔히 하는 실수와 궁금증을 모아보았습니다.

- 맞은 경우(AC): 채점 프로그램은 제출한 소스 코드로부터 얻은 프로그램에 테스트 케이스를 입력하였을 때 나오는 문자열과 정답 문자열을 비교해 일치하면 "Accepted" 판정을 내립니다. 이때, 라인 끝에 출력하는 여분의 공백 하나는 무시됩니다. 즉, 정답이 "1 2"인 경우 "1 2 " 문자열도 허용됩니다. for(i=0;i<n;i++) cout << arr[i] << " "; 와 같은 형태에서 마지막에 여분의 공백이 출력될 수 있는 것을 신경쓰지 않아도 된다는 뜻입니다. 이는 대회를 포함한 모든 온라인 저지의 공통 사항입니다.

- 틀린 경우(WA): 답을 틀렸다면, 문제를 해결하는 알고리즘이 잘못되었을 수도 있고 코드를 잘못 구현했을 수도 있습니다. 만약 주어진 예시조차 틀리게 나온다면, 디버깅을 통해 각 과정별로 변수가 어떤 값을 담고 있나 확인해 보세요. 디버깅은 학부 과정에서는 실습하지 않지만, 코딩을 시작했다면 빨리 배워두는게 좋습니다. 디버깅에 숙달되면 구현 실수정도는 평균 1분이면 잡아낼 수 있습니다. 하지만 예시는 올바르게 나오는데 WA를 받는다면, 문제의 범위 조건 안에서 만들 수 있는 가장 유별나고 사악한 테스트 케이스를 만들어서 대입해 보세요. 자신의 프로그램에 다양한 테스트 케이스를 집어넣어보는 과정은 매우 중요합니다. 범위의 경계에 있는 경계값을 넣어보거나, 프로그램에서 특별히 반례가 될 만한 데이터가 있지는 않을지 생각해 봅시다.

- MLE(Memory Limit Exceeded): 프로그램 내에서 너무 많은 메모리를 할당하지 마세요. 컴퓨터의 메모리는 무한하지 않습니다. MLE의 문구를 보시거나 Java의 OutOfMemoryException을 받았다면, 메모리를 더 적게 사용하여 문제를 해결하는 방법을 생각해 보시기 바랍니다.

- RTE(Runtime Error): 채점 색상이 주황색 계열인 오류는 대부분 런타임에러와 연관이 있으며, Java의 OutOfMemoryException을 제외하면 배열의 인덱스를 벗어나는 경우가 아닌지 확인해 보시기 바랍니다. 런타임 에러의 원인은 다양하기 때문에 콕 찝어 해결책을 제시할 수는 없습니다.

- 허용되는 시간: CPU 클럭은 약 3GHz라는 말을 들어보셨을 겁니다. 이론상 컴퓨터는 초당 약 30억번의 마이크로 연산(micro-operation)을 하고, 매우 간단한 for문은 초당 약 10억번을 돌릴 수가 있습니다. 보통은 for문 안에 다른 연산들도 함께 들어가기 때문에 대략 초당 1억번 정도로 잡습니다. 문제의 시간 제한이 1초라면 약 1억번의 루프를 돌지 않게 주의하는게 좋겠습니다.

- 문제에서 요구하는 범위를 주의깊게 확인하세요. 특히, "입력"과 "출력"란의 경우에는 단 한 글자라도 놓치지 말고 꼼꼼하게 읽도록 합시다. 그리고, 그 제한조건 안에서 만들 수 있는 가장 사악한!! 테스트 케이스가 어떤 경우가 있을지 생각해 봅시다.

- 틀린 소스코드의 채점 상태와 출력 문구를 확인하세요. 특히 CE(컴파일 에러)는 몇 번째 라인에서 무슨 오류가 나왔는지 알려주기 때문에 99%는 바로 잡아낼 수 있고 MLE, TLE, Java의 Error 이름 등을 확인하여 프로그램을 어떻게 수정해야 할 지에 대한 힌트를 얻을 수 있습니다.

- 배열의 끝 주의하기: 원소가 1000개인 배열을 선언할 때 저장할 배열의 인덱스가 확실히 0부터 999라면 n[1000]으로 선언해도 되지만 1부터 1000이라면 n[1001]으로 선언합니다. 간단해 보이지만 적지 않은 학생들이 이 부분에서 틀렸습니다. 좋은 습관은 아예 배열을 항상 넉넉하게 +1로 잡는 것입니다. 변수 하나 정도는 메모리상 큰 차이가 나지 않을 뿐더러 어쩌다가 발생할 수 있는 런타임 에러도 막아줄 수 있기 때문입니다.

- scanf_s: visual studio에서 scanf을 사용하면 오류가 뜨면서 scanf_s를 사용하라고 하는데 이 검사를 끄려면 프로젝트 - [프로젝트] 속성 - C/C++ - SDL 검사를 아니요(/sdl-)로 설정하면 됩니다.

- 스택 오버플로우: 스택은 지역 변수와 함수 호출에 사용되는 메모리입니다. 가끔 스택에 허용되는 크기를 초과하여 스택 오버플로우가 나오기도 하는데 이걸 흔히 볼 수 있는 경우는 C/C++에서 main함수 안에서 배열을 크게 잡았을 때입니다. 이러면 프로그램 실행 즉시 의문의 런타임 에러가 나오지만 입문자들은 코드에 무엇이 잘못되었는지 파악하기가 힘들 수 있습니다. 알고리즘 문제를 풀면서 큰 배열을 선언할 때는 항상 전역으로 선언하는 것이 습관화되는게 좋습니다. 스택 오버플로우를 볼 수 있는 다른 경우는 재귀호출이 무한으로 반복되었을 경우입니다. 재귀함수를 사용했다면 이것도 의심해볼 만합니다.

- 빠른 입출력

어떤 언어는 기본 입출력 속도가 매우 느리기 때문에 입출력만으로 시간초과가 날 수 있습니다.

```
ios_base::sync_with_stdio(false);
cin.tie(0);
```

C++는 C의 버퍼와 동기화를 하기 위하여 추가적인 작업을 하는데, 이 과정이 매우 느립니다.

C++의 cin/cout을 사용하려면 이 두 줄을 main함수의 시작 부분에 넣고 std::endl 대신에 '\n'를 사용해야 합니다.

단, 이렇게 하면 더이상 C 계열의 입출력(printf, scanf, getchar 등)을 사용하면 안 됩니다.

저 함수를 호출하고 나서 C와 C++의 출력을 번갈아 사용해 보면 어떤 결과가 나오는지 확인해볼 수 있습니다.

```
import java.io.*;
import java.util.*;
```

```
public class Main {
  public static void main(String args[]) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
    ...
    int n = Integer.parseInt(br.readLine());
    ...
    StringTokenizer st = new StringTokenizer(br.readLine(), " ");
    int a = Integer.parseInt(st.nextToken());
    ...
    bw.write(sum);
    bw.newLine();
    ...
    bw.flush(); // 마지막에 한 번만 호출
  }
}
```

Java를 사용하고 있다면, Scanner와 System.out.println을 사용하지 말고 위와 같이 BufferedReader, BufferedWriter, StringTokenizer를 이용합니다.

Python을 사용하고 있다면, input 대신 sys.stdin.readline을 사용할 수 있습니다. 단, 이때는 맨 끝의 개행문자까지 같이 입력받기 때문에 문자열을 저장하고 싶을 경우 .rstrip()을 추가로 해 주는 것이 좋습니다.

기타 정보는 fastio를 검색하여 알아볼 수 있습니다.

입력

- 빠른 입출력을 해 보자. 첫 줄에 이 주어진다.
- 다음 N개의 줄에 이하의 두 자연수가 주어진다.

출력

- 한 줄에 하나씩 두 정수의 합을 출력한다.

입력 예시 1

```
5
1 1
12 34
5 500
40 60
1000 1000
```

출력 예시 1

```
2
46
```

505
100
2000