ios_base::sync_with_stido(bool sync);

true

Sync_with_stido(true); //default

- 모든 표준 입출력 stream은 동기화 상태에 있다. 동기화된 스트림들은 각각 상태에 맞는 버퍼를 사용하므로 C, C++의 입출력 방식을 혼용할 수 있다.(C → C stream / C++ C++ Stream)

- 동기화된 스트림은 안정성을 보장한다.
  여러 스레드에서 각각 입출력 연산을 수행하기 때문에 경쟁 상태를 발생하지 않는다.

false

Sync_with_stido(false);

- 동기화 해제 시 C++ stream은 독립적은 버퍼를 갖는다.
  C방식과 C++ 방식을 혼용하면 출력 순서를 보장할 수 없어 오답의 가능성이 생긴다.

ios_base::sync_with_stido(true);

```cpp
#include <iostream>
using namespace std;

int main() {

    ios::sync_with_stdio(true);

    cout<<"a₩n";
    printf("b₩n");
    cout<<"c₩n";

    return 0;
}
```

실행결과

stdout

```
a
b
c
```

ios_base::sync_with_stido(false);

```cpp
#include <iostream>
using namespace std;

int main() {

    ios::sync_with_stdio(false);

    cout<<"a₩n";
    printf("b₩n");
    cout<<"c₩n";

    return 0;
}
```

실행결과

stdout

```
a
c
b
```

cin.tie(NULL), cout.tie(NULL)

```
cin.tie(NULL); cout.tie(NULL);
```

- cin과 cout tie 메소드를 사용하여 묶음 처리를 하여 사용된다.

- tie 메소드를 사용하면 출력 버퍼를 비우는 연산을 하게 된다.
  출력 버퍼를 비우는 연산은 보다 화면에 매끄럽게 출력하기 위한 연산이다.
  이 출력 버퍼를 비우는 연산은 생각보다 많은 시간을 소요한다.
  온라인 저지에서는 화면에 바로 보여지는 것은 중요하지 않고
  무엇이 출력되는가가 중요하기 때문에 해당 연산을 줄이는 것 만으로 실행 시
  간을 줄이는 효과를 볼 수 있다.

- 입력과 출력을 여러 번 번갈아서 반복해야 하는 경우 필수적입니다.

endl vs “\n”

- Endl은 개행문자를 출력하는 것 이외에도 출력 버퍼를 비우는 역할을 한다.
- 출력 버퍼를 비우게 되면 실행 결과를 곧 바로 사용자가 볼 수 있게 해준다.
  해당 작업은 많은 시간을 소요하게 된다.
- 온라인 저지에서는 화면에 바로 보여지는 것은 중요하지 않고
  무엇이 출력되는가가 중요하기 때문에 버퍼를 자주 비울 필요가 없으며
  많은 시간을 소요하게 되므로 단점이 된다.
- 직접 개행문자를 넣는 편이 endl을 쓰는 것 보다 시간소요를 줄일 수 있다.

입출력 연산이 많은 경우 속도 차이

입출력 반복이 많은 문제의 경우 입니다.
소스코드는 동일하고 아래의 소스코드만 추가 했을때의 속도 차이 입니다.

```
ios_base::sync_with_stido(false);
cin.tie(NULL);
cout.tie(NULL);
```

문제	결과	메모리	시간	언어	코드 길이
3 2108	맞았습니다!!	4004 KB	80 ms	C++17 / 수정	838 B
3 2108	맞았습니다!!	4004 KB	208 ms	C++17 / 수정	762 B

문제	결과	메모리	시간	언어	코드 길이
4 20040	맞았습니다!!	3976 KB	100 ms	C++17 / 수정	717 B
4 20040	맞았습니다!!	3976 KB	424 ms	C++17 / 수정	655 B

사용 방법

```cpp
int main()
{
    std::ios::sync_with_stdio(false);
    std::cin.tie(NULL);
    std::cout.tie(NULL);

    // 코드 작성..
}
```