Item 13) Prefer const_iterators to iterators.

컨테이너에 값을 변화시키지 않으려고 하는 경우에는 `const_iterator`를 사용해라!!

- C++11이전에는 `const_iterator`를 지원하는 컨테이너도 많이 없었다.
- 그러나 C++11에서는 `const_iterator`를 사용하는 것이 더 쉽고 안전한 표준이 되었다.

```
#include <iostream>
#include <vector>
#include <algorithm> // std::find

int main() {
    // 벡터를 정의하고 초기화
    std::vector<int> values = {10, 20, 30, 40, 50};

    // 벡터에서 1983을 찾기 위한 const_iterator
    auto it = std::find(values.cbegin(), values.cend(), 1983);

    // 1983이 벡터에 존재하지 않으므로, it는 values.cend()를 가리킴
    if (it == values.cend()) {
        std::cout << "1983 not found in the vector." << std::endl;
    } else {
        // 1983을 찾았을 경우
        std::cout << "1983 found in the vector." << std::endl;
    }

    // 벡터에 1998을 삽입. it는 values.cend()를 가리키므로 벡터의 끝에 삽입됨
    values.insert(it, 1998);

    // 벡터의 내용을 출력
    std::cout << "Vector contents: ";
    for (const auto& value : values) {
        std::cout << value << ' ';
    }
    std::cout << std::endl;

    return 0;
}
```

- 일반 iterator를 호출하기 위해서는 `begin`을 호출하면 되고, const_iterator를 호출하기 위해서는 `cbegin`을 로출하면 된다.

---

confusing point

컨테이너에 삽입을 진행하는데 왜 `const_iterator`가 사용되는 건가요??

```
auto It = std::find(values.cbegin(), values.cend(), 1983);
values.insert(it,1998);
```

- 왜 `const_iterator`를 사용하는가?
  - **읽기 전용** : `const_iterator`를 사용하여 컨테이너의 요소를 읽고, 특정 값을 찾는 과정에서 컨테이너를 수정하지 않겠다는 의도를 명확히 할 수 있다.
- `const_iterator`와 삽입 : `const_iterator`를 사용하여 요소를 찾은 후, 그 위치에 새로운 요소를 삽입할 수 있다. `const_iterator`를 사용하여 요소를 찾는 것은 `const_iterator`가 읽기 전용이기 때문에, 실제 삽입과 같은 수정 작업에는 영향을 미치지 않는다. 삽입 작업은 `iterator`를 사용하여 수행된다.