Item 14) Declare functions noexcept if they won’t emit exceptions.

`noexcept`에 대해서

- 함수가 예외를 던지지 않을 것임을 명시적으로 지정하는데 사용된다. 함수가 예외를 던지지 않음을 선언하면, 컴파일러와 런타임 시스템은 그 함수에 대해 예외 안전성을 보장할 수 있다.
- `noexcept`가 선언된 함수에서 예외가 발생하면, 프로그램은 `std::terminate`를 호출하여 즉시 종료된다. 따라서 `noexcept`를 사용하는 함수는 예외를 던지 않도록 주의해야 한다.

`noexcept`의 장점

C++98 스타일(`throw`)

```
int f(int x) throw();
```

- C++98에서는 함수가 예외를 던지지 않음을 선언할 때 `throw()`를 사용했다.
- 예외가 실제로 발생하면 호출 스택이 되감기(unwind)되고, 프로그램은 종료된다.
- 스택을 되감고 예외를 처리하기 위해 복잡한 메커니즘이 필요하며, 최적화의 여지가 적다.

**C++11 스타일(`noexcept`)

```
int f(int x) noexcept;
```

- C++11에서 도입된 `noexcept`는 함수가 예외를 던지지 않을 것임을 명시적으로 선언한다.
- `noexcept`로 선언된 함수는 예외가 발생하는 경우, 스택을 되감을 필요가 없고 즉시 프로그램을 종료한다.
- 이에 스택을 되감기 위해 필요한 추가 자료구조와 메커니즘이 필요 없으니, 최적화가 더 쉬워진다.

`std::vector`와 예외 안전성

C++98의 예외 안전성

- **예외 안전성**: C++98에서 `std::vector`의 `push_back` 메서드는 요소를 추가할 때 예외 안전성을 보장합니다. `push_back`이 새로운 메모리 블록을 할당하고 기존 요소들을 새 메모리 블록으로 복사할 때, 만약 복사 중 예외가 발생하면, 기존 메모리 블록의 상태는 변경되지 않고 원래 상태를 유지합니다.
- **복사와 파괴**: 복사 중 예외가 발생하면, 기존 메모리 블록의 요소들은 여전히 파괴되지 않으며, 이로 인해 예외 안전성이 보장됩니다.

C++11의 이동 의미

- **이동 의미**: C++11에서는 이동 의미가 도입되어, 복사 연산 대신 이동 연산을 사용할 수 있습니다. 이동 연산은 객체의 자원을 새 객체로 이동시키고, 원래 객체는 빈 상태로 남기기 때문에 성능을 개선할 수 있습니다.
- **예외 안전성의 도전**: 이동 연산이 예외를 던질 수 있는 경우, 이동 연산 중 예외가 발생하면, 기존 메모리 블록의 상태를 복구하기 어렵고, 이로 인해 예외 안전성을 보장할 수 없습니다.

`std::vector::push_back`의 동작

1. **메모리 할당**: `std::vector`의 `push_back` 메서드는 요소를 추가할 때 메모리 용량이 부족하면 새로운, 더 큰 메모리 블록을 할당합니다.
2. **이동 vs 복사**: C++11에서는 복사 대신 이동 연산을 사용하여 기존 요소들을 새 메모리 블록으로 이동시키는 것이 최적화됩니다. 이동 연산이 복사보다 성능이 우수하기 때문입니다.
3. **예외 안전성 문제**: 이동 연산이 예외를 던질 수 있기 때문에, 이동 중 예외가 발생하면, 일부 요소는 새 메모리 블록으로 이동되었지만 원래 메모리 블록의 상태가 손상될 수 있습니다. 이는 예외 안전성을 보장하는 데 문제를 일으킵니다.

`move_if_noexcept`와 예외 안전성

- `move_if_noexcept`: C++11에서는 이동 연산이 예외를 던지지 않을 경우에만 이동을 수행하고, 그렇지 않으면 복사를 수행하도록 설계된 함수입니다. 이를 통해 예외 안전성을 유지하면서도 이동 연산의 성능 이점을 활용할 수 있습니다.
- `std::is_nothrow_move_constructible`: 이 타입 특성(trait)은 타입이 `noexcept`로 이동 생성자를 제공하는지를 검사합니다. 이 정보를 바탕으로 `move_if_noexcept`는 이동 연산을 사용할지 복사 연산을 사용할지를 결정합니다.

C++98 vs C++11 최적화 전략

- **C++98**: 예외 안전성을 보장하기 위해 `push_back`에서 모든 요소를 복사하고, 복사 중 예외가 발생하면 원래 상태를 유지합니다.

- **C++11**: 이동 연산이 `noexcept`로 선언된 경우에는 이동 연산을 사용하여 성능을 개선할 수 있습니다. 이동 연산이 예외를 던지지 않을 때만 이동을 수행하여 예외 안전성을 보장합니다.

`noexcept`의 유연성

```
void setup();
void cleanup();

void doWork() noexcept
{
        setup();
        ...
        cleanup();
}
```

- `noexcept`로 호출된 함수 안에, `noecept`로 호출되지 않는 함수가 들어가도 된다.
- 컴파일러는 *setup*함수와 *cleanup*함수가 `noexcept`로 호출되지 않더라도, `noexcept`로 선언된 것으로 간주한다.

⭐⭐⭐⭐⭐⭐
실제로 noexcept를 사용하니, 백준에서 최적화가 진행이 된 것 같다. 백준에서 사용해보자!!

제출 번호	아이디	문제	결과	메모리	시간	언어	코드 길이	제출한 시간
83965717	busalee	2 2350	맞았습니다!!	4464 KB	96 ms	C++17 / 수정	2035 B	10초 전
83965537	busalee	2 2350	맞았습니다!!	4464 KB	92 ms	C++17 / 수정	2051 B	4분 전
83965490	busalee	2 2350	맞았습니다!!	4464 KB	92 ms	C++17 / 수정	2042 B	5분 전
83965402	busalee	2 2350	컴파일 에러			C++17 / 수정	1723 B	7분 전
71636491	busalee	2 2350	맞았습니다!!	4464 KB	112 ms	C++17 / 수정	1696 B	8달 전

Solve안에 있는 모든 함수를 noexcept를 하니 속도가 많이 빨라졌고, 그냥 Solve만 진행을 해도 최적화가 많이 됨.

- 112ms -> 96ms -> 92ms