Item 11) Prefer deleted functions to private undefined ones.

특정 함수를 선언을 하고 호출을 허용하지 않게 하는 경우

C++98

C++98에서는 `private`로 선언을 한 뒤에 정의를 하지 않는 방법으로 호출을 허용하지 않았다.

Ex) istream 객체

```
template <class charT, class traits = char_traits<charT>>
class basic_ios : public ios_base {
public:
    // ...
private:
    basic_ios(const basic_ios&); // not defined
    basic_ios& operator=(const basic_ios&); // not defined
};
```

즉 함수를 `private`로 선언을 한 뒤에, code를 정의하지 않는다면, 외부에서 함수를 호출하지 못하게 할 수 있다.

C++11

C++11에서는 `= delete`를 통해서, 특정 함수를 호출하지 못하게 할 수 있다. 대부분의 경우 copy constructor와 copy assignment operator를 호출하지 못하게 한다.

```
template <class charT, class traits = char_traits<charT>>
class basic_ios : public ios_base {
public:
    // ...
    basic_ios(const basic_ios&) = delete;
    basic_ios& operator=(const basic_ios&) = delete;
    // ...
};
```

`delete`와 `private`의 차이점은 뭔가요?

- `delete`된 함수는 절대로 사용될 수 없는 함수이다. 특히 `friend`함수로 정의된 경우에도 접근이 불가능한 경우가 생긴다.
- 관습적으로는 `delete`함수는 "public"으로 두는 것이 더 좋다. 실제로 delete가 안되는 것은 아니지만, compiler가 경고 메시지를 남기며, public으로 선언된 경우, 컴파일러가 더 나은 error message를 주는 경우가 있다.
- `delete`를 사용하는 가장 중요한 이유는 class의 멤버함수 뿐만 아니라 모든 함수를 `delete`할 수 있기 때문이다.

```
bool isLucky(int number); // 원래 함수
bool isLucky(char) = delete; // char 타입의 인자를 거부
bool isLucky(bool) = delete; // bool 타입의 인자를 거부
bool isLucky(double) = delete; // double과 float 타입의 인자를 거부

if(isLucky('a')) // error
if(isLucky(true)) // error
if(isLucky(3.5f)) // error
```

- 위의 경우를 살펴보면, int를 argument로 넣어서 bool을 return하고 싶은 함수이다.
  이에, delete를 사용하게 된다면 특정 type이 argument로 들어가는 것을 방지할 수 있다.