**항목 24. 타입 변환이 모든 매개변수에 대해 적용되어야 한다면 비 멤버 함수를 선언하자.**

클래스에서 암시적 타입 변환을 지원하는 것은 일반적으로 안 좋다.

이 규칙에도 예외가 있는데, 가장 흔한 예외 중 하나가 숫자 타입을 만들 때이다.

예를 들어 유리수를 나타내는 클래스에서, 정수에서 유리수로의 암시적 변환은 크게 잘못된 것은 아닐 것이다. C++가 기본으로 제공하는 int -> double 변환과 별반 다르지 않기 때문이다.

다음은 이런 결정에 따라 만든 Rational 클래스이다.

|  |
| --- |
| class Rational {  public:             Rational(                       int numerator = 0, // 생성자에 일부러 explicit 를 붙이지 않았습니다.                       int denominator = 1); // int에서 Rational로의 암시적 변환을 허용하기 위함.               int numerator() const // 분자 및 분모에 대한 접근용 함수.             int denominator() const // (22)    private:             ...  }; |

유리수를 나타내는 클래스이므로 덧셈, 곱셈 등 수치 연산을 기본으로 제공하려고 한다.  
그런데 이들을 어떤 식으로 지원해야 할까? 멤버함수? 비멤버함수? 비멤버 프렌드 함수?

먼저, operator\*를 Rational의 멤버 함수로 만드는 예.

|  |
| --- |
| class Rational {  public:             ...             const Rational operator\* (const Rational& rhs) const; (3)(20)(21)참고  }; |

이렇게 설계해 두면 유리수 곱셈을 아주 쉽게 할 수 있다.

|  |
| --- |
| Rational oneEnglish(1, 8);  Rational oneHalf(1, 2);    Rational result = oneHalf \* oneEnglish; // ok  result = result \* oneEnglish; // ok |

혼합형(mixed-mode) 수치 연산도 가능하게 하고 싶다. 즉, Rational을 int 같은 것과도 곱하고 싶다.. 하지만 혼합형 수치 연산은 에러가 나타난다.

|  |
| --- |
| result = oneEnglish \* 2; // ok  result =  2 \* oneHalf; // error |

곱셈은 교환법칙이 성립해야 하는데…

위 원인은 예제를 함수 형태로 바꾸어 써 보면 바로 드러납니다.

|  |
| --- |
| result = oneHalf.operator(2); // ok  result = 2.operator\*(oneHalf); // error |

첫 줄에서 oneHalf 객체는 operator\* 함수를 멤버로 갖고 있는 클래스의 인스턴스이므로, 컴파일러는 이 함수를 호출합니다.  
하지만 2번째줄에서 정수 2에는 클래스 같은 것이 연관되어 있지 않기 때문에, operator\* 멤버 함수도 있을 리가 없습니다.  
**컴파일러는 아래처럼 호출할 수 있는 비 멤버 버전의 operator\*(네임스페이스 혹은 전역 유효범위에 있는 operator\*)도 찾아봅니다.**

|  |
| --- |
| Result = operator \* (2, oneHalf); // error |

예제에서 int와 Rational을 취하는 비멤버 버전의 operator\*가 없으므로 탐색은 실패하고 컴파일에러가 나게 된다.

**암시적 타입 변환(implicit type conversion).**   
컴파일러는 Rational::operator\* 에서 int가 넘겨졌으며, 함수 쪽에선 Rational을 요구한다는 사실을 알고 있고, 이 int를 Rational 클래스의 생성자에 주어 호출하면 Rational로 둔갑할 수 있다는 사실도 알고 있다. 즉, 다음과 같이 작성된 코드인 것이다.

|  |
| --- |
| const Rational temp(2); // 2롤부터 임시 Rational 객체를 생성한다.  result = oneHalf \* temp; // oneHalf.operator\*(temp); 와 같다. |

물론 컴파일러가 이렇게 동작한 것은 명시호출(explicit)로 선언되지 않은 생성자각 있기 때문이다.  
Rational 생성자가 만약 명시호출 생성자였으면 다음의 코드 중 어느 쪽도 컴파일 되지 않는다

|  |
| --- |
| .result =  oneHalf \* 2; // error(명시호출 생성자에 의해) 2를 Rational로 바꿀수 없다..  result =  2 \* oneHalf; // error(문제도 같다) |

이렇게 하면 혼합형 수치 연산은 지원되지 않지만, 최소한 2문장의 동작은 일관되게 유지된다.

등작도 일관되게 유지하고, 혼합형 수치 연산도 제대로 지원해야 한다.  
즉 위 2문장이 전부 컴파일 되어야 한다.  
암시적 타입 변환에 대해 매개변수가 먹혀 들려면, **매개변수 리스트에 들어 있어야만 한다**는 것입니다.   
그러니까 호출되는 멤버 함수를 갖고 있는 (즉, this가 가리키는) 객체에 해당하는 암시적 매개변수에는 암시적 변환이 먹히지 않습니다.

혼합형 수치 연산을 지원하려고 할 때, 바로 operator\*를 비멤버 함수로 만들어서, 컴파일러 쪽에서 **모든** 인자에 대해 암시적 타입 변환을 수행하도록 내버려 두는 것입니다.

|  |
| --- |
| class Rational {             ...  // operator\* 가 없다.  };    **const Rational operator\* (const Rational& lhs, const Rational& rhs)** // 이제는 비멤버 함수임.  {             return Rational(lhs.numerator() \* rhs.numerator(),                                  lhs.denominator()( \* rhs.denominator());  }    Rational oneHalf(1, 2);  Rational result;    result = oneHalf \* 2; // 이건 원래 ok.  result = 2 \* oneHalf; // ok |

operator\* 함수는 Rational 클래스의 프렌드 함수로 두어도 될까요?  
지금의 예제에서는 ‘아니오’라고 해야 옳습니다.  
operator\*는 완전히 Rational의 public 인터페이스만을 써서 구현할 수 있기 때문입니다.

**요약.**

\* 어떤 함수에 들어가는 모든 매개변수(this 포인터가 가리키는 객체도 포함해서)에 대해 타입 변환을 해 줄 필요가 있다면, 그 함수는 비멤버이어야 합니다.