**항목 21. 함수에서 객체를 반환해야 할 경우에 참조자를 반환하려고 들지 말자.**

항목 20.을 보게 된 사용자는 모든 코드는 ‘오직 참조에 의한 전달’만으로 이루어져야 한다는 생각 때문에 도를 넘을 수 있다. 예를 들어 실제로 존재하지 않는 객체의 참조를 던질 수 있다.

ex) 유리수를 나타내는 클래스

|  |
| --- |
| class Rational {  public:             Rational(int numerator = 0, int denominator = 1); // 생성자가 explicit로 선언되지 않은 이유는 (24)             ...  private:             int n, d; // 분자 및 분모             friend const Rational operator\* (const Rational& lhs, const Rational& rhs); // 반환타입이 const인 이유는 (3)  }; |

이 클래스의 operator\*는 곱셈 결과를 값으로 반환하도록 되어 있다.

값이 아닌 참조자를 반환할 수 있으면 비용 부담은 없을 것입니다.   
그러나 참조자는 그냥 이름입니다. 존재하는 객체에 붙는 다른 이름입니다.

operator\*가 참조자를 반환하도록 만들어졌다면, 이 함수가 반환하는 **참조자는 반드시 이미 존재하는 Rational 객체의 참조자여야 합니다.**

그럼 반환될 객체는 어디에 있을까요?

|  |
| --- |
| Rational a(1, 2); // a = 1/2  Rational b(3, 5); // b = 3/5    Rational c = a \* b; // c는 3/10 이어야 합니다. |

객체에 대한 참조자를 operator\*에서 반환할 수 있으려면, 그 객체를 직접 생성해야 한다는 것입니다.

함수 수준에서 새로운 객체를 만드는 방법은 딱 2가지 뿐입니다.   
하나는 스택에 만드는 것이고,  
또 하나는 힙에 만드는 것입니다.

우선 전자의 방법은, 스택에 객체를 만들려면 지역 변수를 정의하면 됩니다.

|  |
| --- |
| const Rational& operator\*(const Rational& lhs, const Rational& rhs) // 어이없는 코드  {             Rational result(lhs.n \* rhs.n, lhs.d \* rhs.d);             return result;  } |

 위 예제는 피하자.  
생성자가 불리는 게 싫어서 시작한 일인데, 결국 result가 다른 객체처럼 생성되어야 하잖아요.  
더 심각한 문제는, 이 연산자 함수는 result에 대한 참조자를 반환하는데, result는 지역 객체입니다. 즉, 함수가 끝날 때 덩달아 소멸되는 객체죠.

그러니까 이 **operator\*는 현재 온전한 Rational 객체에 대한 참조자를 반환하지 않습니다**.

이 함수를 호출한 쪽은 그 즉시 미정의 동작에 빠지게 됩니다.

 후자의 방법, 함수가 반환할 객체를 힙에 생성했다가, 그 녀석의 참조자를 반환하는 것입니다.

|  |
| --- |
| const Rational$ operator\*(const Rational& lhs, const Rational& rhs) // 또 일났네요!  {             Rational \*result = new Rational(lhs.n \* rhs.n, lhs.d \* rhs.d);             return \*result;  } |

여전히 생성자가 한번 호출되기는 매한가지입니다.  
new로 할당한 메모리를 초기화할 때 생성자가 호출되니 말입니다.  
이걸 말고도, new로 만든 객체를 누가 delete해줄까요?

이 함수를 호출하는 쪽에서는 메모리 누출을 막기에는 한계가 있습니다.

|  |
| --- |
| Rational w, x, y, z;  w = x \* y \* z // operator\* (operator\* (x, y), z) 와 같습니다. |

여기서는 한 문장 안에서 operator\* 호출이 2번 일어나고 있기 때문에, new에 짝을 맞추어 delete를 호출하는 작업도 2번이 필요합니다.   
그런데 operator\*의 사용자 쪽에서는 이렇게 할 수 있는 합당한 방법이 없습니다.   
operator\*로부터 반환되는 참조자 뒤에 숨겨진 포인터에 대해서는 사용자가 어떻게 접근할 방법이 없기 때문입니다.

위 2개의 예는, 한 가지 문제를 똑같이 가지고 있었습니다.  
스택 기반으로 하든 힙 기반으로 하든 operator\*에서 반환되는 결과는, 반드시 생성자를 꼭 한 번 호출했을 거예요. 그런데, 필요 없는 생성자 호출을 피해 보자는 것이 아마 처음 세운 목표였을 걸요?

|  |
| --- |
| const Rational$ operator\*(const Rational& lhs, const Rational& rhs) // 경고!  {             static Rational result; // 반환할 참조자가 가리킬 정적 객체.             result = ...;             return result;  } |

위 코드는 스레드 안정성 문제가 있습니다   
또한 아래와 같은 문제가 있습니다.

|  |
| --- |
| bool operator==(const Rational& lhs, const Rational& rhs);  Rational a, b, c, d;  ...  if ((a\*b) == (c\*d)) {             두 유리수 쌍의 곱이 서로 같으면 적절한 처리를 수행  } else {             다르면 적절한 처리를 수행  } |

((a\*b) == (c\*d)) 표현식이 **항상** true 값을 냅니다.   
a,b,c,d에 어떤 값이 들어가도 마찬가지입니다.

위의 표현식을 아래처럼 바꾸면

|  |
| --- |
| if (operator==(**operator\*(a,b), operator\*(c,d)**)) |

operator== 가 호출될 때, operator\*가 활성화되어 있을 것이고, 각각의 호출을 통해 operator\*안에 정의된 정적 Rational 객체의 참조자가 반환될 것입니다.  
operator==가 비교하는 피연산자는 operator\* 안의 정적 Rational 객체의 값 입니다.  
이 둘이 같지 않은 것이 더 이상합니다.

새로운 개체를 반환해야 하는 함수를 작성하는 방법에는 정도(正道)가 있습니다.   
바로 **‘새로운 객체를 반환하게 만드는 것’**이죠  
그러니까 Rational의 operator\*는 아래처럼 혹은 비슷하게 작성해야 합니다.

|  |
| --- |
| inline const Rational operator\* (const Rational& lhs, const Rational& rhs)  {             return Rational(lhs.n \* rhs.n, lhs.d \* rhs.d);  } |

이 코드에도 반환 값을 생성하고 소멸시키는 비용이 들어 있습니다.   
그러나 끝까지 따져 보면 여기에 들어가는 비용은 올바른 동작에 지불되는 작은 비용입니다.

C++에서도 다 컴파일러 구현자들이 가시적인 동작 변경을 가하지 않고도 기존 코드의 수행 성능을 높이는 최적화를 적용할 수 있도록 배려해 두었습니다. 그 결과, 몇몇 조건하에서는 이 최적화 메커니즘에 의해 operator\*의 반환 값에 대한 생성과 소멸 동작이 안전하게 제거될 수 있습니다.(**반환 값 최적화(return value optimization) RVO**)

**요약.**

\* 지역 스택 객체에 대한 포인터나 참조자를 반환하는 일, 혹은 힙에 할당된 객체에 대한 참조자를 반환하는 일, 또는 지역 정적 객체에 대한 포인터나 참조자를 반환하는 일은 그런 객체가 두 개 이상 필요해질 가능성이 있다면 절대로 하지마세요 (항목 4를 보시면 지역 정적 객체에 대한 참조자를 반환하도록 설계된 올바른 코드 예제를 찾을 수 있습니다. 최소한 단일 스레드 환경에서는 통합니다.)