27장. 타입 2

01_ 연산자 오버로딩
02_클래스와 관련된 형변환

complex 클래스

- 연산자 오버로딩을 위한 예제 클래스

```
// 복소수 클래스
class Complex
{
public:
          // 생성자
          Complex(int realPart, int imaginaryPart)
                    : real(realPart), imaginary(imaginaryPart)
          {}

          // 접근자들
          int Real(int realPart) {real = realPart; return real;}
          int Imaginary(int ImaginaryPart) {imaginary = ImaginaryPart;
return real;}

          int Real() const {return real;}
          int Imaginary() const {return imaginary;}

private:
          int real;           // 실수부
          int imaginary;      // 허수부
};
```

피연산자가 두 개인 연산자(1)

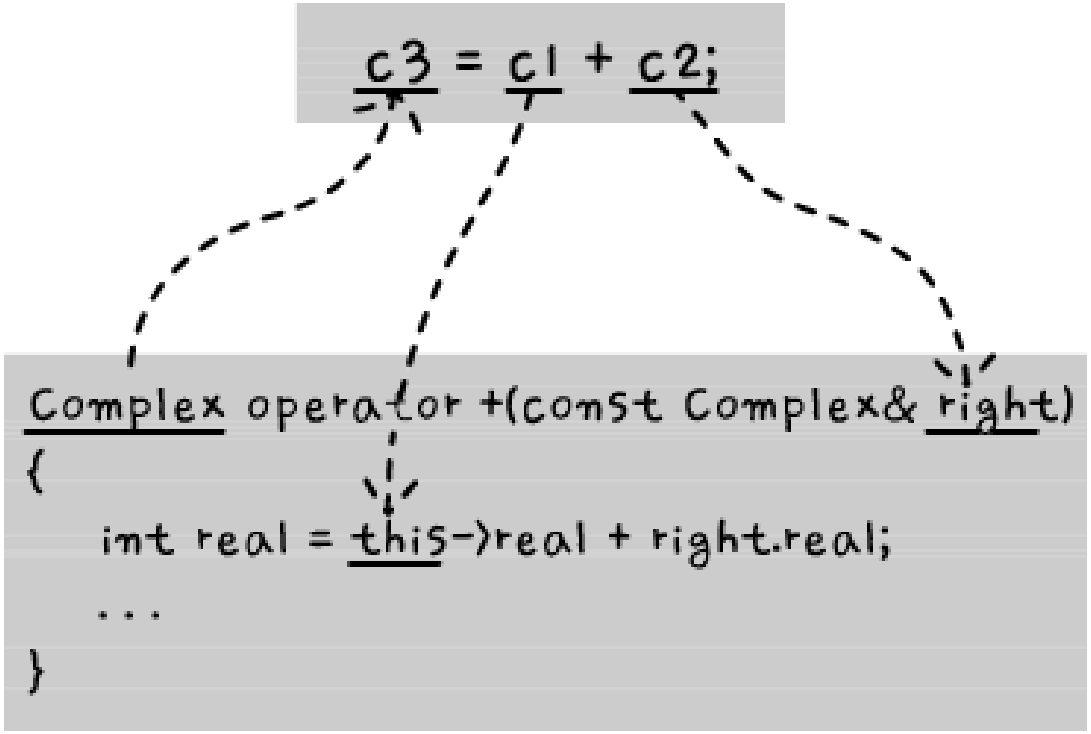
- 피연산자가 두 개인 + 연산자를 오버로딩 하는 예

```
class Complex
{
          // 중간 생략
          Complex operator+(const Complex& right)
          {
                    // 실수부와 허수부를 각각 더한다.
                    int real = this->real + right.real;
                    int imag = this->imaginary + right.imaginary;

                    // 결과를 보관한 복소수 객체를 반환한다.
                    return Complex(real, imag);
          }
```

```
Complex c1(1, 1);
Complex c2(2, 2);
Complex c3(0, 0);

// + 연산자를 사용한 덧셈
c3 = c1 + c2;        // c3 = (3, 3)
c3 = c1.operator +(c2);
```

피연산자가 두 개인 연산자(2)

- 피연산자와 인자의 매칭



피연산자가 한 개인 연산자(1)

- 전치형과 후치형의 ++ 연산자를 오버로딩 하는 예

```
class Complex
{
          // 중간 생략
          // ++c 의 경우 (전치 연산)
          Complex operator++()
          {
                    // 실수부의 값을 먼저 더한다.
                    this->real++;

                    // 현재 값을 반환하다.
                    return *this;
          }

          // c++ 의 경우 (후치 연산)
          Complex operator++(int)
          {
                    // 현재값을 먼저 보관한다.
                    Complex prev( this->real, this->imaginary);

                    // 실수부의 값을 더한다.
                    this->real++;

                    // 보관한 값을 반환한다.
                    return prev;
          }
```

피연산자가 한 개인 연산자(2)

- 전치형과 후치형의 ++ 연산자를 오버로딩 하는 예

```
Complex c1(1, 1);
Complex prefix(0, 0);
Complex postfix(0, 0);

// 전치연산
prefix = ++c1;       // prefix = c1 = (2,1)

// 후치 연산
postfix = c1++;      // postfix = (2,1), c1 = (3,1)
```

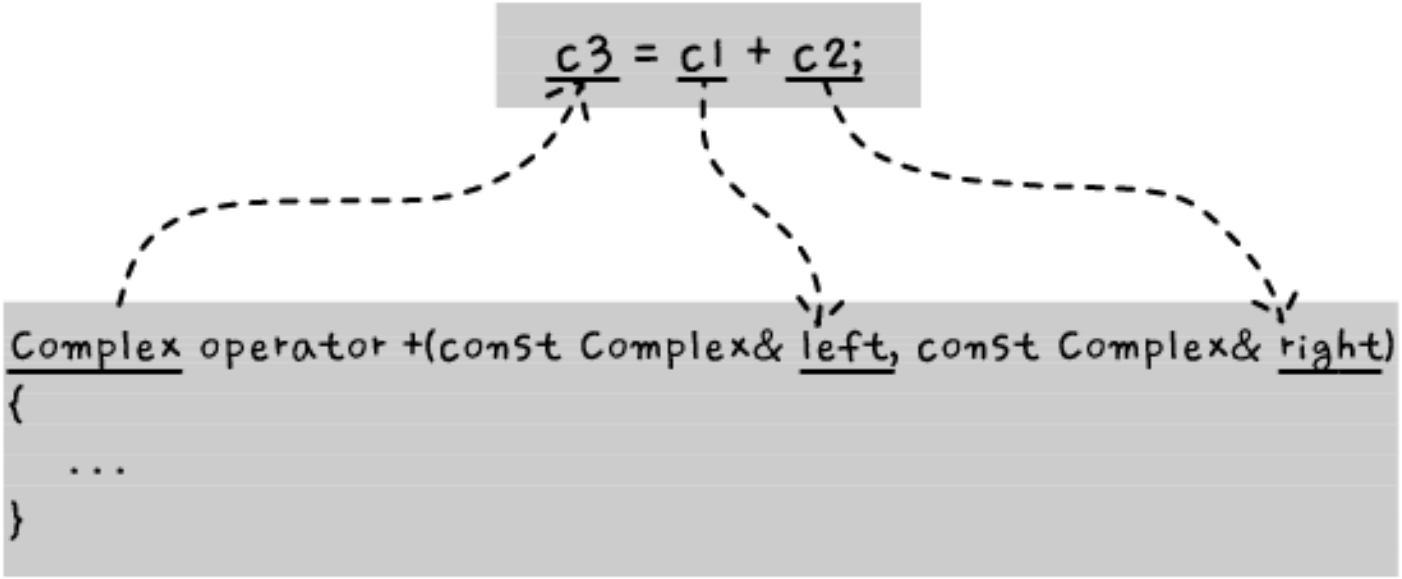
일반 함수를 사용한 연산자 오버로딩(1)

- 멤버 함수가 아닌 일반 함수를 사용해서 + 연산자를 오버로딩 하는 예

```
Complex operator+(const Complex& left, const Complex& right)
{
          // 실수부와 허수부를 각각 더한다.
          int real = left.real + right.real;
          int imag = left.imaginary + right.imaginary;

          // 결과를 보관한 복소수 객체를 반환한다.
          return Complex(real, imag);
}

int main()
{
          Complex c1(1, 1);
          Complex c2(2, 2);
          Complex c3(0, 0);

          // + 연산자를 사용한 덧셈
          c3 = c1 + c2;        // c3 = (3, 3)
          c3 = operator +(c1, c2);

          return 0;
}
```

일반 함수를 사용한 연산자 오버로딩(2)

- 피연산자와 인자의 매칭

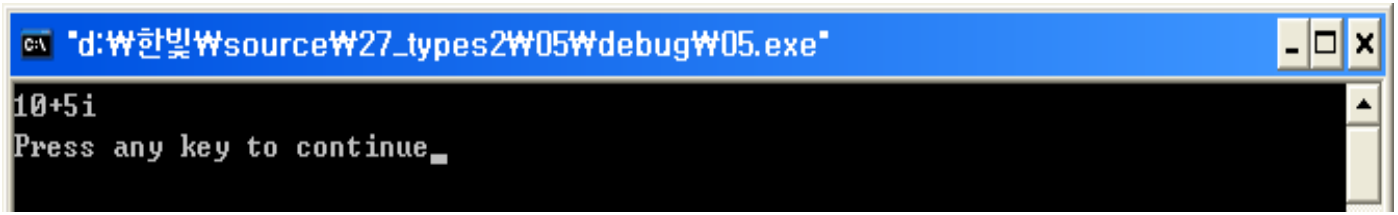


멤버 함수로 만들 수 없는 경우(1)

- 오른쪽 피연산자만 클래스 타입이라면 일반 함수를 사용해서 오버로딩 할 수 밖에 없다.

```
ostream& operator<<(ostream& o, const Complex& right)
{
          o << right.Real() << showpos << right.Imaginary() << "i" << noshowpos;
          return o;
}

int main()
{
          Complex c1(10, 5);

          cout << c1 << "\n";

          return 0;
}
```

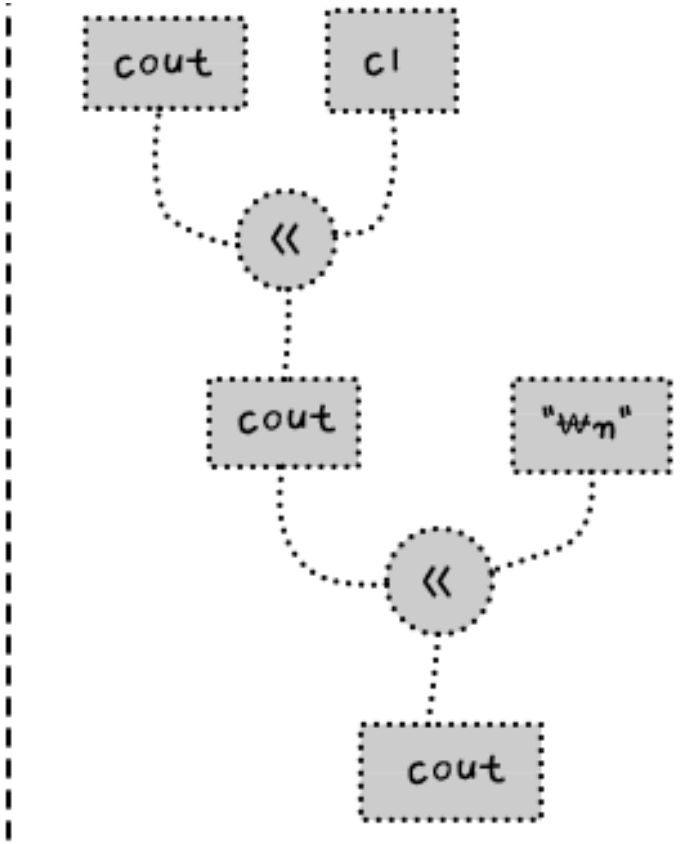
- 실행 결과



멤버 함수로 만들 수 없는 경우(2)

- 연산 과정

```
cout << cl << "₩n";
```



연산자 오버로딩의 규칙

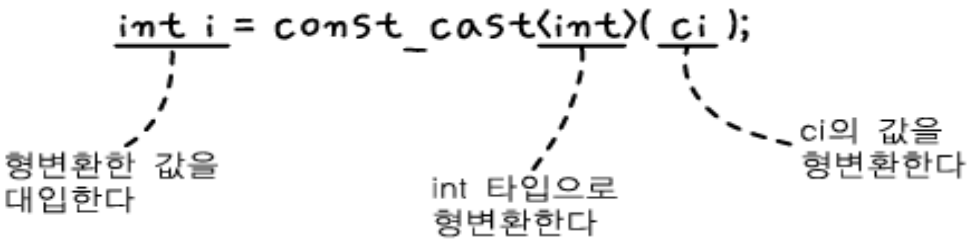
- 오버로딩이 가능한 연산자가 제한되어 있다. (p.818 참조)
- 기본 타입의 연산 방법은 바꿀 수 없다.
  - 그러므로, 피연산자가 모두 기본 타입인 연산자 함수는 만들 수 없다.
- 기존 연산자의 의미를 헤치지 않는 것이 좋다.
  - 예) + 연산자를 뺄셈의 용도로 오버라이딩 하는 것은 좋지 않다.

C++의 새로운 형변환 연산자

- const_cast를 사용해서 변수의 const 속성이나 volatile의 속성을 제거할 수 있다.

```
const int ci = 100;
int i = const_cast<int>( ci );
```

- einterpret_cast를 사용해서 위험한 형변환도 할 수 있다.



- static_cast는 가장 일반적인 형태의 형변환에 사용한다.

```
int a, b;
a = reinterpret_cast<int>( &b );
```

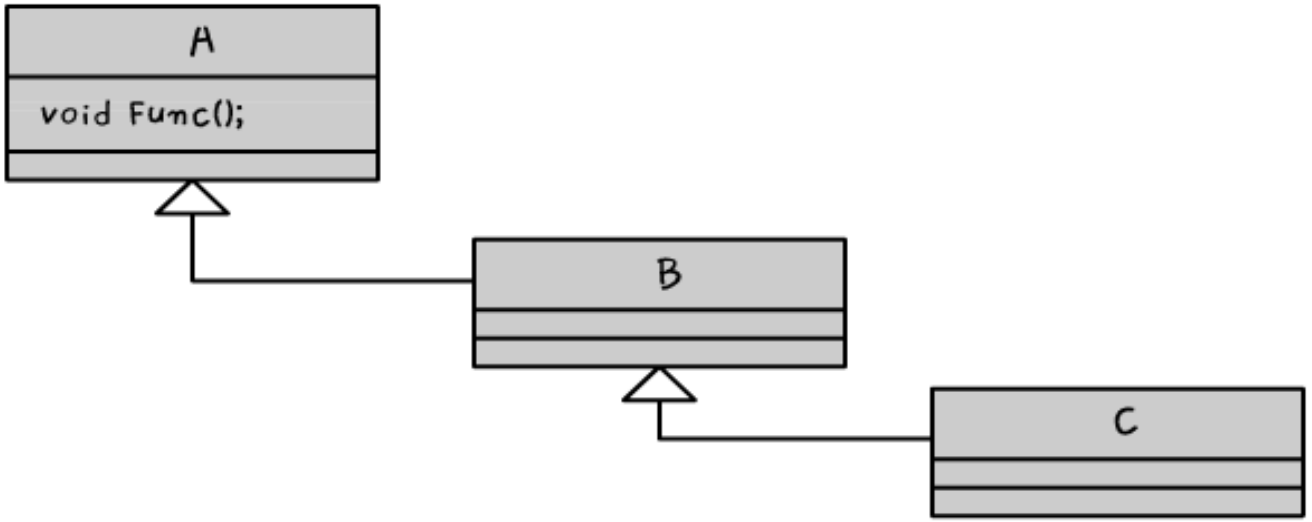
  - A 타입에서 B 타입으로의 암시적인 형변환이 가능하다면 static_cast를 사용해서 B 타입에서 A 타입으로 형변환할 수 있다.

```
double d = 30.0;
char c;
c = static_cast<char>( d );
```

dynamic_cast(1)

- dynamic_cast의 사용법을 알아보기 위해서 만든 클래스들

```
class A
{
public:
          virtual void Func() {}
};

class B : public A
{
};

class C : public B
{
};
```



dynamic_cast(2)

- 올바르지 않은 형변환에 대해서는 NULL을 반환하거나 bad_cast 예외 객체를 던진다.

```
// C 객체를 생성해서 A*에 담는다.
A* pa1 = new C;

// A 객체를 생성해서 A*에 담는다.
A* pa2 = new A;

// pa1을 C* 타입으로 형변환시킨다.
C* pc1 = dynamic_cast<C*>( pa1 );       // 성공

// pa2를 C* 타입으로 형변환시킨다.
C* pc2 = dynamic_cast<C*>( pa2 );       // 실패 : NULL 반환

try
{
        // *pa2를 C& 타입으로 형변환시킨다.
        C& rc1= dynamic_cast<C&>( *pa2);        // 실패 : bad_cast 예외 발생
}
catch(bad_cast& e)
{
}
```

내 클래스를 다른 타입으로 형변환하기

- Complex 객체를 int 로 형변환하는 함수를 제공하는 예

```
class Complex
{
          // 중간 생략
          operator int()
          {
                    return this->real;
          }
```

```
Complex c1(10, 5);

int i;
i = c1;              // i = 10
i = c1.operator int();
```

다른 타입을 내 클래스로 형변환하기

- int를 Complex 타입으로 형변환하는 함수를 제공하는 예

```
class Complex
{
          // 중간 생략
          ComplexI int i )
                    : real( i ), imaginary( 0 )
          {}
```

```
int i = 5;
Complex c( 0, 0 );

c = i;                // c = ( 5, 0 )
c = Complex( i );
```

Q&A