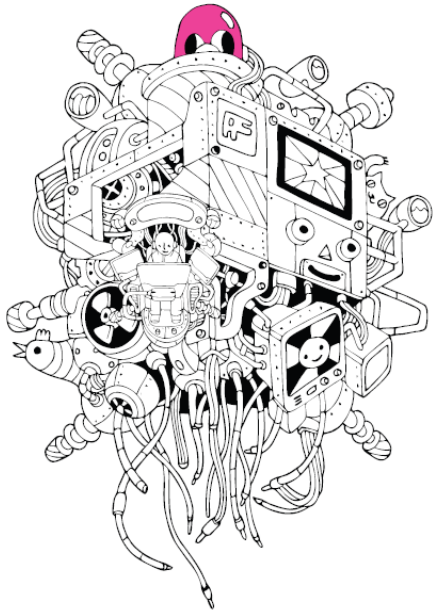
윤성우의
열혈 C++ 프로그래밍



윤성우 저 열혈강의 C++ 프로그래밍 개정판

Chapter 10. 연산자 오버로딩1

윤성우의

열혈 C++ 프로그래밍

Chapter 10-1. 연산자 오버로딩의 이해와 유형

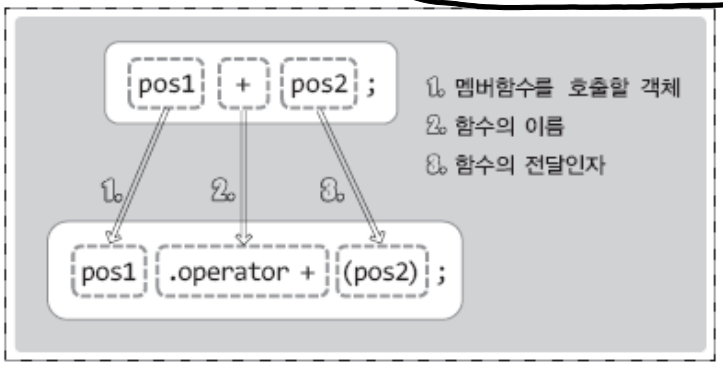
윤성우 저 열혈강의 C++ 프로그래밍 개정판

operator+ 라는 이름의 함수


```
class Point
{
private:
    int xpos, ypos;
public:
    Point(int x=0, int y=0) : xpos(x), ypos(y)
    { }
    void ShowPosition() const
    {
        cout<<'['<<xpos<<", "<<ypos<<']'<<endl;
    }
    Point operator+(const Point &ref)      // operator+라는 이름의 함수
    {
        Point pos(xpos+ref.xpos, ypos+ref.ypos);
        return pos;
    }
};
```

```
int main(void)
{
    Point pos1(3, 4);
    Point pos2(10, 20);
    Point pos3=pos1.operator+(pos2);
    pos1.ShowPosition();
    pos2.ShowPosition();
    pos3.ShowPosition();
    return 0;
}
```

```
int main(void)
{
    Point pos1(3, 4);
    Point pos2(10, 20);
    Point pos3=pos1+pos2;
    pos1.ShowPosition();
    pos2.ShowPosition();
    pos3.ShowPosition();
    return 0;
}
```

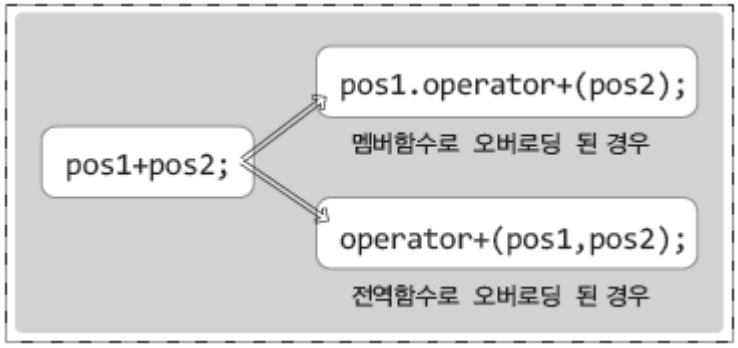



```
[3, 4]
[10, 20]
[13, 24]
```

실행결과

연산자 오버로딩에서 이야기하는 함수호출의 규칙을 이해하는 것이 중요!

연산자를 오버로딩 하는 두 가지 방법



오버로딩 형태에 따라서 스스로 변환!

```
class Point
{
private:
    int xpos, ypos;
public:
    Point(int x=0, int y=0) : xpos(x), ypos(y)
    { }
    void ShowPosition() const
    {
        cout<<'['<<xpos<<", "<<ypos<<']'<<endl;
    }
    friend Point operator+(const Point &pos1, const Point &pos2);
};

Point operator+(const Point &pos1, const Point &pos2)
{
    Point pos(pos1.xpos+pos2.xpos, pos1.ypos+pos2.ypos);
    return pos;
}
```

```
int num = 3 + 4;
Point pos3 = pos1 + pos2;
```

이렇듯 피연산자에 따라서 진행이 되는 + 연산의 형태가 달라지므로 연산자 오버로딩이라 한다.

```
[3, 4]
[10, 20]
[13, 24]
```

실행결과

```
int main(void)
{
    Point pos1(3, 4);
    Point pos2(10, 20);
    Point pos3=pos1+pos2;
    pos1.ShowPosition();
    pos2.ShowPosition();
    pos3.ShowPosition();
    return 0;
}
```

오버로딩이 불가능한 연산자의 종류

.	멤버 접근 연산자
.*	멤버 포인터 연산자
::	범위 지정 연산자
?:	조건 연산자(3항 연산자)
sizeof	바이트 단위 크기 계산
typeid	RTTI 관련 연산자
static_cast	형변환 연산자
dynamic_cast	형변환 연산자
const_cast	형변환 연산자
reinterpret_cast	형변환 연산자

오버로딩 불가능!

=	대입 연산자
()	함수 호출 연산자
[]	배열 접근 연산자(인덱스 연산자)
->	멤버 접근을 위한 포인터 연산자

멤버함수의 형태로만 오버로딩 가능!

연산자를 오버로딩 하는데 있어서의 주의사항

√ 본래의 의도를 벗어난 형태의 연산자 오버로딩은 좋지 않다!

√ 연산자의 우선순위와 결합성은 바뀌지 않는다.

√ 매개변수의 디폴트 값 설정이 불가능하다.

√ 연산자의 순수 기능까지 빼앗을 수는 없다.

```
int operator+(const int num1, const int num2)
{
    return num1*num2;
}
```

정의 불가능한 형태의 함수

윤성우의

열혈 C++ 프로그래밍

Chapter 10-4. cout, cin 그리고 endl의 정체

윤성우 저 열혈강의 C++ 프로그래밍 개정판

cout과 endl 이해하기


```
class ostream
{
public:
    void operator<< (char * str)
    {
        printf("%s", str);
    }
    void operator<< (char str)
    {
        printf("%c", str);
    }
    void operator<< (int num)
    {
        printf("%d", num);
    }
    void operator<< (double e)
    {
        printf("%g", e);
    }
    void operator<< (ostream& (*fp)(ostream &ostm))
    {
        fp(*this);
    }
};
ostream& endl(ostream &ostm)
{
    ostm<<'\n';
    fflush(stdout);
    return ostm;
}
ostream cout;
```

이름공간 mystd 안에 선언되었다고 가정!

예제에서 cout과 endl을 흉내내었으니, 예제의 분석을 통해서 이 둘의 실체를 이해할 수 있다.

```
int main(void)
{
    using mystd::cout;
    using mystd::endl;

    cout<<"Simple String";
    cout<<endl;
    cout<<3.14;
    cout<<endl;
    cout<<123;
    endl(cout);
    return 0;
}
```

실행결과

```
Simple String
3.14
123
```

```
cout.operator<<("Simple String");
cout.operator<<(3.14);
cout.operator<<(123);

cout.operator<<(endl);
```

cout<<123<<endl<<3.14<<endl;


```
class ostream
{
public:
    ostream& operator<< (char * str)
    {
        printf("%s", str);
        return *this;
    }
    ostream& operator<< (char str)
    {
        printf("%c", str);
        return *this;
    }
    ostream& operator<< (int num)
    {
        printf("%d", num);
        return *this;
    }
    ostream& operator<< (double e)
    {
        printf("%g", e);
        return *this;
    }
    ostream& operator<< (ostream& (*fp)(ostream &ostm))
    {
        return fp(*this);
    }
};
ostream& endl(ostream &ostm)
{
    ostm<<'\n';
    fflush(stdout);
    return ostm;
}
```

***this 를 반환함으로써, 연이은 오버로딩 함수의 호출이 가능해진다.**

cout<<123<<endl<<3.14<<endl;

<<, >> 연산자의 오버로딩


```
class Point
{
private:
    int xpos, ypos;
public:
    Point(int x=0, int y=0) : xpos(x), ypos(y)
    { }
    void ShowPosition() const
    {
        cout<<'['<<xpos<<", "<<ypos<<']'<<endl;
    }
    friend ostream& operator<<(ostream&, const Point&);
};

ostream& operator<<(ostream& os, const Point& pos)
{
    os<<'['<<pos.xpos<<", "<<pos.ypos<<']'<<endl;
    return os;
}
```

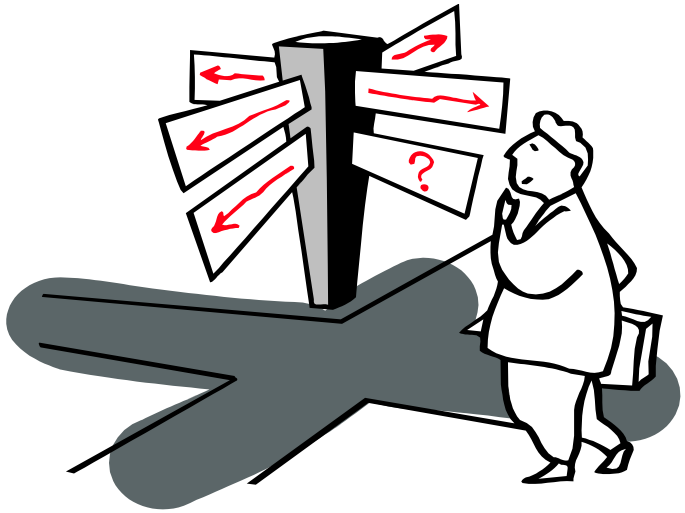
오버로딩 이므로

```
int main(void)
{
    Point pos1(1, 3);
    cout<<pos1;
    Point pos2(101, 303);
    cout<<pos2;
    return 0;
}
```

실행결과

```
[1, 3]
[101, 303]
```

Point 클래스를 대상으로 하는 << 연산자의 오버로딩 사례를 보인다!



Chapter 10이 끝났습니다. 질문 있으신지요?