Week 5-2 : Name Spaces, Operator Overloading

Part2. Operator Overloading

기존의 연산자(+,-,*,/,=…) 를 함수의 이름으로 지정하여 사용함
객체를 연산하는 것처럼 보일 수 있음

Operators that can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Operators that cannot be overloaded			
.	.*	::	?:

멤버함수 기반으로만 오버로딩이 가능한 연산자 : =, (), [], ->

1. 연산자 오버로딩의 두 형태

① 멤버함수에 의한 연산자 오버로딩

```
#include <iostream>
using namespace std;
class Point
{
private:
        int x,y;
```

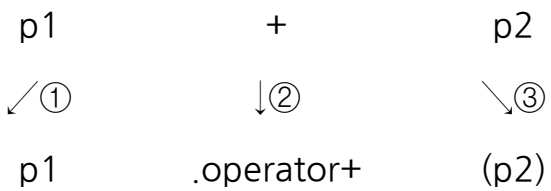
```
public:
        Point(int argX,int argY);
        void ShowPosition();
        Point operator+(Point &p);
};

Point::Point(int argX,int argY)
{
        x=argX;
        y=argY;
}

void Point::ShowPosition()
{
        cout<<x<<" "<<y<<endl;
}
Point Point::operator+(Point &p)
{
        Point temp(x+p.x,y+p.y);
        return temp;
}

int main(void)
{
        Point p1(1,1);
        Point p2(2,2);
        Point p3=p1+p2;
        p3.ShowPosition();

        return 0;
}
```



① 멤버함수를 호출할 객체

② 함수의 이름
③ 함수의 전달 인자

② 전역함수에 의한 연산자 오버로딩

```
#include <iostream>
using namespace std;
class Point
{
private:
	int x,y;
public :
	Point(int argX,int argY);
	int getX();
	int getY();
	void ShowPosition();
	friend Point operator+(Point &p);
};

Point::Point(int argX,int argY)
{
	x=argX;
	y=argY;
}

void Point::ShowPosition()
{
	cout<<x<<" "<<y<<endl;
}
int Point::getX()
{
	return x;
}
int Point::getY()
{
	return y;
}
Point operator+(Point &p1,Point &p2)
{
	Point temp(p1.x+p2.x,p1.y+p2.y);
```

```
        return temp;
}

int main(void)
{
        Point p1(1,1);
        Point p2(5,2);
        Point p3=p1+p2;
        p3.ShowPosition();

        return 0;
}
```

friend Point operator+(Point &p);
private으로 선언된 p에 접근하기 위해 friend 선언

p1	+	p2
operator+	(p1	,p2)

2. 단항연산자의 오버로딩

전위증가	후위증가
++p	p++

p.operator++()
 	
Point& operator++()
{
xpos+=1;
ypos+=1;
return *this
}
	p.operator++(int)
	
	const Point operator++(int)
	{
	const Point retobj(xpos, ypos); //
	const Point retobj(*this);
	xpos+=1;
	ypos+=1;
	return retobj;
	}

operator++(Point &ref)
 	
Point& operator--(Point &ref)
{
ref.xpos-=1;
ref.ypos-=1;
return ref;
}
	operator++(Point &ref, int)
	
	const Point operator++(Point &ref, int)
	{
	const Point retobj(ref);
	ref.xpos+=1;
	ref.ypos+=1;
	return retobj;
	}

①

return *this;

this : 객체 자신을 가리키는 포인터

*this : 포인터가 가리키는 대상을 참조 = 자기 자신을 리턴

②

전위연산과 후위연산은 매개변수 안에서 int로 구별

③

전위연산 반환 : 참조값

후위연산 반환 : 값, 임시 객체

Case3. cout, cin 오버로딩

```
#include <iostream>
using std::endl;
using std::cout;

using std::ostream;

class Point{
private:
	int x,y;
public:
	Point(int _x=0, int _y=0):x(_x), y(_y) {}
	friend ostream& operator<<(ostream& os, const Point& p);
};

ostream& operator<<(ostream& os, const Point& p){
	os<<"("<<p.x<<","<<p.y<<")"<<endl;
	return os;
}

int main() {
	Point p(1,3);
	cout<<p;
	return 0;
}
```

cout : ostream의 객체/ cin : istream의 객체

[실습하기]

1. 다음 아래의 코드를 조건에 맞게 변경시키시오.

- point class에 z멤버 함수를 추가하시오.
- x, y, z 멤버 변수를 private로 바꾸고 그 변수를 접근할 수 있는 접근자 getX(), getY(), getZ()를 작성하시오.
- operator+뿐만 아니라 operator-, operator*를 추가적으로 선언한다. 멤버 변수 x, y, z가 private이므로 접근자를 사용해서 연산을 한다.
- namespace point를 선언하고 그 안에 Point p1(3,3,3), p2(2,2,2),p3(0,0,0);를 선언한다. print()함수를 선언하여 p3의 멤버 변수들을 출력한다.

- main함수에서는 namespace point에 접근하여 p1 p2에 +연산하고 p3에 집어 넣고 print(), p1 p2에 -연산하고 p3에 집어 넣고 print(), p1 p2에 *연산하고 p3에 집어 넣고 print()을 차례 수행하는 code를 작성한다.

```
#include <iostream>
using namespace std;
class Point
{
public:
        int x,y;
         Point(int argX,int argY);
         int getX();
         int getY();
         void ShowPosition();
         friend Point operator+(Point &p);
};

Point::Point(int argX,int argY)
{
         x=argX;
         y=argY;
}

void Point::ShowPosition()
{
         cout<<x<<" "<<y<<endl;
}
int Point::getX()
{
         return x;
}
int Point::getY()
{
         return y;
}
Point operator+(Point &p1,Point &p2)
{
         Point temp(p1.x+p2.x,p1.y+p2.y);
         return temp;
}

int main(void)
{
         Point p1(1,1);
         Point p2(5,2);
         Point p3=p1+p2;
         p3.ShowPosition();

         return 0;
}
```

2. 도서관 관리 프로그램 구현하려고 한다. 책 관리를 위한 번호는 '[도서분야] 상위번호.하위번호' 형태로 부여 된다. 각각의 번호는 도서분야(3자리수), 상위번호(3자리수), 하위번호(4자리수)의 구성을 띈다.

참고 ☞

1. 자리수를 맞추기 위해 setw()을 사용하시오

ex. input >> setw(3) >> number.areaCode;

2. (,), 공백은 ignore한다

ex. input.ignore(2); // skip) and space

BookNumber.h

```
#include <iostream>
#include <string>
using namespace std;

class BookNumber
{
    friend ostream &operator<<( ostream &, const BookNumber & );
    friend istream &operator>>( istream &, BookNumber & );
private:
    string bookArea;
    string number1;
    string number2;
};
```

출력결과 ☞