- 연산자 오버로딩 소개
- 이항 연산자 오버로딩
- 단항 연산자의 오버로딩
- cout, cin, endl 구현
- 배열 인덱스 연산자 오버로딩
- 대입 연산자 오버로딩

Jong Hyuk Park

Jong Hyuk Park

연산자 오버로딩 (operator overloading)

❖ C++에서는 기존의 C 언어에서 제공하고 있는 연산자에 대하여 그 의미를 다시 부여하는 것을 "**연산자 오버로딩**" 또는 "연산자 중복(재정의)"라 함

- 연산자 오버로딩은 기본적으로 함수의 오버로딩과 같이 연산자도 하나의 함수라는 개념을 사용하여 중복 정의
- 중복되는 연산자 함수는 클래스의 멤버함수나 프렌드 함수로 정의
- 함수 이름 대신에 operator 키워드를 사용하고 다음에 연산자를 기술

```
반환형 operator 연산자(가인수 리스트);
```

❖ 두 가지 형태로 표현

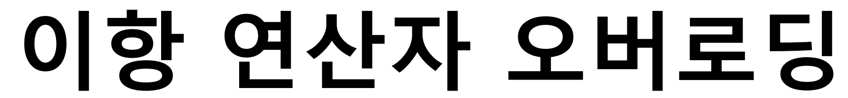
- 멤버함수에 의한 오버로딩
- 전역함수에 의한 오버로딩, friend 함수 선언

연산자 오버로딩 정의 예

```
class Complex  {
        .........
        // + 연산자 오버로딩
        Complex operator +(Complex x);    // 멤버함수
        // friend Complex operator+(Complex x,Complex y);  // 전역함수
        .........
};

Complex a, b, c;
c = a + b; // 중복된 연산자를 사용한 복소수 덧셈 연산 표현
```

❖ 복소수형 클래스 + 연산자 오버로딩 정의 예

- operator + 는 함수와 같이 간주
- c = a + b는 c = a.operator + (b); 로 기술할 수도 있음
  - 객체 a의 멤버함수 operator + 를 호출하고 인수 b를 전달



이항 연산자 오버로딩

Jong Hyuk Park

멤버함수 연산자 오버로딩 예

```
#include <iostream>
using std::endl;
using std::cout;
class Point {
        int x, y;
public:
        Point(int _x=0, int _y=0)   { x = _x; y = _y; }
        void ShowPosition();
        Point operator+(const Point& p);
};
void Point::ShowPosition()
{
        cout<<x<<" "<<y<<endl;
}
Point Point::operator+(const Point& p)
{
        Point temp;
         temp.x = x + p.x;
        temp.y = y + p.y;
        return temp;
}
int main(void)
{
        Point p1(1, 2);
        Point p2(2, 1);
        Point p3=p1+p2;   // p3 = p1.operator+(p2);
        p3.ShowPosition();
        return 0;
}
```

```
#include <iostream>
using std::endl;
using std::cout;
class Point {
        int x, y;
public:
        Point(int _x=0, int _y=0) : x(_x), y(_y) { }
        void ShowPosition();
        Point operator+(const Point& p);
};
void Point::ShowPosition()
{
        cout<<x<<" "<<y<<endl;
}
Point Point::operator+(const Point& p)
{
        Point temp(x+p.x, y+p.y);
        return temp;
}

int main(void)
{
        Point p1(1, 2);
        Point p2(2, 1);
        Point p3=p1+p2;   // p3 = p1.operator+(p2);
        p3.ShowPosition();
        return 0;
}
```

전역함수 연산자 오버로딩 예

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

class Point {
	int x, y;
public:
	Point(int _x=0, int _y=0):x(_x), y(_y){}
	void ShowPosition();
	friend Point operator+(const Point& p1, const Point& p2);
};
void Point::ShowPosition(){
	cout<<x<<" "<<y<<endl;
}

Point operator+(const Point& p1, const Point& p2)
{
	Point temp(p1.x+p2.x, p1.y+p2.y);
	return temp;
}
int main(void)
{
	Point p1(1, 2);
	Point p2(2, 1);
	Point p3=p1+p2;   // p3 = operator+(p1,p2)
	p3.ShowPosition();

	return 0;
}
```

복소수 연산 예 (1)

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

class Complex {
   double re, im;
public:
    Complex(double r, double i)
                              { re = r; im = i; }
    Complex()             { re = 0; im = 0; }
    Complex operator +(const Complex& c);
                              // + 연산자 중복
    Complex operator -(const Complex& c);
                             // - 연산자 중복
     void show()
           { cout << re << " + i" << im << '\n'; }
};

Complex Complex::operator +(const Complex& c)
{
   Complex tmp;
   tmp.re = re + c.re;
   tmp.im = im + c.im;
   return tmp;
}
```

```
Complex Complex::operator -(const Complex& c)
{
   Complex tmp;

   tmp.re = re - c.re;
   tmp.im = im - c.im;
   return tmp;
}

int main()
{
   Complex a(1.2,10.6), b(2.3, 5), c(2.0,4.4);
   Complex sum, dif;

   sum = a + b;          // 연산자 함수 + 호출
   cout << "a + b = "; sum.show();
   dif = a - b;           // 연산자 함수 - 호출
   cout << "a - b = "; dif.show();

   sum = a + b + c;   // 연산자 함수 + 호출
   cout << "a + b + c = "; sum.show();

   return 0;
}
```

복소수 연산 예 (2)

```
a + b = 3.5 + i15.6
a - b = -1.1 + i5.6
a + b + c = 5.5 + i20
```

복소수 덧셈, 뺄셈을 위해 연산자 +, -를 중복 정의하였다.

각 연산자 함수가 Complex 형의 객체를 반환하므로

a + b + c 와 같이 연속적인 연산이 가능하다. 각 연산자는

다음과 같이 연산자 함수를 호출한다.

```
     a + b     =>  a.operator +(b)
     a - b     =>  a.operator -(b)
   a + b + c => a.operator +(b).operator +(c)
```

연산자 함수의 오버로딩 예

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

class Point {
        int x, y;
public:
        Point(int _x=0, int _y=0) : x(_x), y(_y) { }
        void ShowPosition();
        Point operator+(const Point& p);  // + 연산자 1
        Point operator+(const int v);         // + 연산자 2
};
void Point::ShowPosition()
{
        cout<<x<<" "<<y<<endl;
}

Point Point::operator+(const Point& p)
{
        Point temp(x+p.x, y+p.y);
        return temp;
}

Point Point::operator+(const int d)
{
        Point temp(x+d, y+d);
        return temp;
}
```

```
int main(void)
{
        Point p1(1, 2);
        Point p2(2, 1);

        Point p3=p1+p2;   // p3 = p1.operator+(p2);
        p3.ShowPosition();

        Point p4=p1+10;   // p4 = p1.operator+(10);
        // Point p4 = 10 + p1 ?
        p4.ShowPosition();

        return 0;
}
```

교환법칙 적용 예

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

class Point {
	int x, y;
public:
	Point(int _x=0, int _y=0) : x(_x), y(_y) { }
	void ShowPosition();
	Point operator+(const Point& p);  // + 연산자 1
	Point operator+(const int d);         // + 연산자 2
	friend Point operator+(const int d, const Point&
	p);
	                                                    // + 연산자 3
};
void Point::ShowPosition()
{
	cout<<x<<" "<<y<<endl;
}

Point Point::operator+(const Point& p)
{
	Point temp(x+p.x, y+p.y);
	return temp;
}
Point Point::operator+(const int d)
{
	Point temp(x+d, y+d);
	return temp;
}
```

```
Point operator+(const int d, const Point& p)
{
	return p+d;
}


int main(void)
{
	Point p1(1, 2);
	Point p2(2, 1);

	Point p3=p1+p2;   // p3 = p1.operator+(p2);
	p3.ShowPosition();

	Point p4=p1+10;   // p4 = p1.operator+(10);
	p4.ShowPosition();

	Point p5=10+p2;   // p4 = operator+(10,p2);
	p5.ShowPosition();

	return 0;
}
```

교환 법칙 해결하기

```
/*
   Associative2.cpp
*/
#include <iostream>
using std::endl;
using std::cout;

class Point {
private:
	int x, y;
public:
	Point(int _x=0, int _y=0):x(_x), y(_y){}
	void ShowPosition();
	Point operator+(int val); //operator+라는 이름의
	함수
	friend Point operator+(int val, const Point& p);
};
void Point::ShowPosition() {
	cout<<x<<" "<<y<<endl;
}
Point Point::operator+(int val) {
	Point temp(x+val, y+val);
	return temp;
}

Point operator+(int val, Point& p)
{
	return p+val;
}
```

```
int main(void)
{
	Point p1(1, 2);
	Point p2=p1+3;
	p2.ShowPosition();

	Point p3=3+p2;
	p3.ShowPosition();

	return 0;
}
```

교환 법칙 해결하기

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

class AAA{
	char name[20];
public:
	AAA(char* _name){
		strcpy(name, _name);
		cout<<name<<" 객체 생성"<<endl;
	}
	~AAA(){
		cout<<name<<" 객체 소멸"<<endl;
	}
};

int main(void)
{
	AAA aaa("aaa Obj");
	cout<<"--------임시 객체 생성 전---------"<<endl;
	AAA("Temp Obj");
	cout<<"--------임시 객체 생성 후---------"<<endl;
	return 0;
}
```

```
aaa object 객체 생성
--------임시 객체 생성 전---------
Temp object 객체 생성
Temp object 객체 생성
--------임시 객체 생성 후---------
aaa object 객체 소멸
```

교환 법칙 해결하기

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

class Point {
private:
	int x, y;
public:
	Point(int _x=0, int _y=0):x(_x), y(_y){}
	void ShowPosition();
	Point operator+(int val); //operator+라는 이름의
	함수
	friend Point operator+(int val, const Point& p);
};
void Point::ShowPosition() {
	cout<<x<<" "<<y<<endl;
}
Point Point::operator+(int val) {
	//Point temp(x+val, y+val);
	//return temp;
	return Point(x+val, y+val);
}

Point operator+(int val, Point& p)
{
	return p+val;
}
```

```
int main(void)
{
	Point p1(1, 2);
	Point p2=p1+3;
	p2.ShowPosition();

	Point p3=3+p2;
	p3.ShowPosition();

	return 0;
}
```

관계 연산자 예

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

class Point {
	int x, y;
public:
	Point(int _x=0, int _y=0) : x(_x), y(_y) { }
	void ShowPosition();
	Point operator+(const Point& p);  // + 연산자 1
	Point operator+(const int v);        // + 연산자 2
	 int operator<(const Point& p);     // < 연산자
};
void Point::ShowPosition()
{
	cout<<x<<" "<<y<<endl;
}
Point Point::operator+(const Point& p)
{
	Point temp(x+p.x, y+p.y);
	return temp;
}
Point Point::operator+(const int d)
{
	Point temp(x+d, y+d);
	return temp;
}
```

```
Point Point::operator<(const Point& p)
{
	return ((x < p.x) && (y < p.y));
}


int main(void)
{
	Point pt(1, 2);

	if (pt < pt+10)  // pt.operator<(pt.operator+(10))
	   cout << "pt < pt+10₩n";
	 else
	   cout << "pt >= pt+10₩n";

	return 0;
}
```

단항 연산자의 오버로딩

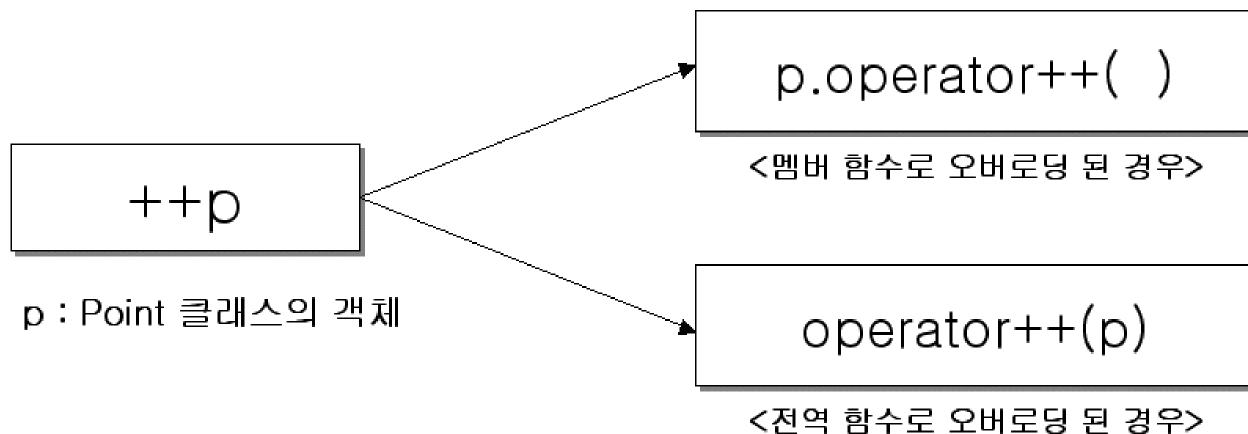
Jong Hyuk Park

단항 연산자 중복

❖ ++, --, ~, ! 등과 같은 **단항 연산자(unary operator)**도 이항 연산자와 같이 중복 정의

❖ 멤버함수에 의한 오버로딩
- 단항 연산자의 경우 하나의 오퍼랜드는 연산자 함수를 정의하는 객체가 되므로 연산자 멤버함수의 인수가 없게 됨

❖ 전역함수에 의한 오버로딩
- 연산자 전역함수에 하나의 오퍼랜드가 전달



단항 연산자 예 1

```
#include <iostream>
using std::endl;
using std::cout;
class Point {
private:
	int x, y;
public:
	Point(int _x=0, int _y=0):x(_x), y(_y){}
	void ShowPosition();
	Point& operator++();
	friend Point& operator--(Point& p);
};
void Point::ShowPosition(){
	cout<<x<<" "<<y<<endl;
}
Point& Point::operator++()
{
	x++;  y++;
	return *this;
}
Point& operator--(Point& p)
{
	p.x--;     p.y--;
	return p;
}
```

```
int main(void)
{
	Point p(1, 2);
	++p;   //p의 x, y 값을 1씩 증가.
	       // p.operator++()
	p.ShowPosition();  //2, 3

	--p;    //p의 x, y 값을 1씩 감소.
	        // operator--(p)
	p.ShowPosition();  //1, 2

	++(++p);  // (p.operator++()).operator++()
	p.ShowPosition();  //3, 4

	--(--p); // operator--(operator--(p))
	p.ShowPosition();  //1, 2

	return 0;
}
```

단항 연산자 예 2

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

class Point {
	int x, y;
public:
	Point(int _x=0, int _y=0) : x(_x), y(_y) { }
	void ShowPosition();
	Point operator-(const Point& p);   // 이항 연산자 -
	Point operator-();         // 단항연산자 -
};
void Point::ShowPosition()
{
	cout<<x<<" "<<y<<endl;
}

Point Point::operator-(const Point& p)
{
	Point temp(x-p.x, y-p.y);
	return temp;
}

Point Point::operator-()
{
	Point temp(-x, -y);
	return temp;
}
```

```
int main(void)
{
	Point p1(3, 8);
	Point p2(2, 1);

	Point p3=p1-p2;   // p3 = p1.operator-(p2);
	p3.ShowPosition();

	Point p4=-p1;   // p4 = p1.operator-();
	p4.ShowPosition();

	return 0;
}
```

단항 연산자의 오버로딩

❖ 선 연산과 후 연산의 구분

```
++p  →  p.operator++();
p++  →  p.operator++(int);
```

```
--p  →  p.operator--();
p--  →  p.operator--(int);
```

단항 연산자의 오버로딩

```
class Point {
private:
    int x, y;
public:
    Point(int _x=0, int _y=0):x(_x), y(_y){}
    void ShowPosition();
    Point& operator++();
    Point operator++(int);
};
void Point::ShowPosition(){
    cout<<x<<" "<<y<<endl;
}

Point& Point::operator++(){
    x++;
    y++;
    return *this;
}
Point Point::operator++(int){
    Point temp(x, y);  // Point temp(*this);
    x++;
    y++;
    return temp;
}
```

```
int main(void)
{
    Point p1(1, 2);
    (p1++).ShowPosition();
    p1.ShowPosition();

    Point p2(1, 2);
    (++p2).ShowPosition();
    return 0;
}
```

Jong Hyuk Park

입출력 연산자 개요

❖ C++의 **입출력 연산자 <<, >>** 도 연산자 오버로딩에 의해 구현된 것

- 이름영역 std의 클래스 ostream, istream 클래스에서 정의
- 미리 정의된 자료형에 대해 입출력

❖ 입출력 연산자 <<, >> 로 사용자 정의 객체에 적용하려면 아래와 같은 전역함수(friend 함수) 정의

```
ostream& operator<<(ostream& os, class
      ob)
{
    ......
    return os;
}
```

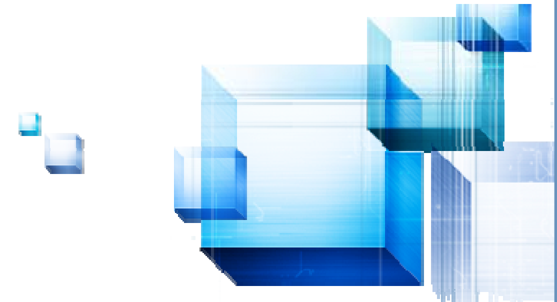
```
istream& operator>>(istream& is, class
      ob)
{
    ......
    return is;
}
```

ostream 구현 예 1

```
#include<stdio.h>

namespace mystd  //mystd라는 이름공간 시작.
{
	char* endl="₩n";

	class ostream // 클래스 ostream 정의
	{
	public:
		void operator<<(char * str) {
			printf("%s", str);
		}
		void operator<<(int i) {
			printf("%d", i);
		}
		void operator<<(double i) {
			printf("%e", i);
		}
	};

	ostream cout;  //ostream 객체 생성
} // mystd 이름공간 끝.
```

```
using mystd::cout;
using mystd::endl;

int main()
{
	cout<<"Hello World \n";
		// cout.operator<<("Hello World \n");
	cout<<3.14;
	cout<<endl;
	cout<<1;
	cout<<endl;

	return 0;
}
```

ostream 구현 예 2



```
#include<stdio.h>

namespace mystd  //mystd라는 이름공간 시작.
{
	char* endl="₩n";
	class ostream // 클래스 ostream 정의
	{
	public:
		ostream& operator<<(char * str) {
			printf("%s", str);
			return *this;
		}
		ostream& operator<<(int i) {
			printf("%d", i);
			return *this;
		}
		ostream& operator<<(double i) {
			printf("%e", i);
			return *this;
		}
	};

	ostream cout;  //ostream 객체 생성
} // mystd 이름공간 끝.
```

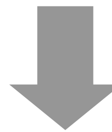
```
using mystd::cout;
using mystd::endl;

int main()
{
	cout<<"Hello World"<<endl<<3.14<<endl;

	return 0;
}
```

사용자 정의 객체 출력 예

❖ <<, >> 연산자의 오버로딩

- Point 객체를 기반으로 하는 <<, >> 입 출력 연산

```
cout<<p → cout.operator<<(p);   // (x)
cout<<p → operator<<(cout, p);  // (o)
```



```
ostream& operator<<(ostream& os, const Point& p)
```

사용자 정의 객체 출력 예

```
#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(void)
{
	Point p(1, 3);
	cout<<p;   // operator<<(cout, p);

	return 0;
}
```

Jong Hyuk Park

인덱스 연산자

❖ 기본 자료형 데이터 저장 배열 클래스

- IdxOverloading1.cpp

❖ 객체 저장할 수 있는 배열 클래스

- IdxOverloading2.cpp

```
arr[i] → arr.operator[](i);
```

배열 클래스 예

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

const int SIZE=3;  // 저장소의 크기.

class Arr {
private:
	int arr[SIZE];
	int idx;
public:
	Arr():idx(0){}
	int GetElem(int i);   // 요소를 참조하는 함수.
	void SetElem(int i, int elem);  // 저장된 요소를 변경하는 
	함수.
	void AddElem(int elem);  // 배열에 데이터 저장하는 함수.
	void ShowAllData();
};
int Arr::GetElem(int i){
	return arr[i];
}
void Arr::SetElem(int i, int elem){
	if(idx<=i){
		cout<<"존재하지 않는 요소!"<<endl;
		return;
	}
	arr[i]=elem;
}
```

```
void Arr::AddElem(int elem){
	if(idx>=SIZE) {
		cout<<"용량 초과!"<<endl;
		return ;
	}
	arr[idx++]=elem;
}
void Arr::ShowAllData(){
	for(int i=0; i<idx; i++)
		cout<<"arr["<<i<<"]="<<arr[i]<<endl;
}

int main(void)
{
	Arr arr;
	arr.AddElem(1); arr.AddElem(2); arr.AddElem(3);
	arr.ShowAllData();

	// 개별 요소 접근 및 변경
	arr.SetElem(0, 10); arr.SetElem(1, 20); arr.SetElem(2, 
	30);

	cout<<arr.GetElem(0)<<endl;
	cout<<arr.GetElem(1)<<endl;
	cout<<arr.GetElem(2)<<endl;

	return 0;
}
```

배열 인덱스 연산자 오버로딩 예

```
#include <iostream>
using std::endl;
using std::cout;
const int SIZE=3;  // 저장소의 크기.

class Arr {
private:
	int arr[SIZE];
	int idx;
public:
	Arr():idx(0){}
	int GetElem(int i);   // 요소를 참조하는 함수.
	void SetElem(int i, int elem);  // 저장된 요소를 변경하는
	함수.
	void AddElem(int elem);  // 배열에 데이터 저장하는 함수.
	void ShowAllData();
	int& operator[](int i); // 배열 요소에 접근
};
int Arr::GetElem(int i){
	return arr[i];
}
int& Arr::operator[](int i){
	return arr[i];
}
void Arr::SetElem(int i, int elem){
	if(idx<=i){
		cout<<"존재하지 않는 요소!"<<endl;
		return;
	}
	arr[i]=elem;
}
```

```
void Arr::AddElem(int elem){
	if(idx>=SIZE) {
		cout<<"용량 초과!"<<endl;
		return ;
	}
	arr[idx++]=elem;
}
void Arr::ShowAllData(){
	for(int i=0; i<idx; i++)
		cout<<"arr["<<i<<"]="<<arr[i]<<endl;
}

int main(void)
{
	Arr arr;
	arr.AddElem(1); arr.AddElem(2); arr.AddElem(3);
	arr.ShowAllData();

	// 개별 요소 접근 및 변경
	arr[0]=10;   // arr.operator[](0);
	arr[1]=20;  arr[2]=30;

	cout<<arr[0]<<endl;
	cout<<arr[1]<<endl;
	cout<<arr[2]<<endl;

	return 0;
}
```

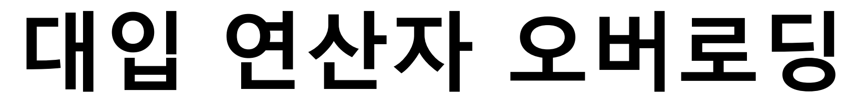
사용자 정의 객체 저장 배열 예 (1)

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

/***************** Point Class **************************/
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<<"]";
	return os;
}

/***************** PointArr Class *************************/
const int SIZE=5;  // 저장소의 크기.

class PointArr {
private:
	Point arr[SIZE];
	int idx;
```

```
public:
	PointArr():idx(0){}
	void AddElem(const Point& elem);
	void ShowAllData();
	Point& operator[](int i); // 배열 요소에 접근.
};

void PointArr::AddElem(const Point& elem){
	if(idx>=SIZE) {
		cout<<"용량 초과!"<<endl;
		return ;
	}
	arr[idx++]=elem;
}
void PointArr::ShowAllData(){
	for(int i=0; i<idx; i++)
		cout<<"arr["<<i<<"]="<<arr[i]<<endl;
}

Point& PointArr::operator[](int i){
	return arr[i];
}
```

사용자 정의 객체 저장 배열 예 (2)

```
int main(void)
{
	PointArr arr;

	arr.AddElem(Point(1, 1));
	arr.AddElem(Point(2, 2));
	arr.AddElem(Point(3, 3));
	arr.ShowAllData();

	// 개별 요소 접근 및 변경
	arr[0]=Point(10, 10);
	arr[1]=Point(20, 20);
	arr[2]=Point(30, 30);

	cout<<arr[0]<<endl;
	cout<<arr[1]<<endl;
	cout<<arr[2]<<endl;

	return 0;
}
```



대입 연산자 오버로딩

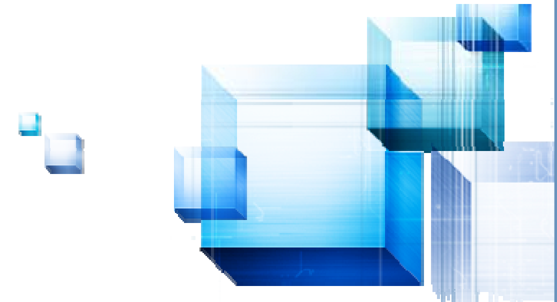
Jong Hyuk Park

디폴트 대입 연산자

- ❖ 클래스 정의 시 기본적으로(디폴트로) '=' 대입 연산자 제공
  - 멤버 대 멤버 복사

- ❖ 디폴트 대입 연산자는 디폴트 복사 생성자와 유사
  - 기본적으로는 얕은 복사(shallow copy)
    - 포인터 멤버변수를 갖는 소멸자 호출 시 문제 발생 가능성
  - 깊은 복사(deep copy)로 문제 해결

디폴트 대입 연산자 예



```
#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<<"]";
	return os;
}
int main(void)
{
	Point p1(1, 3);
	Point p2(10, 30);
	cout<<p1<<endl;
	cout<<p2<<endl;

	p1=p2;  // p1.operator=(p2)
	cout<<p1<<endl;

	return 0;
}
```

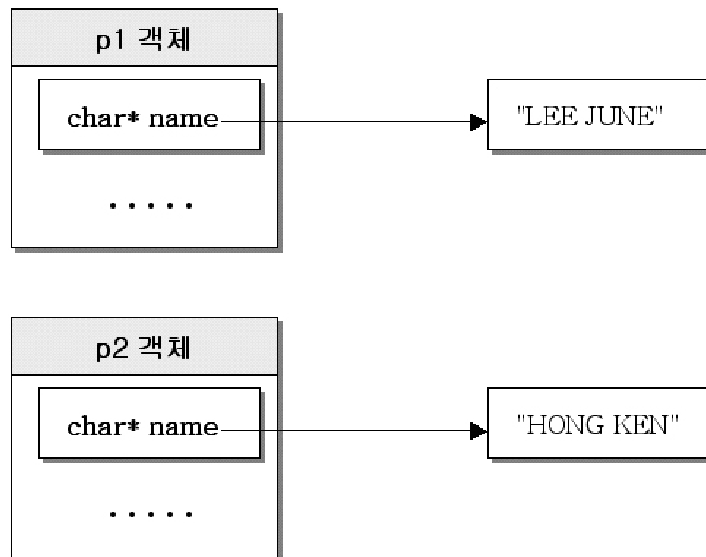
```
Point& Point::operator=(const Point& p)
{
	x=p.x;
	y=p.y;
	return *this;
}
```

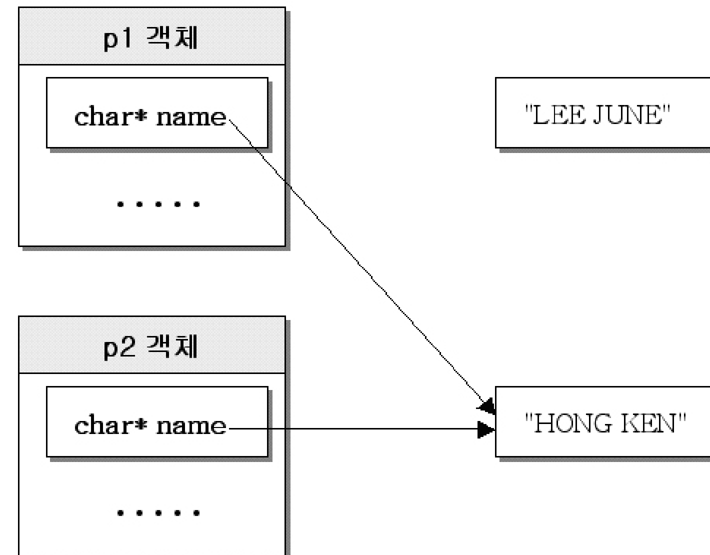
디폴트 대입 연산자 예: 얕은 복사 (1)

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

class Person {
private:
	char* name;
public:
	Person(char* _name);
	Person(const Person& p);
	~Person();
	friend ostream& operator<<(ostream& os, const Person& p);
};

Person::Person(char* _name){
	name= new char[strlen(_name)+1];
	strcpy(name, _name);
}
Person::Person(const Person& p){
	name= new char[strlen(p.name)+1];
	strcpy(name, p.name);
}
Person::~Person(){
	delete[] name;
}
```

```
ostream& operator<<(ostream& os, const Person&
	p)
{
	os<<p.name;
	return os;
}

int main()
{
	Person p1("LEE JUNE");
	Person p2("HONG KEN");

	cout<<p1<<endl;
	cout<<p2<<endl;

    p1=p2;  // 문제의 원인, p1.operator=(p2);

	cout<<p1<<endl;

	return 0;
}   // 소멸자 호출 에러 !!
```

디폴트 대입 연산자 예: 얕은복사 (2)





p1=p2 실행 전

p1=p2 실행 후

디폴트 대입 연산자 예: 깊은 복사

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

class Person {
private:
	char* name;
public:
	Person(char* _name);
	Person(const Person& p);
	~Person();
	Person& operator=(const Person& p);
	friend ostream& operator<<(ostream& os, const
	Person& p);
};

Person::Person(char* _name){
	name= new char[strlen(_name)+1];
	strcpy(name, _name);
}
Person::Person(const Person& p){
	name= new char[strlen(p.name)+1];
	strcpy(name, p.name);
}
Person::~Person(){
	delete[] name;
}
```

```
Person& Person::operator=(const Person& p){
	delete []name;
	name= new char[strlen(p.name)+1];
	strcpy(name, p.name);
	return *this;
}

ostream& operator<<(ostream& os, const Person& p)
{
	os<<p.name;
	return os;
}

int main()
{
	Person p1("LEE JUNE");
	Person p2("HONG KEN");

	cout<<p1<<endl;
	cout<<p2<<endl;

	p1=p2;  // p1.operator=(p2);

	cout<<p1<<endl;
	return 0;
}
```

프로젝트 과제 3

- ❖ 프로젝트 과제2에 10-9절의 은행계좌 관리 과제 프로그램의 내용을 추가하여라.
  - ❖ Container 클래스, 대입 연산자 오버로딩

  - ❖ 9개의 파일로 분리(9-6절)

Q & A

Jong Hyuk Park