10장. 연산자 오버로딩

김미경
부산대학교
ddosun@pusan.ac.kr

10-1 연산자 오버로딩의 의미

```
/*   1_OpIntro.cpp   */
class Point {
private:
    int x, y;
public:
    Point(int _x=0, int _y=0):x(_x), y(_y){}
    void ShowPosition();
    void operator+(int val);
};

void Point::ShowPosition() {
    cout<<x<<" "<<y<<endl;
}

void Point::operator+(int val) {
    x+=val;
    y+=val;
}
```

```
int main(void)
{
    Point p(3, 4);
    p.ShowPosition();

    p.operator+(10);
    p.ShowPosition();
    return 0;
}
```

10-2 연산자 오버로딩 두 가지 방법

- **멤버 함수에 의한 오버로딩**

```
///*   2_멤버함수연산자오버로딩.cpp   *///////
class Point {
private:
    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.ShowPosition();

    return 0;
}
```

10-2 연산자 오버로딩 두 가지 방법

- **전역 함수에 의한 오버로딩**

```
///*   3_전역함수연산자오버로딩.cpp   *///////
class Point {
private:
    int x, y;
public:
    Point(int _x=0, int _y=0):x(_x), y(_y){}
    void ShowPosition();
    friend Point operator-(const Point&, const Point&);
};
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.ShowPosition();

    return 0;
}
```

10-2 연산자 오버로딩 두 가지 방법

- **연산자 오버로딩의 주의 사항**
  - **본 의도를 벗어난 연산자 오버로딩!**
  - **연산자 우선 순위와 결합성은 유지된다.**
  - **디폴트 매개변수 설정이 불가능하다.**
  - **디폴트 연산자들의 기본 기능 변경 불가**

```
int operator+(int a, int b)       // 정의 불가능한 함수
{
    return a+b+3;
}
```

10-3 단항 연산자의 오버로딩

- **증가, 감소 연산자 오버로딩**

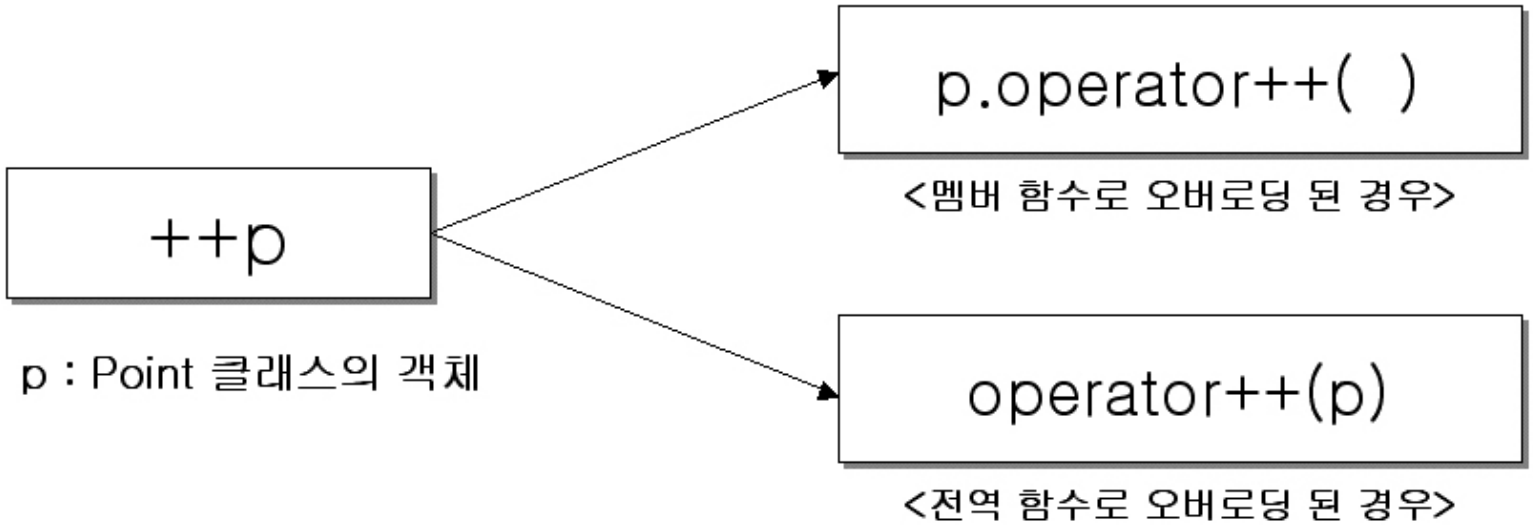

그림 10-7

10-3 단항 연산자의 오버로딩

```
/* 4_전치증가연산자.cpp*/
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.ShowPosition();       //2, 3

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

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

    --(--p);
    p.ShowPosition();       //1, 2
    return 0;
}
```

10-3 단항 연산자의 오버로딩

- **선 연산과 후 연산의 구분**

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

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

10-3 단항 연산자의 오버로딩

```
4_전치증감.cpp 에 후치증가 추가
p++은 멤버 함수로, p- 는 전역으로 만들기
class Point{
      Point operator++(int); //후치
      friend Point operator—(Point& p, int); //후치
};
Point Point::operator++(int)
{
        Point temp(x,y);  //Point temp(*this);
         x++;
         y++;
        return temp;
}
//전역함수
Point operator--(Point& p, int ) //후치
{
      Point temp(p);
      p.x--;
      p.y--;
      return temp;
}
```

```
int main(void)
{
        Point p1(3,4);
        p1.ShowPosition();

        ++p1;
        cout <<  " 전치 연산자 ++p1후 p1 =  " ;
        p1.ShowPosition();
        cout << endl;

        cout <<  " p1++의 결과 : " ;
        (p1++).ShowPosition();
        cout << "후치 (p1++)실행 후 p1 :" ;
        p1.ShowPosition();
        cout << endl;

        --p1;
        cout << "전치 연산자 --p1후 p1 = ";
        p1.ShowPosition();
        cout << endl;

        cout << "후치p1--의 결과 :";
        (p1--).ShowPosition();
        cout << "(p1--)실행 후 p1 :" ;
        p1.ShowPosition();
        cout << endl;
}
```

10-4 교환 법칙 해결하기

- **교환 법칙의 적용**
  - **Associative1.cpp**
  - **Associative2.cpp**

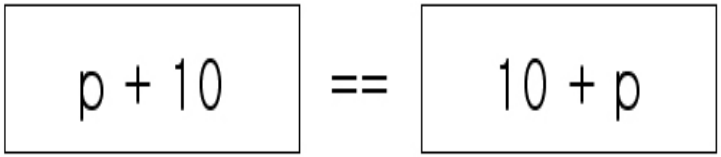

그림 10-9

```
1  /////*   Associative1.cpp   */////
2  #include <iostream>
3  using std::endl;
4  using std::cout;
5
6  class Point {
7  private:
8      int x, y;
9  public:
10     Point(int _x=0, int _y=0):x(_x), y(_y){}
11     void ShowPosition();
12     Point operator+(int val); //operator+라는 이름의 함수
13 };
14 void Point::ShowPosition() {
15     cout<<x<<" "<<y<<endl;
16 }
17 Point Point::operator+(int val) {
18     Point temp(x+val, y+val);
19     return temp;
20 }
21
22 int main(void)
23 {
24     Point p1(1, 2);
25     Point p2=p1+3;
26     p2.ShowPosition();
27
28     return 0;
29 }
```

10-4 교환 법칙 해결하기

- **임시 객체의 생성**
  - **TempObj.cpp**

Point(3, 4);

```
////*   TempObj.cpp   *////
#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;
}
```

10-4 교환 법칙 해결하기

- **임시 객체 생성의 적용**
  - **6_상수더하기_교환법칙.cpp에 적용**

```
Point Point::operator+(int val)
{
    return Point(x+val, y+val);
}
```

```
/* 6_상수 더하기 교환법칙 */
#include <iostream>
using namespace std;
class Point{
    int x; int y;
public:
    Point(int _x=0, int _y=0):x(_x), y(_y){ }
    void ShowPosition();
    Point operator+(int val);//교환법칙 성립되게 하려면
    Point operator+(const Point& p);
    friend Point operator+(int val, Point& p);
};
void Point::ShowPosition()
{
    cout << x << "," << y << endl;
}
Point Point::operator+(int val)
{
    Point temp(x+val, y+val);
    return temp;
}
Point Point::operator+(const Point& p)
{
    Point temp(x+p.x, y+p.y);
    return temp;
}
Point operator+(int val, Point& p)
{
    return p+val;
}
```

```
int main(){
    Point p1(3,4);
    p1.ShowPosition();

    Point p2 = p1 +3;
    cout <<endl << "p1+3 = p2 : " ;
    p2.ShowPosition();

    Point p3 = 3 + p2;
    cout <<endl << "3 + p2 = p3 : " ;
    p3.ShowPosition();
}
```

10-5 cout, cin 그리고 endl의 비밀

```
#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;
}
```

10-5 cout, cin 그리고 endl의 비밀

- **<<, >> 연산자의 오버로딩**
  - **Point 객체를 기반으로 하는 <<, >> 입 출력 연산**
  - **OpOverloading6.cpp**

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

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

10-5 cout, cin 그리고 endl의 비밀

```
 1  /*8_출 력 연 산 자 */
 2  #include <iostream>
 3  using namespace std;
 4  class Point{
 5      int x; int y;
 6  public:
 7      Point(int _x=0, int _y=0):x(_x), y(_y){ }
 8      friend ostream& operator<<(ostream& os, Point& p);
 9  };
10  ostream& operator<<(ostream& os, Point& p)
11  {
12      os << " [ " << p.x << " , "  << p.y<< " ] " << endl;
13      return os;
14  }
15  int main()
16  {
17      Point p1(1,3);
18      cout << p1 ;
19  }
```

10-6 인덱스 연산자

- 기본 자료형 데이터 저장 배열 클래스
  - IdxOverloading1.cpp

- 객체 저장할 수 있는 배열 클래스
  - IdxOverloading2.cpp

arr[i] ➔ arr.operator[](i);

IdxOverloading1.cpp

```
//*   9_IdxOverloading1.cpp   */
#include <iostream>
using std::endl;using std::cout;
const int SIZE=3;  // 저장소의 크기.
class Arr {
        int arr[SIZE];    int idx;
public:
        Arr():idx(0){}
        int GetElem(int i);   // 요소를 참조하는 함수.
        void SetElem(int i, int elem); //저장된 요소 변경함수.
        void AddElem(int elem);  // 배열에 데이터 저장 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;
}
```

IdxOverloading2.cpp

- 앞 예제에
- [ ]연산자 오버로딩 추가
- int& operator[](int i);

  int& Arr::operator[](int i){

  return arr[i];

  }

  main의 개별 요소 접근 부분 수정

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

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[1]=20;
    arr[2]=30;

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

    return 0;
}
```

10-7 대입 연산자 오버로딩

- 디폴트 대입 연산자
  - 멤버 대 멤버 복사
  - DefaultSubOp.cpp

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

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

DefaultSubOp.cpp

```
 1  /*9_디 폴 드 대 입 연 산 자 */
 2  #include <iostream>
 3  using namespace std;
 4  class Point{
 5      int x; int y;
 6  public:
 7      Point(int _x=0, int _y=0):x(_x), y(_y){ }
 8      friend ostream& operator<<(ostream& os, Point& p);
 9  };
10  ostream& operator<<(ostream& os, Point& p)
11  {
12      os << "[ " << p.x << " , "  << p.y<< " ] " << endl;
13      return os;
14  }
15  int main()
16  {
17      Point p1(1,3);
18      cout << "p1 : "  << p1 ;
19
20      Point p2(10,30);
21      cout << "p2 : "  << p2 ;
22
23      p1=p2;
24      cout << "p1=p2 후  p1 :"  << p1 << endl;
25
26  }
```

10-7 대입 연산자 오버로딩

- **디폴트 대입 연산자의 문제점**

```
/* 10_디폴트대입연산자문제점 */
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, Person& p)
{
      os << "name  : "  << p.name  << endl;
}
```

```
int main()
{
   Person p1("LEE JUNE");
   Person p2("HONG KEN");
   cout<<p1<<endl;
   cout<<p2<<endl;

   p1=p2;  // 문제의 원인

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

10-7 대입 연산자 오버로딩

- **디폴트 대입 연산자의 문제점**

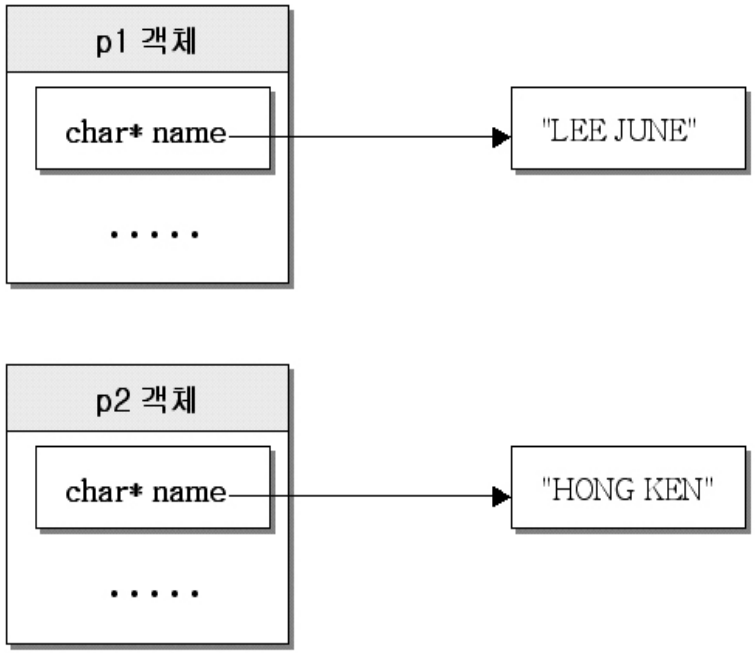

그림 10-13

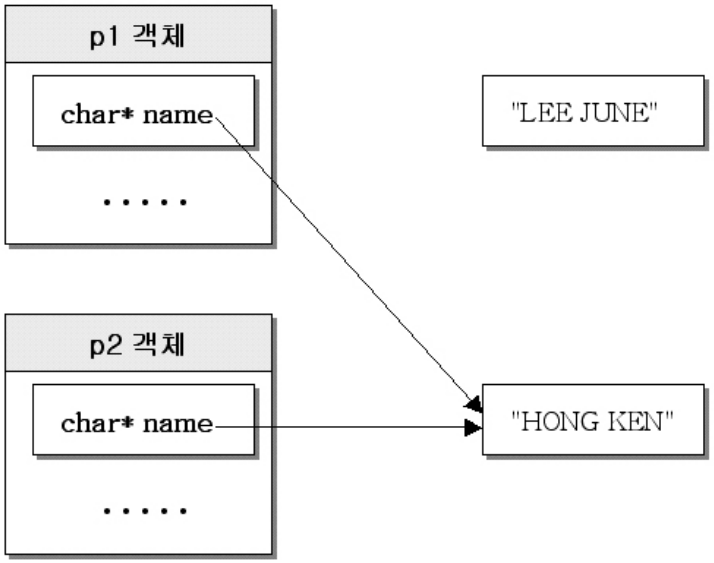

그림 10-14

10-7 대입 연산자 오버로딩

- **깊은 복사(Deep Copy)를 하는 대입 연산자**

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