4-1. 객체와 friend 함수

♲ this 포인터

this포인터는 어떤 객체에 의해 멤버함수가 호출되었는지 멤버함수를 호출한 객체의 주소값을 저장한 채 멤버함수의 매개변수로 전달되는 포인터 변수이다. 멤버함수는 다른 모든 멤버변수나 멤버함수에 접근할 때 this포인터를 통해 접근한다. this 포인터는 컴파일러에 의해 자동으로 선언되고 사용되어 진다. 단, 멤버호출시 사용자가 this 포인터를 명시적으로 기술하는 것도 가능하다.

1. this 포인터의 원리 및 사용방법

설명 : this포인터는 멤버함수가 호출될 때 해당 멤버함수의 첫 번째 매개변수가 된다. this 포인터에는 멤버함수를 호출한 객체의 주소값이 보관되며 모든 멤버함수는 반드시 this포인터를 매개변수로 받아가야 한다.

예제 1-1 : this포인터를 명시적으로 사용한 예제

```
#include <iostream.h>

class day3 {
	int nai;
	double ki;
public :
	// 모든 멤버함수는 첫번째 매개변수로 자신을 호출한 객체의 주소를 받아와
	// this라는 포인터에 보관 시킨다.
	void set(int a, double b) {	// void set ( day3 *this, int a, double b) {
		// 멤버함수 내에서 다른 멤버를 호출할 때 this포인터가 가리키는 객체의 멤버를
		// 불러온다.
		nai = a;		// this->nai = a;
		this->ki = b;	// this-> 는 생략 가능하지만 직접 써도 된다.
	}
	void print( ) {		// void print( day3 *this) {
		cout << "나이 = " << nai << ", 키 = " << ki << endl;
	}
};

void main ( ) {
	day3 ob1, ob2;
	// 모든 멤버함수는 호출될때 자신을 호출한 객체의 주소값을 첫번째 매개변수로 받아간다.
	ob1.set ( 10, 10.5 );		// ob1.set ( &ob1, 10, 10.5 );
	ob1.print( );			// ob1.print( &ob1 );
	ob2.set ( 30, 30.5 );		// ob2.set ( &ob2, 30, 30.5 );
	ob2.print( );			// ob2.print( &ob2 );
}
```

실 행 결 과

```
x=0, y=0
x=10, y=20
x=0, y=0
x=55, y=66
Press any key to continue
```

예제 1-2 :

```
#include <iostream.h>

class day3 {
    int x, y;
public :
    day3() {
        x=0, y=0;
        print();                    // print(this);
    }
    void set(int a, int b) {        // void set( day3 *this, int a, int b)
        this->x=a;
        this->y=b;
    }
    void print() {                  // void print( day3 *this ) {
        cout << "x=" << x << ", y=" << y << endl;
                                    // cout << "x=" << this->x  << ", y=" << this->y << endl;
    }
};
void main ( ) {
    day3 ob1, ob2;

    ob1.set(10,20);     // ob1.set( &ob1, 10, 20);
    ob1.print();        // ob1.print( &ob1 );

    ob2.set(55,66);
    ob2.print();        // ob2.print( &ob2 );
}
```

실 행 결 과

```
x=0, y=0
x=10, y=20
x=0, y=0
x=55, y=66
Press any key to continue
```

♲ friend 함수

class의 private 멤버 변수는 같은 객체의 멤버함수만이 접근 가능하다는 점에서 객체지향 언어의 데이터 은닉성과 캡슐화를 지원한다. 즉, private 멤버 변수의 접근을 class의 내부로 국한시킴으로서 클래스 내부의 수정이 클래스 외부영역에 전혀 영향을 주지 못하기 때문에 프로그램의 유지보수가 쉬워지게 된다. 그러나 경우에 따라서 부득이하게 멤버함수가 아닌 일반함수가 private멤버에 접근해야 하는 경우가 있다. 이때 사용되는 함수가 friend 함수 이다. friend 함수는 일반함수 이면서 클래스의 private 멤버를 호출해 사용할 수 있다.

2. friend 함수의 사용방법

설명 : 일반함수의 원형을 class내부에 선언하면서 함수의 앞쪽에 friend 키워드를 붙이면 friend 함수가 된다. 함수를 선언하는 위치는 private, public, protected 와 관계없이 아무곳에서나 가능하다.

예제 2-1 : friend 함수를 이용해 private 멤버변수의 값에 접근하는 예제

```
#include <iostream.h>

class ADD {
                int su1, su2;
        public:
                void get_val(int, int);
                friend int add(ADD ob);

};

void ADD::get_val(int a, int b) {
        su1 = a;
        su2 = b;
};

int add(ADD ob) {
        return ob.su1 + ob.su2;
}

int main() {
        ADD obj;
        obj.get_val(10, 20);
        cout << add(obj) << endl;
        return 0;
}
```

실 행 결 과

```
30
Press any key to continue
```

예제 2-2 : friend 함수를 이용해 좌표를 초기화 시키는 예제

```
#include <iostream.h>

class Point {
        int xp, yp;
public :
        Point(int x, int y) { xp = x, yp = y; }
        friend void reset(Point &p);  // 클래스 Point의 프렌드 함수
        void prn() { cout << "x 좌표 : " << xp << ", y 좌표 : " << yp << endl; }
};

void reset(Point &p) {
        p.xp = 0;
        p.yp = 0;
}

int main()
{
        Point p(10, 20);
        p.prn();
        reset(p);  // 프렌드 함수 reset() 호출
        p.prn();
        return 0;
}
```

실 행 결 과

```
x 좌표 : 10, y 좌표 : 20
x 좌표 : 0, y 좌표 : 0
Press any key to continue
```

♲ 정적멤버 변수와 정적멤버 함수

정적멤버 변수는 클래스로 만들어진 객체들 사이에서 마치 전역변수와 같이 공용해 사용할 수 있는 멤버 변수이다. 정적멤버 함수도 정적멤버 변수와 같이 모든 객체가 공용해 사용 할 수 있으며 정적멤버 함수와 정적멤버 변수는 객체의 생성과 관계없이 프로그램의 시작과 동시에 만들어 진다.

1. 정적멤버는 클래스로 만들어진 모든 객체가 공유해 사용 할 수 있는 멤버이다.
2. 정적멤버는 객체의 생성과 관계없이 프로그램의 시작과 동시에 만들어 지고 프로그램이 종료되면 사라진다.
3. 정적멤버 변수가 동작하는 원리와 형태는 전역변수와 동일하다. 단, 정적멤버 변수의 경우 객체지향 언어의 데이터 은닉성을 만족시키기 위해 클래스로 만들어진 객체 또는 정적멤버함수를 통해서만 접근해 사용 할 수 있다.
4. 정적멤버 변수는 초기값을 부여 할 수 있다. 단 초기값을 부여하는 행위는 클래스 밖에서 이루어 져야 한다.
5. 정적멤버 함수는 정적멤버변수에 대한 접근을 위해 사용한다.

3. 정적멤버 변수와 정적 멤버 함수

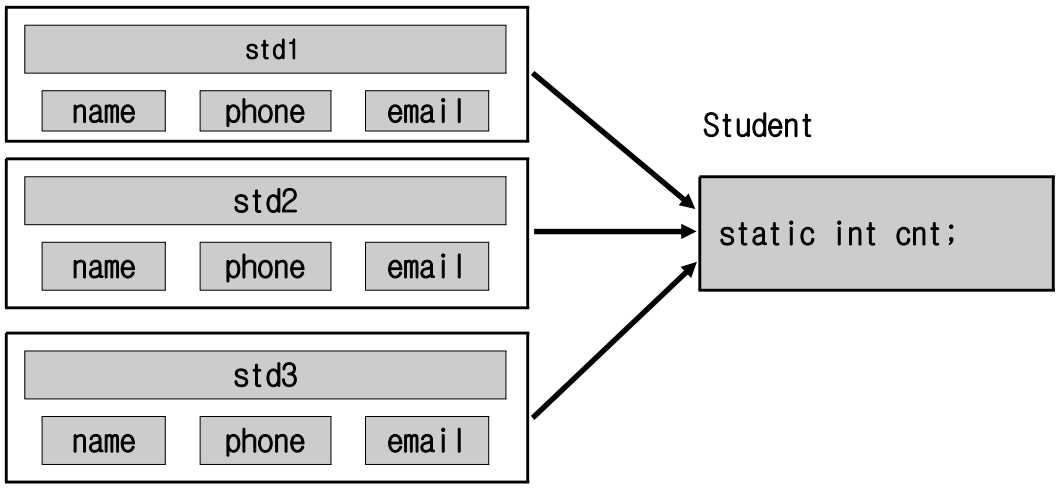
예제 3-1 : 정적 멤버변수의 사용 예제

```
#include<iostream.h>
#include<string.h>

class Student {
	char name[30];
	char phone[20];
	char email[30];
public:
	static int cnt;
	Student(char *na="박지성", char *hp="010-444-5555", char *em="pjs2010@kkk.com");
	void prn();
};

int Student::cnt = 0;

Student::Student(char *na, char *hp, char *em) {
	strcpy(name, na);
	strcpy(phone, hp);
	strcpy(email, em);
	cnt++;
}

void Student::prn( ) {
	cout<<"₩n이름   : " << name;
	cout<<"₩n핸드폰 : " << phone;
	cout<<"₩n이메일 : " << email;
	cout<<"₩n현재까지 등록된 인원수 " << cnt;
	cout<<"₩n";
}

int main( ) {
	Student std1("김연아","010-111-2222" ,"yeonha@ggg.com");
	std1.prn();
	Student std2("이효리","010-333-4444" ,"hyori78@ddd.com");
	std2.prn();
	Student std3;
	std3.prn();
	Student::cnt = 10;  // 변경가능
	cout << "₩n cnt => " << Student::cnt << endl;
	return 0;
}
```

실 행 결 과

```
이름    : 김연아
핸드폰 : 010-111-2222
이메일 : yeonha@ggg.com
현재까지 등록된 인원수 1

이름    : 이효리
핸드폰 : 010-333-4444
이메일 : hyori78@ddd.com
현재까지 등록된 인원수 2

이름    : 박지성
핸드폰 : 010-444-5555
이메일 : pjs2010@kkk.com
현재까지 등록된 인원수 3

 cnt => 10
Press any key to continue
```

예제 3-1 해설: 공용 static 멤버 변수를 이용한 등록 인원수 파악 프로그램 Ex.11의 개념도



예제 3-2 : 정적멤버함수의 사용

```
#include<iostream.h>
#include<string.h>

class Student {
        char name[30];
        char phone[20];
        char email[30];
        static int cnt;   // 정적 멤버 변수
public:
        Student(char *na="박지성", char *hp="010-444-5555", char *em="pjs2010@kkk.com");
        void prn();
        static void cnt_prn();   // 정적 멤버 함수
```

```
};

int Student::cnt = 0;

Student::Student(char *na, char *hp, char *em) {
	strcpy(name, na);
	strcpy(phone, hp);
	strcpy(email, em);
	cnt++;
}

void Student::prn( ) {
	cout << "₩n이름   : " << name << "₩n핸드폰 : " << phone
	     <<"₩n이메일 : " << email << endl;
}

void Student::cnt_prn() {
	cout<<"₩n현재까지 등록된 인원수 : " << cnt << endl;
}

int main( ) {
	Student::cnt_prn();
	Student std1("김연아","010-111-2222" ,"yeonha@ggg.com");
	std1.prn();
	Student std2("이효리","010-333-4444" ,"hyori78@ddd.com");
	std2.prn();
	cout << "₩n[중간 인원수 파악]=> ";
	Student::cnt_prn();
	Student std3;
	std3.prn();
	// Student::cnt = 10;  불가능
	Student::cnt_prn();
	return 0;
}
```

실 행 결 과

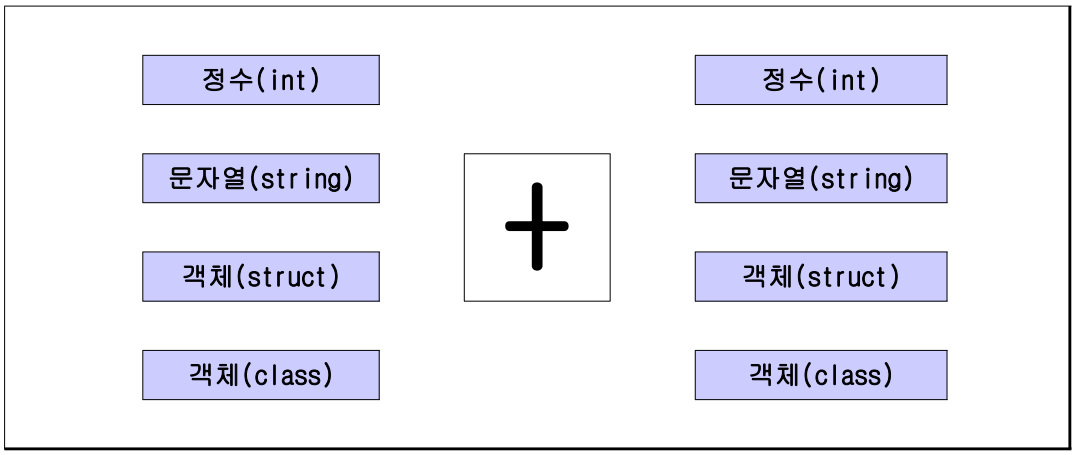
```
현재까지 등록된 인원수 : 0

이름   : 김연아
핸드폰 : 010-111-2222
이메일 : yeonha@ggg.com

이름   : 이효리
핸드폰 : 010-333-4444
이메일 : hyori78@ddd.com

[중간 인원수 파악]=>
현재까지 등록된 인원수 : 2

이름   : 박지성
핸드폰 : 010-444-5555
이메일 : pjs2010@kkk.com

현재까지 등록된 인원수 : 3
Press any key to continue
```

4-2. 연산자 오버로딩

♲ 연산자 오버로딩

이미 사용되고 있는 함수와 동일한 이름으로 인수의 개수나 자료형을 다르게 하여 함수를 정의 하는 것을 함수의 오버로딩이라 한다. C++ 에서는 연산자도 함수로 취급함으로 함수와 동일한 방법으로 연산자도 오버로딩하여 기능을 추가 할 수 있다. 즉, 연산자 오버로딩이란 프로그램의 내용과 적용되는 자료형에 따라 프로그램을 문제 중심으로 표현하기 위해 기존 C언어에서 제공되는 연산자의 의미를 변경하여 사용할 수 있도록 하는 기능이다.
예) “<<” 연산자 (왼쪽 쉬프트와 출력)

+ 연산자 다중정의(overloading) 의 예



오버로딩을 할 수 있는 연산자의 종류

오버로딩이 가능한 연산자	오버로딩이 불가능한 연산자
+, -, *, /, %, ^, &, ~, |, =, <, +=, -=, *=, /=, ^=, &=, |=, <<, >>, <<=, >>=, ==, !=, <=, >=, &&, ||, ++, --, [ ], ( ), ,, ->	:: = scope연산자 ? : = 삼항조건 연산자 . = 구조체멤버 연산자 sizeof = sizeof 연산자

오버로딩이 불가능한 연산자 중 :: (scope) 연산자와 . (구조체멤버 연산자) 연산자는 C++의 대표적 기능이라고 할 수 있는 클래스에 관련된 연산자이기 때문에 클래스를 기반으로 재정의 되는 연산자 오버로딩의 재료로 쓰기에는 무리가 따른다. 또는 삼항 조건 연산자는 피연산자가 셋이나 되기 때문에 오버로딩을 하더라도 까다롭기 때문에 재정의를 못하도록 되어 있다. 하지만 이 연산자들을 사용하지 못한다 하더라도 오버로딩의 재료로 쓸 수 있는 연산자는 많은 편이다.

연산자 오버로딩의 장점

① c++ 언어의 확장성을 증가시킨다
② 자연스럽게 프로그램을 표현할 수 있게 한다
③ 선언방법이 일반 멤버함수와 같다

연산자 오버로딩의 특징

① 연산자 오버로딩이란 C++언어에서 기본 자료형으로 사용하고 있는 연산자를 의미를 변경하여 재정의 하는 것을 말한다. 따라서 기존 C++ 의 연산자를 사용하지 않으면 에러가 발생 한다.
(사용자가 새로운 연산자를 만들 수 없다.)
② 연산자가 취하게 되어있는 오퍼랜드(피연산자)의 개수와 위치는 변경이 불가능하다.
③ 연산자 오버로딩 함수는 디폴트 인수를 가질 수 없다
④ 연산자 오버로딩을 하더라도 연산자의 우선순위는 원래 연산자의 우선순위가 그대로 유지된다.
⑤ C++언어에서는 연산자를 함수로 취급하므로 연산자 오버로딩 방법은 함수정의 방법과 동일하다. 따라서 연산자 오버로딩 시 매개변수의 자료형에 따라 호출될 연산자 함수가 결정된다.
⑥ 연산자 오버로딩 함수에 있어서 매개변수로 전달되는 값은 피연산자 이다.
⑦ 연산자가 오버로딩 될 수 있는 경우는 연산의 대상이 기존에 정의되어 있지 않은 연산에 대해서 이다. 즉, 기본적으로 정의되어 있는 연산은 바꿀 수 없다

연산자 오버로딩 시 주의사항 : 연산자를 재정의 하게 되면 어떠한 경우에는 복잡한 것을 간단하게 또는 산만한 것을 좀 더 논리적으로 표현해 낼 수 있다. 하지만 그에 따라 아래와 같은 부작용이 있다.

① 연산자를 여러 형태로 다중 정의해 놓으면 코드 자체가 이해하기 어렵게 되기도 한다.
② 다른 사람이 자신의 작품을 분석할 때 일일이 그 연산자가 어떻게 정의되어 있는지를 조사해 보아야 한다.
③ 그 코드가 정의 되어 있는 소스 파일이 없이 라이브러리 형태로 되어 있다면 연산자의 정의 내용을 알 수가 없어진다.
④ 주석을 달아 ③의 문제를 해결 할 수도 있지만 너무 복잡하게 되면 일정시간이 지난후에는 자기 자신도 이해하지 못하는 경우가 생기기도 한다.

때문에 연산자 오버로딩은 꼭 필요한 경우가 아니면 자제하는 것이 좋으며 우리가 연산자 오버로딩을 공부하는 이유는 라이브러리나 I/O stream, 또는 남이 구축해놓은 클래스를 좀 더 신속하게 이해하고 적극적으로 활용하기 위해서이다

연산자 오버로딩 선언 형식

```
리턴자료형 operator 연산자(인수1, 인수2, ....) ;

// class 내에 연산자 오버로딩 함수를 정의할 경우
class 클래스명 {
      ....
public :
      리턴자료형 operator 연산자(인수1, 인수2, ....) { ... }
};
```

```
// class 내에는 함수의 원형만 만들고 클래스 밖에 연산자 오버로딩 함수를 정의할 경우
class 클래스명 {
        ....
public :
        리턴자료형 operator 연산자(인수1, 인수2, ....);
};

리턴자료형 클래스명::operator 연산자(인수1, 인수2, ....) { ... }
```

1. friend 함수를 이용한 연산자 오버로딩

설명 : friend 함수를 이용하면 멤버함수가 아니어도 객체의 멤버변수에 접근이 가능함으로 friend 함수를 이용해 연산자 오버로딩 작업을 한다.

예제 1-1 : friend 함수를 이용한 객체와 정수에 대한 + 연산자 오버로딩

```
#include <iostream.h>
class day3 {
        int x, y;
public :
        day3(int a, int b) {
                x = a;
                y = b;
        }
        void print( ) {
                cout << "x = " << x << ", y = " << y << endl;
        }
        friend day3 operator + (day3 temp, int data);   // 객체 + 정수 일때 호출되는 함수
};

day3 operator + (day3 temp, int data) { // friend 함수를 이용해 구현한 연산자 오버로딩 함수
        cout << "난 객체랑 정수랑 더할 때 나오는 연산자 오버로딩 함수다." << endl;
        temp.x = temp.x + data;
        temp.y = temp.y + data;
        return temp;
}

void main () {
        day3 ob1(10,20), ob2(0,0);
        ob2 = ob1 + 10; // day3와 int를 더할 수 있는 +연산자의 오버로딩 함수가 호출된다.
        ob2.print();
}
```

실 행 결 과

```
난 객체랑 정수랑 더할 때 나오는 연산자 오버로딩 함수다.
x = 20, y = 30
Press any key to continue
```

예제 1-2 : friend 함수를 이용한 객체와 정수에 대한 - 연산자 오버로딩

```
#include <iostream.h>
class day3 {
	int x, y;
public :
	day3(int a, int b) {
		x = a;
		y = b;
	}
	void print( ) {
		cout << "x = " << x << ", y = " << y << endl;
	}
	friend day3 operator - (day3 temp, int data);   // 객체 - 정수일 때 호출되는 함수
};
day3 operator - (day3 temp, int data) { // friend 함수를 이용해 구현한 연산자 오버로딩 함수
	cout << "난 객체랑 정수랑 뺄 때 나오는 연산자 오버로딩 함수다." << endl;
	temp.x = temp.x - data;
	temp.y = temp.y - data;
	return temp;
}

void main () {
	day3 ob1(10,20), ob2(0,0);
	ob2 = ob1 - 5;  // day3와 int를 뺄 수 있는 -연산자의 오버로딩 함수가 호출된다.
	ob2.print();
}
```

실 행 결 과

```
난 객체랑 정수랑 뺄 때 나오는 연산자 오버로딩 함수다.
x = 5, y = 15
Press any key to continue
```

예제 1-3 : friend 함수를 이용한 정수와 객체에 대한 + 연산자 오버로딩

```
#include <iostream.h>
class day3 {
	int x, y;
public :
	day3(int a, int b) {
		x = a,  y = b;
	}
	void print( ) {
		cout << "x = " << x << ", y = " << y << endl;
	}
	friend day3 operator + (int data, day3 temp);   // 정수 + 객체일 때 호출되는 함수
};
day3 operator + (int data, day3 temp) { // friend 함수를 이용해 구현한 연산자 오버로딩 함수
	cout << "난 정수와 객체를 더할 때 나오는 연산자 오버로딩 함수다." << endl;
	temp.x = temp.x + data;
	temp.y = temp.y + data;
	return temp;
}
void main () {
	day3 ob1(10,20), ob2(0,0);
	ob2 = 10 + ob1; // int와 day3를 더할 수 있는 +연산자의 오버로딩 함수가 호출된다.
	ob2.print();
}
```

실 행 결 과

```
난 정수와 객체를 더할 때 나오는 연산자 오버로딩 함수다.
x = 20, y = 30
Press any key to continue
```

예제 1-4 : friend 함수를 이용한 객와 객체에 대한 + 연산자 오버로딩

```
#include <iostream.h>
class day3 {
	int x, y;
public :
	day3(int a, int b) {    x = a,  y = b;  }
	void print( ) { cout << "x = " << x << ", y = " << y << endl; }
	friend day3 operator + (day3 temp, int data);   // 객체 + 정수 일때 호출되는 함수
	friend day3 operator - (day3 temp, int data);   // 객체 - 정수 일때 호출되는 함수
	friend day3 operator + (int data, day3 temp);   // 정수 + 객체 일때 호출되는 함수
	friend day3 operator + (day3 temp1, day3 temp2);// 객체 + 객체 일때 호출되는 함수
};
day3 operator + (day3 temp, int data) { // friend 함수를 이용해 구현한 연산자 오버로딩 함수
	cout << "난 객체랑 정수랑 더할 때 나오는 연산자 오버로딩 함수다." << endl;
	temp.x = temp.x + data;
	temp.y = temp.y + data;
	return temp;
}
day3 operator - (day3 temp, int data) { // friend 함수를 이용해 구현한 연산자 오버로딩 함수
	cout << "난 객체랑 정수랑 뺄 때 나오는 연산자 오버로딩 함수다." << endl;
	temp.x = temp.x - data;
	temp.y = temp.y - data;
	return temp;
}
day3 operator + (int data, day3 temp) { // friend 함수를 이용해 구현한 연산자 오버로딩 함수
	cout << "난 정수와 객체를 더할 때 나오는 연산자 오버로딩 함수다." << endl;
	temp.x = temp.x + data;
	temp.y = temp.y + data;
	return temp;
}
day3 operator + (day3 temp1, day3 temp2) { // friend 함수를 이용해 구현한 연산자 오버로딩 함수
	day3 im(0,0);
	cout << "난 객체와 객체를 더할 때 나오는 연산자 오버로딩 함수다." << endl;
	im.x = temp1.x + temp2.x;
	im.y = temp1.y + temp2.y;
	return im;
}
void main () {
	day3 ob1(10,20), ob2(0,0);
	ob2 = ob1 + 10; // day3와 int를 더할 수 있는 +연산자의 오버로딩 함수가 호출된다.
	ob1.print();
	ob2.print();
	ob2 = ob2 - 5;  // day3와 int를 뺄 수 있는 -연산자의 오버로딩 함수가 호출된다.
	ob2.print();
	ob2 = 10 + ob2; // int와 day3를 더할 수 있는 +연산자의 오버로딩 함수가 호출된다.
	ob2.print();
	ob2 = ob1 + ob2; // day3와 day3를 더할 수 있는 +연산자의 오버로딩 함수가 호출된다.
	ob2.print();
}
```

실 행 결 과

```
난 객체랑 정수랑 더할 때 나오는 연산자 오버로딩 함수다.
x = 10, y = 20
x = 20, y = 30
난 객체랑 정수랑 뺄 때 나오는 연산자 오버로딩 함수다.
x = 15, y = 25
난 정수와 객체를 더할 때 나오는 연산자 오버로딩 함수다.
x = 25, y = 35
난 객체와 객체를 더할 때 나오는 연산자 오버로딩 함수다.
x = 35, y = 55
Press any key to continue
```

예제 1-5 : 멤버 함수를 이용한 객체와 객체에 대한 + 연산자 오버로딩

```
#include<iostream.h>

class Complex {
	int real;
	int image;
public :
	Complex() { real = image = 0; }
	Complex(int rr , int ii);
	void prn( );
	Complex operator +(Complex second);
};
void Complex::prn( )  {
	cout<<"(" << real <<" + " << image << "i)" << endl ;
}
Complex::Complex(int rr, int ii)  {
	real=rr;
	image=ii;
}
Complex Complex::operator +(Complex second)
{
	Complex res; //결과값을 저장하기 위한 객체변수
	res.real  = this->real  + second.real;
	res.image = this->image + second.image;
	return res;
}

void main() {
	Complex x(10, 20), y(20, 40);
	Complex z1;     //더한 결과를 저장하기 위한 변수
	z1 = x + y;  // 연산자 함수 호출
	x.prn();
	y.prn();
	cout<<"₩n[ z = x+y 결과 ]₩n";
	z1.prn();

	Complex z2;
	cout<<"₩n[x.operator+(y) 결과 ]₩n";
	z2 = x.operator+(y);
	z2.prn();
}
```

실 행 결 과

```
(10 + 20i)
(20 + 40i)

[ z = x+y 결과 ]
(30 + 60i)

[x.operator+(y) 결과 ]
(30 + 60i)
Press any key to continue
```

예제 1-6 : +연산자와 -연산자의 오버로딩을 이용한 세 객체의 연속적인 연산 예제

```
#include<iostream.h>

class Point {
	int xp, yp;
public :
	Point()  { xp = yp = 0; }  // 생성자1
	Point(int x, int y) { xp = x, yp = y; } // 생성자2
	void get_xy(int &x, int &y)  { x = xp; y = yp; }
	Point operator +(Point ob);  // + 연산자 중복
	Point operator -(Point ob);  // - 연산자 중복
};

Point Point::operator +(Point ob) {
	Point tmp;
	tmp.xp = xp + ob.xp;  // this->xp + ob.xp
	tmp.yp = yp + ob.yp;  // this->yp + ob.yp
	return tmp;
}

Point Point::operator -(Point ob) {
	Point tmp;
	tmp.xp = xp - ob.xp;  // this->xp - ob.xp
	tmp.yp = yp - ob.yp;  // this->yp - ob.yp
	return tmp;
}

void main()
{
	Point p, p1, p2(10, 20), p3(100, 200), p4(5, 5);
	int x, y;
	p = p2 + p3 + p4;  // + 연산자 함수 호출
	p.get_xy(x, y);
	cout << "Wn[ p = p2 + p3 + p4] x값 : " << x << " , y값 : " << y << endl;

	p1 = p2 - p3 - p4;  // - 연산자 함수 호출
	p1.get_xy(x, y);
	cout << "Wn[ p1 = p2 - p3 - p4] x값 : " << x << " , y값 : " << y << endl;
}
```

실 행 결 과

```
[ p = p2 + p3 + p4 ] x값 : 115, y값 : 225

[ p1 = p2 - p3 - p4 ] x값 : -95 y값 : -185

Perss any key to continue...
```

예제 1-7 : (객체+정수) 연산과 관계 연산자 “<” 의 오버로딩 예제

```
#include<iostream.h>

class Point {
	int xp, yp;
public :
	Point()  { xp = yp = 0; }                          // 생성자1
	Point(int x, int y) { xp = x, yp = y; } // 생성자2
	void get_xy(int &x, int &y)  { x = xp; y = yp; }
	Point operator +(int digit);                       // + 연산자 중복(객체+정수)
	int operator <(Point ob);                          // < 연산자 중복
};

Point Point::operator +(int digit) {
	Point tmp;
	tmp.xp = xp + digit;                               // this->xp + digit
	tmp.yp = yp + digit;                               // this->yp + digit
	return tmp;
}

int Point::operator <(Point ob) {
	return ((xp < ob.xp) && (yp < ob.yp));
}

void main()
{
	Point p(10, 20);
	int x, y;
	p.get_xy(x, y);
	cout << "[p객체]    x값 : " << x << " , y값 : " << y << endl;

	(p+10).get_xy(x, y);
	cout << "[p객체+10] x값 : " << x << " , y값 : " << y << endl;

	cout << "₩n[ p와 p+10의 비교결과 ]₩n";
	if(p < p+10)
		cout << "p < p+10 ₩n";
	else
		cout << "p >= p+10 ₩n";
	cout << "₩n(p < p+10)의 결과는 " << (p< p+10) << endl;
}
```

실 행 결 과

```
[p객체]        x값 : 10, y값 : 20
[p객체+10]     x값 : 20, y값 : 30

[ p와 p+10의 비교결과 ]
p < p+10

( p< p+10 )의 결과는 1
Perss any key to continue...
```

예제 1-8 : ++연산자와 --연산자의 오버로딩 예제

```
#include<iostream.h>
class Counter {
	int value;
public :
	Counter() { value = 0; }
	Counter(int n)  { value = n; }
	int val() { return value; }
	void operator ++() { ++value; }  // ++ 연산자 중복
	void operator --() { --value; }  // -- 연산자 중복
};

void main() {
	Counter ob1(10), ob2;
	++ob1;  // 연산자 함수 ++ 호출
	--ob2;  // 연산자 함수 -- 호출

	cout << "ob1 : " << ob1.val() << endl;
	cout << "ob2 : " << ob2.val() << endl;
}
```

실 행 결 과

```
ob1 : 11
ob2 : -1
Perss any key to continue...
```

예제 1-9 : '-' 연산자를 이항 연산자 함수와 단항 연산자 함수로 오버로딩한 예제

```
#include<iostream.h>
class Point {
	int xp, yp;
public :
	Point()  { xp = yp = 0; }
	Point(int x, int y) { xp = x, yp = y; }
	void get_xy(int &x, int &y)  { x = xp; y = yp; }
	Point operator -(Point ob);  // - 이항 연산자 중복
	Point   operator -();          // - 단항 연산자 중복
};
Point Point::operator -(Point ob) {  // - 이항 연산자 함수
	Point tmp;
	tmp.xp = xp - ob.xp;  // this->xp - ob.xp
	tmp.yp = yp - ob.yp;  // this->yp - ob.yp
	return tmp;
}
Point Point::operator -() {  // - 단항 연산자 함수
	xp = -xp;
	yp = -yp;
	return *this;
}
```

```
void main() {
	Point p1(10, 20), p2;
	int x, y;
	(p1-p2).get_xy(x, y);
	cout << "[p1-p2 객체]  x값 : " << x << " , y값 : " << y << endl;

	(-p1).get_xy(x, y);
	cout << "[-p2 객체]    x값 : " << x << " , y값 : " << y << endl;
}
```

실 행 결 과

```
[p1-p2 객체]   x값 : 10, y값 : 20
[-p1 객체]     x값 : -10, y값 : -20
Perss any key to continue...
```

예제 1-10 : ++, --, ( ) 연산자를 오버로딩 한 예제

```
#include<iostream.h>

class Counter {
	int value;
public :
	Counter() { value = 0; }
	Counter(int n)  { value = n; }
	void operator ++() { ++value; }  // ++ 연산자 중복
	void operator --() { --value; }  // -- 연산자 중복
	int operator ()() { return value; } // ( ) 연산자 중복
};

void main() {
	Counter ob1(10), ob2;
	++ob1;  // 연산자 함수 ++ 호출
	--ob2;  // 연산자 함수 -- 호출

	cout << "ob1 : " << ob1() << endl;  // 연산자 함수 ()호출
	cout << "ob2 : " << ob2() << endl;  // 연산자 함수 ()호출
}
```

실 행 결 과

```
ob1 : 11
ob2 : -1

Perss any key to continue...
```

예제 1-11 : ( ) 연산자를 이항 연산자로 재정의하여, 이름 테이블의 내용을 검색하여 이름과 순번을 출력하는 프로그램

```
#include<iostream.h>
#include<string.h>
#include<stdlib.h>
#define MAX 100
struct Name {
	char *name;
	int no;
};
class NameTable {
	Name table[MAX];
	int cnt;
public:
	NameTable() { cnt = 0; }
	void add(char *str, int n);
	int search(char *str);
	int operator ()(char *str); // ( ) 연산자 중복
};
void NameTable::add(char *str, int n) {
	if(cnt < MAX-1) {
		table[cnt].name = new char[strlen(str)+1];
		strcpy(table[cnt].name, str);
		table[cnt].no = n;
		cnt++;
	}
	else {
		cout << "₩n Symbol Table Overflow!!₩n";
		exit(1);
	}
}
int NameTable::search(char *str) {
	for(int i=0; i<cnt; i++)
		if(strcmp(str, table[i].name) == 0)
		return table[i].no;
	return -1;
}
int NameTable::operator ()(char *str) {
	cout << str << "₩t";
	return search(str);
}
void dash() { cout << "-----------₩n"; }

void main() {
	NameTable nameT;
	nameT.add("박지성", 10);		nameT.add("김연아", 11);
	nameT.add("이효리", 12);		nameT.add("존코너", 13);
	nameT.add("유재석", 14);
	cout << "이름₩t번호₩n";
	dash();
	cout << nameT("박지성") << endl; // () 연산자 함수 호출
	cout << nameT("이효리") << endl; // () 연산자 함수 호출
	cout << nameT("희동구") << endl; // () 연산자 함수 호출
	return 0;
}
```

실 행 결 과

```
이름            번호
------------------
박지성           10
이효리           12
희동구           -1
Perss any key to continue...
```

예제 1-12 : “[ ]” 연산자를 오버로딩 하여 객체로 배열을 구현한 예제

```
#include <iostream.h>
#include <stdlib.h>
class Array {
        int *p;   // 배열의 시작 주소
        int size; // 배열의 크기
public:
        Array(int num);  // 생성자
        int &operator [](int i);  // [ ] 연산자 중복 선언
};
Array::Array(int num) {
        p = new int[num];
        if(!p) {
                cout << "메모리 할당이 실패했소!!₩n";
                exit(1);
        }
        size = num;
}
int &Array::operator[](int i) {  // [ ] 연산자 중복 정의
        if(i<0 || i>=size) {   // 배열의 범위 체크
                cout << "₩n 배열의 범위를 넘어섰소!!₩n";
                exit(1);
        }
        return p[i];
}

void main() {
        Array n(10);   // 10개의 원소를 가지는 객체배열
        for(int i=0; i<10; i++)
                n[i] = i*i;    // [ ] 연산자함수 호출, 값 대입

        cout << endl;

        for(i=0; i<11; i++)
                cout << n[i] << "   ";  // [ ] 연산자함수 호출
}
```

실 행 결 과

```
0  1  4  9  16  25  36  49  64  81
 배열의 범위를 넘어섰소!!
Perss any key to continue...
```

예제 1-13 : 연산자 오버로딩을 이용한 string class의 구현 - 1 (문자열과 문자열을 연결하는 프로그램)

```
#include <iostream.h>
#include <string.h>

class String{
       char *str;
       int len;
public:
              String();   // 생성자
              String(const char * const Str);  // 생성자
              String(int n);   // 생성자
              String(const String &);     // 생성자
              ~String();     // 소멸자
              void Print();  // 문자열을 화면에 출력함
              int GetLen(){return len;}  // 문자열 크기를 리턴함
              char *GetString(){return str;}  // 문자열을 리턴함
              String operator = (String data);  // 대입 연산자 오버로딩
              String operator+(const String &second);  // 덧셈 연산자 오버로딩
              friend ostream & operator <<(ostream &os, String &temp); // << 연산자 오버로딩

};

String::String() {
              str = NULL;
              len = 0;
}

String::~String() {
       if(str)
              delete str;
}

String::String(int n) {
       len = n;
       str = new char[len + 1];
       memset(str, '₩0', len);
}

String::String(const char * const Str) {
       len = strlen(Str);
       str = new char[len + 1];
       strcpy(str, Str);
}

String::String(const String &temp) {
   len = temp.len;
   str = new char[len + 1];
   strcpy(str, temp.str);
}

void String::Print() {
       cout << str << endl;
}
```

```
String String::operator = (String data) {
	len = data.GetLen();  // data객체  문자열의 크기를 알고
	str = new char[len + 1];  // 그 크기만큼 메모리를 할당한 후
	strcpy(str, data.GetString());  // data객체의 문자열 복사
	return *this;  // 복사된 객체 반환
}

String String::operator+(const String &second)  { // + 연산자 오버로딩
	int tot_len = len + second.len;		//두 문자열의 길이를 구함
	//두 문자열을 모두 저장할 만한 크기의 string객체 생성
	String temp(tot_len);
	strcpy(temp.str, str);  //왼쪽 객체의 문자열을 임시 객체에 복사
	strcat(temp.str, second.str);   //오른쪽 객체의 문자열을 임시 객체에 추가
	return temp;    //임시 객체 값 반환
}

void main()
{
	cout << "두 문자열을 +로 연결하기₩n";
	String str1("C++ ");
	String str2("Programming");
	String str3;
	str3 = str1 + str2;
	cout << "문자열1 :";    str1.Print();
	cout << "문자열2 :";    str2.Print();
	cout << "----------------------₩n";
	cout << "[문자열1 + 문자열2] => "; str3.Print();
}
```

실 행 결 과

```
두 문자열을 +로 연결하기
문자열1 :C++
문자열2 :Programming
----------------------
[문자열1 + 문자열2] => C++ Programming
Press any key to continue
```

예제 1-14 : 연산자 오버로딩을 이용한 string class의 구현 - 2 (>, < ,== 연산자로 문자열을 비교하는 프로그램)

```
#include <iostream.h>
#include <string.h>

class String{
	char *str;
public:
		String(const char * const Str);  // 생성자
		~String();     // 소멸자
		void Print();  // 문자열을 화면에 출력
		String operator = (String data);  // 대입 연산자 오버로딩
		bool operator<(const String &second);  // < 연산자 오버로딩
		bool operator>(const String &second);  // > 연산자 오버로딩
		bool operator==(const String &second);  // == 연산자 오버로딩
};
```

```
String::~String() {
	if(str)
		delete str;
}
String::String(const char * const Str) {
	str = new char[strlen(Str) + 1];
	strcpy(str, Str);
}
void String::Print() {
	cout << str << "Wt";
}
bool String::operator<(const String &second)  { // < 연산자 오버로딩
	if(strcmp(str, second.str) < 0)
		return 1;
	else
		return 0;
}
bool String::operator>(const String &second)  { // > 연산자 오버로딩
	if(strcmp(str, second.str) > 0)
		return 1;
	else
		return 0;
}
bool String::operator==(const String &second)  { // == 연산자 오버로딩
	if(strcmp(str, second.str) == 0)
		return 1;
	else
		return 0;
}

void main()
{
	cout << "<< 두 문자열의 비교 >>Wn";
	String str1("Orange");  str1.Print();
	String str2("Apple");          str2.Print();
	if(str1 < str2)
		cout << "=> 두 번째 문자열이 크다." << endl;
	else
		cout << "=> 첫 번째 문자열이 크다." << endl;

	String str3("Blue");    str3.Print();
	String str4("Yellow");         str4.Print();
	if(str3 > str4)
		cout << "=> 첫 번째 문자열이 크다." << endl;
	else
		cout << "=> 두 번째 문자열이 크다." << endl;

	String str5("Watch");          str5.Print();
	String str6("Clock");          str6.Print();

	if(str5 == str6)
		cout << "=> 두 문자열은 같다." << endl;
	else
		cout << "=> 두 문자열은 다르다." << endl;
}
```

실 행 결 과

```
<< 두 문자열의 비교 >>
Orange  Apple   => 첫 번째 문자열이 크다.
Blue    Yellow  => 두 번째 문자열이 크다.
Watch   Clock   => 두 문자열은 다르다.
Press any key to continue
```

예제 1-15 : 연산자 오버로딩을 이용한 string class의 구현 - 3

```
#include <iostream.h>
#include <string.h>

class String{
       char *str;
       int len;
public:
              String();   // 생성자
              String(const char * const Str);  // 생성자
              String(int n);   // 생성자
              String(const String &);     // 생성자
              ~String();     // 소멸자
              void Print();  // 문자열을 화면에 출력함
              void GetInput();  // 문자열을 콘솔에서 입력받음
              int GetLen(){return len;}  // 문자열 크기를 리턴함
              char *GetString(){return str;}  // 문자열을 리턴함
              String operator = (String data);  // 대입 연산자 오버로딩
              String operator += (String data);  // 증가 연산자 오버로딩
              String operator+(const String &second);  // 덧셈 연산자 오버로딩
              operator const char* () {       return str; };  // const char* 연산자 오버로딩
              friend ostream & operator <<(ostream &os, String &temp); // << 연산자 오버로딩
              char operator [](int n);  // 첨자 위치의 문자 정보 알려줌
};

String::String() {
              str = NULL;
              len = 0;
}

String::~String() {
       if(str)
              delete str;
}

String::String(int n) {
       len = n;
       str = new char[len + 1];
       memset(str, '₩0', len);
}

String::String(const char * const Str) {
       len = strlen(Str);
       str = new char[len + 1];
       strcpy(str, Str);
}

String::String(const String &temp) {
   len = temp.len;
   str = new char[len + 1];
   strcpy(str, temp.str);
}
```

```
void String::Print() {
	cout << str << endl;
}
void String::GetInput() {
	char temp[80];
	cin >> temp;
	len = strlen(temp);
	str = new char[len + 1];
	strcpy(str, temp);
}

String String::operator = (String data) {
	len = data.GetLen();  // data객체  문자열의 크기를 알고
	str = new char[len + 1];  // 그 크기만큼 메모리를 할당한 후
	strcpy(str, data.GetString());  // data객체의 문자열 복사
	return *this;  // 복사된 객체 반환
}

String String::operator += (String data) {
	char temp[512];
	strcpy(temp, str);  // 기존의 데이터를 temp에 복사하고
	len += data.GetLen();  // 현재 크기에 data객체 문자열의 크기만큼 추가
	str = new char[len + 1];  // 새로 str을 메모리에 할당
	strcpy(str, temp);  // temp를 복사하고
	strcat(str, data.GetString());  // data객체의 문자열 복사
	return *this;  // 연결된 문자열 반환
}

String String::operator+(const String &second)  { // + 연산자 오버로딩
	int tot_len = len + second.len;		//두 문자열의 길이를 구함
	//두 문자열을 모두 저장할 만한 크기의 string객체 생성
	String temp(tot_len);
	strcpy(temp.str, str);  //왼쪽 객체의 문자열을 임시 객체에 복사
	strcat(temp.str, second.str);   //오른쪽 객체의 문자열을 임시 객체에 추가
	return temp;    //임시 객체 값 반환
}

ostream & operator << (ostream &os, String &temp) {
	os << temp.str;
	return os;
}

char String::operator [](int n) {  // 첨자 위치의 문자 정보
	//지정된 첨자가 문자열의 길이보다 크면
	if( n > len )
		return str[len - 1];
	else
		return str[n];
}

void main()
{
	String *data1 = new String;
	String *data2 = new String;
```

```
        cout << "두 문자열을 +=로 연결하기₩n";
        cout << "문자열1 :";
        data1->GetInput();  // data 문자열 입력
        cout << "문자열2 :";
        data2->GetInput();  // data2 문자열 입력
        cout << "------------------------" << endl;
        *data1 += *data2;   // *data.operator(String *data2) 형태로 호출
        cout << "[문자열1 += 문자열2] => ";
        data1->Print();    // 연결된 문자열 객체 data 출력
        cout << "₩nchar* 로 출력 : ";
        cout << data1->operator const char*() << endl;  // char* 로 출력

        cout << "₩n========================" << endl;
        cout << "두 문자열을 +로 연결하기₩n";
        String str1("C++ ");
        String str2("Programming");
        String str3;
        str3 = str1 + str2;
        cout << "문자열1 :";    str1.Print();
        cout << "문자열2 :";    str2.Print();
        cout << "------------------------₩n";
        cout << "[문자열1 + 문자열2] => "; str3.Print();
        cout << "₩nstr3[7] = " << str3[7] << endl;
}
```

실 행 결 과

```
두 문자열을 +=로 연결하기
문자열1 :코리아팀
문자열2 :화이팅
------------------------
[문자열1 += 문자열2] => 코리아팀화이팅

char* 로 출력 : 코리아팀화이팅

========================
두 문자열을 +로 연결하기
문자열1 :C++
문자열2 :Programming
------------------------
[문자열1 + 문자열2] => C++ Programming

str3[7] = g
Press any key to continue
```