**연산자 오버로딩**

c++에서는 사용자 정의 연산자를 사용가능

(리턴타입) operator(연산자) (연산자가 받는 인자)

bool MyString::operator==(MyString& str){

return !compare(str);

}

str1 == str2

str1.operator==(str2)로 변환됨.

**사칙연산시 리턴타입이 중요**

class Num이 있을 때

Num& operator+ (const Num& n){

num+=n.num;

return \*this;

}

이면 레퍼런스 반환으로 속도는 더 빨라지지만

b+a+b를 했을때

(b.+(a)).+(b)가 일어나므로

b에 a가 더해지게 돼어 +(b) 시에 사실상 a가 한번 더 더해진다

이런 결과를 막기위해 **사칙연산시 리턴타입은 값을 반환하는 것이 좋다.**

**대입연산시 에는 레퍼런스를 반환하는 것이좋다.**

불필요한 복사를 방지한다.

**friend 키워드**

friend로 정의된 클래스나 함수들은 원래 클래스의 private로 정의된 변수나 함수들에 접근이 가능.

그러나 friend로 정의한 클래스의 private에 접근이 가능해지진 않는다(일방적 관계)

**이항 연산자**

num1 + num2 는 2가지로 구현가능하다

num1.operator+(b); //멤버함수

operator+(a, b); //외부함수

2가지중 하나를 골라야하며

자기자신을 리턴하지않는 사칙연산 연산자는 외부함수,

자기자신을 리턴하는 += 같은 연산자는 멤버함수로 선언하는게 원칙

>>, <<연산자

ostream 클래스를 수정할 수는 없으므로 외부함수로 추가해야한다.

Complex operator<<(std::ostream& os, const Complex& c){

os << "( " << c.real << " , " <<c.img << " ) ";

return os;

}

**첨자 연산자.**

[]안에 넣는수를 첨자라고 부름 (subscript)

char& MyString::operator[](const int index) { return string\_content[index]; }

**int Wrapper 클래스 - 타입 변환 연산자**

Wrapper 기본자료형을 객체로 다룰 때 사용.

class Int

{

int data;

// some other data

public:

Int(int data) : data(data) {}

Int(const Int& i) : data(i.data) {}

};

//일일이 int의 operator와 같은 동작을 하는 함수를 오버로딩하는 건 너무 비효율적

**타입 변환 연산자**

operator (변환타입)() 으로 정의

ex) operator int() {return data}

이러면 operator 연산시 int로 변환해 처리.

**전위/후위 증감연산자**

a++와 ++a 구분.

**operator++() //전위**

**operator++(int) //후위**

int는 아무의미없이 구분을 하기위해 존재함. 인수의 이름조차 정할필요 없음.

단 전위 연산의 경우 값이 바뀐 자신을 리턴하고

후위연산흔 값이 바뀌기 전의 객체를 리턴해야함.

A& operator++() { //전위

// A ++ 을 수행한다.

return \*this;

}

A operator++(int) { //후위 즉 복사생성자 호출로 인해 더 느림

A temp(A);

// A++ 을 수행한다.

return temp;

**C++ 스타일의 캐스팅 (static\_cast 등등)**

4가지 캐스팅 존재

**static\_cast** : 우리가 흔히 생각하는,

언어적 차원에서 지원하는 일반적인 타입 변환

**const\_cast** : 객체의

상수성(const) 를 없애는 타입 변환.쉽게 말해 const int 가 int 로 바뀐다.

**dynamic\_cast** : 파생 클래스 사이에서의 다운 캐스팅

**reinterpret\_cast** : 위험을 감수하고 하는 캐스팅으로 서로 관련이 없는 포인터들 사이의 캐스팅 등

디폴트 인자 (default argument)

// N 차원 배열의 제작

/\*

a[2][3][4] 같은 것이 있을 떄

a.[](2) -> x

x.[](3) -> y

y.[](4) -> (int)가 나와야한다.x와 y는 int 일 수 없다->wrapper클래스 Int를 적용한다.