객체지향 프로그래밍과 자료구조

Ch 4. 연산자 오버로딩 (Operator Overloading)

정보통신공학과

교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- ◆ 함수 오버로딩
- ◆기본 연산자의 오버로딩
  - Unary operators: **++, --**
  - Binary operators: **+, -, *, /, %, ==, !=**
  - 연산자오버로딩 멤버함수 구현
- ◆ **Friends 함수, Friend 클래스, 자동 형변환**
  - Friend functions, friend classes
  - Constructors for automatic type conversion
- ◆ 기타 연산자 오버로딩
  - **<<** and **>>**
  - **= , [], ++, --**

함수오버로딩 (Function Overloading)
연산자 오버로딩 (Operator Overloading)

함수 오버로딩 (1)

◆ 함수 오버로딩 (function overloading)

- 동일한 함수 이름을 사용하여도 전달되는 매개 변수의 자료형, 순서가 다르면 서로 다른 함수로 구분하여 사용될 수 있게 함
- 예)

```
double avg(double d1, double d2);
double avg(double d1, double d2, double d3);
double avg(int i1, int i2);
double avg(int i1, int i2, int i3);
```

◆ 함수 서명 (function signature)

- 함수 이름 + 매개 변수 (parameter)의 자료형과 순서 정보로서 함수 서명 구성
- 함수의 반환형 (return type), 매개 변수 (parameter)에 const 선언 등은 함수 서명 정보에 포함되지 않음

함수 오버로딩 (2)

◆ 컴파일러의 함수 오버로딩 처리 기준

- 컴파일러가 동일한 함수 이름을 가지는 여러 함수들 중에서 가장 적합한 함수를 선택하여 함수 호출에 연결
- 함수 오버로딩을 처리하는 기준
  - 정확하게 일치하는 (함수 이름과 매개 변수의 자료형 및 순서가 동일)한 함수 서명의 함수를 선정
  - 정확하게 일치하는 함수 서명을 가진 함수가 없는 경우, 매개 변수의 자료형을 승격 (promotion)시켜 일치하는 함수가 있으면 이를 선택 (예: int -> double, short -> double, float ->double, char * -> string)

◆ 함수 오버로딩의 활용

- 템플릿 함수 (template function): 동일한 구조의 함수 (알고리즘)에 전달되는 인수의 자료형만 다를 경우에 하나의 템플릿 함수를 구현하고, 인수의 자료형에 따라 컴파일러가 자동적으로 필요한 함수를 생성하여 사용할 수 있게 함
- 수학 공식에서 익숙한 연산자 (operator)를 사용자가 구현한 클래스에 사용할 수 있게 하는 연산자 오버로딩 (operator overloading)에 활용

연산자 오버로딩 개요

◆ 연산자

- 기본 연산자: +, -, *, /, %, ==, != etc.
- 연산자를 함수로 생각할 수 있음

◆ 연산자를 사용한 연산

- 예) x + 7
- "+" 는 x와 7 두 개의 연산수 (operand)를 가지는 이진 연산자
- 수학 공식에서 사용하며, 세계 공통어로 모든 사람이 이해할 수 있음

◆ 연산자를 함수와 같이 생각하면: +(x, 7)

- "+" 는 함수 이름
- x, 7 은 함수에 전달되는 인수 (arguments)
- 함수 "+"는 전달된 인수들에 대한 덧셈 계산을 한 후 그 결과를 반환

연산자 오버로딩

◆ 기본 제공 (built-in) 연산자

- 예) **+, -, *, /, = , %, ==, !=**
- C++ 프로그래밍 언어에서 기본적으로 제공되는 자료형에 사용
- 하나의 인수를 사용하는 일진 연산자 (unary operator) 또는 두 개의 인수를 사용하는 이진 연산자 (binary operator)

◆기본 제공 연산자를 일반 클래스에 사용

- C++ 프로그램에서 구현하는 클래스에 기본 연산자를 사용하는 것이 가능함
- 예) 복소수를 위한 class Cmplx 객체를 +, -, *, / 연산자를 사용하여 계산을 하도록 할 수 있음
  - 수학 공식에서 익숙한 연산자를 사용하므로 쉽게 이해할 수 있음
  - 사용자 편의성이 증가된 (user friendly) 인터페이스 제공

◆ 연산의 의미를 직관적으로 이해할 수 있는 연산자를 사용하여야 함

연산자 오버로딩이 필요한 이유

◆ 간략한 표현과 쉬운 이해

- 수학 공식에서 사용하는 연산자를 사용함으로써 간단하고 이해하기 쉽게 프로그램 소스코드를 구현

```
/* What you have used */

double mA[N][N], mB[N][N], mC[N][N],
    mD[N][N], mE[N][N], mF[N][N];

getMtrx(mA, N);
getMtrx(mB, N);

mtrxAdd(mA, mB, mC, N);
mtrxSubtract(mA, mB, mD, N);
mtrxMultiply(mA, mB, mE, N);
mtrxInverse(mA, mF, N);
```

```
/* What can be possible with */
// Class definition for Mtrx
 Mtrx mA, mB, mC, mD, mE, mInv;

 cin >> mA;
 cin >> mB;

 mC = mA + mB;
 mD = mA - mB;
 mE = mA * mB;
 mInv = ~mA;
```

연산자 오버로딩이 가능한 연산자
연산자 오버로딩이 가능하지 않은 연산자

◆ 연산자 오버로딩이 가능한 연산자

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

◆ 연산자 오버로딩이 가능하지 않은 연산자

.	.*	::	?:

연산자의 오버로딩 구현 (1)

◆ 연산자 오버로딩 구현

- 함수 오버로딩과 유사
- 연산자 앞에 키워드 operator를 붙여 함수 이름 구성

◆ 연산자 오버로딩을 정의하는 함수 원형 예:

```
const Cmplx operator+(const Cmplx & c1, const Cmplx & c2);
```

- **+** 연산자를 class **Cmplx** 에 사용할 수 있도록 연산자 오버로딩 정의
- 전달되는 파라메터에 **const**를 선언하여 전달된 인수를 읽기만 하고, 변경할 수 없도록 정의
- 반환되는 자료형으로 class Cmplx 선언
  - class Cmplx 객체 2개에 덧셈을 계산하여 결과인 class Cmplx 객체 반환

연산자의 오버로딩 구현 (2)

◆ 연산자 오버로딩 구현의 2가지 방법

- case 1: 클래스의 멤버함수가 아닌 일반 함수 오버로딩으로 구현
  - 일반 함수 오버로딩과 같이 전달되는 인수의 자료형과 순서에 따라 구분
  - 개별적인 함수로 관리됨
  - 클래스 선언에서 일반 함수로 구현된 오버로딩을 포함하지 않음

- case 2: **클래스의 멤버 함수로 구현**
  - 연산자 오버로딩이 클래스 선언에 포함됨
  - 객체 지향형 프로그래밍 원칙에 잘 부합함

Class Cmplx를 위한 "+" 연산자를 일반 함수 오버로딩으로 구현하는 경우

◆ class Cmplx 를 위한 "+" 연산자 오버로딩을 일반 함수 오버로딩으로 구현하는 경우

```
const Cmplx operator+(const Cmplx &c1, const Cmplx &c2)
{
    double real, imag;

    real = c1.real + c2.real;
    imag = c1.imag + c2.imag;

    return Cmplx(real, imag);
}
```

연산자 오버로딩을 클래스의 멤버함수로의 구현

◆**Cmplx c1(3, 4), c2(1, 2), c3;**
c3 = c1 + c2;

- "**+**" is overloaded as member operator
- think of as: **c3 = c1+(c2);**

◆ **"+" 연산자 오버로딩을 헤더 파일의 클래스 선언에 포함**

- **const Cmplx operator+(const Cmplx& c)**;
- Notice only ONE argument

class Cmplx의 함수 오버로딩을 멤버 함수로 구현

```
/**  Cmplx.h  */
#include <iostream>

using namespace std;

class Cmplx
{
public:
    Cmplx(double real=0.0, double imagin=0.0); // constructor
    const Cmplx operator+(const Cmplx &);
    const Cmplx operator-(const Cmplx &);
    void printCmplx();
private:
    double real;
    double imag;
};
```

```
/**  Cmplx.cpp */

#include <iostream>
#include "Cmplx.h"

using namespace std;

Cmplx::Cmplx(double r, double i) :real(r), imag(i) { } // constructor

const Cmplx Cmplx::operator+(const Cmplx &c)
{
    Cmplx result;

    result.real = real + c.real;
    result.imag = imag + c.imag;

    return result;
}

const Cmplx Cmplx::operator-(const Cmplx &c)
{
    Cmplx result;

    result.real = real - c.real;
    result.imag = imag - c.imag;

    return result;
}

void Cmplx::printCmplx()
{
    printf("Complex(%.2lf, %.2lf)", real, imag);
}
```

```
/**  main.cpp  */

#include <iostream>
#include "Cmplx.h"

using namespace std;

void main()
{
    Cmplx c1 (3.3, 4.4), c2(1.1, 2.2), c3, c4;

    cout << " c1 = ";
    c1.printCmplx();     cout << endl;
    cout << " c2 = ";
    c2.printCmplx();     cout << endl;

    c3 = c1 + c2;
    cout << " c3 = c1 + c2 = ";
    c3.printCmplx();     cout << endl;

    c4 = c1 - c2;
    cout << " c4 = c1 - c2 = ";
    c4.printCmplx();     cout << endl;

}
```

<Result of Execution>

```
c1 = Complex(3.30, 4.40)
c2 = Complex(1.10, 2.20)
c3 = c1 + c2 = Complex(4.40, 6.60)
c4 = c1 - c2 = Complex(2.20, 2.20)
```

const 멤버함수,
() 연산자 오버로딩

const 멤버함수

◆왜 const 멤버 함수가 필요한 가?

- const 함수는 클래스의 데이터 멤버를 변경할 수 없게 하며, 불필요한 변경을 방지하여 데이터 멤버를 보호
- const 객체는 ***const*** 멤버함수만 호출할 수 있음

◆객체 지향형 프로그래밍의 좋은 프로그래밍 원칙

- 데이터 멤버를 변경하지 않아도 되는 멤버 함수는 모두 const 멤버 함수로 선언

◆키워드 ***const*** 를 멤버함수 선언 뒤에 표시

const 멤버 함수의 예

```
class TelNum
{
   friend ostream& operator<<(ostream& fout, const TelNum& TelNum);
public:
   TelNum(int nc, int rn, int sn, int ln);
   TelNum() {}
   ~TelNum();
   void setNation_code(int nc) { nation_code = nc; }
   void setRegion_no(int rn) { region_no = rn; }
   void setSwitch_no(int sn) { switch_no = sn; }
   void setLine_no(int ln) { line_no = ln; }
   int getNation_code() const { return nation_code; }
   int getRegion_no() const { return region_no; }
   int getSwitch_no() const { return switch_no; }
   int getLine_no() const { return line_no; }
   void print(ostream& fout);
   TelNum& operator=(const TelNum& right);
   bool operator>(const TelNum& right);
   bool operator<(const TelNum& right);
   bool operator==(const TelNum& right);
   bool operator!=(const TelNum& right);
private:
   int nation_code;
   int region_no;
   int switch_no;
   int line_no;
};
```

() 연산자 오버로딩

◆ 함수호출 연산자 ( )

- Must be overloaded as member function
- Allows use of class object like a function
- Can overload for all possible numbers of arguments

◆ 예:

- class BigRand
  - **BigRand big_rand;**
  - **big_rand()** that returns non-duplicated big random number with offset

() 연산자 오버로딩의 예 – BigRand

```
/* class BigRand */
#ifndef BIGRAND
#define BIGRAND
#include <iostream>
using namespace std;

class BigRand
{
public:
    BigRand(int num_bigRand, int offset);
    int operator()(); // returns big random number
private:
    bool *flag;
    int num_big_rand;
    int offset;
    int cur_index; // current index
};
#endif
```

```
/* BigRand.cpp */
#include "BigRand.h"
#include <time.h>

BigRand::BigRand (int num_big_rand, int offset)
{
    this->num_big_rand = num_big_rand;
    this->offset = offset;

    flag = (bool *)new bool[num_big_rand];
    for (int i = 0; i < num_big_rand; i++)
    flag[i] = false;
    srand(time(0));
}

int BigRand::operator()()
{
    int rand_h, rand_l, big_rand, big_rand_with_offset;
    do
    {
        rand_h = rand();
        rand_l = rand();
        big_rand = rand_h << 15 | rand_l;
        big_rand = big_rand % num_big_rand;
    } while (flag[big_rand] == true);
    flag[big_rand] = true;
    big_rand_with_offset = big_rand + offset;

    return big_rand_with_offset;
}
```

```
/* main() */

#include <iostream>
#include <iomanip>
#include "BigRand.h"

#define NUM_BIG_RAND 100
#define BIG_RAND_OFFSET 0
void main()
{
    BigRand big_rand(NUM_BIG_RAND, BIG_RAND_OFFSET);

    for (int i = 1; i <= NUM_BIG_RAND; i++)
    {
        cout << setw(4) << big_rand();
        if (i % 20 == 0)
            cout << endl;
    }
}
```

```
  29  94  24  51  81  91  57  77  65  92  42  26  71  14   7  98  55  99  69  96
  64  49  66  22  58  38  13  63  33  10  19   4  27  50  73  17   9  67  56  40
  39  79  82  90  48  41  76  23  62  70  59  35  46  87  44  89  52  11  53  84
   0  78  37  34  75  61  45   6  97  31  12  16  28  20  47  85  30  21   2  86
  15  80  88  32  74  36  83   3  54   5  18  95  60  43   8  93  72   1  25  68
계속하려면 아무 키나 누르십시오 . . .
```

Friend 함수 (Friend Function)
멤버함수의 호출과 반환에서의 참조
(call-by-reference, return-by-reference)
Friend 클래스 (Friend Class)

Friend 함수 (1)

◆클래스의 멤버함수가 아닌 일반 함수를 friend 함수로 선언

- 클래스의 연산자 오버로딩을 멤버함수가 아닌 일반 함수 오버로딩으로 구현하는 경우
  - 데이터 멤버를 읽거나 변경할 때 항상 접근자 (accessor)와 변경자/설정자 (mutator) 멤버함수를 사용하여야 하므로 구현이 번거로움
  - 데이터 멤버의 이름으로 직접 읽거나 쓸 수 없고 멤버 함수를 거쳐야 하므로 함수 호출에 따른 추가 부담이 항상 발생

◆Friend 함수에게는 private 영역의 데이터 멤버를 이름으로 직접 접근하고 변경할 수 있게 허용함

- 접근자/설정자 멤버 함수의 호출에 따른 추가 부담이 발생하지 않아 성능 관점에서 효율적임
- 멤버함수가 아닌 일반 함수에게 private 영역의 데이터 멤버 직접 접근 및 변경이 허용되므로 friend 함수 선언은 매우 신중하여야 함

Friend 함수 (2)

◆ 클래스의 friend 함수 선언 및 구현 (1)

- 클래스 선언에서 friend로 선언할 함수의 함수 원형을 포함시키고 함수 원형 앞에 키워드 *friend* 를 표시
- 클래스 선언 내부에 포함되어 있지만 그 클래스의 멤버 함수는 아님

```
/** Cmplx.h */
#include <iostream>

using namespace std;

class Cmplx
{
   friend ostream & operator<< (ostream &, const Cmplx &);
   friend istream & operator>> (istream &, Cmplx &);
public:
   Cmplx(double real=0.0, double imag=0.0); // constructor
   const Cmplx operator+(const Cmplx &);
   const Cmplx operator-(const Cmplx &);
private:
   double real;
   double imag;
};
```

◆ 클래스의 friend 함수 선언 및 구현 (2)

```
/* Cmplx.cpp */
ostream& operator<<(ostream& fout, const Cmplx &c)
{
   fout << "Complex (" << c.real << ", " << c.imag << ")" << endl;
   return fout;
}

istream& operator>>(istream& fin, Cmplx &c)
{
   fin >> c.real >> c.imag; // chained operation
   return fin;
}
```

```
/**  main.cpp */
. . . .
   Cmplx c1, c2, c3, c4;

   cout << " input c1.real and c1.imag: ";
   cin >> c1;
   cout << " c1 : " << c1 << endl; // chained operation

   cout << " input c2.real and c2.imag: ";
   cin >> c2;
   cout << " c2 : " << c2 << endl; // chained operation
. . . .
```

멤버 함수의 호출과 반환에서의 참조 (reference)

- **Syntax:**
  double& functionA(double& variable);
  - double& and double are different
  - Must match in function declaration and heading
- **참조가 반환된 경우 그 참조의 값이 저장된 메모리 공간이 존재하여야 함**
  - 참조로 반환값을 전달한 함수에서 그 반환값이 저장되어 있는 메모리 공간을 유지하여야 함
  - 반환값으로 계산식 (예: "x+5")을 반환할 수 없으며, 그 계산식의 결과값이 저장된 메모리 공간의 주소 정보를 전달하여야 함

전달받은 참조 인수를 반환 값으로 전달

◆ Chained operation을 사용할 수 있도록 구현

```
Cmplx c1, c2, 3;
cin >> c1 >> c2 >> c3;
cout << c1 << c2 << c3;
```

◆ Example function definition:

```
/* Cmplx.cpp */
ostream& operator<<(ostream& fout, const Cmplx &c)
{
    fout << "Complex (" << c.real << ", " << c.imag << ")" << endl;
    return fout;
}

istream& operator>>(istream& fin, Cmplx &c)
{
    fin >> c.real >> c.imag;  // chained operation
    return fin;
}
```

Friend 함수의 순수성 (purity)

◆Friends 함수와 객체 지향형 프로그래밍 원칙

- 객체 지향형 프로그래밍 원칙에서는 모든 연산자와 함수들을 클래스의 멤버 함수로 선언하고 구현하는 것을 원칙으로 함
- friend 함수 선언은 클래스의 멤버함수가 아닌 일반 함수에게 클래스 멤버함수와 동일한 권한을 부여
- friend 함수 선언은 객체 지향형 프로그래밍 원칙을 위배하는 것으로 간주할 수 있음

◆friend 함수 선언이 가지는 의미

- 클래스 선언 내부에서 friend 함수를 선언하므로 캡슐화 개념에는 부합
- 데이터 멤버의 접근과 변경에서 멤버 함수를 거치지 않아도 되므로 추가적인 함수 호출 및 반환 부담이 없어 성능 향상
- friend로 선언하는 함수는 신뢰성이 있는 함수로 제한하여야 함

Friend 클래스

◆friend 클래스로 선언되면 그 클래스의 모든 멤버 함수가 friend 함수로 선언 됨

- 예)

  class F is friend of **class C**

  - All member functions of class F are friends of C
  - 반대의 경우는 성립하지 않음 (class F가 class C의 friend 클래스로 선언되는 것이 class C가 class F의 friend가 되는 것을 의미하지는 않음)
  - Friendship은 다른 클래스가 부여하는 것이며, 스스로 받아 올 수 없음

◆Syntax: *friend class F*

- Goes inside class definition of "authorizing" class

대입 연산자 (=) 오버로딩
배열 클래스
인덱싱 연산자 ([]) 오버로딩

대입 연산자 (=) 오버로딩

◆ 기본 제공 대입 연산자 오버로딩

- Default assignment operator:
  - Member-wise copy
  - Member variables from one object → corresponding member variables from other

◆ 단순 클래스에서는 별도의 대입 연산자 구현이 필요 없음

◆ 클래스 객체 생성에서 포인터와 동적 메모리 할당이 포함되어 있는 경우 별도의 대입 연산자 구현이 필요함

◆ Example of "operator=" overloading (1)

```
/** Cmplx.h */

#ifndef CMPLX_H
#define CMPLX_H
#include <iostream>

using namespace std;

class Cmplx
{
    friend istream & operator>> (istream &, Cmplx &);
    friend ostream & operator<< (ostream &, const Cmplx &);
public:
    Cmplx(double real=0.0, double imagin=0.0); // constructor
    const Cmplx operator+(const Cmplx &);
    const Cmplx operator-(const Cmplx &);
    const Cmplx operator*(const Cmplx &);
    const Cmplx operator/(const Cmplx &);
    const Cmplx operator=(const Cmplx&);
private:
    double real;
    double imag;
};

#endif
```

◆ Example of "operator=" overloading (2)

```
const Cmplx Cmplx::operator=(const Cmplx& c)
{
    real = c.real;
    imag = c.imag;

    return *this;
}
```

```
/**  main.cpp  */

#include <iostream>
#include "Cmplx.h"

using namespace std;

void main()
{
    Cmplx c1, c2, c3, c4, c5, c6;

    cin >> c1 >> c2;

    cout << endl;
    cout << " c1 = " << c1 << endl;
    cout << " c2 = " << c2 << endl;

    c3 = c1 + c2;
    cout << " c3 = c1 + c2 = " << c3 << endl;

    c4 = c1 - c2;
    cout << " c4 = c1 - c2 = " << c4 << endl;

    c5 = c1 * c2;
    cout << " c5 = c1 * c2 = " << c5 << endl;

    c6 = c1 / c2;
    cout << " c6 = c1 / c2 = " << c6 << endl;
}
```

```
Input real and imag in double type: 3.3 4.4
Input real and imag in double type: 1.1 2.2

 c1 = Complex( 3.30,  4.40)
 c2 = Complex( 1.10,  2.20)
 c3 = c1 + c2 = Complex( 4.40,  6.60)
 c4 = c1 - c2 = Complex( 2.20,  2.20)
 c5 = c1 * c2 = Complex(-6.05, 12.10)
 c6 = c1 / c2 = Complex( 2.20, -0.40)
```

배열 클래스와 인덱싱 연산자 [ ]

◆ 배열 클래스

- 클래스로 구현한 객체들을 배열로 구성하기 위하여 배열 클래스를 구현
  - 예) class Cmplx -> class CmplxArray
    class Date -> class DateArray
- 배열의 각 원소를 인덱스를 사용하여 접근 및 변경
  - 예) class CmplxArray cmplexes(6);
    cin >> cmplexes[0] >> cmplexes[1];
    cmplexes[2] = complexes[0] + complexes[1];

◆ 배열 클래스 객체를 인덱스로 접급/변경하기 위해서는 연산자 [ ] 오버로딩이 구현되어야 함

- To be used with objects of your class
- Operator must return a reference!
- **operator[]** must be a member function!
- 배열을 구성하는 원소들의 클래스에 배열 클래스가 friend로 선언되어 있어야 원소들의 데이터 멤버를 이름으로 직접 접근/변경할 수 있음

◆ Example : CmplxArray (1)

```
/**  Cmplx.h  */

#ifndef CMPLX_H
#define CMPLX_H
#include <iostream>

using namespace std;
class CmplxArray;
class Cmplx
{
    friend ostream & operator<< (ostream &, const Cmplx &);
    friend istream & operator>> (istream &, Cmplx &);
    friend class CmplxArray;
public:
    Cmplx(double real=0.0, double imagin=0.0); // constructor
    const Cmplx operator+(const Cmplx &);
    const Cmplx operator-(const Cmplx &);
    const Cmplx operator*(const Cmplx &);
    const Cmplx operator/(const Cmplx &);
private:
    double real;
    double imag;
};

#endif
```

◆ Example : CmplxArray (2)

```
/**  CmplxArray.h  */
#ifndef CMPLXARRAY_H
#define CMPLXARRAY_H

#include <iostream>
#include "Cmplx.h"

using namespace std;

class CmplxArray
{
public:
    CmplxArray(int size); // constructor
    CmplxArray(const CmplxArray &obj); // copy constructor
    ~CmplxArray();
    int size() {  return cmplxArrySize; }
    Cmplx &operator[](int  sub);
private:
    Cmplx *pCA;
    int cmplxArrySize;
    void subError(); // out of subscript range
};

#endif
```

◆ Example : CmplxArray (5)

```
/**  CmplxArray.cpp  */

#include "CmplxArray.h"
#include "Cmplx.h"

CmplxArray::CmplxArray(int size) // constructor
{
    cmplxArrySize = size;
    this->pCA = new Cmplx[size];
    for (int i=0; i<size; i++) {
    this-> pCA[i].real = 0.0;
    this-> pCA[i].imag = 0.0;
    }
}

CmplxArray::CmplxArray(const CmplxArray &obj) // constructor
{
     cmplxArrySize = obj.cmplxArrySize;
     this-> pCA = new Cmplx[cmplxArrySize];
     for (int i=0; i<cmplxArrySize; i++) {
    this-> pCA[i] = obj.pCA[i];   // *(pCA+i) = obj.pCA[i];
     }
}

CmplxArray::~CmplxArray()  // destructor
{
     if (cmplxArrySize > 0)
         delete [] pCA;
}
```

◆ Example : CmplxArray (6)

```
void CmplxArray::subError()
{
    cout << "ERROR: Subscript out of range.₩n";
    exit(0);
}

Cmplx &CmplxArray::operator [](int sub)
{
    if (sub < 0 || sub >= cmplxArrySize) // checking validity of range
        subError();
    return pCA[sub];
}
```

◆ Example : CmplxArray (7)

```
#include <iostream>
#include "CmplxArray.h"
#include "Cmplx.h"

using namespace std;

void main()
{
    CmplxArray CMPLXES(4);

    cout << "\n Input real and imag in double type: ";
    cin >> CMPLXES[0];
    cout << " CMPLXES[0] : " << CMPLXES[0] << endl;

    cout << " Input real and imag in double type: ";
    cin >> CMPLXES[1];
    cout << " CMPLXES[1] : " << CMPLXES[1] << endl;

    CMPLXES[2] = CMPLXES[0] + CMPLXES[1];
    CMPLXES[3] = CMPLXES[0] - CMPLXES[1];

    cout << " CMPLXES[2] : " << CMPLXES[2] << endl;
    cout << " CMPLXES[3] : " << CMPLXES[3] << endl;

}
```

```
Input real and imagin in double type: 3.4 5.6
CMPLXES[0] : Complex (3.4, 5.6)

Input real and imagin in double type: 1.2 2.3
CMPLXES[1] : Complex (1.2, 2.3)

CMPLXES[2] : Complex (4.6, 7.9)

CMPLXES[3] : Complex (2.2, 3.3)
```

증감 연산자 (++, --) 오버로딩

증감연산자의 오버로딩

◆ Each operator has two versions

- Prefix notation: **++x;**
  - **y = ++x;**
  - **same as x = x+1; y = x;**
- Postfix notation: **x++;**
  - **y = x++;**
  - **same as y = x; x = x+1**

◆ Must distinguish in overload

- Standard overload method → Prefix
  - **int operator++();**
- Add 2d parameter of type int → Postfix
  - **int operator++(int dumy);**
  - Just a marker for compiler!
  - Specifies postfix is allowed

◆ Operator overloading for Date (1)

```
/**  Date.h   */
#include <iostream>
using namespace std;

class Date
{
    friend istream &operator>>(istream &, Date &);
    friend ostream &operator<<(ostream &, const Date &);
public:
    Date();
    Date(int y, int m, int d);
    int getElapsedDays(); // get elapsed days from AD 1. 1. 1
    const Date operator+(int dd);
    const Date operator-(int dd);
    const Date operator++(); // prefix
    const Date operator++(int dummy);  // postfix
    const Date operator--();  // prefix
    const Date operator--(int dummy);  // postfix
    const Date operator=(const Date rightSide);
    bool operator>(const Date rightSide);
    bool operator<(const Date rightSide);
    bool operator==(const Date rightSide);
private:
    int year;
    int month;
    int day;
};
```

◆ Operator overloading for Date (2)

```
/**  Date.cpp  (1)  */

const Date Date::operator+(int dd)
{
      int y, m, d;
      int days_month[13] = { 0, 31, 28, 31,
           30, 31, 30, 31, 31, 30, 31, 30, 31 };
      y = year;
      if (isLeapYear(y))
           days_month[2] = 29;
      m = month;
      d = day + dd;
      while (d > days_month[m])
      {
          d -= days_month[m];
          m++;
          if (m > 12)
          {
              m = 1;
              y++;
              isLeapYear(y) ? days_month[2] = 29
                            : days_month[2] = 28;
          }
      }  // end while
      return Date(y, m, d);
}
```

```
/**  Date.cpp  (2)  */

const Date Date::operator-(int dd)
{
    int y, m, d;
    int days_month[13] = { 0, 31, 28, 31,
         30, 31, 30, 31, 31, 30, 31, 30, 31 };
    y = year;
    if (isLeapYear(y))
        days_month[2] = 29;
    m = month;
    d = day - dd;
    while (d < 1)
    {
        m--;
        if (m < 1)
        {
            y--;
            m = 12;
            isLeapYear(y) ? days_month[2] = 29
                          :  days_month[2] = 28;
        }  // end if
        d += days_month[m];
    }  // end while
    return Date(y, m, d);
}
```

◆ Operator overloading for Date (3)

```
/**  Date.cpp  (3)  */

const Date Date::operator++()
  // prefix
{
     *this = *this + 1;
     return (*this);
}

const Date Date::operator++(int dummy)
  // postfix
{
     Date d = *this;
     ++(*this);

     return d;
}
```

```
/**  Date.cpp  (4)  */

const Date Date::operator--()
  // prefix
{
     *this = *this - 1;

     return(*this);
}

const Date Date::operator--(int dummy)
  // postfix
{
    Date d = *this;

    --(*this);

    return d;
}
```

◆ Operator overloading for Date (4)

```
/**  Date.cpp  (5)  */

const Date Date::operator=(const Date rightSide)
{
     int y, m, d;

     y = rightSide.year;
     m = rightSide.month;
     d = rightSide.day;

     if (isValidDate(y, m, d))
     {
          this->year = y;
          this-> month = m;
          this-> day = d;

          return *this;
     } else {
          return Date(0, 0, 0);
     }
}
```

◆ Operator overloading for Date (5)

```
bool Date::operator>(Date d)
{
        int days1, days2;

        days2 =
        d.getElapsedDays();
        days1 = getElapsedDays();
        if (days1 > days2)
                return true;
        else
                return false;
}
bool Date::operator<(Date d)
{
        int days1, days2;
        days2 =
        d.getElapsedDays();
        days1 = getElapsedDays();
        if (days1 < days2)
                return true;
        else
                return false;
}
```

```
bool Date::operator==(Date d)
{
        int days1, days2;
        days2 = d.getElapsedDays();
        days1 = getElapsedDays();
        if (days1 == days2)
                return true;
        else
                return false;
}
```

◆ class DateArray

```
/* DateArray.h */
#ifndef DATE_ARRAY_H
#define DATE_ARRAY_H

#include <iostream>
#include "Date.h"

class DateArray
{
public:
        DateArray(int size); // constructor
        DateArray(const DateArray &obj);
        ~DateArray();
        int size() { return arraySize; }
        Date &operator[](int  sub);
        void sort();
private:
        bool isValidIndex(int index);
        Date *pDate;
        int arraySize;
};

#endif
```

```
/* DateArray.cpp (1) */

#include "DateArray.h"
#include "Date.h"

DateArray::DateArray(int size) // constructor
{
        arraySize = size;
        pDate = new Date[size];
        for (int i = 0; i<size; i++) {
                pDate[i].year = 0;
                pDate[i].month = 0;
                pDate[i].day = 0;
        }
}

DateArray::DateArray(const DateArray &obj) // constructor
{
        arraySize = obj.arraySize;
        pDate = new Date[arraySize];
        for (int i = 0; i<arraySize; i++) {
                pDate[i] = obj.pDate[i];   // *(pDate+i) = obj.pDate[i];
        }
}

DateArray::~DateArray()  // destructor
{
        if (arraySize > 0)
                delete[] pDate;
}
```

```
/* DateArray.cpp (2) */

bool DateArray::isValidIndex(int sub)
{
        if (sub < 0 || sub >= arraySize)
        {
                cout << "ERROR: Subscript ( " << sub << " ) out of range !!\n";
                exit(0);
        }
        else
          return true;


}

Date &DateArray::operator [](int sub)
{
        if (isValidIndex(sub)) // checking validity of range
                return pDate[sub];
        exit(0);
}
```

```
/* DateArray.cpp (3) */

void DateArray::sort()
{
        int i, j, mx;
        Date tmp;
        int min_date, min_index;

        for (i = 0; i<arraySize - 1; i++)
        {
                min_date = i;
                for (j = i + 1; j<arraySize; j++)
                {
                        if (pDate[j] < pDate[min_date])
                        {
                                min_date = j;
                        }
                }

                if (min_date != i)
                {
                        tmp = pDate[i];
                        pDate[i] = pDate[min_date];
                        pDate[min_date] = tmp;
                }
        }
}
```

```
/* main.cpp (1) */
#include <iostream>
#include <fstream>
#include <iomanip>
#include <time.h>
#include "Date.h"
#include "DateArray.h"

using namespace std;
#define NUM_DATES 8

int main()
{
        int daysToChristmas;
        DateArray dates(NUM_DATES);
        time_t currentTime;
        struct tm *time_and_date;
        int year, month, day;
        ofstream fout;

        fout.open("output.txt");
        if (fout.fail())
        {
                cout << "Fail to open an output file (output.txt)\n";
                exit(1);
        }
```

```
/* main.cpp (2) */

        time(&currentTime);
        time_and_date = localtime(&currentTime);
        year = time_and_date->tm_year + 1900;
        month = time_and_date->tm_mon + 1;
        day = time_and_date->tm_mday;
        Date newYearDay(year, 1, 1), today(year, month, day);
        Date AD010101(1, 1, 1, "AD010101"), thisChristmas;

        fout <<  today << endl;

        thisChristmas = Date(today.getYear(), 12, 25);
        if (today == thisChristmas)
        {
                fout << "Happy Christmas !!" << endl;
        }
        else
        {
                daysToChristmas = thisChristmas.getElapsedDays() - today.getElapsedDays();
                if (daysToChristmas > 0)
                   fout << "Sorry today is not Christmas - You should wait " << daysToChristmas
                        << " days !!" << endl;
                else
                   fout << "Sorry this year's Christmas was " << -daysToChristmas
                         << " days before !!" << endl;
        }

        fout << endl;
        dates[0] = today;
        fout << "dates[0] = today;\n" << dates[0] << endl << endl;
```

```
/* main.cpp (3) */

        dates[1] = ++today; dates[1].setName("dates[1]");
        fout << "dates[1] = ++today;\n" << dates[1] << endl;
        fout << "Currently " << today << endl << endl;

        dates[2] = today++; dates[2].setName("dates[2]");
        fout << "dates[2] = today++;\n" << dates[2] << endl;
        fout << "Currently " << today << endl << endl;

        dates[3] = today--; dates[3].setName("dates[3]");
        fout << "dates[3] = today--;\n" << dates[3] << endl;
        fout << "Currently " << today << endl << endl;

        dates[4] = --today; dates[4].setName("dates[4]");
        fout << "dates[4] = --today;\n" << dates[4] << endl;

        fout << "Currently " << today << endl;

        dates[5] = newYearDay;
        dates[6] = thisChristmas;
        dates[7] = AD010101;

        fout << endl << "Before sorting dates: " << endl;
        for (int i = 0; i < NUM_DATES; i++)
        {
                fout << dates[i] << endl;
        }
```

```
/* main.cpp (4) */

        dates.sort();
        fout << endl << "After sorting dates: " << endl;
        for (int i = 0; i < NUM_DATES; i++)
        {
                fout << dates[i] << endl;
        }

        fout << endl;
        fout.close();

        return 0;
}
```

```
Date [2021. 9.14,  (Tuesday)]
Sorry today is not Christmas - You should wait 102 days !!

dates[0] = today;
Date [2021. 9.14,  (Tuesday)]

dates[1] = ++today;
Date [2021. 9.15, dates[1] (Wednesday)]
Currently Date [2021. 9.15,  (Wednesday)]

dates[2] = today++;
Date [2021. 9.15, dates[2] (Wednesday)]
Currently Date [2021. 9.16,  (Thursday)]

dates[3] = today--;
Date [2021. 9.16, dates[3] (Thursday)]
Currently Date [2021. 9.15,  (Wednesday)]

dates[4] = --today;
Date [2021. 9.14, dates[4] (Tuesday)]
Currently Date [2021. 9.14,  (Tuesday)]

Before sorting dates:
Date [2021. 9.14,  (Tuesday)]
Date [2021. 9.15, dates[1] (Wednesday)]
Date [2021. 9.15, dates[2] (Wednesday)]
Date [2021. 9.16, dates[3] (Thursday)]
Date [2021. 9.14, dates[4] (Tuesday)]
Date [2021. 1. 1,  (Friday)]
Date [2021.12.25,  (Saturday)]
Date [   1. 1. 1, AD010101 (Monday)]

After sorting dates:
Date [   1. 1. 1, AD010101 (Monday)]
Date [2021. 1. 1,  (Friday)]
Date [2021. 9.14, dates[4] (Tuesday)]
Date [2021. 9.14,  (Tuesday)]
Date [2021. 9.15, dates[2] (Wednesday)]
Date [2021. 9.15, dates[1] (Wednesday)]
Date [2021. 9.16, dates[3] (Thursday)]
Date [2021.12.25,  (Saturday)]
```

Homework 4

Homework 4

4.1 operator overloading for Class complex number

1) Define a class Cmplx that contains double type private data members: real and imag. Class Cmplx has operator overloading of input/output operator (>>, <<), arithmetic operators (+, -, *, /, ~ (conjugate)), equality operator (==, !=), assignment operator (=).

2) Define a class CmplxArray, that include a given number of Cmplx instances. The CmplxArray should provide operator overloading of [].

3) In main() function, specify 7 instances of Cmplx array as cmplxs[0..6].

4) Using input operator overloading (cin >> cmplxs[]), input two complex numbers, and initialize the first two complex numbers of the array.

5) Using the cmplxs[0] and cmplxs[1], perform arithmetic operations (+, -, *, /, ~), and store the results at cmplxs[2], cmplxs[3], cmplxs[4], cmplxs[5], and cmplxs[6], respectively.

6) Using the cmplxs[0] and cmplxs[1], perform operators == and !=, print out the results.

7) Output the calculated result of cmplxs[] using operator <<.

(Note. Header file and main() function should include following sample source code.)

```
/** Cmplx.h */
#ifndef CMPLX_H
#define CMPLX_H
#include <iostream>

using namespace std;
class CmplxArray;
class Cmplx
{
    friend ostream & operator<< (ostream &, const Cmplx &);
    friend istream & operator>> (istream &, Cmplx &);
    friend class CmplxArray;
public:
    Cmplx(double real=0.0, double imag=0.0); // constructor
    double mag() const; // return the magnitude
    const Cmplx operator+(const Cmplx &);
    const Cmplx operator-(const Cmplx &);
    const Cmplx operator*(const Cmplx &);
    const Cmplx operator/(const Cmplx &);
    const Cmplx operator~(); // conjugate of this complex
    bool operator==(const Cmplx &);
    bool operator!=(const Cmplx &);
    bool operator<(const Cmplx &);
    bool operator>(const Cmplx &);
    const Cmplx operator=(const Cmplx &);
private:
    double real;
    double imag;
};

#endif
```

4.1 (cont.) operator overloading for Class complex number (CmplxArray.h)

```
/**  CmplxArray.h  */

#ifndef CMPLXARRAY_H
#define CMPLXARRAY_H

#include <iostream>
#include "Cmplx.h"

using namespace std;

class CmplxArray
{
public:
    CmplxArray(int size); // constructor
    ~CmplxArray();
    int size() { return cmplxArrySize; }
    Cmplx &operator[](int);
    void print(ostream& fout);
    void sort();
private:
    Cmplx *pCA;
    int cmplxArrySize;
    bool isValidIndex(int indx);
};

#endif
```

4.1 (cont.) operator overloading for Class complex number (main.c())

```
/**  main.cpp  (1) */
#include <iostream>
#include <fstream>
#include "CmplxArray.h"
#include "Cmplx.h"

using namespace std;

void main()
{
    ofstream fout;
    ifstream fin;
    CmplxArray cmplxs(7);

    fin.open("input.txt");
    if (fin.fail())
    {
        cout << "Error in opening intput.txt !!" << endl;
        exit;
    }
    fin >> cmplxs[0] >> cmplxs[1];
```

```
/**  main.cpp  (2) */
    cmplxs[2] = cmplxs[0] + cmplxs[1];
    cmplxs[3] = cmplxs[0] - cmplxs[1];
    cmplxs[4] = cmplxs[0] * cmplxs[1];
    cmplxs[5] = cmplxs[0] / cmplxs[1];
    cmplxs[6] = ~cmplxs[0];

    cout << "cmplxs[0] = " << cmplxs[0] << endl;
    cout << "cmplxs[1] = " << cmplxs[1] << endl;
    cout << "cmplxs[2] = cmplxs[0] + cmplxs[1] = " << cmplxs[2] << endl;
    cout << "cmplxs[3] = cmplxs[0] - cmplxs[1] = " << cmplxs[3] << endl;
    cout << "cmplxs[4] = cmplxs[0] * cmplxs[1] = " << cmplxs[4] << endl;
    cout << "cmplxs[5] = cmplxs[0] / cmplxs[1] = " << cmplxs[5] << endl;
    cout << "cmplxs[6] = ~cmplxs[0] (conjugate) = " << cmplxs[6] << endl;

    if (cmplxs[0] == cmplxs[1])
        cout << "cmplxs[0] is equal to cmplxs[1]" << endl;
    else
        cout << "cmplxs[0] is not equal to cmplxs[1]" << endl;

    cmplxs[1] = cmplxs[0];
    cout << "After cmplxs[1] = cmplxs[0]; ==> " << endl;
    if (cmplxs[0] == cmplxs[1])
        cout << "cmplxs[0] is equal to cmplxs[1]" << endl;
    else
        cout << "cmplxs[0] is not equal to cmplxs[1]" << endl;
    fin.close();
}
```

◆ 실행 결과

```
cmplxs[0] =    1.10 +    2.20j
cmplxs[1] =    3.30 +    4.40j
cmplxs[2] = cmplxs[0] + cmplxs[1] =    4.40 +    6.60j
cmplxs[3] = cmplxs[0] - cmplxs[1] =   -2.20 -    2.20j
cmplxs[4] = cmplxs[0] * cmplxs[1] =   -6.05 +   12.10j
cmplxs[5] = cmplxs[0] / cmplxs[1] =    0.44 +    0.08j
cmplxs[6] = ~cmplxs[0] (conjugate) =    1.10 -    2.20j
cmplxs[0] is not equal to cmplxs[1]
After cmplxs[1] = cmplxs[0]; ==>
cmplxs[0] is equal to cmplxs[1]
```

4.2 Operator overloading for Class Mtrx

1) Define a class Mtrx that contains a private data member of 2-dimentional array to store double data type. Class Mtrx has operator overloading of input/output operator (>>, <<), arithmetic operator (+, -, *), transpose operator(~), equality operator (==, !=), assignment operator (=).

2) Define a class MtrxArray, that include a given number of Mtrx instances. The MtrxArray should provide operator overloading of [].

3) In main() function, define a MtrxArray of 7 elements of Mtrx elements, MtrxArray mtrx(7).

4) Using “>>” operator, input mtrx[0], mtrx[1] and mtrx[2] from input data file “Matrix_data.txt”.

5) Using the operator (‘+’), perform mtrx[3] = mtrx[0] + mtrx[1].

6) Using the operator (‘-’), perform mtrx[4] = mtrx[0] - mtrx[1].

7) Using the operator (‘*’), perform mtrx[5] = mtrx[0] * mtrx[2].

8) Using the operator (‘~’), perform mtrx[6] = ~mtrx[5].

9) Using the operators == and !=, check mtrx[5] == mtrx[6] and mtrx[5] != mtrx[6], and print out the results.

10) Printout each operation result using operator <<.

4.2 (cont) Header File ("Mtrx.h")

```
/**  Class_Mtrx.h     */
#define MAX_SIZE 100
#include <string>
using namespace std;
class MtrxArray;
class Mtrx {
    friend ostream & operator<< (ostream &, const Mtrx &);
    friend istream& operator>> (istream&, Mtrx&);
    friend class MtrxArray;
public:
    Mtrx(); // default constructor
    Mtrx(string nm, double *pA, int num_row, int num_col);
    ~Mtrx();
    void init(int n_row, int n_col);
    void set_name(string nm) { name = nm; }
    string get_name() const { return name; }
    int get_n_row() const { return n_row; }
    int get_n_col() const { return n_col; }
    const Mtrx operator+(const Mtrx&);
    const Mtrx operator-(const Mtrx&);
    const Mtrx operator*(const Mtrx&);
    const Mtrx operator~(); // transpose()
    const Mtrx& operator=(const Mtrx&);
    bool operator==(const Mtrx&);
    bool operator!=(const Mtrx&);
private:
    string name;
    int n_row;
    int n_col;
    double **dM;
};
```

4.2 (cont) Header File (“MtrxArray.h”)

```
/**  MtrxArray.h    */
#ifndef MTRX_ARRAY_H
#define MTRX_ARRAY_H
#include <iostream>
#include "Mtrx.h"

using namespace std;

class Mtrx;

class MtrxArray
{
public:
    MtrxArray(int array_size); // constructor
    ~MtrxArray();  // destructor
    Mtrx &operator[](int);
private:
    Mtrx *pMtrx;
    int mtrxArrySize;
    bool isValidIndex(int index);
};
#endif
```

4.2 (cont) main.cpp

```
/** main.cpp (1) */

#include <iostream>
#include <fstream>
#include <string>
#include "Mtrx.h"
#include "MtrxArray.h"
using namespace std;

#define NUM_MTRX 7

int main()
{
    ifstream fin;
    ofstream fout;
    int n_row, n_col;

    fin.open("Matrix_data.txt");
    if (fin.fail())
    {
        cout << "Error in opening input data file !!" << endl;
        exit;
    }

    fout.open("Result.txt");
    if (fout.fail())
    {
        cout << "Error in opening output data file !!" << endl;
        exit;
    }
```

4.2 (cont) main.cpp

```
/** main.cpp (2) */

    MtrxArray mtrx(NUM_MTRX);

    fin >> mtrx[0] >> mtrx[1] >> mtrx[2];
    mtrx[0].set_name("mtrx[0] =");
    mtrx[1].set_name("mtrx[1] =");
    mtrx[2].set_name("mtrx[2] =");
    fout << mtrx[0] << endl;
    fout << mtrx[1] << endl;
    fout << mtrx[2] << endl;

    mtrx[3] = mtrx[0] + mtrx[1];
    mtrx[3].set_name("mtrx[3] = mtrx[0] + mtrx[1] =");
    fout << mtrx[3] << endl;

    mtrx[4] = mtrx[0] - mtrx[1];
    mtrx[4].set_name("mtrx[4] = mtrx[0] - mtrx[1] =");
    fout << mtrx[4] << endl;

    mtrx[5] = mtrx[0] * mtrx[2];
    mtrx[5].set_name("mtrx[5] = mtrx[0] * mtrx[2] =");
    fout << mtrx[5] << endl;

    mtrx[6] = ~mtrx[5];
    mtrx[6].set_name("mtrx[6] = ~mtrx[5] (transposed matrix) =");
    fout << mtrx[6] << endl;
```

4.2 (cont) main.cpp

```
/** main.cpp (3) */

    if (mtrx[5] == mtrx[6])
        fout << "mtrx[5] and mtrx[6] are equal.₩n";
    if (mtrx[5] != mtrx[6])
        fout << "mtrx[5] and mtrx[6] are not equal.₩n";

    fin.close();
    fout.close();

    return 0;
}
```

4.2 (cont) 입력 데이터 파일, 출력 결과

Matrix_data.txt

```
5  7
1.0 2.0 3.0 4.0 5.0 6.0 7.0
2.0 3.0 4.0 5.0 1.0 7.0 8.0
3.0 2.0 5.0 3.0 2.0 4.0 6.0
4.0 3.0 2.0 7.0 2.0 1.0 9.0
5.0 4.0 3.0 2.0 9.0 6.0 9.0

5  7
1.0 0.0 0.0 0.0 0.0 1.0 2.0
0.0 1.0 0.0 0.0 0.0 2.0 3.0
0.0 0.0 1.0 0.0 0.0 3.0 4.0
0.0 0.0 0.0 1.0 0.0 4.0 5.0
0.0 0.0 0.0 0.0 1.0 5.0 6.0

7  5
1.0 2.0 3.0 4.0 5.0
6.0 7.0 2.0 3.0 4.0
5.0 1.0 7.0 8.0 3.0
2.0 5.0 3.0 2.0 4.0
6.0 4.0 3.0 2.0 7.0
2.0 1.0 9.0 5.0 4.0
3.0 2.0 9.0 6.0 9.0
```

Result.txt

```
mtrx[0] =
┌                                  ┐
   1.00   2.00   3.00   4.00   5.00   6.00   7.00
   2.00   3.00   4.00   5.00   1.00   7.00   8.00
   3.00   2.00   5.00   3.00   2.00   4.00   6.00
   4.00   3.00   2.00   7.00   2.00   1.00   9.00
   5.00   4.00   3.00   2.00   9.00   6.00   9.00
└                                  ┘

mtrx[1] =
┌                                  ┐
   1.00   0.00   0.00   0.00   0.00   1.00   2.00
   0.00   1.00   0.00   0.00   0.00   2.00   3.00
   0.00   0.00   1.00   0.00   0.00   3.00   4.00
   0.00   0.00   0.00   1.00   0.00   4.00   5.00
   0.00   0.00   0.00   0.00   1.00   5.00   6.00
└                                  ┘

mtrx[2] =
┌                          ┐
   1.00   2.00   3.00   4.00   5.00
   6.00   7.00   2.00   3.00   4.00
   5.00   1.00   7.00   8.00   3.00
   2.00   5.00   3.00   2.00   4.00
   6.00   4.00   3.00   2.00   7.00
   2.00   1.00   9.00   5.00   4.00
   3.00   2.00   9.00   6.00   9.00
└                          ┘

mtrx[3] = mtrx[0] + mtrx[1] =
┌                                  ┐
   2.00   2.00   3.00   4.00   5.00   7.00   9.00
   2.00   4.00   4.00   5.00   1.00   9.00  11.00
   3.00   2.00   6.00   3.00   2.00   7.00  10.00
   4.00   3.00   2.00   8.00   2.00   5.00  14.00
   5.00   4.00   3.00   2.00  10.00  11.00  15.00
└                                  ┘

mtrx[4] = mtrx[0] - mtrx[1] =
┌                                  ┐
   0.00   2.00   3.00   4.00   5.00   5.00   5.00
   2.00   2.00   4.00   5.00   1.00   5.00   5.00
   3.00   2.00   4.00   3.00   2.00   1.00   2.00
   4.00   3.00   2.00   6.00   2.00  -3.00   4.00
   5.00   4.00   3.00   2.00   8.00   1.00   3.00
└                                  ┘

mtrx[5] = mtrx[0] * mtrx[2] =
┌                                  ┐
  99.00  79.00 172.00 124.00 160.00
  94.00  81.00 193.00 144.00 161.00
  84.00  64.00 153.00 124.00 134.00
  87.00  93.00 149.00 118.00 165.00
 141.00 111.00 212.00 162.00 226.00
└                                  ┘

mtrx[6] = ~mtrx[5] (transposed matrix) =
┌                                  ┐
  99.00  94.00  84.00  87.00 141.00
  79.00  81.00  64.00  93.00 111.00
 172.00 193.00 153.00 149.00 212.00
 124.00 144.00 124.00 118.00 162.00
 160.00 161.00 134.00 165.00 226.00
└                                  ┘

mtrx[5] and mtrx[6] are not equal.
```