객체지향프로그래밍과 자료구조 (실습)

Lab. 4 (보충설명)
Class Mtrx and Class MtrxArray
with Operator Overloading

정보통신공학과
교수 김 영 탁

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

Outline

◆연산자 오버로딩

◆class Mtrx with Operator Overloading

◆class MtrxArray with Operator Overloading

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 2

O-O Programming & Data Structure
Prof. Young-Tak Kim

연산자 오버로딩
(Operator Overloading)

◆ **Operators +, -, %, ==,<<, >>, ^, ~ etc.**
- Really just functions!

◆ **Simply "called" with different syntax:**
x + 7
- "+" is binary operator with **x** and **7** as operands
- We "like" this notation as humans

◆ **Think of it as:**
+(x, 7)
- "+" is the function name
- x, 7 are the arguments
- Function "+" returns "sum" of it's arguments

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 3

O-O Programming & Data Structure
Prof. Young-Tak Kim

연산자 오버로딩

◆ ***Built-in*** **operators**
- e.g., **+, -, = , %, ==, /, ***
- Already work for C++ built-in types
- In standard "binary" notation

◆ **We can overload them!**
- To work with user-defined class/types!
- To add "Mtrx" types, or "Cmplx" types
  - As appropriate for our needs
  - In "notation" we're comfortable with

◆ **Always overload with "*similar actions*"!**

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 4

O-O Programming & Data Structure
Prof. Young-Tak Kim

연산자 오버로딩이 필요한 이유

◆ **간략한 표현과 쉬운 이해**

● using the usual mathematical operators, concise notations are possible

```
/* What you have use */

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;
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 5

O-O Programming & Data Structure
Prof. Young-Tak Kim

Operators that can be Overloaded and that cannot be overloaded

◆ **Operators that can be overloaded**

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

◆ **Operators that cannot be overloaded**

.   .*   ::   ?:

O-O Programming & Data Structure
Prof. Young-Tak Kim

연산자의 오버로딩 구현 (1)

◆Overloading operators

- VERY similar to overloading functions
- Operator itself is "name" of function

◆Example Declaration:

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

- Overloads **+** for operands of type **Cmplx**
- Uses constant reference parameters for efficiency
- Returned value is type Cmplx
    - Allows addition of "Cmplx" objects

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 7

O-O Programming & Data Structure
Prof. Young-Tak Kim

연산자의 오버로딩 구현 (2)

◆ **연산자 오버로딩 구현의 2가지 방법**

- case 1: implementation as a **non-member function**
  - similar to usual function overloading
  - standalone function
  - operator overloading is not included in class definition
  - generally simple to implement

- case 2: implementation as a **member function**
  - operator overloading is included in class definition
  - principles suggest member operators, to maintain the "sprit" of object-oriented programming

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 8

O-O Programming & Data Structure
Prof. Young-Tak Kim

"+" 연산자의 오버로딩

◆**Given previous example:**

- Note: overloaded "+" *NOT* **member function**
- Definition is "more involved" than simple "add"
  - Requires issues of Cmplx type addition
  - Must handle negative/positive values

◆**Operator overload definitions generally very simple**

- Just perform "addition" particular to "your" type

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 9

O-O Programming & Data Structure
Prof. Young-Tak Kim

Class Cmplx를 위한 "+" 연산자

◆ 개별적인 함수 오버로딩으로 연산자 오버로딩 구현

```cpp
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);
}
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 10

O-O Programming & Data Structure
Prof. Young-Tak Kim

클래스의 멤버함수로 연산자 오버로딩을 구현

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

- If "**+**" overloaded as member operator:
    - Variable/object cost is calling object
    - Object tax is single argument
- Think of as: **c3 = c1+(c2);**

◆ **Declaration of "+" in class definition (header file):**

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

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 11

O-O Programming & Data Structure
Prof. Young-Tak Kim

**class Cmplx
with operator overloadings**

◆ Example Cmplx (1)

```cpp
/** 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;

};
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 13

O-O Programming & Data Structure
Prof. Young-Tak Kim

◆ Example Cmplx (2)

```cpp
/**  Cmplx.cpp */

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

using namespace std;

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

ostream &operator<<(ostream &output, const Cmplx &c)
{
        output << "Complex (" << c.real << ", "
               << c.imag << ")" << endl;

        return output;
}

istream &operator>>(istream &input, Cmplx &c)
{
        input >> c.real;
        input >> c.imag;

        return input;
}
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 14

O-O Programming & Data Structure
Prof. Young-Tak Kim

◆ Example Cmplx (3)

```cpp
/* Cmplx.cpp (cont.) */

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;
}
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 15

O-O Programming & Data Structure
Prof. Young-Tak Kim

◆ Example Cmplx (4)

<Result of Execution>

```
input c1.real and c1.imagin: 3.3 4.4
c1 : Complex (3.3, 4.4)

input c2.real and c2.imagin: 1.1 2.2
c2 : Complex (1.1, 2.2)

c3 = c1 + c2: Complex (4.4, 6.6)

c4 = c1 - c2: Complex (2.2, 2.2)
```

```cpp
/** main.cpp */

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

using namespace std;

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

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

        cout << "input c2.real and c2.imag: ";
        cin >> c2;
        cout << "c2 : " << c2 << endl;

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

        c4 = c1 - c2;
        cout << "c4 = c1 - c2: " << c4 << endl;
}
```

const Functions

◆ **When to make function const?**

- Constant functions not allowed to alter class member data

- Constant objects can ONLY call *constant member functions*

◆ **Good style dictates:**

- Any member function that will NOT modify data should be made *const*

◆ **Use keyword *const* after function declaration and heading**

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 17

O-O Programming & Data Structure
Prof. Young-Tak Kim

class Mtrx and class MtrxArray
with operator overloadings

class Mtrx

```cpp
/** Class_Mtrx.h */
#ifndef MTRX_H
#define MTRX_H
#include <iostream>
#include <fstream>
using namespace std;
#define MAX_SIZE 100

class Mtrx {
public:
    Mtrx(string nm, int num_row, int num_col);
    Mtrx(string nm, double dA[], int num_row, int num_col);
    Mtrx(istream& fin);
    ~Mtrx(); // destructor
    int getN_row() const { return n_row; }
    int getN_col() const { return n_col; }
    void fprintMtrx(ostream& fout);
    void setName(string nm) { name = nm;};
    string getName() { return name;};
    const Mtrx add(const Mtrx&);
    const Mtrx sub(const Mtrx&);
    const Mtrx multiply(const Mtrx&);
    const Mtrx transpose();
private:
    string name;
    int n_row;
    int n_col;
    double **dM;
};
#endif
```

Adva
Yeun

class Mtrx 멤버함수 구현

```cpp
/**  Matrix.cpp  (1) */
#include "Class_Mtrx.h"
#include <iostream>
#include <iomanip>

using namespace std;
typedef double * DBLPTR;

Mtrx::Mtrx(string nm, int num_row, int num_col)
 : name(nm), n_row(num_row), n_col(num_col)
{
    int i, j;
    //cout <<"Mtrx constructor (int size: "
    //        << size << ")\n";
    dM = new DBLPTR[n_row];
    for (i=0; i<n_row; i++)
    {
        dM[i] = new double[n_col];
    }
    for (i=0; i<n_row; i++) {
        for (j=0; j<n_col; j++) {
            dM[i][j] = 0.0;
        }
    }
    // cout <<"End of Mtrx constructor... \n";
}
```

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

Mtrx::~Mtrx()
{
    // cout << "destructor of Mtrx ("
    //         << name << ")" << endl;
    /*
    for (int i=0; i<n_row; i++)
        delete [] dM[i];
    delete [] dM;
    */
}
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 20

O-O Programming & Data Structure
Prof. Young-Tak Kim

```cpp
/**  Matrix.cpp  (3) */
Mtrx::Mtrx(istream& fin)
{
    // DBLPTR *dM; /* defined in class, as private data member
    int i, j, size_row, size_col, num_data, cnt;
    double d;

    //cout <<"Mtrx constructor (double **dA, int size: " << size << ") \n";
    fin >> size_row >> size_col;

    n_row = size_row;
    n_col = size_col;
    dM = new DBLPTR[n_row];
    for (i = 0; i<n_row; i++)
    {
        dM[i] = new double[n_col];
    }
    for (i = 0; i<n_row; i++) {
        for (j = 0; j<n_col; j++) {
            if (fin.eof())
                dM[i][j] = 0.0;
            else
            {
                fin >> d;
                dM[i][j] = d;
            }
        }
    }
    //cout <<"End of Mtrx constructor... \n";
}
```

Advanced Networking Tech. Lab.
Yeungnam University (*yuANTL*)

Lab. 4 - 21

O-O Programming & Data Structure
Prof. Young-Tak Kim

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

#define SETW 6
void Mtrx::fprintMtrx(ostream& fout)
{
    unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;
    unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;

    fout << name << endl;
    for (int i=0; i< n_row; i++) {
        for (int j=0; j< n_col; j++)
        {
            fout.setf(ios::fixed);
            fout.precision(2);
            if ((i==0) && (j==0))
                fout << a6 << a3 << setw(SETW) << dM[i][j];
            else if ((i==0) && (j== (n_col-1)))
                fout << setw(SETW) <<  dM[i][j] << a6 << a4;
            else if ((i>0) && (i<(n_row-1)) && (j==0))
                fout << a6 << a2 << setw(SETW) << dM[i][j];
            else if ((i>0) && (i<(n_row-1)) && (j== (n_col-1)))
                fout << setw(SETW) <<  dM[i][j] << a6 << a2;
            else if ((i==(n_row-1)) && (j==0))
                fout << a6 << a6 << setw(SETW) <<  dM[i][j];
            else if ((i==(n_row-1)) && (j==(n_col-1)))
                fout << setw(SETW) <<  dM[i][j] << a6 << a5;
            else
                fout << setw(SETW) <<  dM[i][j];
        }
        fout << endl;
    }
    fout << endl;
}
```

출력 결과	확장 완성형 코드
―	0xA6, 0xA1
│	0xA6, 0xA2
┌	0xA6, 0xA3
┐	0xA6, 0xA4
┘	0xA6, 0xA5
└	0xA6, 0xA6

```
MtrxA =
   ┌  1.00   2.00   3.00   4.00   5.00┐
   │  2.00   3.00   4.00   5.00   1.00│
   │  3.00   2.00   5.00   3.00   2.00│
   │  4.00   3.00   2.00   7.00   2.00│
   └  5.00   4.00   3.00   2.00   9.00┘
```

Yeungnam University (*yuANTL*)   *Lab. 4 - 22*

O-O Programming & Data Structure
Prof. Young-Tak Kim

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

const Mtrx Mtrx::add(const Mtrx& mA)
{
    int i, j;

    Mtrx mR("mR", n_row, n_col);

    for (i=0; i<n_row; i++) {
        for (j=0; j<n_col; j++) {
            mR.dM[i][j] = dM[i][j] + mA.dM[i][j];
        }
    }

    return mR;
}
```

```cpp
/**   Matrix.cpp  (6) */
const Mtrx Mtrx::sub(const Mtrx& mA)
{
    int i, j;

    Mtrx mR("mR", n_row, n_col);

    for (i=0; i<n_row; i++) {
        for (j=0; j<n_col; j++) {
            mR.dM[i][j] = dM[i][j] - mA.dM[i][j];
        }
    }

    return mR;
}
```

Advanced Networking Tech. Lab.
Yeungnam University (*yuANTL*)

Lab. 4 - 23

O-O Programming & Data Structure
Prof. Young-Tak Kim

```cpp
/**   Matrix.cpp  (7) */

const Mtrx Mtrx::multiply(const Mtrx& mA)
{
    int i, j, k;

    Mtrx mR("mR", n_row, mA.n_col);

    for (i=0; i<n_row; i++) {
        for (j=0; j<mA.n_col; j++) {
            mR.dM[i][j] = 0.0;
            for (k=0; k<n_col; k++) {
                mR.dM[i][j] += dM[i][k] * mA.dM[k][j];
            }
        }
    }

    return mR;
}
```

```cpp
/**   Matrix.cpp  (8) */

const Mtrx Mtrx::transpose()
{
    int i, j;

    Mtrx mR("mR", n_col, mA.n_row);

    for (i=0; i<n_row; i++) {
        for (j=0; j<mA.n_col; j++) {
            mR.dM[j][i] = dM[i][j];
        }
    }

    return mR;
}
```

Advanced Networking Tech. Lab.
Yeungnam University (*yuANTL*)

Lab. 4 - 24

O-O Programming & Data Structure
Prof. Young-Tak Kim

class Mtrx 응용 프로그램

```c
/**  main.c (1) */
#include <iostream>
#include <fstream>
#include "Class_Mtrx.h"
using namespace std;

void main()
{
    ifstream fin;
    ofstream fout;

    fin.open("Matrix_5x5_data.txt");
    if (fin.fail())
    {
        cout << "Error in opening Matrix_5x5_data.txt !!" << endl;
        exit;
    }

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

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 25

O-O Programming & Data Structure
Prof. Young-Tak Kim

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

    Mtrx mtrxA(fin);
    mtrxA.setName("MtrxA =");
    fout <<" MtrxA:\n";
    mtrxA.fprintMtrx(fout);

    Mtrx mtrxB(fin);
    mtrxB.setName("MtrxB =");
    fout <<" MtrxB:\n";
    mtrxB.fprintMtrx(fout);

    int n_row = mtrxA.getN_row();
    int n_col  = mtrxB.getN_col();

    Mtrx mtrxC("", n_row, n_col);
    mtrxC = mtrxA.add(mtrxB);
    mtrxC.setName("MtrxC = mtrxA.add(mtrxB) =");
    mtrxC.fprintMtrx(fout);

    Mtrx mtrxD("", n_row, n_col);
    mtrxD = mtrxA.sub(mtrxB);
    mtrxD.setName("MtrxD = mtrxA.sub(mtrxB) =");
    mtrxD.fprintMtrx(fout);
```

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

    Mtrx mtrxE("",  n_row, n_col);
    mtrxE = mtrxA.multiply(mtrxB);
    mtrxE.setName("MtrxC = mtrxA.multiply(mtrxB) =");
    mtrxE.fprintMtrx(fout);

    fout.close();
} // end of main()
```

```
5   5
1.0 2.0 3.0 4.0 5.0
2.0 3.0 4.0 5.0 1.0
3.0 2.0 5.0 3.0 2.0
4.0 3.0 2.0 7.0 2.0
5.0 4.0 3.0 2.0 9.0

5   5
1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 1.0
```

(Input data)

```
MtrxA =
⌈  1.00   2.00   3.00   4.00   5.00⌉
│  2.00   3.00   4.00   5.00   1.00│
│  3.00   2.00   5.00   3.00   2.00│
│  4.00   3.00   2.00   7.00   2.00│
⌊  5.00   4.00   3.00   2.00   9.00⌋

MtrxB =
⌈  1.00   0.00   0.00   0.00   0.00⌉
│  0.00   1.00   0.00   0.00   0.00│
│  0.00   0.00   1.00   0.00   0.00│
│  0.00   0.00   0.00   1.00   0.00│
⌊  0.00   0.00   0.00   0.00   1.00⌋

MtrxC = mtrxA.add(mtrxB) =
⌈  2.00   2.00   3.00   4.00   5.00⌉
│  2.00   4.00   4.00   5.00   1.00│
│  3.00   2.00   6.00   3.00   2.00│
│  4.00   3.00   2.00   8.00   2.00│
⌊  5.00   4.00   3.00   2.00  10.00⌋

MtrxD = mtrxA.sub(mtrxB) =
⌈  0.00   2.00   3.00   4.00   5.00⌉
│  2.00   2.00   4.00   5.00   1.00│
│  3.00   2.00   4.00   3.00   2.00│
│  4.00   3.00   2.00   6.00   2.00│
⌊  5.00   4.00   3.00   2.00   8.00⌋

MtrxE = mtrxA.multiply(mtrxB) =
⌈  1.00   2.00   3.00   4.00   5.00⌉
│  2.00   3.00   4.00   5.00   1.00│
│  3.00   2.00   5.00   3.00   2.00│
│  4.00   3.00   2.00   7.00   2.00│
⌊  5.00   4.00   3.00   2.00   9.00⌋

mtrxA and mtrxE are same
```

(Output result)

Advanced Networking Tech. Lab.
Yeungnam University (yuANTL)

Lab. 4 - 26

O-O Programming & Data Structure
Prof. Young-Tak Kim

class Mtrx와 연산자 오버로딩

```cpp
/**   Class_Mtrx.h */

#define 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, int n_row, int n_col);
        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;
};
```

class Mtrx를 위한 operator<<()

```cpp
ostream& operator<<(ostream& fout, Mtrx& m)
{
    unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;
    unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;
    fout << m.get_name() << endl;
    for (int i=0; i< n_row; i++) {
        for (int j=0; j< n_col; j++)
        {
            fout.setf(ios::fixed);
            fout.precision(2);
            if ((i==0) && (j==0))
                fout << a6 << a3 << setw(SETW) << m.dM[i][j];
            else if ((i==0) && (j== (n_col-1)))
                fout << setw(SETW) << m.dM[i][j] << a6 << a4;
            else if ((i>0) && (i<(n_row-1)) && (j==0))
                fout << a6 << a2 << setw(SETW) << m.dM[i][j];
            else if ((i>0) && (i<(n_row-1)) && (j== (n_col-1)))
                fout << setw(SETW) << m.dM[i][j] << a6 << a2;
            else if ((i==(n_row-1)) && (j==0))
                fout << a6 << a6 << setw(SETW) << m.dM[i][j];
            else if ((i==(n_row-1)) && (j==(n_col-1)))
                fout << setw(SETW) << m.dM[i][j] << a6 << a5;
            else
                fout << setw(SETW) << m.dM[i][j];
        }
        fout << endl;
    }
    fout << endl;
    return fout;
}
```

출력 결과	확장 완성형 코드
─	0xA6, 0xA1
│	0xA6, 0xA2
┌	0xA6, 0xA3
┐	0xA6, 0xA4
┘	0xA6, 0xA5
└	0xA6, 0xA6

```
┌ 1.00  2.00  3.00  4.00  5.00┐
│ 2.00  3.00  4.00  5.00  1.00│
│ 3.00  2.00  5.00  3.00  2.00│
│ 4.00  3.00  2.00  7.00  2.00│
└ 5.00  4.00  3.00  2.00  9.00┘
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 28

O-O Programming & Data Structure
Prof. Young-Tak Kim

class Mtrx를 위한 operator+()

```
const Mtrx Mtrx::operator+(Mtrx& mA)
{
    int i, j;

    Mtrx mR("mR", n_row, n_col);

    for (i=0; i<n_row; i++) {
        for (j=0; j<n_col; j++) {
            mR.dM[i][j] = dM[i][j] + mA.dM[i][j];
        }
    }

    return mR;
}
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 29

O-O Programming & Data Structure
Prof. Young-Tak Kim

class MtrxArray

```cpp
/* 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); // constructor
    ~MtrxArray();  // destructor
    Mtrx &operator[](int);
    int getSize() {return mtrxArrySize;}
private:
    Mtrx *pMtrx;
    int mtrxArrySize;
};
#endif
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 30

O-O Programming & Data Structure
Prof. Young-Tak Kim

```cpp
/**  MtrxArray.cpp (1) */
#include "MtrxArray.h"
#include "Mtrx.h"

MtrxArray::MtrxArray(int arraySize) // constructor
{
      mtrxArrySize = arraySize;
      pMtrx = new Mtrx[arraySize];
}

MtrxArray::~MtrxArray()
{
      //cout << "MtrxArray :: destructor" << endl;
      if (pMtrx != NULL){
            delete[] pMtrx;
      }
}

Mtrx &MtrxArray::operator [](int sub)
{
      if (sub < 0 || sub >= mtrxArrySize)
      {
            cout << "ERROR: Subscript out of range.\n";
            exit(0);
      }
      return pMtrx[sub];
}
```

Advanced Networking Tech. Lab.
Yeungnam University (*yuANTL*)

Lab. 4 - 31

O-O Programming & Data Structure
Prof. Young-Tak Kim

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;
        }
```

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 32

O-O Programming & Data Structure
Prof. Young-Tak Kim

```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;

        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;
}
```

input data file (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
```

O-O Programming & Data Structure
Prof. Young-Tak Kim

실행 결과

```
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.
```

O-O Programming & Data Structure
Prof. Young-Tak Kim

Lab. 4 Oral Test

Q 4.1 C++ 프로그래밍에서 연산자 오버로딩 (operator overloading)의
필요성에 대하여 예를 들어 설명하라. (핵심포인트: 사용자 편의성)

Q 4.2 C++ 클래스인 class Mtrx의 friend 함수로 operator<<()를 구현에서 call-
by-reference , return-by-reference, const 를 사용하는 이유에 대하여
설명하라. (핵심 포인트: C++ 클래스에 대한 operator<<() 함수의 구현에서
call-by-reference , return-by-reference, const를 사용 이유, chained
operation이 가능하도록 하기 위한 구조)

Q 4.3 C++ 클래스인 class Mtrx의 멤버함수로 덧셈 계산을 위한 '+' 연산자의
operator overloading 함수를 구현하는 방법에 대하여 설명하라.
(핵심포인트: 전달되는 인수, 내부 실행 절차, 결과의 반환)

Q 4.4 C++ 클래스인 class Mtrx의 멤버함수로 대입연산을 위한 '=' 연산자의
operator overloading 함수를 구현하는 방법에 대하여 설명하라.
(핵심포인트: 전달되는 인수, 내부 실행 절차, 결과의 반환)

Advanced Networking Tech. Lab.
Yeungnam University (YU-ANTL)

Lab. 4 - 36

O-O Programming & Data Structure
Prof. Young-Tak Kim