객체지향형 프로그래밍과 자료구조 Lab. 4

4.1 Class Mtrx

(1) class Mtrx 정의:

```
/**   Class_Mtrx.h */

#ifndef CLASS_MTRX_H
#define CLASS_MTRX_H
#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 n_row, int n_col);
        Mtrx(istream& fin);
        ~Mtrx();
        void init(int n_row, int n_col);
        void set_name(string nm) { name = nm; }
        string get_name() { 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~(); // returns transposed matrix
        const Mtrx& operator=(const Mtrx&);
        bool operator==(const Mtrx&);
        bool operator!=(const Mtrx&);
private:
        string name;
        int n_row;
        int n_col;
        double **dM;
};

#endif
```

(2) 클래스 멤버 함수 구현

- class Mtrx 선언은 "Mtrx.h" header 파일에 class Mtrx 의 멤버함수들은 Class_Mtrx.cpp 파일에 구현할 것.
- Mtrx(string nm, int n_row, int n_col) 생성자는 인수로 이름 (nm), 행렬의 크기 (n_row, n_col)을 전달받고, 데이터 멤버를 초기화 하며, 주어진 크기의 2 차원 배열을 동적으로 생성하여 그 주소를 이중 포인터 dM 의 값으로 설정.
- Mtrx(istream& fin) 생성자는 주어진 입력 데이터 파일 객체로부터 행렬의 크기 (n_row, n_col)와 데이터를 입력 받아, 주어진 행렬의 크기에 맞는 2 차원 double 형 배열을 동적으로 생성.
- init(int n_row, int n_col) 멤버함수는 주어진 행렬의 크기에 따라 double 자료형의 n_row x n_col 크기 2 차원 배열을 동적으로 생성하고, 초기 값으로 각 원소값을 0.0 으로 설정.
- class Mtrx 에 friend 로 선언되어 있는 operator>>() 함수는 각각 지정된 입력 파일로 행렬의 크기 (n_row, n_col)를 읽고, 동적으로 2 차원 배열을 생성하며, 원소들을 차례로 읽어 행렬의 초기값으로 설정.

- class Mtrx 에 friend 로 선언되어 있는 operator<<() 함수는 행렬의 이름과 원소들을 지정된 파일로 출력. 행렬의 출력에서는 확장완성형 코드를 사용하여 행렬 모습인 대괄호 ([,])가 표시되도록 하며, 행렬의 이름을 함께 출력할 것.
- operator+() 연산자 오버로딩 멤버 함수는 행렬의 덧셈을 계산. 기준이 되는 객체의 행렬과 인수로 주어진 행렬의 합을 계산하여 반환.
- operator-() 연산자 오버로딩 멤버 함수는 행렬의 뺄셈을 계산. 기준이 되는 객체의 행렬과 인수로 주어진 행렬의 차을 계산하여 반환.
- operator*() 연산자 오버로딩 멤버 함수는 행렬의 곱셈을 계산. 기준이 되는 객체의 행렬과 인수로 주어진 행렬의 곱을 계산하여 반환.
- operator~() 연산자 오버로딩 멤버 함수는 치환 행렬을 생성하여 반환.
- operator=() 연산자 오버로딩 멤버 함수는 대입 (assign) 연산을 구현
- operator==()과 operator!=() 연산자 오버로딩 멤버 함수는 equality 와 not-equality 연산을 구현

4.2 Class MtrxArray

(1) class MtrxArray 정의

```
/** MtrxArray.h */
#ifndef MTRXARRAY_H
#define MTRXARRAY_H

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

using namespace std;

class Mtrx;

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

(2) class MtrxArray 멤버 함수 구현

- MtrxArray(int arraySize)는 class Mtrx 객체를 arraySize 개수 만큼 관리할 수 있는 배열을 생성. 생성된 배열의 원소는 행렬이 크기 (n_row, n_col)이 설정되어 있는 않은 상태이며, 따라서 초기화되어 있지 않은 상태임
- operator[] 연산자 오버로딩 함수는 행렬의 배열을 인덱스를 사용하여 접근 및 설정할 수 있게 하며, 인덱스가 행렬의 배열 크기 범위 이내에 있는가를 isValidIndex(int index) 멤버 함수를 사용하여 확인
- getSize() 멤버함수는 mtrxArraySize 값을 반환하여 행렬의 배열 크기를 알려줌

4.3 main() 함수

1) 파일 입력, class Mtrx 객체의 배열 생성

- class Mtrx 의 객체 7 개를 배열로 구성하도록 Mtrx mtrx[NUM_MTRX]를 생성
- 입력파일 ("Matrix_data.txt")로부터 3 개의 행렬에 대한 크기 (size_m, size_n)와 행렬의 원소 데이터를 파일로부터 ">>" 연산자를 사용하여 각각 입력 받아 행렬 mtrx[0], mtrx[1], mtrx[2]를 초기화
- 생성된 class Mtrx 배열의 행렬 mtrx[0], mtrx[1], mtrx[2]를 출력하여 정확하게 생성되었는지 확인

2) class Mtrx 멤버 함수 실행 및 행렬 계산 결과 출력

- mtrx[3]은 mtrx[0]과 mtrx[1]의 행렬 덧셈을 + 연산자 오버로딩을 사용하여 계산하고, 결과를 저장한 후, class Mtrx 의 << 연산자 오버로딩 기능을 사용하여 출력
- mtrx[4]는 mtrx[0]과 mtrx[1]의 행렬 뺄셈을 - 연산자 오버로딩을 사용하여 계산하고, 결과를 저장한 후, class Mtrx 의 << 연산자 오버로딩 기능을 사용하여 출력.
- mtrx[5]는 mtrx[0]와 mtrx[2]의 행렬 곱셈을 * 연산자 오버로딩을 사용하여 계산하고, 결과를 저장한 후, class Mtrx 의 << 연산자 오버로딩 기능을 사용하여 출력.
- mtrx[6]은 mtrx[5]의 전치 행렬을 ~연산자 오버로딩을 사용하여 생성하여 대입시키고, 결과를 저장한 후, class Mtrx 의 << 연산자 오버로딩 기능을 사용하여 출력.
- mtrx[5]와 mtrx[6]가 동일한가를 "==" 연산자와 "!=" 연산자를 사용하여 판단하고, 그 결과를 출력.

```
/** main.cpp */

#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;
        }

        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;
}
```

4.4 입력 데이터 파일과 실행 결과

```
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
```

```
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.
```

<2021-2 객체지향형 프로그래밍과 자료구조 Lab 4 Oral Test>

학번		성명		점수	

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

(2) C++ 클래스인 class Mtrx 의 friend 함수로 operator<<()를 구현에서 call-by-reference , return-by-reference, const 를 사용하는 이유에 대하여 예를 들어 설명하라. (핵심 포인트: C++ 클래스에 대한 operator<<() 함수의 구현에서 call-by-reference , return-by-reference, const 를 사용 이유, chained operation 이 가능하도록 하기 위한 구조)

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

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