**Lab 4**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 객체지향프로그래밍과자료구조 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.09.24.금 |

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* Mtrx 객체를 배열로 생성  \* 연산자 오버로딩으로 연산  \* 행렬 특성 출력  \* Programmed by J. H. Kim  \* Last updated : 2021-09-24 \*/  #include <iostream>  #include <fstream>  #include <string>  #include "Class\_Mtrx.h"  #include "MtrxArray.h"  using namespace std;  #define NUM\_MTRX 8  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;  // 덧셈  mtrx[3] = mtrx[0] + mtrx[1];  mtrx[3].set\_name("mtrx[3] = mtrx[0] + mtrx[1] =");  fout << mtrx[3] << endl;  // 뺄셈  mtrx[4] = mtrx[0] - mtrx[1];  mtrx[4].set\_name("mtrx[4] = mtrx[0] - mtrx[1] =");  fout << mtrx[4] << endl;  // 곱셈  mtrx[5] = mtrx[0] \* mtrx[2];  mtrx[5].set\_name("mtrx[5] = mtrx[0] \* mtrx[2] =");  fout << mtrx[5] << endl;  // 전치  mtrx[6] = ~mtrx[5];  mtrx[6].set\_name("mtrx[6] = ~mtrx[5] (transposed matrix) =");  fout << mtrx[6] << endl;  // 동일한 행렬인지 확인  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";  // 행렬 정보  mtrx[6].mtrxStats(fout);  mtrx[6].set\_name("Add status");  fout << mtrx[6] << endl;  fin.close();  fout.close();  return 0;  } |
| /\* 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&);  void mtrxStats(ostream&);  private:  string name;  int n\_row;  int n\_col;  double\*\* dM;  };  #endif |
| /\* Class\_Mtrx.cpp \*/  #include <iomanip>  //#include <stdbool.h>  #include "Class\_Mtrx.h"  using namespace std;  #define SETW 10  typedef double\* DBLPTR;  ostream& operator<<(ostream& fout, const Mtrx& m) {  unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;  unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;  fout << m.name << endl;  for (int i = 0; i < m.n\_row; i++) {  for (int j = 0; j < m.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 == (m.n\_col - 1)))  fout << setw(SETW) << m.dM[i][j] << a6 << a4;  else if ((i > 0) && (i < (m.n\_row - 1)) && (j == 0))  fout << a6 << a2 << setw(SETW) << m.dM[i][j];  else if ((i > 0) && (i < (m.n\_row - 1)) && (j == (m.n\_col - 1)))  fout << setw(SETW) << m.dM[i][j] << a6 << a2;  else if ((i == (m.n\_row - 1)) && (j == 0))  fout << a6 << a6 << setw(SETW) << m.dM[i][j];  else if ((i == (m.n\_row - 1)) && (j == (m.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;  }  istream& operator>>(istream& fin, Mtrx& m) {  // 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;  m.n\_row = size\_row;  m.n\_col = size\_col;  m.dM = new DBLPTR[m.n\_row];  for (i = 0; i < m.n\_row; i++)  m.dM[i] = new double[m.n\_col];  for (i = 0; i < m.n\_row; i++) {  for (j = 0; j < m.n\_col; j++) {  if (fin.eof())  m.dM[i][j] = 0.0;  else {  fin >> d;  m.dM[i][j] = d;  }  }  }  //cout <<"End of Mtrx constructor... \n";  return fin;  }  Mtrx::Mtrx() {  name = "mR";  n\_col = 0;  n\_row = 0;  dM = new DBLPTR[1];  }  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";  }  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";  }  Mtrx::~Mtrx() {  // cout << "destructor of Mtrx ("  // << name << ")" << endl;  /\*  for (int i=0; i<n\_row; i++)  delete [] dM[i];  delete [] dM;  \*/  }  void Mtrx::init(int n\_row, int n\_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";  }  const Mtrx Mtrx::operator+(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;  }  const Mtrx Mtrx::operator-(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;  }  const Mtrx Mtrx::operator\*(const Mtrx& mA) {  int i, j, k;  Mtrx mR("mR", n\_row, n\_row);  for (i = 0; i < n\_row; i++) {  for (j = 0; j < 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;  }  const Mtrx Mtrx::operator~() {  int i, j;  Mtrx mR("mR", n\_col, n\_row);  for (i = 0; i < this->n\_row; i++) {  for (j = 0; j < this->n\_col; j++) {  mR.dM[j][i] = this->dM[i][j];  }  }  return mR;  }  const Mtrx& Mtrx::operator=(const Mtrx& mA) {  int i, j;  // Mtrx mR("mR", mA.n\_row, mA.n\_col);  this->n\_row = mA.n\_row;  this->n\_col = mA.n\_col;  this->init(this->n\_row, this->n\_col);  for (i = 0; i < mA.n\_row; i++) {  for (j = 0; j < mA.n\_col; j++)  this->dM[i][j] = mA.dM[i][j];  }  return \*this;  }  bool Mtrx::operator==(const Mtrx& mA) {  int i, j;  bool flag = true;  Mtrx mR("mR", n\_row, n\_col);  for (i = 0; i < n\_row; i++) {  for (j = 0; j < n\_col; j++) {  if (mR.dM[i][j] != mA.dM[i][j])  flag = false;  }  }  return flag;  }  bool Mtrx::operator!=(const Mtrx& mA) {  int i, j;  bool flag = true;  Mtrx mR("mR", n\_row, n\_col);  for (i = 0; i < n\_row; i++) {  for (j = 0; j < n\_col; j++) {  if (mR.dM[i][j] != mA.dM[i][j])  flag = false;  }  }  return !flag;  }  void Mtrx::mtrxStats(ostream& fout) {  int i, j, k;  double mean = 0.0;  double sum = 0.0;  double medi = 0.0;  double sq\_sum\_avg = 0.0;  double diff\_sq\_sum = 0.0;  double var, std\_dev;  int num\_elements = n\_row \* n\_col;  int element;  Mtrx mR("Mtrx status", n\_row + 1, n\_col + 1);  for (i = 0; i < this->n\_row; i++) {  for (j = 0; j < this->n\_col; j++) {  mR.dM[i][j] = this->dM[i][j];  mR.dM[i][n\_col] += this->dM[i][j];  mR.dM[n\_row][n\_col] += this->dM[i][j];  diff\_sq\_sum +=  (this->dM[i][j] - mean) \* (this->dM[i][j] - mean);  }  }  for (j = 0; j < n\_col; j++) {  for (i = 0; i < n\_row; i++) {  mR.dM[n\_row][j] += this->dM[i][j];  }  }  sum = mR.dM[n\_row][n\_col];  mean = sum / (double)num\_elements;    var = diff\_sq\_sum / (double)num\_elements;  std\_dev = sqrt(var);  \*this = mR;  fout << "Mtrx status" << endl;  fout << "avg : " << mean << ", var : " << var << ", st\_dev : " << std\_dev << endl;  } |
| /\* MtrxArray.h \*/  #ifndef MTRXARRAY\_H  #define MTRXARRAY\_H  #include <iostream>  #include "Class\_Mtrx.h"  using namespace std;  class Mtrx;  class MtrxArray {  public:  MtrxArray(int arraySize); // constructor  ~MtrxArray(); // destructor  Mtrx& operator[](int);  int getSize() { return mtrxArraySize; }  private:  Mtrx\* pMtrx;  int mtrxArraySize;  bool isValidIndex(int index);  };  #endif |
| /\* MtrxArray.cpp \*/  #include "MtrxArray.h"  #include "Class\_Mtrx.h"  MtrxArray::MtrxArray(int arraySize) { // constructor  mtrxArraySize = arraySize;  pMtrx = new Mtrx[arraySize];  }  MtrxArray::~MtrxArray(){  //cout << "MtrxArray :: destructor" << endl;  if (pMtrx != NULL)  delete[] pMtrx;  }  void subError() {  cout << "ERROR: Subscript out of range.₩n";  exit(0);  }  bool MtrxArray::isValidIndex(int index) {  if (index < 0 || index >= mtrxArraySize)  return false;  else  return true;  }  Mtrx& MtrxArray::operator [](int sub) {  if (isValidIndex(sub))  return pMtrx[sub];  else  subError();  } |
|  |
|  |

**2. 2021-2 객체지향형 프로그래밍과 자료구조 실습 Oral Test**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 학번 | 21812158 | 성명 | 김주환 | 점수 |  |

|  |
| --- |
| (1) C++ 프로그래밍에서 연산자 오버로딩 (operator overloading)의 필요성에 대하여 각각 예 를 들어 설명하라.  간략히 표현하고 쉽게 이해할 수 있다.  수학 공식에서 사용하는 연산자를 사용함으로써 간단하고 이해하기 쉽게 프로그램 소스코드를 구현할 수 있다.  이를 통해 사용자의 편의성을 증대할 수 있다. |
| (2) C++ 클래스인 class Mtrx 의 friend 함수로 operator<<()를 구현에서 call-by-reference, return-by-reference, const를 사용하는 이유에 대하여 예를 들어 설명하라.   * call-by-reference & return-by-reference 사용 이유 참조가 반환된 경우 그 참조의 값이 저장된 메모리 공간이 존재하여야하고, 참조로 반환값을 전달한 함수에서 그 반환값이 저장되어 있는 메모리 공간을 유지해야 하기 때문에 레퍼런스를 사용한다. 전달 인수와 반환형을 통일시켜주지 않으면, << 연산이 연속으로 일어날 경우 Chain이 끊긴다. * const사용 이유 연산에 사용되는 원본 데이터가 손상되어 연산에 차질이 생기지 않도록 하기 위해 사용한다. * 결론 Chained operation을 위해 사용한다. |
| (3) C++ 클래스인 class Mtrx의 멤버함수로 덧셈 계산을 위한 ‘+’ 연산자의 operator overloading 함수를 구현하는 방법에 대하여 설명하라.   * 전달 인수 피 연산자(const Mtrx& mA) * 내부 실행 절차 연산 값을 전달할 객체를 생성한다. 이중 반복을 통해 연산을 진행한다. * 결과 반환 Chained operation을 위해Mtrx를 반환한다. 값을 반환하면 한 줄에 Chaining이 가능하다.  Mtrx 가 아닌 Mtrx 레퍼런스를 반환하면, Chained operation 도중에 문제가 있는 식에 대해 예외처리가 불가능하다. 예를 들어, (a + b) = C; 가 있다. |
| (4) C++ 클래스인 class Mtrx 의 멤버함수로 대입연산을 위한 ‘=’ 연산자의 operator overloading 함수를 구현하는 방법에 대하여 설명하라.   * 전달 인수 피연산자(const Mtrx& mA) * 내부 실행 절차 전달받을 객체 정보를 초기화 한다. 이중 반복을 통해 값을 복사한다. * 결과 반환 Chained operation을 위해 Mtrx 레퍼런스를 반환한다. 레퍼런스를 반환하면 한 줄에 Chaining이 가능하다. |