**Homework 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 | 2021.09.29.수 |

1. operator overloading for Class complex number

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* class를 이용한 복소수 계산  \* Programmed by J. H. Kim  \* Last updated : 2021-09-29\*/  #include <iostream>  #include <fstream>  #include "CmplxArray.h"  #include "Cmplx.h"  using namespace std;  void main()  {  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];  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();  } |
| /\*\* 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 |
| /\*\* 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 << c.real << " ";  if (c.imag >= 0.0) output << "+ " << c.imag << "j" << endl;  else output << "- " << -1 \* c.imag << "j" << endl;  return output;  }  istream& operator>>(istream& input, Cmplx& c) {  input >> c.real;  input >> c.imag;  return input;  }  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;  }  const Cmplx Cmplx::operator\*(const Cmplx& c) {  Cmplx result;  result.real = real \* c.real - imag \* c.imag; result.imag = real \* c.imag + c.real \* imag;  return result;  }  const Cmplx Cmplx::operator/(const Cmplx& c) {  Cmplx result;  double temp;  temp = c.real \* c.real + c.imag \* c.imag;  result.real = (real \* c.real + imag \* c.imag) / temp; result.imag = (imag \* c.real - real \* c.imag) / temp;  return result;  }  const Cmplx Cmplx::operator~() {  Cmplx result;  result.real = real; result.imag = -imag;  return result;  }  bool Cmplx::operator==(const Cmplx& c) {  if ((real == c.real) && (imag == c.imag))  return true;  else  return false;  }  bool Cmplx::operator!=(const Cmplx& c) {  if ((real == c.real) && (imag == c.imag))  return false;  else  return true;  }  bool Cmplx::operator<(const Cmplx& c) {  double mag1, mag2;    mag1 = sqrt(real \* real + imag \* imag);  mag2 = sqrt(c.real \* c.real + c.imag \* c.imag);  if (mag1 < mag2)  return true;  else  return false;  }  bool Cmplx::operator>(const Cmplx& c) {  double mag1, mag2;  mag1 = sqrt(real \* real + imag \* imag);  mag2 = sqrt(c.real \* c.real + c.imag \* c.imag);  if (mag1 < mag2)  return false;  else  return true;  }  const Cmplx Cmplx::operator=(const Cmplx& c) {  this->real = c.real; this->imag = c.imag;  return \*this;  } |
| /\*\* 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);  ~CmplxArray();  int size() { return cmplxArrySize; }  Cmplx &operator[](int);  void print(ostream& fout);  void sort();  private:  Cmplx\* pCA;  int cmplxArrySize;  bool isValidIndex(int indx);  };  #endif |
| /\*\* 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;  }  bool CmplxArray::isValidIndex(int indx)  {  if (indx < 0 || indx >= cmplxArrySize)  {  cout << "ERROR: the given index is out of range.₩n";  exit(0);  }  else  return true;  }  Cmplx& CmplxArray::operator [](int indx)  {  if (isValidIndex(indx))  return pCA[indx];  }  void CmplxArray::print(ostream& fout)  {  for (int i = 0; i < cmplxArrySize; i++)  {  fout << pCA[i] << endl;  }  } |
|  |
|  |

2. Operator overloading for Class Mtrx

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* 행렬 생성및 계산  \* Programmed by J. H. Kim  \* Last updated : 2021-09-29\*/  #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." << endl;  if (mtrx[5] != mtrx[6]) fout << "mtrx[5] and mtrx[6] are not equal." << endl;  fin.close();  fout.close();  return 0;  } |
| /\* Mtrx.h \*/  #ifndef M  #define M  #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;  };  #endif |
| /\* Mtrx.cpp \*/  #include <iostream>  #include <iomanip>  //#include <stdbool.h>  #include "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, double\* pA, 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() {  // 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", \*dM, 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", \*dM, 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", \*dM, 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];  // cout << mR.dM[i][j] << endl;  }  // cout << endl;  }  }  return mR;  }  const Mtrx Mtrx::operator~() {  int i, j;  Mtrx mR("mR", \*dM, 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", \*dM, 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", \*dM, 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;  } |
| /\*\* 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 mtrxArraySize;  bool isValidIndex(int index);  };  #endif |
| /\* MtrxArray.cpp \*/  #include "MtrxArray.h"  #include "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();  } |
|  |
|  |