**Lab 3**

|  |
| --- |
| 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 | 2021.09.17.금 |

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* 2차원 배열로 행렬을 구현  \* 기본연산과 전치행렬 연산  \* Programmed by J. H. Kim  \* Last updated : 2021-09-17 \*/  #include <iostream>  #include <fstream>  #include "Class\_Mtrx.h"  using namespace std;  void main()  {  ifstream fin;  ofstream fout;  fin.open("Matrix\_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;  }  // 기본 행렬 데이터 불러오기  Mtrx mtrxA(fin);  mtrxA.setName("MtrxA");  int n\_row = mtrxA.getN\_row();  int n\_col = mtrxA.getN\_col();  mtrxA.fprintMtrx(fout);  Mtrx mtrxB(fin);  mtrxB.setName("MtrxB");  mtrxB.fprintMtrx(fout);  Mtrx mtrxC(fin);  mtrxC.setName("MtrxC");  mtrxC.fprintMtrx(fout);  // 행렬 덧셈  Mtrx mtrxD(mtrxA.getN\_row(), mtrxB.getN\_col());  mtrxD = mtrxA.add(mtrxB);  mtrxD.setName("MtrxD = mtrxA.add(mtrxB)");  mtrxD.fprintMtrx(fout);  // 행렬 뺄셈  Mtrx mtrxE(mtrxA.getN\_row(), mtrxB.getN\_col());  mtrxE = mtrxA.sub(mtrxB);  mtrxE.setName("MtrxE = mtrxA.sub(mtrxB)");  mtrxE.fprintMtrx(fout);  // 행렬 곱셈  Mtrx mtrxF(mtrxA.getN\_row(), mtrxC.getN\_col());  mtrxF = mtrxA.multiply(mtrxC);  mtrxF.setName("MtrxF = mtrxA.multiply(mtrxC)");  mtrxF.fprintMtrx(fout);  // 전치 행렬 생성  Mtrx mtrxT(mtrxF.getN\_col(), mtrxF.getN\_row());  mtrxT = mtrxT.Transpos(mtrxF);  mtrxT.setName("MtrxT = mtrxT.Transpos(mtrxF)");  mtrxT.fprintMtrx(fout);  fout.close();  } // end of main() |
| /\* Class\_Mtrx.h \*/  #ifndef MTRX\_H  #define MTRX\_H  #include <iostream>  #include <fstream>  using namespace std;  #define MAX\_SIZE 100  class Mtrx {  public:  Mtrx(int num\_row, int num\_col); // 클래스 생성자  Mtrx(double\* dA, int num\_data, int num\_row, int num\_col); // 생성자  Mtrx(istream& fin); // 생성자  ~Mtrx(); // destructor  int getN\_row() { return n\_row; } // 열 정보 반환  int getN\_col() { return n\_col; } // 행 정보 반환  void fprintMtrx(ostream& fout); // 포멧에 맞춰 출력  void setName(string nm) { name = nm; }; // 이름 설정  string getName() { return name; }; // 이름 반환  Mtrx add(const Mtrx&); // 덧셈 연산  Mtrx sub(const Mtrx&); // 뺄셈 연산  Mtrx multiply(const Mtrx&); // 곱셈 연산  Mtrx Transpos(const Mtrx&); // 전치행렬 연산  private:  string name;  int n\_row;  int n\_col;  double\*\* dM;  };  #endif |
| /\* Class\_Mtrx.cpp \*/  #include "Class\_Mtrx.h"  #include <iostream>  #include <iomanip>  using namespace std;  typedef double\* DBLPTR;  Mtrx::Mtrx(int num\_row, int num\_col) {  int i, j;  //cout <<"Mtrx constructor (int size: "  // << size << ")₩n";  n\_row = num\_row;  n\_col = num\_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++) {  dM[i][j] = 0.0;  }  }  // cout <<"End of Mtrx constructor...₩n";  }  Mtrx::~Mtrx() {  // cout << "destructor of Mtrx ("  // << name << ")" << endl;  // 주석처리하지 않으면 중간에 소멸자를 호출하여 기능을 수행하지 못함  /\*  for (int i=0; i<n\_row; i++)  delete [] dM[i];  delete [] dM;  \*/  }  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";  }  #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 << " =\n";  for (int i = 0; i < n\_row; i++) {  for (int j = 0; j < n\_col; j++) {  fout.setf(ios::fixed);  fout.precision(1);  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;  }  Mtrx Mtrx::add(const Mtrx& mA) {  int i, j;  Mtrx mR(n\_row, n\_col);  mR.setName("R");  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;  }  Mtrx Mtrx::sub(const Mtrx& mA) {  int i, j;  Mtrx mR(n\_row, n\_col);  mR.setName("R");  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;  }  Mtrx Mtrx::multiply(const Mtrx& mA) {  int i, j, k;  Mtrx mR(n\_row, mA.n\_col);  mR.setName("R");  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;  }  Mtrx Mtrx::Transpos(const Mtrx& mF) {  Mtrx mR(this->n\_row, this->n\_col);  for (int i = 0; i < this->n\_row; i++) {  for (int j = 0; j < this->n\_col; j++) {  mR.dM[j][i] = mF.dM[i][j];  }  }  return mR;  } |
| 텍스트, 전자기기, 키보드이(가) 표시된 사진  자동 생성된 설명 - Matrix\_data.txt |

텍스트, 전자기기, 계산기, 키보드이(가) 표시된 사진

자동 생성된 설명

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

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

|  |
| --- |
| **(1) C/C++ 프로그래밍에서 사용되는 포인터 (pointer)의 사용 용도에 대하여 설명하라. (핵심포인트: 함수의 호출과 반환에서 사용, 동적 메모리 할당, 자기참조 구조체/클래스)**   * 개요 포인터란 값으로 주소를 가지는 변수 포인터는 자료형이 지정되며, 포인터의 자료형에 따라 가리키는 내용을 다르게 해석함 주로 함수의 호출과 반환, 동적 메모리 할당, 자기참조 구조체/클래스에 사용된다. * 함수의 호출과 반환 함수 호출 시 인수 값을 복사하지 않고, 인수가 저장된 곳의 주소를 전달하므로 인수 내용의 복사에 걸리는 시간 부담이 없으며, 메모리 공간 사용도 효율적임 호출된 함수에서는 포인터를 사용하여 간접참조로 인수 내용을 직접 변경할 수 있음      * 동적 메모리 할당 * 자기참조 구조체/클래스 내부 데이터 멤버 중에 자기 자신과 동일한 구조의 구조체/클래스 객체를 가리키는 포인터가 포함되어 있는 것 예) 연결형 리스트 노드, 이진 트리노드, 스킵 리스트의 쿼드노드 |
| **(2) C/C++ 프로그래밍에서 사용되는 포인터에 +/- 1의 덧셈/뺄셈 연산이 어떤 의미를 가지는가에 대하여 예를 들어 설명하라. (핵심포인트: 포인터의 자료형 (예: char, int, double, struct Student, class Person)에 따라 포인터가 가리키는 메모리 블록의 크기가 달라짐)**   * 개요 포인터의 연산은 기본 연산과 다른 형식으로 동작한다. * 덧셈 그 포인터의 자료형 크기만큼 앞으로 이동 * 뺄셈 그 포인터의 자료형 크기만큼 뒤로 이동 |
| **(3) C++ 클래스의 데이터 멤버로 2차원 동적 배열이 사용되어야 하는 경우, 생성자에서 생성하는 방법과 소멸자에서 2차원 동적 배열을 어떻게 삭제하는 방법에 대하여 각각 예를 들어 설명하라. (핵심 포인트: C++ 클래스 생성자에서 2차원 배열의 동적 생성과 삭제)**   * 생성 double 포인터의 동적 배열을 지정된 크기만큼 생성해준다. 그리고 각 포인터마다 다시 double 자료형의 동적 배열을 지정된 크기만큼 생성해준다. 마지막으로, 원하는 값으로 배열의 값을 초기화한다.      * 소멸 생성 순서의 반대로 한다. 최초 생성한 double 포인터 자료형 동적 배열의 각 인덱스에 할당된 동적 배열을 모두 delete 한다. 이후 남은 최초 생성한 double 포인터 자료형 동적 배열을 delete 한다. |
| **(4) Windows 환경에서 실행 중인 프로그램 (process)의 가상 메모리 맵에 설명하라. (핵심 포인트: 가상 메모리 멥의 각 구간이 어떤 용도로 사용되는가, 스택과 힙에서의 메모리 할당에 따른 주소 변화에 대한 설명)**   * 개요 프로그램 실행 중에 시스템 내부에서 관리되는 메모리를 사용함 이 프로그램이 프로세스다. 이 프로그램의 가상 메모리 맵의 구성을 알아볼 필요가 있다. 이를 통해 실제 메모리 할당 장소, 포인터가 가르키는 곳의 메모리 영역이 어떤 곳인지 정확히 이해 가능하다. * 각 구간의 용도 프로그램 실행 시 전체 4GB의 메모리를 할당한다. 이는 운영체제나 프로그래밍 환경에 따라 다르다.(0x0000 0000 ~ 0xFFFF FFFF) Kernel Land : 상부 2GB는 운영체제가 사용하는 공간이다. 사용자 프로그램에서는 직접 접근할 수 없도록 보호된다.  그 아래는 실제 프로그램 실행 시 생성되거나 사용할 수 있는 부분이다.  Stack : 함수 실행 시 함수에서 사용하는 지역 변수나 함수 호출 시 전달되는 인수들이 저장되는 곳이다. 여기서는 함수가 실행될 때마다 Stack Frame이 생성되고 거기서 지역변수와 인수가 저장된다. Heap : 동적 메모리 할당 시 사용하는 부분이다.  Program Image : 실제 작성한 프로그램 실행 코드다.  그 위는 라이브러리나 스레드 관련된 부분이다. DLL : 프로그램 실행 시 include한 메모리와 시스템 함수가 필요할 경우 장착되어 사용될 수 있도록 하는 것이다.  PEB : 프로그램 실행 시 메모리가 사용되면서, 여러가지 상태 정보를 관리하는데, 이 정보를 여기 저장한다. TEB : 프로세스 안에서 여러 스레드를 생성해서 병렬 처리를 할 경우 스레드가 관리 되어야한다. 이 경우 스레드 관련 관리 정보를 여기 저장한다. * 스택과 힙에서의 메모리 할당에 따른 주소변화 Stack은 주소가 작아지는쪽으로 자라나고 Heap은 주소가 커지는 쪽으로 자라난다. |