**Lab 08**

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

|  |  |
| --- | --- |
| 과목명 | 프로그래밍 언어 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.04.30.금 |

**8.1 2차원 배열의 동적 생성, 파일 입출력**

**8.1.1 2 차원 배열의 파일 입력, 행렬의 화면/파일 출력**

1) 입력 데이터 파일 (예: mtrx\_input.txt)로부터 지정된 행렬의 크기 (size\_row와 size\_col)에 따라 size\_row x size\_col 개의 double 데이터를 fscanf() 함수를 사용하여 읽고, 2차원 배열을 동적으로 생성한 후, 동적 생성된 2차원 배열에 저장하는 함수 double \*\* fGetMtrx(FILE \*fin, int \*row\_size, int \*col\_size)를 작성하라.

// 원하는 크기의 2차원 배열 동적 할당

double\*\* createDynamicDoubleMatrix(int row\_size, int col\_size) {

double\*\* ppdm = NULL;

ppdm = (double\*\*)calloc(row\_size, sizeof(double\*));

for (int i = 0; i < row\_size; i++) {

ppdm[i] = (double\*)calloc(col\_size, sizeof(double));

}

return ppdm;

}

// 파일로부터 원하는 크기의 행렬 데이터 불러와서 저장

double\*\* fGetMtrx(FILE\* fin, int\* row\_size, int\* col\_size) {

double d = 0.0;

fscanf(fin, "%d %d", row\_size, col\_size);

double\*\* dm = createDynamicDoubleMatrix(\*row\_size, \*col\_size);

for (int i = 0; i < \*row\_size; i++) {

for (int j = 0; j < \*col\_size; j++) {

if (fscanf(fin, "%lf", &d) != EOF) dm[i][j] = d;

}

}

return dm;

}

2) fGetMtrx() 함수의 실행 결과로 동적으로 생성된 행렬과 그 행렬의 크기 (size\_row, size\_column)이 return-by-pointer 방식으로 반환되어야 한다. fGetMtrx() 함수에서는 지정된 입력 데이터 파일(예: Array\_2D\_input.txt)로부터 fscanf() 함수를 사용하여, 각size\_row x size\_column개의 실수형(double)데이터를 읽어 2차원 배열을 초기화하는 기능이 포함된다.

3) 동적으로 생성된 2차원 배열을 삭제하고, 메모리를 반환하는 함수 void deleteDynMtrx(double\*\* dM, int row\_size, int col\_size)를 작성하라.

// 생성한 2차원 동적 배열 해제

void deleteDynMtrx(double\*\* dm, int row\_size, int col\_size) {

for (int i = 0; i < row\_size; i++) free(dm[i]);

free(dm);

}

**8.1.2 2 차원 배열의 파일 입력, 행렬의 화면/파일 출력, 행렬 원소의 통계 분석**

1) 2차원 배열의 크기 (size\_row와 size\_col)에 따라 화면과 파일로 출력하는 함수 void printMtrx(double \*\*dM, int size\_row, int size\_col)와 void fprintMtrx(FILE \*fout, double \*\*mA, int size\_row, int size\_col)을 작성하라.

// 행렬 데이터를 실행창에 출력

void printMtrx(double\*\* dm, int row\_size, int col\_size) {

unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;

unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

if ((i == 0) && (j == 0)) {

printf("%c%c%7.2lf", a6, a3, dm[i][j]);

}

else if ((i == 0) && j == (col\_size - 1)) {

printf("%7.2lf%c%c", dm[i][j], a6, a4);

}

else if ((i > 0) && (i < row\_size - 1) && (j == 0)) {

printf("%c%c%7.2lf", a6, a2, dm[i][j]);

}

else if ((i > 0) && (i < row\_size - 1) && (j == (col\_size - 1))) {

printf("%7.2lf%c%c", dm[i][j], a6, a2);

}

else if ((i == row\_size - 1) && (j == 0)) {

printf("%c%c%7.2lf", a6, a6, dm[i][j]);

}

else if ((i == row\_size - 1) && (j == (col\_size - 1))) {

printf("%7.2lf%c%c", dm[i][j], a6, a5);

}

else {

printf("%7.2lf", dm[i][j]);

}

if (col\_size == 1) {

if (i == 0)

printf("%c%c", a6, a4);

else if (i == row\_size - 1)

printf("%c%c", a6, a5);

else

printf("%c%c", a6, a2);

}

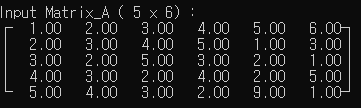
}

printf("\n");

}

}

2) 행렬의 각 값은 최소 8자리를 확보하고, 소수점 이하 2자리를 출력하며, 오른쪽 정렬로 출 력되도록 할 것. 이 때, 행렬을 표시하기 위하여, 확장 완성형 코드를 사용할 것.



**8.2 동적 2 차원 배열을 사용한 행렬의 계산**

**8.2.1 행렬 데이터의 파일 입력, 동적 생성**

1) 3개의 행렬의 데이터를 포함하는 입력 데이터 파일 (mtrx\_input.txt)로부터 차례로 행렬의 크기 (size\_row와 size\_col)와 행렬의 초기화 데이터를 입력받아 3개의 행렬의 파일 입력과 동적 생성은 double \*\* fGetMtrx(FILE \*fin, int \*row\_size, int \*col\_size) 함수를 사용할 것.

2) main() 함수에서는 파일로부터 입력 받아 생성된 3개의 행렬을 printMtrx() 함수를 사용하 여 출력하도록 하라.

**8.2.2 행렬 연산 (덧셈, 뺄셈, 곱셈)**

1) 2개의 size\_row x size\_col 크기의 행렬 A, B의 덧셈을 계산하여 그 결과를 동적으로 생성된 행렬 R에 저장하고, R의 주소를 반환하는 함수 double \*addMtrx(double \*\*mA, double 2 \*\*mB, int size\_row, int size\_col)를 작성하라. main() 함수에서는 행렬의 덧셈 결과를 출력하 도록 하라.

// 행렬 덧셈

double\*\* addMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int col\_size) {

double\*\* dm3 = createDynamicDoubleMatrix(row\_size, col\_size);

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

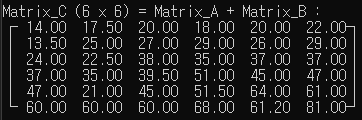
dm3[i][j] = dm1[i][j] + dm2[i][j];

}

}

return dm3;

}



2) 2개의 size\_row x size\_col 크기의 행렬 A, B의 뺄셈을 계산하여 그 결과를 동적으로 생성된 행렬 R에 저장하고, R의 주소를 반환하는 함수 double \*\*subtractMtrx(double \*\*mA, double \*\*mB, int size\_row, int size\_col)를 작성하라. main() 함수에서는 행렬의 뺄셈 결과를 출력하도록 하라.

// 행렬 뺄셈

double\*\* subtractMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int col\_size) {

double\*\* dm3 = createDynamicDoubleMatrix(row\_size, col\_size);

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

dm3[i][j] = dm1[i][j] - dm2[i][j];

}

}

return dm3;

}

화살이(가) 표시된 사진

자동 생성된 설명

3) size\_row x size\_K 크기의 행렬 A와 size\_K x size\_col 크기의 행렬 C의 곱셈을 계산하여 그 결과를 동적으로 생성된 size\_row x size\_col 크기의 행렬 R에 저장하고, R의 주소를 반환하는 함수 double \*\*multiplyMtrx(double \*\*mA, double \*\*mB, int size\_row, int size\_k, int size\_col)를 작성하라. main() 함수에서는 행렬의 곱셈 결과를 출력하도록 하라.

// 행렬 곱셈

double\*\* multiplyMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int temp, int col\_size) {

double\*\* dm3 = createDynamicDoubleMatrix(row\_size, col\_size);

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

for (int k = 0; k < temp; k++) {

dm3[i][j] += dm1[i][k] \* dm2[k][j];

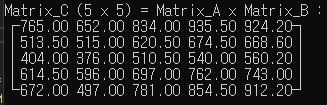
}

}

}

return dm3;

}



**8.3 첨가 행렬과 가우스 소거법 및 역행렬을 사용한 선형 방정식 해**

**8.3.1 Augmented Matrix의 준비 및 입력**

1) 주어진 직병렬 전자회로의 선형방정식 해법을 위한 첨가행렬 준비: 5개의 저항으로 구성된 직•병렬 전자회로에서 각 저항을 통하여 흐르는 전류의 양을 계산하는 선형 연립방정식을 정리하고, 그 계수 행렬의 정보 (선형방정식의 개수)와 계수 행렬의 각 원소 값을 입력데이터 파일 (augMtrx.txt)에 저장하라.

2) 입력파일 (augMtrx.txt) 파일을 열고, 이 파일에 포함되어 있는 첨가 행렬의 정보 (선형 방 정식의 개수 (N), 각 선형 방정식의 계수 (N + 1)에 해당하는 double type의 값 (총 N x (N + 1) 개)를 입력하여, 동적으로 생성한 N x (N + 1) 크기의 double type 2차원 배열 (double \*\*augMtrx)에 저장하는 함수 void getAugmentedMatrixData(FILE \*fin, double \*\*augMtrx, int size\_N)을 작성하라.

// 첨가행렬 데이터 파일로부터 불러오기

void getAugmentedMatrixData(FILE\* fin, double\*\* augMtrx, int size\_N) {

double d = 0.0;

if (fin == NULL) {

printf("Error in getDoubleMatrixData() - file pointer is NULL !!\n");

exit(-1);

}

for (int m = 0; m < size\_N; m++)

for (int n = 0; n <= size\_N; n++) {

if (fscanf(fin, "%lf", &d) != EOF)

augMtrx[m][n] = d;

}

}

**8.3.2 Gauss Elimination**

1) Gauss Elimination을 사용하여 선형 방정식 시스템의 해를 구하기 위하여, pivoting() 함수를 작 성하라.

// pivoting 과정 수행

void pivoting(double\*\* augMtrx, int size\_N, int p, int\* piv\_found) {

double xmax = 0.0, xtemp;

int j, k, max\_row;

xmax = fabs(augMtrx[p][p]); // 해당 위치 계수의 절대값 저장

max\_row = p; // 해당 위치 저장

for (j = p + 1; j < size\_N; j++) {

if (fabs(augMtrx[j][p]) > xmax) { // 저장한 절대값보다 큰 값이 존재할 경우

xmax = fabs(augMtrx[j][p]); // 더 큰 절대값을 저장

max\_row = j; // 해당 위치 저장

}

}

if (fabs(xmax) < EPSILON) // 값이 0이거나 0에 근접한 경우

\*piv\_found = 0; // 해가 없음

else { // 그 외

\*piv\_found = 1; // 해가 있음

if (max\_row != p) {

for (k = p; k <= size\_N; k++) { // swap

xtemp = augMtrx[p][k];

augMtrx[p][k] = augMtrx[max\_row][k];

augMtrx[max\_row][k] = xtemp;

}

}

}

}

2) Gauss Elimination을 사용하여 선형 방정식 시스템의 해를 구하기 위하여, diagonalize() 함수를 작성하라.

void diagonalize\_FileOut(FILE\* fout, double\*\* augMtrx, int size\_N, int\* solExist) {

int j, k, p = 0;

double pivWeight, w;

// Pivoting 그리고 Scaling

for (p = 0; (\*solExist) && (p < size\_N); p++) {

fprintf(fout, "\nPivoting at p = %d\n", p);

pivoting(augMtrx, size\_N, p, solExist);

fprintMtrx(fout, augMtrx, size\_N, size\_N + 1);

if (\*solExist) { // 해가 존재

// 해당 위치의 원소 scaling

if (augMtrx[p][p] != 1.0) { // 이미 scaling된 값이 아닌 경우

pivWeight = augMtrx[p][p];

augMtrx[p][p] = 1.0; // scaling

for (k = p + 1; k <= size\_N; k++) {

augMtrx[p][k] /= pivWeight; // 이후의 값도 나누기

}

}

}

else { // 해가 없음

break;

}

fprintf(fout, "\nAfter scaling at p = %d\n", p);

fprintMtrx(fout, augMtrx, size\_N, size\_N + 1);

// Diagonalize

for (j = 0; j < size\_N; j++) {

if ((j != p) && (augMtrx[j][p] != 0.0)) {

w = augMtrx[j][p];

augMtrx[j][p] = 0.0;

for (k = p + 1; k <= size\_N; k++) {

augMtrx[j][k] =

augMtrx[j][k] - w \* augMtrx[p][k];

}

}

}

fprintf(fout, "\nAfter diagonalizing at p = %d\n", p);

fprintMtrx(fout, augMtrx, size\_N, size\_N + 1);

}

}

3) 파일로부터 입력을 받아 구성하였던 augMtrx를 Gauss Elimination 방법을 사용하여 해를 구하고, 이를 출력하라.

텍스트이(가) 표시된 사진

자동 생성된 설명

**8.3.3 역행렬을 사용한 해법**

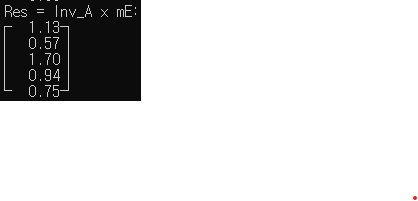
1) 위 8.3.1에서 주어진 선형연립방정식의 계수 행렬 A의 값을 double형 2차원 배열 mA에 저장하고, 이 mA의 역행렬을 계산하여 inv\_A에 저장하도록 하라.

2) 주어진 선형 연립 방정식을 AX = E로 정리하고, double 형 1차원 배열 E를 정의하라.

3) 주어진 선형 연립 방정식을 X = A-1B를 계산하는 방법을 사용하여 해를 구하라.

4) 이렇게 산출된 해를 위 8.3.2에서 구한 해와 비교하라.

텍스트이(가) 표시된 사진

자동 생성된 설명

// main.cpp(2)

// 주어진 행렬 데이터를 파일로부터 읽고 쓰는 작업을 수행한다.

void test\_2D\_DynamicArray\_FileIO(FILE\* fout) {

const char\* matrixDataFile = "mtrx\_input.txt";

FILE\* fin;

int a\_row\_size, a\_col\_size;

int b\_row\_size, b\_col\_size;

double\*\* mA, \*\* mB;

fin = fopen(matrixDataFile, "r"); // 파일 읽기모드로 연다.

if (fin == NULL) {

printf("Error in opening input.txt file !!\n");

exit(-1);

}

mA = fGetMtrx(fin, &a\_row\_size, &a\_col\_size);

printf("Input Matrix\_A ( %d x %d) : \n", a\_row\_size, a\_col\_size);

printMtrx(mA, a\_row\_size, a\_col\_size);

printf("\n");

mB = fGetMtrx(fin, &b\_row\_size, &b\_col\_size);

printf("Input Matrix\_B ( %d x %d) : \n", b\_row\_size, b\_col\_size);

printMtrx(mB, b\_row\_size, b\_col\_size);

printf("\n");

fclose(fin); // 파일 닫기

}

// 주어진 행렬의 덧셈, 뺄셈 연산을 수행한다.

void test\_MatrixAdditionSubtraction(FILE\* fout) {

const char\* matrixDataFile = "mtrx\_nxn\_input.txt";

FILE\* fin;

double\*\* mA, \*\* mB, \*\* mC, \*\* mD;

int a\_row\_size, a\_col\_size;

int b\_row\_size, b\_col\_size;

int c\_row\_size, c\_col\_size;

int d\_row\_size, d\_col\_size;

fin = fopen(matrixDataFile, "r"); // 파일 읽기모드로 연다.

if (fin == NULL) {

printf("Error in opening input.txt file !!\n");

exit(-1);

}

mA = fGetMtrx(fin, &a\_row\_size, &a\_col\_size); // 2차원 배열을 동적 할당하고 파일로부터 값을 불러온다.

printf("Input Matrix\_A ( %d x %d) : \n", a\_row\_size, a\_col\_size);

printMtrx(mA, a\_row\_size, a\_col\_size); // 불러온 행렬 데이터를 실행창에 출력한다.

printf("\n");

mB = fGetMtrx(fin, &b\_row\_size, &b\_col\_size);

printf("Input Matrix\_B ( %d x %d) : \n", b\_row\_size, b\_col\_size);

printMtrx(mB, b\_row\_size, b\_col\_size);

printf("\n");

if ((a\_row\_size != b\_row\_size) || (a\_col\_size != b\_col\_size)) {

printf("Error in input matrix dimension: row\_size and/or col\_size are not equal !!\n");

fclose(fin);

return;

}

// 덧셈

c\_row\_size = a\_row\_size;

c\_col\_size = b\_col\_size;

mC = addMatrix(mA, mB, a\_row\_size, a\_col\_size);

printf("Matrix\_C (%d x %d) = Matrix\_A + Matrix\_B : \n", c\_row\_size, c\_col\_size);

printMtrx(mC, c\_row\_size, c\_col\_size);

printf("\n");

// 뺄셈

d\_row\_size = a\_row\_size;

d\_col\_size = b\_col\_size;

mD = subtractMatrix(mA, mB, a\_row\_size, a\_col\_size);

printf("Matrix\_D (%d x %d) = Matrix\_A - Matrix\_B : \n", d\_row\_size, d\_col\_size);

printMtrx(mD, d\_row\_size, d\_col\_size);

printf("\n");

/\*

\* 파일명 : main.cpp

\* 목적 및 기본 기능

\* 2차원 동적 배열을 생성하고 파일의 행렬 데이터를 알맞게 입/출력 가능한지 확인하고

\* 주어진 행렬의 덧셈, 뺄셈, 곱셈 연산을 실행한다.

\* Gauss-Jordan 소거법을 수행하고 이를 이용하여 역행렬을 구하고 이를 검증한다.

\* 작성자 : 김주환(21812158)

\* 작성일 : 2021년 4월 30일

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include "Mtrx.h"

#include "GaussJordan\_LinearSystem.h"

const char\* outputFile = "OutputResult.txt";

void test\_2D\_DynamicArray\_FileIO(FILE\* fout);

void test\_MatrixAdditionSubtraction(FILE\* fout);

void test\_MatrixMultiplication(FILE\* fout);

void test\_GaussJordanElimination\_LinearSystem(FILE\* fout);

void test\_InvMtrx\_GaussJordanElimination\_LinearSystem(FILE\* fout);

int main(void) {

FILE\* fin, \* fout;

int menu;

fout = fopen(outputFile, "w");

if (fout == NULL) {

printf("Error in opening input.txt file !!\n");

exit(-1);

}

while (1) {

printf("Testing Matrix Operations with 2-Dimensional Dynamic Array\n");

printf(" 1: Test 2-D Dynamic Array Creation for Matrix with File I/O\n");

printf(" 2: Test Matrix Addition, Subtraction\n");

printf(" 3: Test Matrix Multiplication\n");

printf(" 4: Test Gauss-Jordan Elimination for Linear System\n");

printf(" 5: Test Inverse Matrix with Gauss-Jordan Elimination for Linear System\n");

printf(" 0: Quit\n");

printf("Input menu (0 to quit) : ");

scanf("%d", &menu);

if (menu == 0)

break;

printf("\n");

switch (menu) {

case 1:

test\_2D\_DynamicArray\_FileIO(fout);

break;

case 2:

test\_MatrixAdditionSubtraction(fout);

break;

case 3:

test\_MatrixMultiplication(fout);

break;

case 4:

test\_GaussJordanElimination\_LinearSystem(fout);

break;

case 5:

test\_InvMtrx\_GaussJordanElimination\_LinearSystem(fout);

break;

default:

break;

}

}

fclose(fout);

}

// main.cpp(3)

// 생성한 2차원 동적 배열을 해제

deleteDynMtrx(mA, a\_row\_size, a\_col\_size);

deleteDynMtrx(mB, b\_row\_size, b\_col\_size);

deleteDynMtrx(mC, c\_row\_size, c\_col\_size);

fclose(fin); // 파일 닫기

}

// Gauss-Jordan 소거법을 수행한다.

const char\* augMatrix\_inputFile = "augMtrx.txt";

void test\_GaussJordanElimination\_LinearSystem(FILE\* fout) {

FILE\* fin;

int size\_N, row\_size, col\_size;

double\* solution;

double\*\* augMtrx = NULL;

int i, j, solExist = 1, error = 0;

double d;

fin = fopen(augMatrix\_inputFile, "r");

if (fin == NULL) {

printf("Error in opening input.txt file (%s)!!\n", augMatrix\_inputFile);

exit(-1);

}

fscanf(fin, "%d", &size\_N);

augMtrx = createDynamicDoubleMatrix(size\_N, size\_N + 1); // 첨가행렬을 동적할당

solution = (double\*)calloc(size\_N, sizeof(double)); // 해의 배열을 동적 할당

fprintf(fout, "Augmented Matrix size\_N : %d\n", size\_N);

//fGetDoubleMatrixData(fin, augMtrx, size\_N, size\_N + 1);

getAugmentedMatrixData(fin, augMtrx, size\_N); // 첨가행렬 데이터를 파일로부터 불러온다.

fprintf(fout, "Augmented Matrix : \n");

fprintMtrx(fout, augMtrx, size\_N, size\_N + 1);

printf("Augmented Matrix : \n");

printMtrx(augMtrx, size\_N, size\_N + 1);

diagonalize\_FileOut(fout, (double\*\*)augMtrx, size\_N, &solExist); // 대각화 수행

if (solExist) { // 해가 존재할 경우

fprintf(fout, "The solution of the given linear system:\n");

printf("The solution of the given linear system:\n");

for (i = 0; i < size\_N; i++) {

solution[i] = augMtrx[i][size\_N]; // 해의 배열에 결과값을 대입

fprintf(fout, " x[%d] : %4f\n", i, solution[i]);

printf(" x[%d] : %4f\n", i, solution[i]);

}

}

else { // 해가 존재하지 않을 경우

fprintf(fout, "No unique solution\n");

printf("No unique solution\n");

}

// 동적 할당 해제

for (int i = 0; i < size\_N; i++)

free(augMtrx[i]);

free(augMtrx);

free(solution);

fclose(fin);

}

// main.cpp(4)

// 역행렬 구하는 과정 수행

// const char\* linearSystem\_inputFile = "ElectronicCircuit\_A\_5x5.txt";

const char\* linearSystem\_inputFile = "invMtrx.txt";

void test\_InvMtrx\_GaussJordanElimination\_LinearSystem(FILE\* fout) {

FILE\* fin;

double\*\* mA, \*\* inv\_A, \*\* res, \*\* mE;

int size\_N;

double data;

//fin = fopen("InputData.txt", "r");

fin = fopen(linearSystem\_inputFile, "r"); // 파일 읽기모드로 열기

//fin = fopen("ElectronicCircuit\_B\_5x5.txt", "r");

if (fin == NULL) {

printf("Error in openning %s input file !!\n", linearSystem\_inputFile);

exit(-1);

}

fscanf(fin, "%d", &size\_N); // 행렬 크기 불러오기

mA = (double\*\*)calloc(size\_N, sizeof(double\*));

inv\_A = (double\*\*)calloc(size\_N, sizeof(double\*));

mE = (double\*\*)calloc(size\_N, sizeof(double\*));

res = (double\*\*)calloc(size\_N, sizeof(double\*));

for (int i = 0; i < size\_N; i++) {

mA[i] = (double\*)calloc(size\_N, sizeof(double));

mE[i] = (double\*)calloc(size\_N, sizeof(double));

inv\_A[i] = (double\*)calloc(size\_N, sizeof(double));

res[i] = (double\*)calloc(size\_N, sizeof(double));

}

for (int r = 0; r < size\_N; r++)

for (int c = 0; c < size\_N; c++) {

fscanf(fin, "%lf", &data);

mA[r][c] = data;

}

for (int i = 0; i < size\_N; i++) {

fscanf(fin, "%lf", &data);

mE[i][0] = data;

}

fprintf(fout, "Matrix A: \n");

fprintMtrx(fout, mA, size\_N, size\_N);

printf("Matrix A: \n");

printMtrx(mA, size\_N, size\_N);

invMtrxGaussJordanElim\_FileOut(fout, mA, inv\_A, size\_N); // 역행렬 구하기

fprintf(fout, " Inverse A: \n");

fprintMtrx(fout, inv\_A, size\_N, size\_N);

printf(" Inverse A: \n");

printMtrx(inv\_A, size\_N, size\_N);

fprintf(fout, " mE: \n");

fprintMtrx(fout, mE, size\_N, 1);

printf(" mE: \n");

printMtrx(mE, size\_N, 1);

res = multiplyMatrix(inv\_A, mE, size\_N, size\_N, 1);

fprintf(fout, " Res = Inv\_A x mE: \n");

fprintMtrx(fout, res, size\_N, 1);

printf(" Res = Inv\_A x mE: \n");

printMtrx(res, size\_N, 1);

fprintf(fout, "\n");

// 역행렬이 맞는지 검증

mA = multiplyMatrix(inv\_A, mA, size\_N, size\_N, size\_N);

printf(" I = Inv\_A x mA: \n");

printMtrx(mA, size\_N, size\_N);

// 동적 할당 해제

for (int i = 0; i < size\_N; i++) {

free(mA[i]); free(mE[i]);

free(inv\_A[i]); free(res[i]);

}

free(mA); free(mE);

free(inv\_A); free(res);

fclose(fin);

}

// Mtrx.cpp(2)

// 행렬 데이터를 실행창에 출력

void printMtrx(double\*\* dm, int row\_size, int col\_size) {

unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;

unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

if ((i == 0) && (j == 0)) {

printf("%c%c%7.2lf", a6, a3, dm[i][j]);

}

else if ((i == 0) && j == (col\_size - 1)) {

printf("%7.2lf%c%c", dm[i][j], a6, a4);

}

else if ((i > 0) && (i < row\_size - 1) && (j == 0)) {

printf("%c%c%7.2lf", a6, a2, dm[i][j]);

}

else if ((i > 0) && (i < row\_size - 1) && (j == (col\_size - 1))) {

printf("%7.2lf%c%c", dm[i][j], a6, a2);

}

else if ((i == row\_size - 1) && (j == 0)) {

printf("%c%c%7.2lf", a6, a6, dm[i][j]);

}

else if ((i == row\_size - 1) && (j == (col\_size - 1))) {

printf("%7.2lf%c%c", dm[i][j], a6, a5);

}

else {

printf("%7.2lf", dm[i][j]);

}

if (col\_size == 1) {

if (i == 0)

printf("%c%c", a6, a4);

else if (i == row\_size - 1)

printf("%c%c", a6, a5);

else

printf("%c%c", a6, a2);

}

}

printf("\n");

}

}

// Mtrx.cpp

#ifndef M

#define M

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

// 원하는 크기의 2차원 배열 동적 할당

double\*\* createDynamicDoubleMatrix(int row\_size, int col\_size) {

double\*\* ppdm = NULL;

ppdm = (double\*\*)calloc(row\_size, sizeof(double\*));

for (int i = 0; i < row\_size; i++) {

ppdm[i] = (double\*)calloc(col\_size, sizeof(double));

}

return ppdm;

}

// 파일로부터 원하는 크기의 행렬 데이터 불러와서 저장

double\*\* fGetMtrx(FILE\* fin, int\* row\_size, int\* col\_size) {

double d = 0.0;

fscanf(fin, "%d %d", row\_size, col\_size);

double\*\* dm = createDynamicDoubleMatrix(\*row\_size, \*col\_size);

for (int i = 0; i < \*row\_size; i++) {

for (int j = 0; j < \*col\_size; j++) {

if (fscanf(fin, "%lf", &d) != EOF) dm[i][j] = d;

}

}

return dm;

}

// 생성한 2차원 동적 배열 해제

void deleteDynMtrx(double\*\* dm, int row\_size, int col\_size) {

for (int i = 0; i < row\_size; i++) free(dm[i]);

free(dm);

}

// GaussJordan\_LinearSystem.h

#ifndef GJ

#define GJ

#include <stdio.h>

#include <math.h>

#include "Mtrx.h"

#define EPSILON 0.000001

void getAugmentedMatrixData(FILE\* fin, double\*\* augMtrx, int size\_N);

void pivoting(double\*\* augMtrx, int size\_N, int p, int\* piv\_found);

void diagonalize\_FileOut(FILE\* fout, double\*\* augMtrx, int size\_N, int\* solExist);

void invMtrxGaussJordanElim\_FileOut(FILE\* fout, double\*\* Mtrx, double\*\* inv\_Mtrx, int size\_N);

#endif

// Mtrx.h

#ifndef M

#define M

#include <stdio.h>

double\*\* createDynamicDoubleMatrix(int row\_size, int col\_size);

double\*\* fGetMtrx(FILE\* fin, int\* row\_size, int\* col\_size);

void deleteDynMtrx(double\*\* dm, int row\_size, int col\_size);

void printMtrx(double\*\* dm, int size\_row, int size\_col);

void fprintMtrx(FILE\* fout, double\*\* dm, int row\_size, int col\_size);

double\*\* addMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int col\_size);

double\*\* subtractMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int col\_size);

double\*\* multiplyMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int temp, int col\_size);

#endif

// Mtrx.cpp(4)

// 행렬 뺄셈

double\*\* subtractMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int col\_size) {

double\*\* dm3 = createDynamicDoubleMatrix(row\_size, col\_size);

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

dm3[i][j] = dm1[i][j] - dm2[i][j];

}

}

return dm3;

}

// 행렬 곱셈

double\*\* multiplyMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int temp, int col\_size) {

double\*\* dm3 = createDynamicDoubleMatrix(row\_size, col\_size);

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

for (int k = 0; k < temp; k++) {

dm3[i][j] += dm1[i][k] \* dm2[k][j];

}

}

}

return dm3;

}

#endif

// Mtrx.cpp(3)

// 행렬 데이터를 지정된 파일에 출력

void fprintMtrx(FILE\* fout, double\*\* dm, int row\_size, int col\_size) {

unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;

unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

if ((i == 0) && (j == 0)) {

fprintf(fout, "%c%c%7.2lf", a6, a3, dm[i][j]);

}

else if ((i == 0) && j == (col\_size - 1)) {

fprintf(fout, "%7.2lf%c%c", dm[i][j], a6, a4);

}

else if ((i > 0) && (i < row\_size - 1) && (j == 0)) {

fprintf(fout, "%c%c%7.2lf", a6, a2, dm[i][j]);

}

else if ((i > 0) && (i < row\_size - 1) && (j == (col\_size - 1))) {

fprintf(fout, "%7.2lf%c%c", dm[i][j], a6, a2);

}

else if ((i == row\_size - 1) && (j == 0)) {

fprintf(fout, "%c%c%7.2lf", a6, a6, dm[i][j]);

}

else if ((i == row\_size - 1) && (j == (col\_size - 1))) {

fprintf(fout, "%7.2lf%c%c", dm[i][j], a6, a5);

}

else {

fprintf(fout, "%7.2lf", dm[i][j]);

}

if (col\_size == 1) {

if (i == 0)

fprintf(fout, "%c%c", a6, a4);

else if (i == row\_size - 1)

fprintf(fout, "%c%c", a6, a5);

else

fprintf(fout, "%c%c", a6, a2);

}

}

fprintf(fout, "\n");

}

}

// 행렬 덧셈

double\*\* addMatrix(double\*\* dm1, double\*\* dm2, int row\_size, int col\_size) {

double\*\* dm3 = createDynamicDoubleMatrix(row\_size, col\_size);

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

dm3[i][j] = dm1[i][j] + dm2[i][j];

}

}

return dm3;

}

// GaussJordan\_LinearSystem.cpp(2)

// 대각화 과정 수행

void diagonalize\_FileOut(FILE\* fout, double\*\* augMtrx, int size\_N, int\* solExist) {

int j, k, p = 0;

double pivWeight, w;

// Pivoting 그리고 Scaling

for (p = 0; (\*solExist) && (p < size\_N); p++) {

fprintf(fout, "\nPivoting at p = %d\n", p);

pivoting(augMtrx, size\_N, p, solExist);

fprintMtrx(fout, augMtrx, size\_N, size\_N + 1);

if (\*solExist) { // 해가 존재

// 해당 위치의 원소 scaling

if (augMtrx[p][p] != 1.0) { // 이미 scaling된 값이 아닌 경우

pivWeight = augMtrx[p][p];

augMtrx[p][p] = 1.0; // scaling

for (k = p + 1; k <= size\_N; k++) {

augMtrx[p][k] /= pivWeight; // 이후의 값도 나누기

}

}

}

else { // 해가 없음

break;

}

fprintf(fout, "\nAfter scaling at p = %d\n", p);

fprintMtrx(fout, augMtrx, size\_N, size\_N + 1);

// Diagonalize

for (j = 0; j < size\_N; j++) {

if ((j != p) && (augMtrx[j][p] != 0.0)) {

w = augMtrx[j][p];

augMtrx[j][p] = 0.0;

for (k = p + 1; k <= size\_N; k++) {

augMtrx[j][k] =

augMtrx[j][k] - w \* augMtrx[p][k];

}

}

}

fprintf(fout, "\nAfter diagonalizing at p = %d\n", p);

fprintMtrx(fout, augMtrx, size\_N, size\_N + 1);

}

}

// 역행렬 구하기

void invMtrxGaussJordanElim\_FileOut(FILE\* fout, double\*\* Mtrx, double\*\* inv\_Mtrx, int size\_N) {

int j, k, p = 0;

double pivWeight, w;

double\*\* m\_aug;

m\_aug = (double\*\*)calloc(size\_N, sizeof(double\*));

for (int i = 0; i < size\_N; i++)

m\_aug[i] = (double\*)calloc(size\_N \* 2, sizeof(double));

for (j = 0; j < size\_N; j++) {

// 역행렬을 구하고싶은 행렬

for (k = 0; k < size\_N; k++)

m\_aug[j][k] = Mtrx[j][k];

// 단위 행렬

for (k = size\_N; k < size\_N \* 2; k++)

m\_aug[j][k] = (k == (j + size\_N)) ? 1.0 : 0.0;

}

// GaussJordan\_LinearSystem.cpp

#ifndef GJ

#define GJ

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <Windows.h>

#include "Mtrx.h"

#define EPSILON 0.000001

// 첨가행렬 데이터 파일로부터 불러오기

void getAugmentedMatrixData(FILE\* fin, double\*\* augMtrx, int size\_N) {

double d = 0.0;

if (fin == NULL) {

printf("Error in getDoubleMatrixData() - file pointer is NULL !!\n");

exit(-1);

}

for (int m = 0; m < size\_N; m++)

for (int n = 0; n <= size\_N; n++) {

if (fscanf(fin, "%lf", &d) != EOF)

augMtrx[m][n] = d;

}

}

// pivoting 과정 수행

void pivoting(double\*\* augMtrx, int size\_N, int p, int\* piv\_found) {

double xmax = 0.0, xtemp;

int j, k, max\_row;

xmax = fabs(augMtrx[p][p]); // 해당 위치 계수의 절대값 저장

max\_row = p; // 해당 위치 저장

for (j = p + 1; j < size\_N; j++) {

if (fabs(augMtrx[j][p]) > xmax) { // 저장한 절대값보다 큰 값이 존재할 경우

xmax = fabs(augMtrx[j][p]); // 더 큰 절대값을 저장

max\_row = j; // 해당 위치 저장

}

}

if (fabs(xmax) < EPSILON) // 값이 0이거나 0에 근접한 경우

\*piv\_found = 0; // 해가 없음

else { // 그 외

\*piv\_found = 1; // 해가 있음

if (max\_row != p) {

for (k = p; k <= size\_N; k++) { // swap

xtemp = augMtrx[p][k];

augMtrx[p][k] = augMtrx[max\_row][k];

augMtrx[max\_row][k] = xtemp;

}

}

}

}

// GaussJordan\_LinearSystem.cpp(3)

for (p = 0; p < size\_N; p++) {

if (m\_aug[p][p] != 1.0) { // 해당 위치의 값이 이미 scaling되지 않은 경우

pivWeight = m\_aug[p][p];

m\_aug[p][p] = 1.0; // scaling

for (k = p + 1; k < size\_N \* 2; k++) {

m\_aug[p][k] /= pivWeight; // 이후의 값도 나누기

}

}

fprintf(fout, "After scaling at p = %d\n", p);

fprintMtrx(fout, m\_aug, size\_N, size\_N \* 2);

// diagonalize

for (j = 0; j < size\_N; j++) {

if ((j != p) && (m\_aug[j][p] != 0.0))

{

w = m\_aug[j][p];

m\_aug[j][p] = 0.0; // 해당 열에서 pivot 위치 이외의 자리에 0.0으로 소거

for (k = p + 1; k < 2 \* size\_N; k++) {

m\_aug[j][k] -= w \* m\_aug[p][k];

}

}

}

fprintf(fout, "After diagonalizing at p = %d\n", p);

fprintMtrx(fout, m\_aug, size\_N, size\_N \* 2);

}

// 구한 역행렬을 생성한 2차원 배열에 복사

for (j = 0; j < size\_N; j++) {

for (k = 0; k < size\_N; k++) {

inv\_Mtrx[j][k] = m\_aug[j][k + size\_N];

}

}

}

#endif

Oral Test

|  |
| --- |
| **(1) 행렬의 크기를 파일로부터 읽고, 지정된 크기의 2차원 배열을 동적으로 생성한 후, 행렬 원소의 데이터를 파일로부터 읽어 저장한 후, 생성된 행렬의 주소로 행렬의 크기를 return-by-pointer 방법으로 반환하는 fGetMtrx() 함수의 기능에 대하여 각 단계 별로 상세하게 설명하라**   * 행렬의 크기 fscanf를 통해 파일 데이터 상단에 표시한 크기 데이터를 읽어온다. 이때, 행렬의 크기는 return-by-pointer를 이용하여 반환한다. * 2차원 배열 동적 생성 2중 포인터 배열을 생성 생성한 2중 포인터에 creatDynamicDoubleMatrix()함수를 통해 지정된 크기의 2차원 배열을 동적 할당한다. * 파일로부터 데이터 읽기 생성한 2차원 배열에 원하는 데이터를 파일로부터 불러와서 저장한다. * 작성한 배열 반환 함수의 반환 자료형을 double\*\*로 지정하고 위에서 작성한 배열을 반환한다. |

|  |
| --- |
| **(2) Gauss-Jordan 소거법에서 pivoting을 하는 이유에 대하여 설명하고, 구체적인 절차에 대하여 설명하라.**  Gauss-Jordan 소거법 연산 과정에서 컴퓨터 계산의 반올림 처리 오류 발생을 최소화하기 위하여 pivot 행을 선정할 때 남아있는 행들 중에서 pivot 열 위치의 원소 절대값이 가장 큰 행을 찾고, 이를 pivot 위치의 행과 교체한다.  만약 남아있는 행들 중에서 pivot 열 위치의 원소 값이 가장 큰 pivot 대상 원소 절대값이 0이거나 0에 근접한 경우, 오차가 발생하기 때문에 정확한 해를 구할 수 없는 상태가 되며, 이 경우 단일 해를 구할 수 없는 것으로 판단한다.  pivoting은 p번째 이후의 행들 중에서 p번째 원소의 절대값이 가장 큰 행을 p번째 행으로 선정하고, 필요한 경우 순서를 swap하는 과정을 말한다.   * 해를 구할 수 있는지 판단 pivot 대상 원소의 절대값이 0이거나 0에 근접한 값이면 solExist에 0을 반환하고 반복을 종료시킨다. * swap 조건에 해당하는지 판단 pivot에 해당하는 절대값을 기준으로 잡고 그보다 큰 절대값이 이후에 존재하면 위치를 swap한다. |

|  |
| --- |
| **(3) Gauss-Jordan 소거법에서diagonalization의 절차에 대하여 설명하고, 이를 통하여 선형 방 정식 시스템의 해를 구하게 되는 원리에 대하여 설명하라.**  diagonalization은 p 번째 열에서 1.0으로 scaling된 원소를 제외한 해당 행의 모든 원소를 0.0으로변경하도록 소거 연산을 수행한다.   * 반복 조건 pivot에 해당하는 계수가 0이나 0에 근접한 값이 아니어야한다.(해가 존재해야한다.) * pivoting pivot에 해당하는 계수보다 큰 행이 존재하면 위치를 swap한다. * scaling pivoting한 pivot 위치의 값을 1.0으로 만든다. * diagonalization scaling된 원소를 제외한 동일 행의 모든 원소를 0.0으로 변경하도록 소거연산 수행한다. * solution 모든 과정을 완료한 후 가장 우측 열의 값이 해당 행렬의 solution이다. |

|  |
| --- |
| **(4) Gauss-Jordan 소거법을 사용하여 역행렬을 구하는 절차를 설명하고, 이를 사용하여 선형방정식 시스템의 해를 구하게 되는 원리에 대하여 설명하라.**   * n x 2n 배열 생성 지정된 크기의 첨가 행렬을 생성한다. 크기를 2n으로 지정한 이유는 좌측은 역행렬을 구하고싶은 행렬, 우측은 단위행렬을 생성하기 위함이다. * 역행렬 계산 좌측 부분을 단위행렬로 변환하는 과정을 똑같이 우측의 단위 행렬에 실행해주면 역행렬을 구할 수 있다. * 계산한 역행렬을 원하는 배열에 복사 계산을 마친 우측 배열을 원하는 배열에 반복문을 통해 복사한다. |