2021-1 프로그래밍언어 실습 8

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.
2) fGetMtrx() 함수의 실행 결과로 동적으로 생성된 행렬과 그 행렬의 크기 (size_row, size_column)이 return-by-pointer 방식으로 반환되어야 한다. fGetMtrx() 함수에서는지정된 입력 데이터 파일(예: Array_2D_input.txt)로부터 fscanf() 함수를 사용하여, 각size_rowx 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.

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

출력 결과	확장 완성형 코드	사용 방법
ㅡ	0xa6, 0xa1	printf(" %c%c", 0xa6, 0xa1)
ㅣ	0xa6, 0xa2	printf(" %c%c", 0xa6, 0xa2)
ㄱ	0xa6, 0xa3	printf(" %c%c", 0xa6, 0xa3)
ㄴ	0xa6, 0xa4	printf(" %c%c", 0xa6, 0xa4)
ㄷ	0xa6, 0xa5	printf(" %c%c", 0xa6, 0xa5)
ㄹ	0xa6, 0xa6	printf(" %c%c", 0xa6, 0xa6)

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 사용하여 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

**mB, int size_row, int size_col)를 작성하라. main() 함수에서는 행렬의 덧셈 결과를 출력하도록 하라.

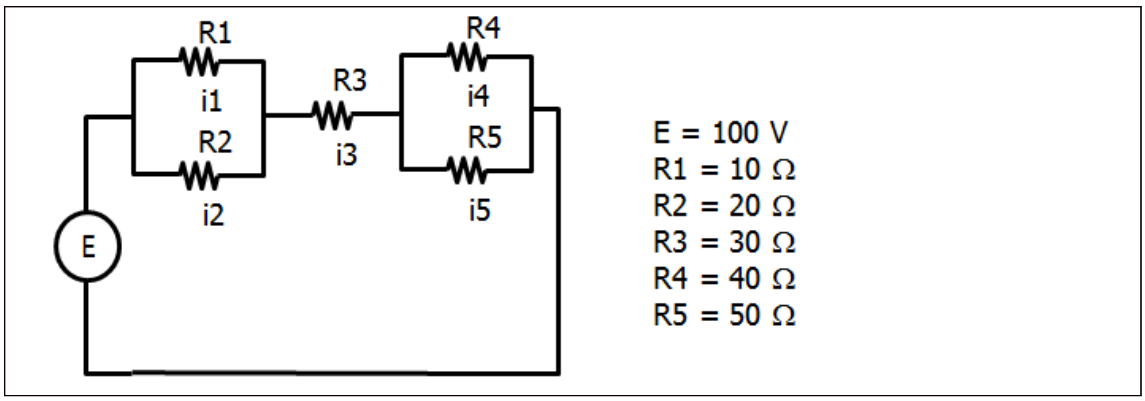
2) 2개의 size_row x size_col 크기의 행렬 A, B의 뺄셈을 계산하여 그 결과를 동적으로 생성된 행렬 R에 저장하고, R의 주소를 반환하는 함수 double **subtractMtrx(double **mA, double **mB, int size_row, int size_col)를 작성하라. main() 함수에서는 행렬의 뺄셈 결과를 출력하도록 하라.

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.

8.3 첨가 행렬과 가우스 소거법 및 역행렬을 사용한 선형 방정식 해

8.3.1 Augmented Matrix 의 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 방정식의 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.

8.3.2 Gauss Elimination

1) Gauss Elimination을 사용하여 선형 방정식 시스템의 해를 구하기 위하여, pivoting() 함수를작성하라.

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

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

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 $X = A^{-1}B$를 계산하는 방법을 사용하여 해를 구하라.
4) 이렇게 산출된 해를 위 8.3.2에서 구한 해와 비교하라.

8.4 main 프로그램과 실행 결과

8.4.1 main() 함수

```
/** main_Matrx_GaussJordan_LinearSystem.cpp */
#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;
                }
        }   // end while
        fclose(fout);
}
```

8.4.2 test_2D_DynamicArray_FileIO(fout)

```
void test_2D_DynamicArray_FileIO(FILE *fout)
{
        const char *matrixDataFile = "mtrxInputData.txt";
        FILE *fin;
        int a_row_size, a_col_size;
        int b_row_size, b_col_size;
        double **mA, **mB;

        fin = fopen(matrixDataFile, "r");
        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);
}
```

8.4.3 test_MatrixAdditionSubtraction(fout)

```
void test_MatrixAdditionSubtraction(FILE *fout)
{
        const char *matrixDataFile = "mtrx_nxn_InputData.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");
	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");

	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;
	}
	// MC = MA x MB
	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");

	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");

	deleteDynMtrx(mA, a_row_size, a_col_size);
	deleteDynMtrx(mB, b_row_size, b_col_size);
	deleteDynMtrx(mC, c_row_size, c_col_size);
	deleteDynMtrx(mD, d_row_size, d_col_size);

	fclose(fin);
}
```

8.4.4 test_MatrixMultiplication(fout)

```
void test_MatrixMultiplication(FILE *fout)
{
	const char *matrixDataFile = "mtrxInputData.txt";
	FILE *fin;

	int a_row_size, a_col_size;
	int b_row_size, b_col_size;
	int c_row_size, c_col_size;
	double **mA, **mB, **mC;

	fin = fopen(matrixDataFile, "r");
	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");

        // MC = MA x MB
        c_row_size = a_row_size;
        c_col_size = b_col_size;
        mC = multiplyMatrix(mA, mB, a_row_size, a_col_size, b_col_size);
        printf("Matrix_C (%d x %d) = Matrix_A x Matrix_B : \n", c_row_size, c_col_size);
        printMtrx(mC, c_row_size, c_col_size);
        printf("\n");

        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);
}
```

8.4.5 test_GaussJordanElimination_LinearSystem(fout)

```
const char *augMatrix_inputFile = "AugMtrx_ElectronicCircuit_B_5x5.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);
        solution = (double *)calloc(size_N, sizeof(double));
        fprintf(fout, "Augmented Matrix size_N : %d\n", size_N);
        //fGetDoubleMatrixData(fin, augMtrx, size_N, size_N + 1);
        getAugmentedMatrixData(fin, augMtrx, size_N);
        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);
        if (solExist) {
                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];
                        fprintf(fout, " x[%d] : %4f\n", i, solution[i]);
                        printf(" x[%d] : %4f\n", i, solution[i]);
                }
        }
        else {
                fprintf(fout, "No unique solution\n");
                printf("No unique solution\n");
        }

        for (int i = 0; i < size_N; i++)
                free(augMtrx[i]);
        free(augMtrx);
        free(solution);
        fclose(fin);
}
```

8.4.6 test_InvMtrx_GaussJordanElimination_LinearSystem(fout)

```
//const char *linearSystem_inputFile = "ElectronicCircuit_A_5x5.txt";
const char *linearSystem_inputFile = "ElectronicCircuit_B_5x5.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");
        //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);
        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);

        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");

        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);
}
```

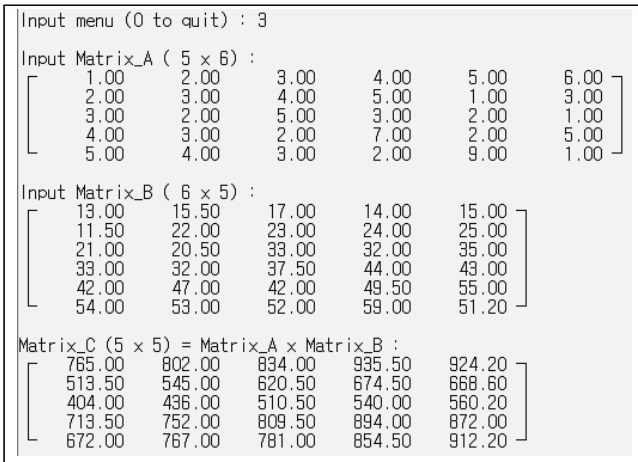
8.4.7 입력 파일 및 출력 결과

```
Testing Matrix Operations with 2-Dimensional Dynamic Array
 1: Test 2-D Dynamic Array Creation for Matrix with File I/O
 2: Test Matrix Addition, Subtraction
 3: Test Matrix Multiplication
 4: Test Gauss-Jordan Elimination for Linear System
 5: Test Inverse Matrix with Gauss-Jordan Elimination for Linear System
 0: Quit
Input menu (0 to quit) : 1

Input Matrix_A ( 5 x 6) :
┌     1.00      2.00      3.00      4.00      5.00      6.00 ┐
│     2.00      3.00      4.00      5.00      1.00      3.00 │
│     3.00      2.00      5.00      3.00      2.00      1.00 │
│     4.00      3.00      2.00      7.00      2.00      5.00 │
└     5.00      4.00      3.00      2.00      9.00      1.00 ┘

Input Matrix_B ( 6 x 5) :
┌    13.00     15.50     17.00     14.00     15.00 ┐
│    11.50     22.00     23.00     24.00     25.00 │
│    21.00     20.50     33.00     32.00     35.00 │
│    33.00     32.00     37.50     44.00     43.00 │
│    42.00     47.00     42.00     49.50     55.00 │
└    54.00     53.00     52.00     59.00     51.20 ┘
```

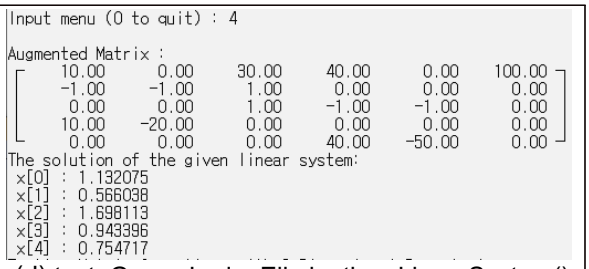
(a) test_2D_DynamicArray_FileIO 실행 결과

```
Input menu (0 to quit) : 2

Input Matrix_A ( 6 x 6) :
┌     1.00      2.00      3.00      4.00      5.00      6.00 ┐
│     2.00      3.00      4.00      5.00      1.00      3.00 │
│     3.00      2.00      5.00      3.00      2.00      1.00 │
│     4.00      3.00      2.00      7.00      2.00      5.00 │
│     5.00      4.00      3.00      2.00      9.00      1.00 │
└     6.00      7.00      8.00      9.00     10.00     11.00 ┘

Input Matrix_B ( 6 x 6) :
┌    13.00     15.50     17.00     14.00     15.00     16.00 ┐
│    11.50     22.00     23.00     24.00     25.00     26.00 │
│    21.00     20.50     33.00     32.00     35.00     36.00 │
│    33.00     32.00     37.50     44.00     43.00     42.00 │
│    42.00     47.00     42.00     49.50     55.00     60.00 │
└    54.00     53.00     52.00     59.00     51.20     70.00 ┘

Matrix_C (6 x 6) = Matrix_A + Matrix_B :
┌    14.00     17.50     20.00     18.00     20.00     22.00 ┐
│    13.50     25.00     27.00     29.00     26.00     29.00 │
│    24.00     22.50     38.00     35.00     37.00     37.00 │
│    37.00     35.00     39.50     51.00     45.00     47.00 │
│    47.00     51.00     45.00     51.50     64.00     61.00 │
└    60.00     60.00     60.00     68.00     61.20     81.00 ┘

Matrix_D (6 x 6) = Matrix_A - Matrix_B :
┌   -12.00    -13.50    -14.00    -10.00    -10.00    -10.00 ┐
│    -9.50    -19.00    -19.00    -19.00    -24.00    -23.00 │
│   -18.00    -18.50    -28.00    -29.00    -33.00    -35.00 │
│   -29.00    -29.00    -35.50    -37.00    -41.00    -37.00 │
│   -37.00    -43.00    -39.00    -47.50    -46.00    -59.00 │
└   -48.00    -46.00    -44.00    -50.00    -41.20    -59.00 ┘
```

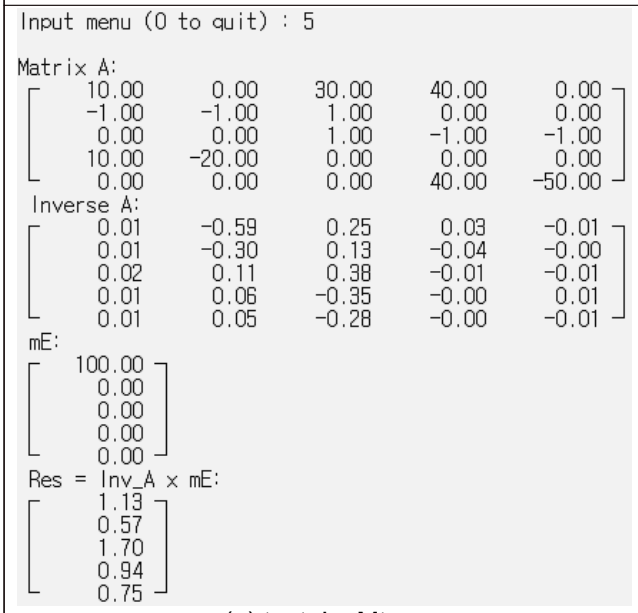
(b) test_MatrixAdditionSubtraction() 실행결과



(c) test_MatrixMultiplication() 실행결과



(d) test_GaussJordanElimination_LinearSystem() 실행 결과



(e) test_InvMtrx_ GaussJordanElimination_LinearSystem() 실행 결과

<Oral Test>

Q8.1 행렬의 크기 (row_size, col_size)를 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 단계별로 상세하게 설명하라.

Q8.2 Gauss-Jordan 소거법에서 pivoting을 하는 이유에 대하여 설명하고, 구체적인 절차에 대하여 설명하라.

Q8.3 Gauss-Jordan 소거법에서diagonalization의 절차에 대하여 설명하고, 이를 통하여 선형 방정식 시스템의 해를 구하게 되는 원리에 대하여 설명하라.

Q8.4 Gauss-Jordan 소거법을 사용하여 역행렬 (inverse matrix)을 구하는 절차를 설명하고, 이를 사용하여 선형방정식 시스템의 해를 구하게 되는 원리에 대하여 설명하라.