

프로그래밍언어 (실습)

실습 7 (보충설명) - 2차원배열 동적생성, 행렬 연산



교수 김 영 탁

영남대학교 정보통신공학과

(Tel : +82-53-810-2497; Fax : +82-53-810-4742

<http://antl.yu.ac.kr/>; E-mail : ytkim@yu.ac.kr)

Outline

- ◆ 2차원 배열의 동적 생성
- ◆ 파일 입출력
- ◆ 행렬의 연산
 - 덧셈
 - 뺄셈
 - 곱셈



실습 7

실습 7.1 행렬 연산을 위한 2차원 배열의 동적 생성, 파일 입출력 (1)

7.1.1 행렬 연산을 위한 2차원 배열의 동적 생성, 주소 확인 및 삭제

- 1) 지정된 크기의 2차원 double 자료형 배열을 동적으로 생성하는 함수 `double ** createDoubleMtrx(int row_size, int col_size)`를 작성하라.
- 2) 2차원 배열의 배열 이름, 첫번째 행의 주소, 첫번째 원소의 주소를 각각 출력하는 함수 `void checkAddr_2D_Array(double **dM, int row_size, int col_size)`를 작성하라.
- 3) 지정된 크기의 2차원 double 자료형 배열을 삭제하는 함수 `void deleteDoubleMtrx(double **dM, int row_size, int col_size)`를 작성하라.

7.1.2 행렬 연산을 위한 double 자료형의 파일 입력

- 1) 지정된 크기의 2차원 double 자료형 배열을 동적으로 생성하고, 입력 데이터 파일로부터 `row_size x col_size` 개의 double 데이터를 `fscanf()` 함수를 사용하여 읽고, 동적으로 생성한 2차원 배열에 저장하는 기능을 함수 `double ** fgetDoubleMtrx(FILE *fin, int row_size, int col_size)`을 구현하라. `fgetMtrx()` 함수는 `createDoubleMtrx()` 함수를 사용할 것.

실습 7.1 행렬 연산을 위한 2차원 배열의 동적 생성, 파일 입출력 (2)

7.1.3 행렬의 화면/파일 출력

- 1) 행렬 (2차원 배열, 크기: row_size x col_size)을 화면으로 출력하는 함수 void printMtrx(double **dM, int size_row, int size_col)을 작성하라. 행렬의 각 값은 최소 8자리를 확보하고, 소수점 이하 2자리를 출력하며, 오른쪽 정렬로 출력되도록 할 것. 이 때, 행렬을 대괄호 ([,])로 표시하기 위하여, 확장 완성형 코드를 사용할 것.

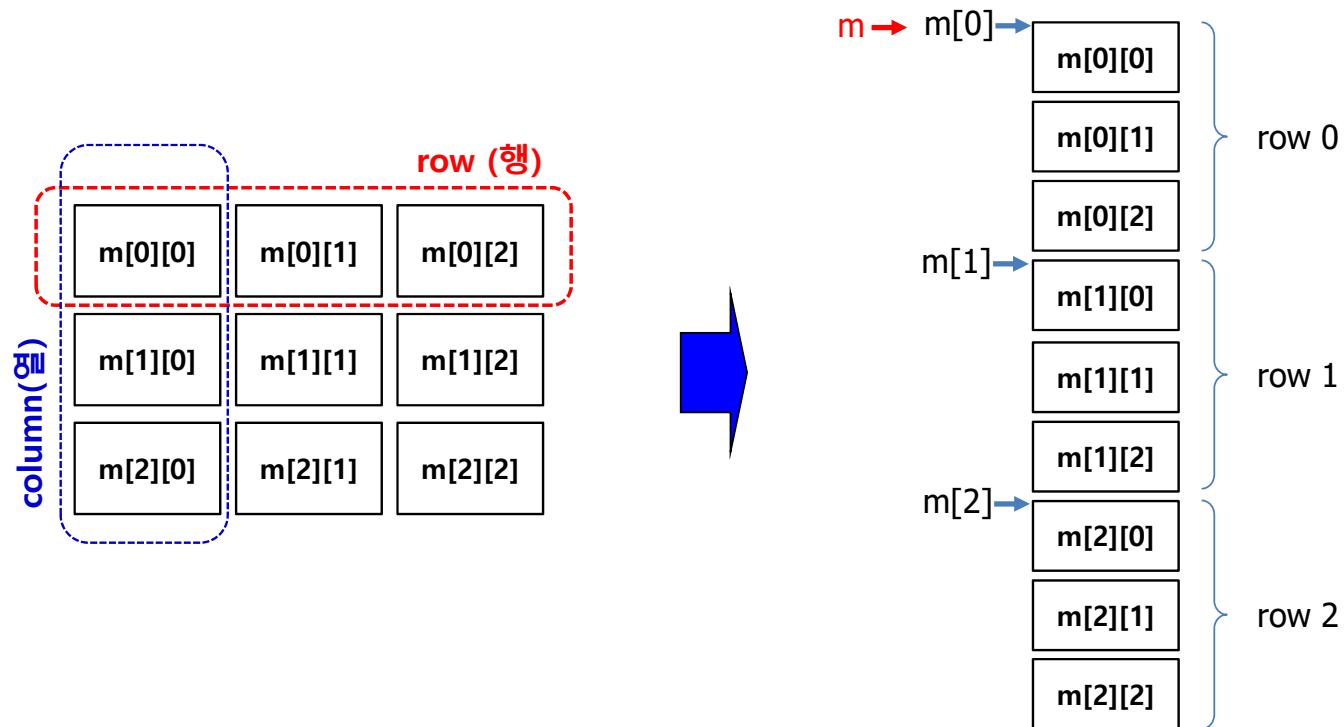
출력 결과	확장 완성형 코드	사용 방법
—	0xa6, 0xa1	printf(" %c%c", 0xa6, 0xa1)
	0xa6, 0xa2	printf(" %c%c", 0xa6, 0xa2)
┌	0xa6, 0xa3	printf(" %c%c", 0xa6, 0xa3)
┐	0xa6, 0xa4	printf(" %c%c", 0xa6, 0xa4)
└	0xa6, 0xa5	printf(" %c%c", 0xa6, 0xa5)
┘	0xa6, 0xa6	printf(" %c%c", 0xa6, 0xa6)

- 2) 행렬 (2차원 배열, 크기: row_size x col_size)을 파일로 출력하는 함수 void fprintMtrx(FILE *fout, double **mA, int row_size, int col_size)을 작성하라. 이 때, 행렬을 대괄호 ([,])로 표시하기 위하여, 확장 완성형 코드를 사용할 것.

2차원 배열과 포인터

◆ 2차원 배열 `int m[3][3]`

◆ row 0, row 1, row 2 ... 순으로 메모리에 저장 (행 우선 방법)



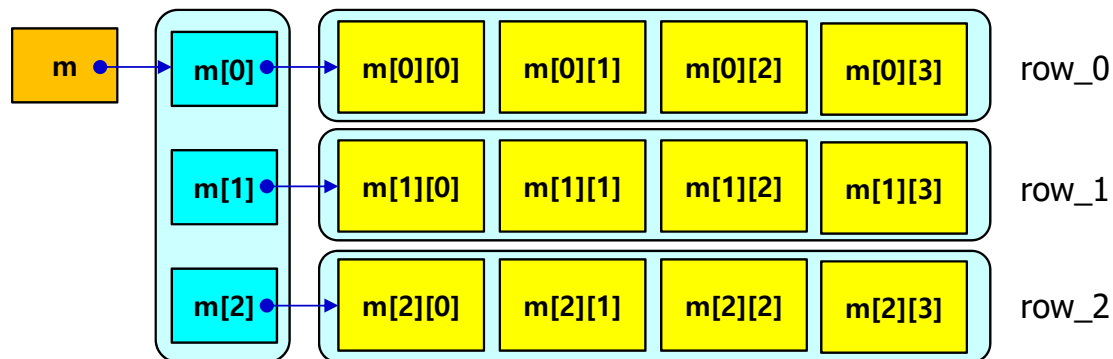
2차원 배열과 포인터

- ◆ 배열 이름 **m**은 **&m[0][0]**
- ◆ **m[0]**는 row 0의 시작 주소
- ◆ **m[1]**은 row 1의 시작 주소
- ◆ **m[2]**는 row 2의 시작 주소

double m[3][4];

m은 3개의 1 차원 배열을 원소로 가지는 2차원 배열이다.

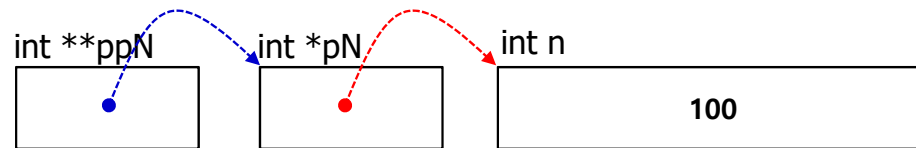
m[]의 각 원소는 4개의 double 형 데이터를 가지는 1차원 배열이다.



이중 포인터 (double pointer)

◆ 이중 포인터(double pointer) : 포인터를 가리키는 포인터

```
int n = 100; // n는 int 형 변수
int *pN = &n; // pN는 n를 가리키는 포인터
int **ppN = &pN; // ppN는 포인터 pN를 가리키는 이중 포인터 (double pointer)
```



`pN` : `int n`의 주소

`*pN` : `pN`가 가리키는 위치의 데이터 (`n`)

`*ppN` : `ppN`이 가리키는 위치의 데이터 (`pN`의 값, `n`의 주소)

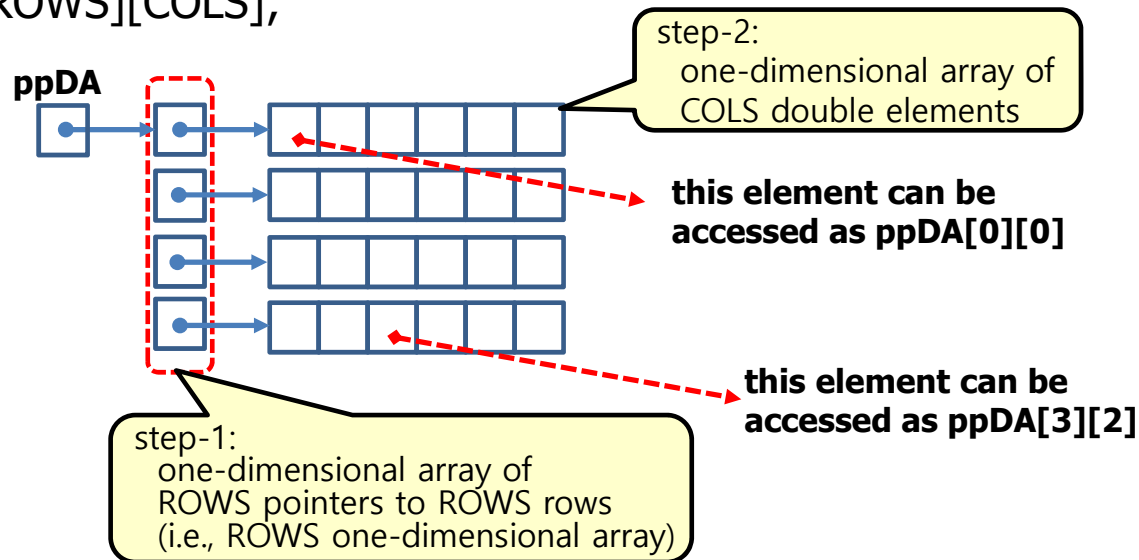
`**ppN` : `ppN`가 가리키는 위치의 데이터 (`pN`의 값, `n`의 주소)가 가리키는 위치의 데이터 (`n`)

A diagram showing the variable `**ppN` enclosed in a blue bracket. A blue dashed arrow points from the text above to the top of the bracket, and a red dashed arrow points from the text below to the bottom of the bracket.

2차원 배열의 동적 생성

◆ 2-dimensional Dynamic Array of double type (1)

- `double DA[ROWS][COLS];`



- can be considered as four rows of one dimensional array: `DA[i][0..4]`
- each row is pointed by a pointer to 1-dimension array:
- Array of pointers to the four 1-dimension arrays of pointers
 - `double **ppDA = (double **)calloc(ROWS, sizeof(double *));`
- To make the 2-dimensional array
 - `ppDA[i] = (double *)calloc(COLS, sizeof(double));`

2차원 배열의 동적 생성

◆ 2차원 실수 (double) 배열의 동적 생성 함수

```
double **createDoubleMtrx(int row_size, int col_size)
{
    double **dM;

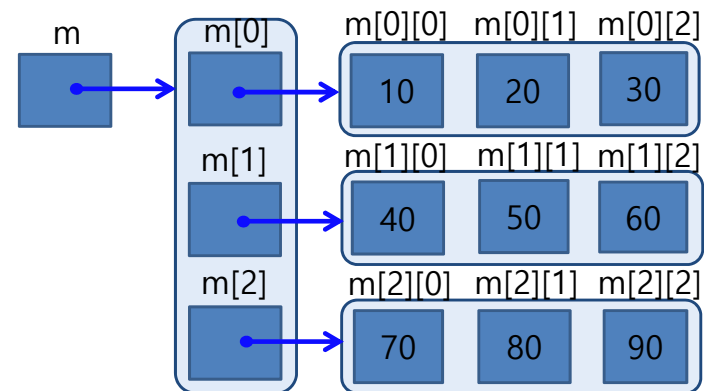
    dM = (double **)calloc(row_size, sizeof(double *));
    for (int r = 0; r < row_size; r++)
    {
        dM[r] = (double *)calloc(col_size, sizeof(double));
    }

    return dM;
}
```

2 차원 배열 원소의 주소

```
void checkAddress_2DArray_for_Matrix()
{
    int m[3][3] =
        { 10, 20, 30, 40, 50, 60, 70, 80, 90 };

    printf("m      = %p\n", m);
    printf("m[0]   = %p\n", m[0]);
    printf("m[1]   = %p\n", m[1]);
    printf("m[2]   = %p\n", m[2]);
    printf("&m[0][0] = %p\n", &m[0][0]);
    printf("&m[1][0] = %p\n", &m[1][0]);
    printf("&m[2][0] = %p\n", &m[2][0]);
    printf("\n");
}
```



```
m      = 0036F9B8
m[0]   = 0036F9B8
m[1]   = 0036F9C4
m[2]   = 0036F9D0
&m[0][0] = 0036F9B8
&m[1][0] = 0036F9C4
&m[2][0] = 0036F9D0
```

동일한 주소 !!

행렬의 파일 입력

```
double ** fgetDoubleMtrx(FILE *fp, int row_size, int col_size)
{
    double d = 0.0;
    double** dM;

    if (fp == NULL)
    {
        printf("Error in getDoubleMatrixData() - file pointer is NULL !!\n");
        exit(-1);
    }

    .... // 동적 2차원 배열 생성을 이곳에 구현할 것.

    for (int m = 0; m < row_size; m++)
        for (int n = 0; n < col_size; n++)
        {
            if (fscanf(fp, "%lf", &d) != EOF)
                dM[m][n] = d;
        }

    return dM;
}
```



확장 완성형 코드를 사용한 행렬의 출력

```
void printMtrx(double mA[][SIZE], int size)
{
    unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;
    unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;

    for (int i=0; i< size_n; i++) {
        for (int j=0; j< size; j++)
        {
            if ((i==0) && (j==0))
                printf("%c%c%c%7.2lf", a6, a3, mA[i][j]);
            else if ((i==0) && j==(size -1))
                printf("%7.2lf%c%c", mA[i][j], a6, a4);
            else if ((i>0) && (i<size-1) && (j==0))
                printf("%c%c%c%7.2lf", a6, a2, mA[i][j]);
            else if ((i>0) && (i<size-1) && (j== (size -1)))
                printf("%7.2lf%c%c", mA[i][j], a6, a2);
            else if ((i==(size-1)) && (j==0))
                printf("%c%c%c%7.2lf", a6, a6, mA[i][j]);
            else if ((i==(size-1)) && (j==(size -1)))
                printf("%7.2lf%c%c", mA[i][j], a6, a5);
            else
                printf("%7.2lf", mA[i][j]);
        }
        printf("\n");
    }
}
```

출력 결과	확장 완성형 코드
—	0xa6, 0xa1
	0xa6, 0xa2
┌	0xa6, 0xa3
┐	0xa6, 0xa4
└	0xa6, 0xa5
┘	0xa6, 0xa6

```
Matrix A :
┌ 5.00  4.00  3.00  2.00  1.00
├ 6.00  7.00  8.00  9.00 10.00
├ 11.00 12.00 13.00 14.00 15.00
├ 16.00 17.00 18.00 19.00 20.00
└ 21.00 22.00 23.00 24.00 25.00
Matrix B :
┌ 1.00  0.00  0.00  0.00  0.00
├ 0.00  1.00  0.00  0.00  0.00
├ 0.00  0.00  1.00  0.00  0.00
├ 0.00  0.00  0.00  1.00  0.00
└ 0.00  0.00  0.00  0.00  1.00
```

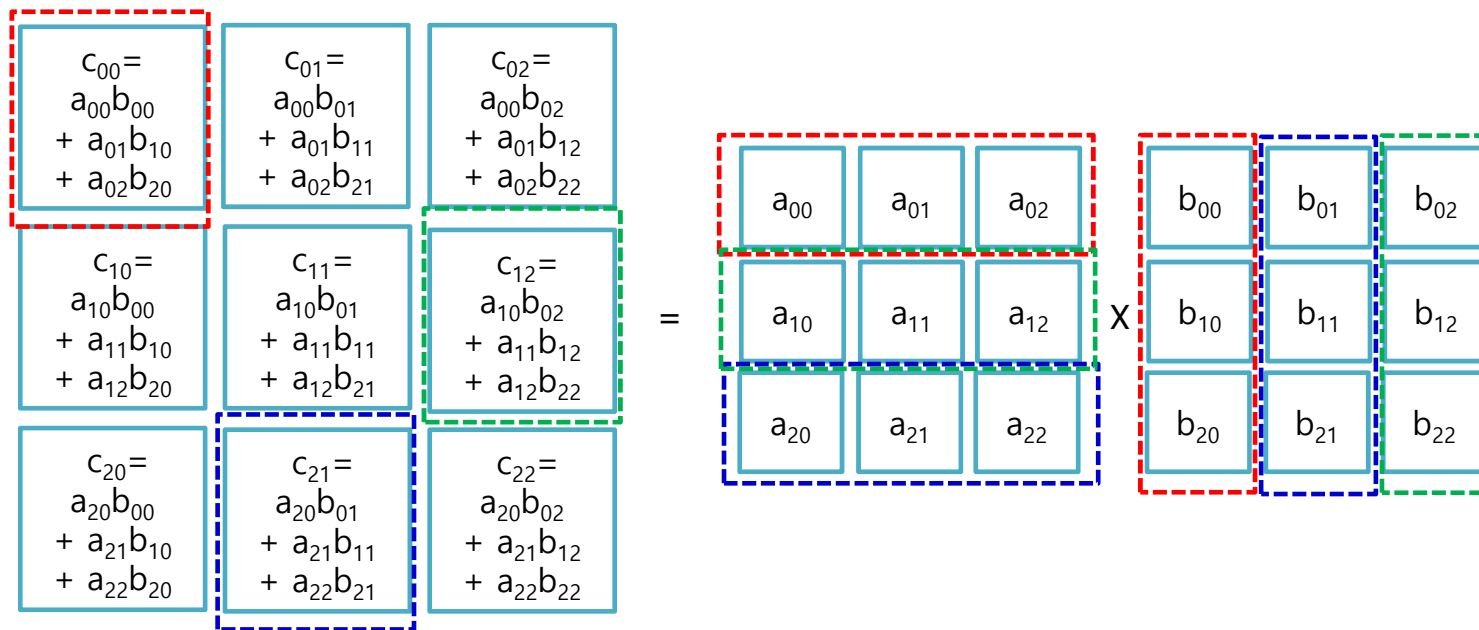
실습 7.2 행렬의 연산

7.2.1 행렬의 덧셈, 뺄셈, 곱셈 연산

- 1) 2개의 $\text{row_size} \times \text{col_size}$ 크기의 행렬 A, B의 덧셈을 계산하여 그 결과를 행렬 R에 저장하는 함수 void **addMtrx**(double **mA, double **mB, double **mR, int row_size, int col_size) 를 작성하라.
- 2) 2개의 $\text{row_size} \times \text{col_size}$ 크기의 행렬 A, B의 뺄셈을 계산하여 그 결과를 행렬 R에 각각 저장하는 함수 void **subMtrx**(double **mA, double **mB, double **mR, int row_size, int col_size) 를 작성하라.
- 3) 2개의 $\text{row_size} \times \text{size_K}$ 크기의 행렬 A와 $\text{size_K} \times \text{col_size}$ 크기의 행렬 B의 곱셈을 계산하여 그 결과를 $\text{row_size} \times \text{col_size}$ 크기의 행렬 R에 저장하는 함수 void **multiplyMtrx**(double **mA, double **mB, double **mR, int row_size, int size_k, int col_size)를 작성하라.

행렬의 곱셈

◆ 행렬의 곱셈 계산



행렬의 덧셈, 뺄셈, 곱셈

```
void addMtrx(double **mA, double **mB, double **mR, int row_size, int col_size)
```

```
{
    for (int r=0; r<row_size; r++)
        for (int c=0; c<col_size; c++)
            mR[r][c] = mA[r][c] + mB[r][c];
}
```

```
void subtractMtrx(double **mA, double **mB, double **mR, int row_size, int col_size)
```

```
{
    for (int r=0; r<row_size; r++)
        for (int c=0; c<col_size; c++)
            mR[r][c] = mA[r][c] - mB[r][c];
}
```

```
void multiplyMtrx(double **mA, double **mB, double **mR, int row_size, int col_size)
```

```
{
    for (int r=0; r<row_size; r++)
        for (int c=0; c<col_size; c++)
        {
            mR[r][c] = 0;
            for (int k=0; k<size; k++)
                mR[r][c] += mA[r][k] * mB[k][c];
        }
}
```



실습 7.3 main() 함수

```
/** main_DynamicTwoDimensionalArray.cpp (1) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "Matrix.h"

void checkAddress_2DimArray_for_Matrix();
void test_2D_DynamicArray_FileIO();
void test_Matrix_Addition_Subtraction();
void test_Matrix_Multiplication();
int main(void)
{
    int menu;

    while (1)
    {
        printf("Testing Matrix Operations with 2-Dimensional Dynamic Array\n");
        printf(" 1: Check addresses of 2-Dim array for Matrix\n");
        printf(" 2: Test 2-D Dynamic Array Creation for Matrix with File I/O\n");
        printf(" 3: Test Matrix Addition, Subtraction\n");
        printf(" 4: Test Matrix Multiplication\n");
        printf("-1: Quit\n");
        printf("Input menu (-1 to quit) : ");
        scanf("%d", &menu);
        if (menu == -1)
            break;
    }
}
```



실습 7.3 main() 함수

```
/** main_DynamicTwoDimensionalArray.cpp (2) */

switch (menu)
{
case 1:
    checkAddress_2DimArray_for_Matrix();
    break;
case 2:
    test_2D_DynamicArray_FileIO();
    break;
case 3:
    test_Matrix_Addition_Subtraction();
    break;
case 4:
    test_Matrix_Multiplication();
    break;
default:
    break;
}
} // end while
}
```



```

void checkAddress_2DimArray_for_Matrix()
{
    int m[3][3] = { 10, 20, 30, 40, 50, 60, 70, 80, 90 };

    printf("m      = %p\n", m);
    printf("m[0]   = %p\n", m[0]);
    printf("m[1]   = %p\n", m[1]);
    printf("m[2]   = %p\n", m[2]);
    printf("&m[0][0] = %p\n", &m[0][0]);
    printf("&m[1][0] = %p\n", &m[1][0]);
    printf("&m[2][0] = %p\n", &m[2][0]);
    printf("\n");
}

```

```

Testing Matrix Operations with 2-Dimensional Dynamic Array
1: Check addresses of 2-Dim array for Matrix
2: Test 2-D Dynamic Array Creation for Matrix with File I/O
3: Test Matrix Addition, Subtraction
4: Test Matrix Multiplication
-1: Quit
Input menu (-1 to quit) : 1
m      = 007EFAF0
m[0]   = 007EFAF0
m[1]   = 007EFAFC
m[2]   = 007EFB08
&m[0][0] = 007EFAF0
&m[1][0] = 007EFAFC
&m[2][0] = 007EFB08

```



void test_2D_DynamicArray_FileIO()

```
{
    const char *matrixDataFile = "mtrxInputData.txt";
    FILE *fin;
    int a_row_size, a_col_size;
    int b_row_size, b_col_size;
    double **dMA, **dMB;

    fin = fopen(matrixDataFile, "r");
    if (fin == NULL)
    {
        printf("Error in opening input.txt file !!\n");
        exit(-1);
    }

    fscanf(fin, "%d %d", &a_row_size, &a_col_size);
    dMA = fgetDoubleMtrx(fin, a_row_size, a_col_size);
    printf("Input Matrix_A ( %d x %d) : \n", a_row_size, a_col_size);
    printMtrx(dMA, a_row_size, a_col_size);
    printf("\n");

    fscanf(fin, "%d %d", &b_row_size, &b_col_size);
    dMB = fgetDoubleMtrx(fin, b_row_size, b_col_size);
    printf("Input Matrix_B ( %d x %d) : \n", b_row_size, b_col_size);
    printMtrx(dMB, b_row_size, b_col_size);
    printf("\n");

    deleteDoubleMtrx(dMA, a_row_size, a_col_size);
    deleteDoubleMtrx(dMB, b_row_size, b_col_size);
    fclose(fin);
}
```

Line	Content
1	5 6
2	1.0 2.0 3.0 4.0 5.0 6.0
3	2.0 3.0 4.0 5.0 1.0 3.0
4	3.0 2.0 5.0 3.0 2.0 1.0
5	4.0 3.0 2.0 7.0 2.0 5.0
6	5.0 4.0 3.0 2.0 9.0 1.0
7	
8	6 5
9	13.0 15.5 17.0 14.0 15.0
10	11.5 22.0 23.0 24.0 25.0
11	21.0 20.5 33.0 32.0 35.0
12	33.0 32.0 37.5 44.0 43.0
13	42.0 47.0 42.0 49.5 55.0
14	54.0 53.0 52.0 59.0 51.2

(a) mtrxInputData.txt

```
Input menu (-1 to quit) : 2
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 ]
```

(b) 화면 출력

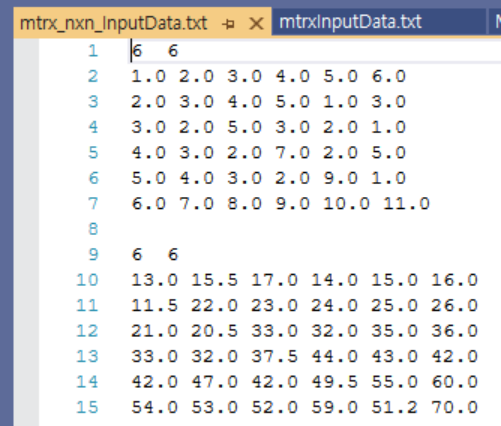
void test_Matrix_Addition_Subtraction()

```
{
    const char *matrixDataFile = "mtrx_nxn_InputData.txt";
    FILE *fin;

    double **dMA, **dMB, **dMC, **dMD;
    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);
    }
    fscanf(fin, "%d %d", &a_row_size, &a_col_size);
    dMA = fgetDoubleMtrx(fin, a_row_size, a_col_size);
    printf("Input Matrix_A ( %d x %d) : \n", a_row_size, a_col_size);
    printMtrx(dMA, a_row_size, a_col_size);
    printf("\n");

    fscanf(fin, "%d %d", &b_row_size, &b_col_size);
    dMB = fgetDoubleMtrx(fin, b_row_size, b_col_size);
    printf("Input Matrix_B ( %d x %d) : \n", b_row_size, b_col_size);
    printMtrx(dMB, b_row_size, b_col_size);
    printf("\n");
}
```



```
mtrx_nxn_InputData.txt  mtrxInputData.txt
1 6 6
2 1.0 2.0 3.0 4.0 5.0 6.0
3 2.0 3.0 4.0 5.0 1.0 3.0
4 3.0 2.0 5.0 3.0 2.0 1.0
5 4.0 3.0 2.0 7.0 2.0 5.0
6 5.0 4.0 3.0 2.0 9.0 1.0
7 6.0 7.0 8.0 9.0 10.0 11.0
8
9 6 6
10 13.0 15.5 17.0 14.0 15.0 16.0
11 11.5 22.0 23.0 24.0 25.0 26.0
12 21.0 20.5 33.0 32.0 35.0 36.0
13 33.0 32.0 37.5 44.0 43.0 42.0
14 42.0 47.0 42.0 49.5 55.0 60.0
15 54.0 53.0 52.0 59.0 51.2 70.0
```

(a) mtrx_nxn_InputData.txt

```
Input menu (-1 to quit) : 3
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
 -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) 화면 출력

```

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 + MB
c_row_size = a_row_size;
c_col_size = b_col_size;
dMC = createDoubleMtrx(c_row_size, c_col_size);
addMtrx(dMA, dMB, dMC, a_row_size, a_col_size);
printf("Matrix_C (%d x %d) = Matrix_A + Matrix_B : \n", c_row_size, c_col_size);
printMtrx(dMC, c_row_size, c_col_size);
printf("\n");

// MC = MA - MB
d_row_size = a_row_size;
d_col_size = b_col_size;
dMD = createDoubleMtrx(d_row_size, d_col_size);
subMtrx(dMA, dMB, dMD, a_row_size, a_col_size);
printf("Matrix_D (%d x %d) = Matrix_A - Matrix_B : \n",
    d_row_size, d_col_size);
printMtrx(dMD, d_row_size, d_col_size);
printf("\n");

deleteDoubleMtrx(dMA, a_row_size, a_col_size);
deleteDoubleMtrx(dMB, b_row_size, b_col_size);
deleteDoubleMtrx(dMD, d_row_size, d_col_size);
deleteDoubleMtrx(dMC, c_row_size, c_col_size);

fclose(fin);
}

```

```

Input menu (-1 to quit) : 3
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) 화면 출력

void test_Matrix_Multiplication()

```
{
    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 **dMA, **dMB, **dMC;

    fin = fopen(matrixDataFile, "r");
    if (fin == NULL)
    {
        printf("Error in opening input.txt file !!\n");
        exit(-1);
    }
    fscanf(fin, "%d %d", &a_row_size, &a_col_size);
    dMA = fgetDoubleMtrx(fin, a_row_size, a_col_size);
    printf("Input Matrix_A ( %d x %d) : \n", a_row_size, a_col_size);
    printMtrx(dMA, a_row_size, a_col_size);
    printf("\n");

    fscanf(fin, "%d %d", &b_row_size, &b_col_size);
    dMB = fgetDoubleMtrx(fin, b_row_size, b_col_size);
    printf("Input Matrix_B ( %d x %d) : \n", b_row_size, b_col_size);
    printMtrx(dMB, b_row_size, b_col_size);
    printf("\n");
}
```

mtrxInputData.txt	Matrix.cpp	Matrix
1	5	6
2	1.0	2.0 3.0 4.0 5.0 6.0
3	2.0	3.0 4.0 5.0 1.0 3.0
4	3.0	2.0 5.0 3.0 2.0 1.0
5	4.0	3.0 2.0 7.0 2.0 5.0
6	5.0	4.0 3.0 2.0 9.0 1.0
7		
8	6	5
9	13.0	15.5 17.0 14.0 15.0
10	11.5	22.0 23.0 24.0 25.0
11	21.0	20.5 33.0 32.0 35.0
12	33.0	32.0 37.5 44.0 43.0
13	42.0	47.0 42.0 49.5 55.0
14	54.0	53.0 52.0 59.0 51.2

(a) mtrxInputData.txt

```
Input menu (-1 to quit) : 4
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 ]

Matrix_C (5 x 5) = Matrix_A x Matrix_B :
[ 765.00  802.00  834.00  935.50  924.20
  513.50  545.00  620.50  674.50  668.60
  404.00  436.00  510.50  540.00  560.20
  713.50  752.00  809.50  894.00  872.00
  672.00  767.00  781.00  854.50  912.20 ]
```

(b) 화면 출력

```

// MC = MA x MB
c_row_size = a_row_size;
c_col_size = b_col_size;
dMC = createDoubleMtrx(c_row_size, c_col_size);
multiplyMtrx(dMA, dMB, dMC, 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(dMC, c_row_size, c_col_size);
printf("\n");

deleteDoubleMtrx(dMA, a_row_size, a_col_size);
deleteDoubleMtrx(dMB, b_row_size, b_col_size);
deleteDoubleMtrx(dMC, c_row_size, c_col_size);

fclose(fin);
}

```

```

Input menu (-1 to quit) : 4
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 ]

Matrix_C (5 x 5) = Matrix_A x Matrix_B :
[ 765.00  802.00  834.00  935.50  924.20 ]
[ 513.50  545.00  620.50  674.50  668.60 ]
[ 404.00  436.00  510.50  540.00  560.20 ]
[ 713.50  752.00  809.50  894.00  872.00 ]
[ 672.00  767.00  781.00  854.50  912.20 ]

```

(b) 화면 출력

Oral Test 7

Oral Test 7

- Q7.1 포인터의 연산과 그 포인터의 자료형이 어떤 관계에 있는지 설명하라.
- Q7.2 함수호출에서의 인수 전달 방식 중 call-by-value와 call-by-pointer의 차이점을 예를 들어 설명하고, call-by-pointer의 장점에 대하여 설명하라.
- Q7.3 동적 메모리 할당 방법을 사용하여, 행렬 계산을 위한 2차원 배열을 동적으로 생성하는 절차와 삭제하는 절차에 대하여 각 단계별로 상세하게 설명하라.
- Q7.4 지정된 파일로 부터 행렬의 크기 (row_size, col_size)를 읽고, 이 행렬 크기에 따라 double 자료형 2차원 배열을 동적으로 구성한 후, row_size x col_size개의 double 자료형 행렬 데이터를 파일로 부터 읽어 동적으로 생성된 파일에 저장하는 절차에 대하여 각 단계별로 상세하게 설명하라.

