객체지향프로그래밍과 자료구조 (실습)

Lab. 3 (보충설명)
동적 2차원 배열과 행렬 연산

정보통신공학과
교수 김 영 탁

(Tel : +82-53-810-2497; Fax : +82-53-810-4742
http://antl.yu.ac.kr/; E-mail : ytkim@yu.ac.kr)

Outline

◆C++ class에서 포인터 사용

- 변수의 간접참조
- 함수의 인수 (argument) 전달, 함수 반환 값 전달
- 동적 배열 생성

◆Class Mtrx

- 동적 2차원 배열 생성
- 행렬 덧셈
- 행렬 뺄셈
- 행렬 곱셈

포인터 (pointer)

◆ 포인터란 ?

- 값으로 주소 (address)를 가지는 변수 (variable)
- 포인터는 자료형 (data type)이 지정되며,
  포인터의 자료형에 따라 가리키는 내용을 다르게 해석함

◆ 포인터 관련 연산자

연산자의 분류	연산자	의미
포인터 관련 연산자 (Operators for pointer)	주소 연산자 **&**	변수나 함수의 주소를 찾아낼 때 사용
	간접참조 연산자 *	포인터가 가리키고 있는 곳의 값을 읽거나 쓸 때 사용
	배열 인덱스 연산자 **[ ]**	포인터를 배열의 이름처럼 사용하여, 동적 배열로 사용할 때

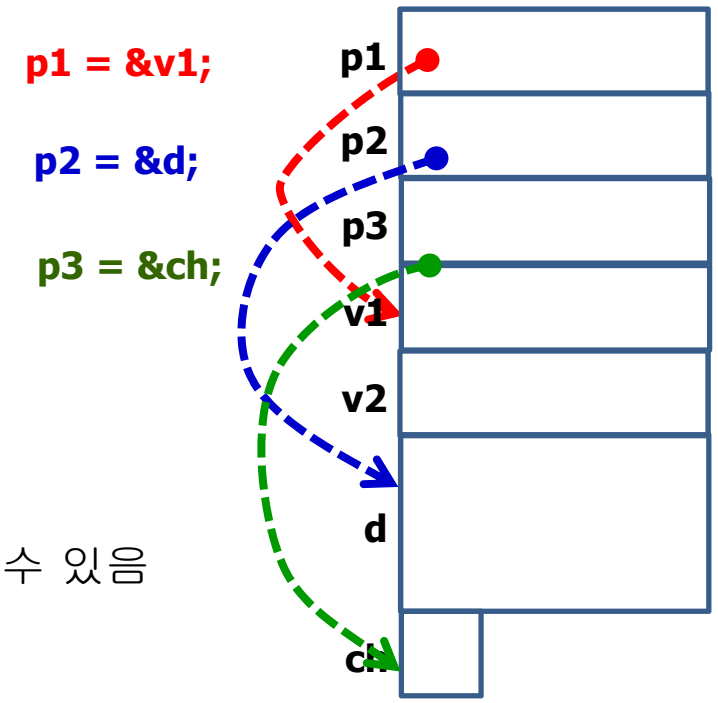
포인터의 자료형, 주소 연산자 (&), 간접참조 연산자 (*)

◆ 자료형이 다른 포인터들의 선언 및 값 설정

```
int *p1;
double *p2;
char *p3;
int  v1, v2;
double d;
char ch;
```

◆ 주소 연산자 (&)

```
p1 = &v1;
p2 = &d;
p3 = &ch;
```



◆ 간접 참조 연산자 (*)

- 포인터를 사용하여 변수를 간접 접근 (참조)
- 포인터가 가리키는 곳의 값을 읽거나 변경할 수 있음

◆ 데이터 변수의 접근 방법

- 변수 이름에 의한 접근

  v1 = 100;
- 포인터와 간접참조 연산자에 의한 접근

  ***p1 = 200;**

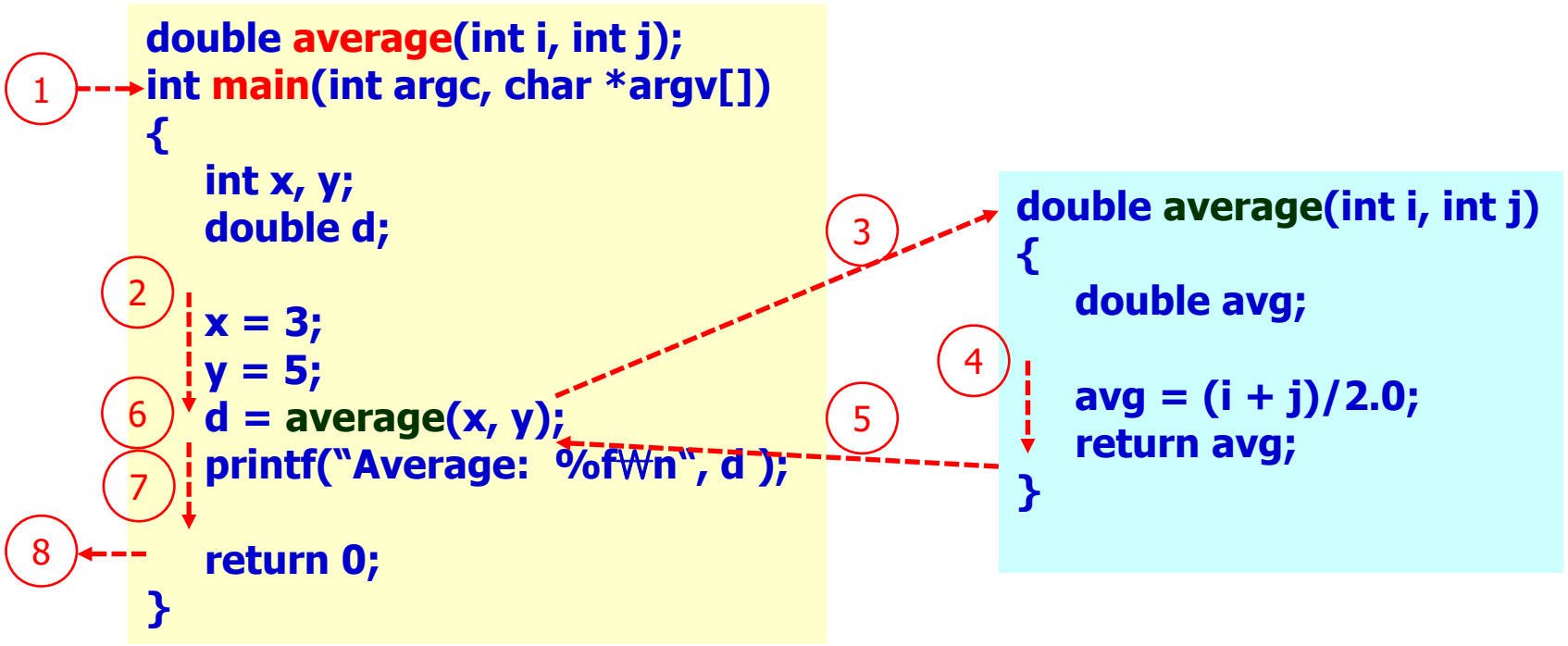
C++ 클래스 멤버함수 호출과 반환에서의 포인터와 참조

◆C/C++ 함수의 호출과 반환에서의 인수 전달 방법

인수 전달 방법	비교
call-by-value return-by-value	함수 호출 시 인수 (argument) 값을 복사하여 전달 인수의 내용이 클 경우 내용 복사에 시간이 많이 걸릴 수 있고, 메모리 사용이 늘어남
call-by-pointer return-by-pointer	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
call-by-reference return-by-reference	call-by-pointer와 유사하게 인수의 내용을 직접 복사하지 않고, 참조 정보를 전달하여 호출된 함수에서 직접 사용할 수 있도록 함.

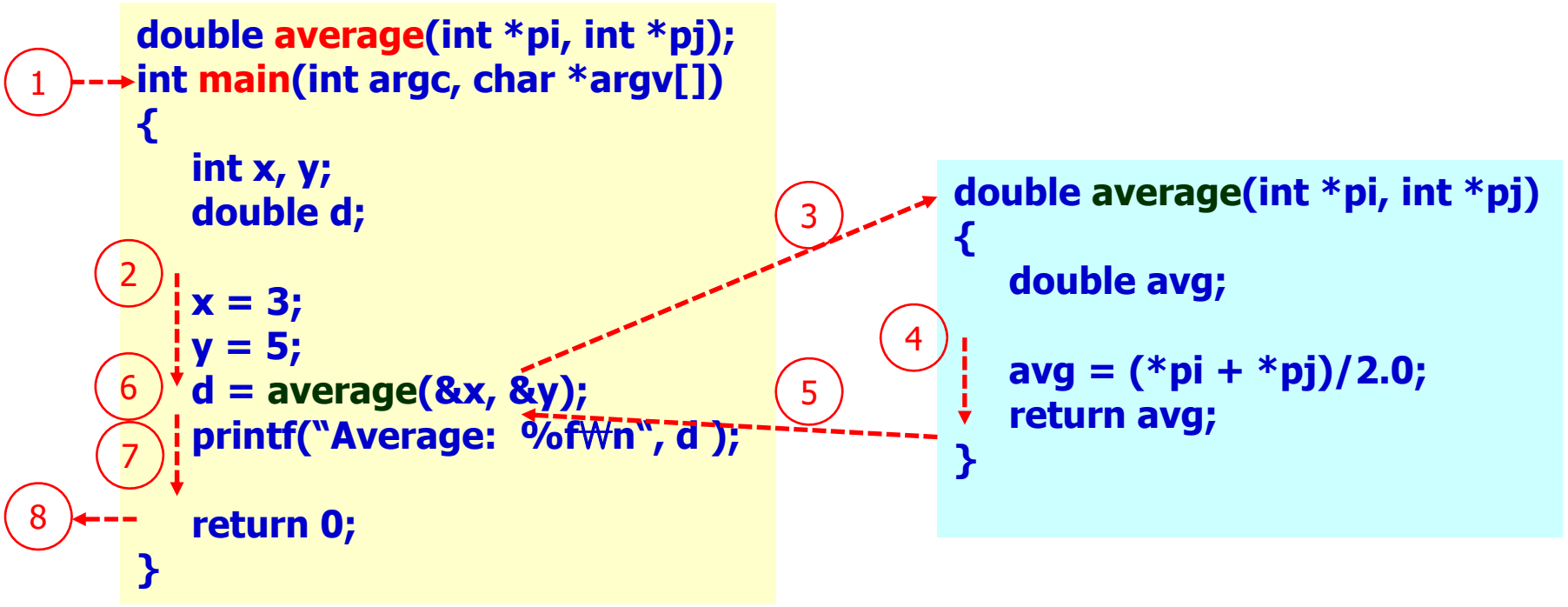
Call-by-Value, Return-by-Value

◆ 인수 (argument)의 값을 복사하여 전달



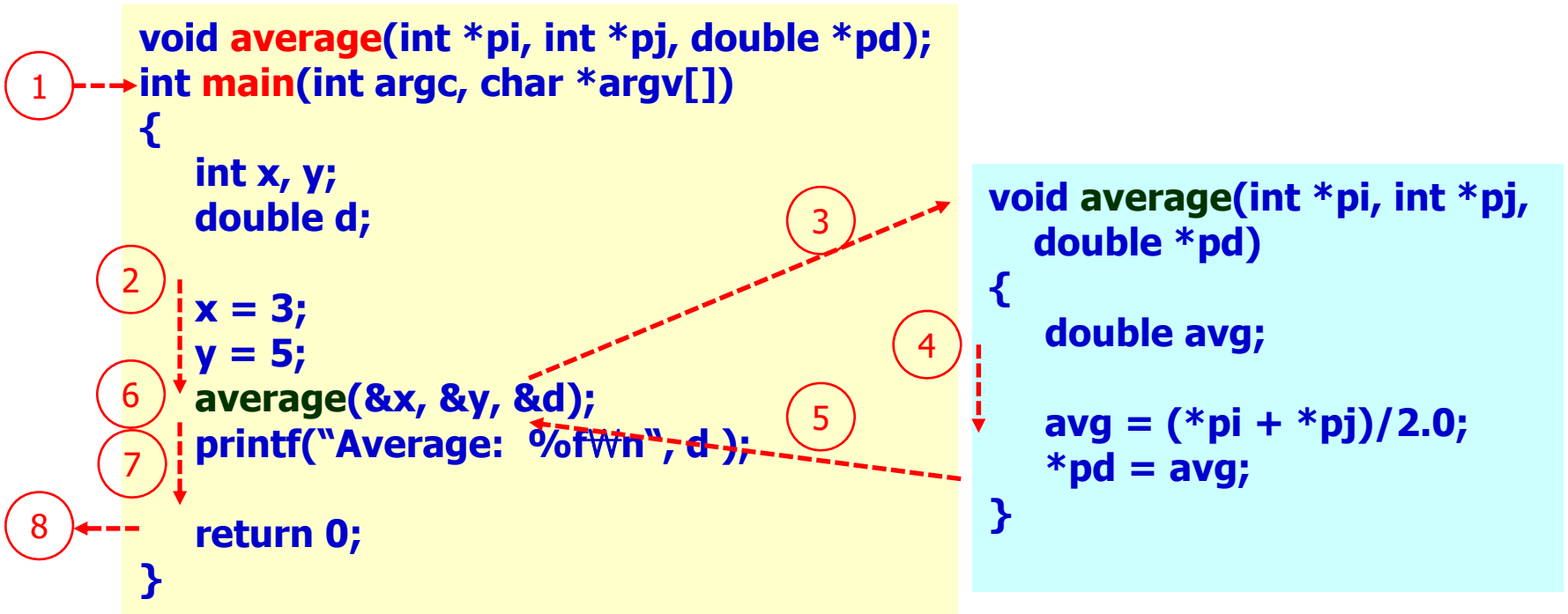
Call-by-Pointer, Return-by-Value

◆ 인수 (argument)의 주소를 전달



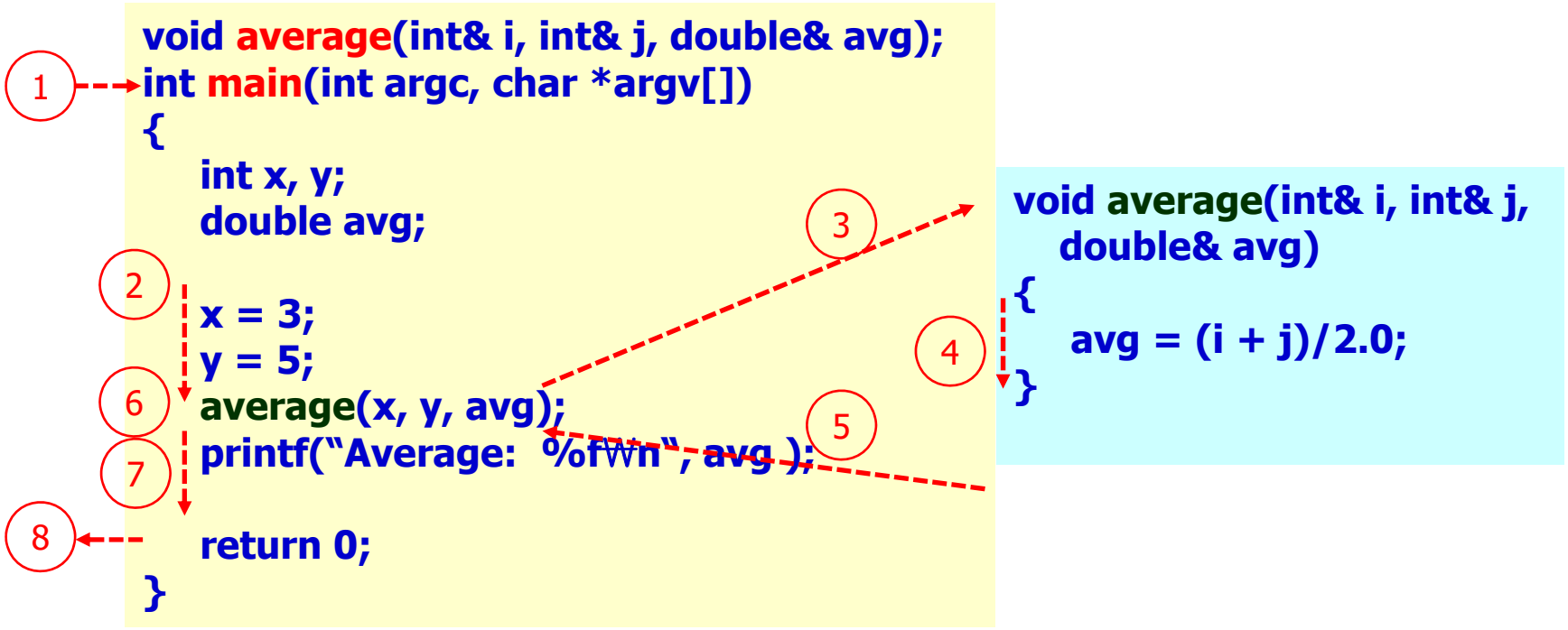
Call-by-Pointer, Return-by-Pointer

◆ 인수(argument)의 주소를 전달



Call-by-Reference

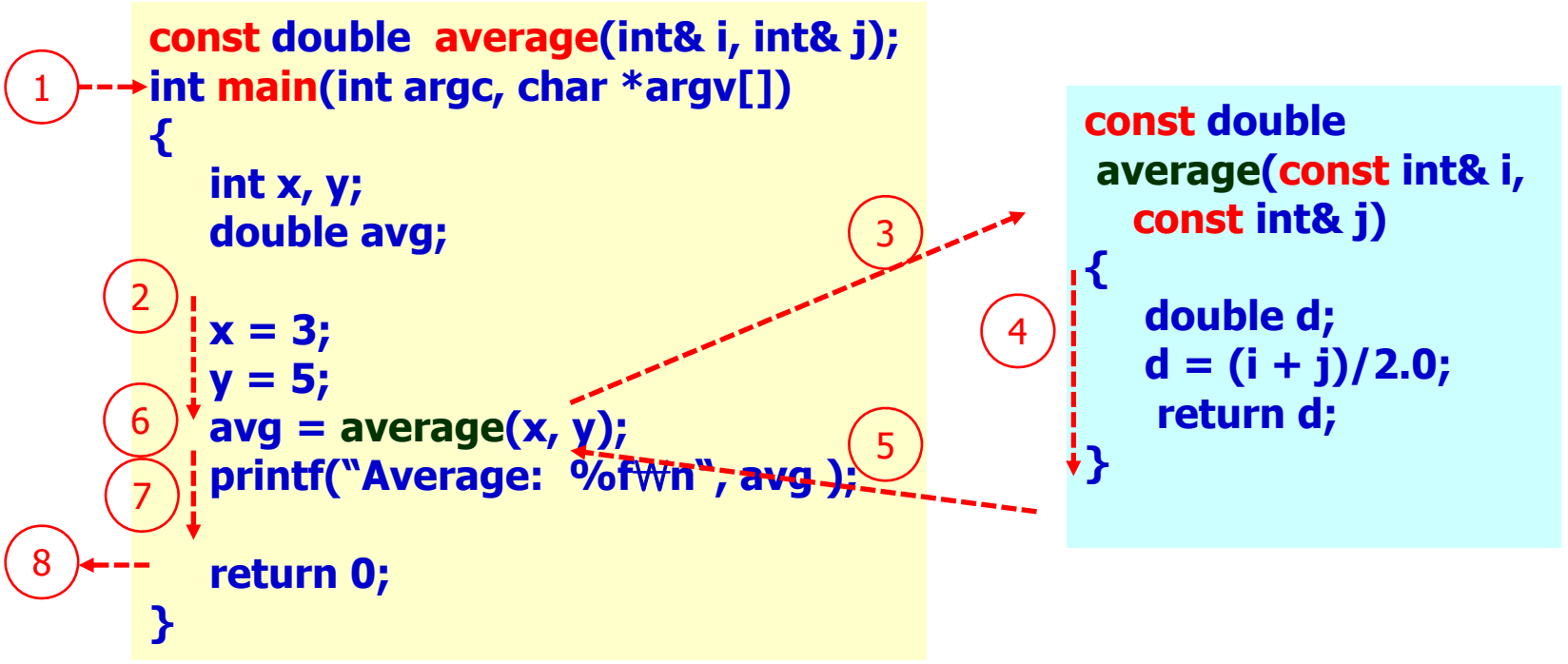
◆ 인수 (argument)의 참조 (reference)를 전달



함수의 호출과 반환에서의 const

◆ const

- const로 지정된 인수는 읽기만 가능하며 변경을 불가능
- 참조로 전달된 인수 값이 비정상적으로 변경되는 것을 방지함



C++ 프로그램에서의 동적 메모리 할당

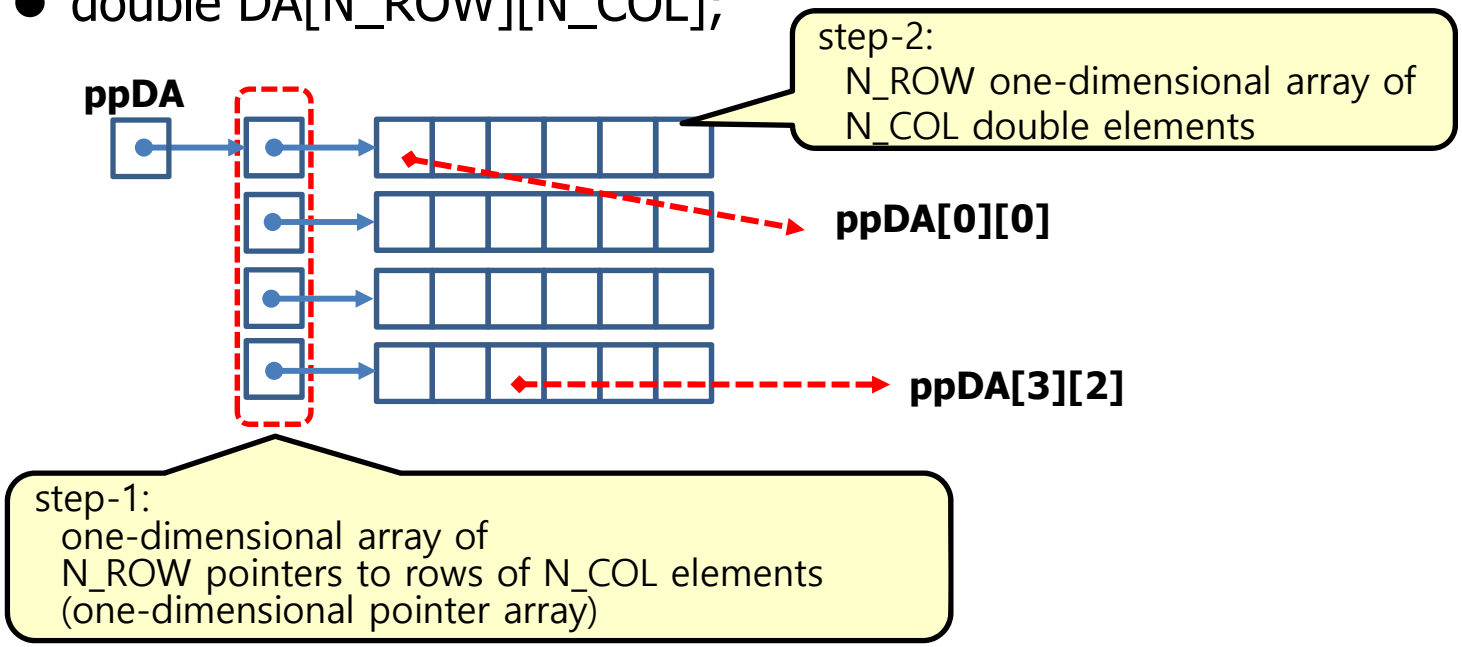
◆C/C++에서의 동적 메모리 할당 관련 함수

분류	함수 원형과 인수	기 능
동적 메모리 블록 할당 및 반환 <stdlib.h>	**void* malloc(size_t size)**	지정된 size 크기의 메모리 블록을 할당하고, 그 시작 주소를 void pointer로 반환
	void *calloc(size_t n, size_t size)	size 크기의 항목을 n개 할당하고, 0으로 초기화 한 후, 그 시작 주소를 void pointer로 반환
	void *realloc(void *p, size_t size)	이전에 할당 받아 사용하고 있는 메모리 블록의 크기를 변경 p는 현재 사용하고 있는 메모리 블록의 주소, size는 변경하고자 하는 크기; 기존의 데이터 값은 유지된다
	void free(void *p)	동적 메모리 블록을 시스템에 반환; p는 현재 사용하였던 메모리 블록 주소
C++ 동적 메모리 블록 할당 및 반환	**void * new int;** **void * new int[10];**	지정된 데이터 타입 또는 데이터 배열을 저장하기 위한 메모리 블록을 할당하고, 그 시작 주소를 void pointer로 반환
	delete p; **delete [] pA;**	포인터로 지정된 변수 또는 배열을 삭제

2차원 배열의 동적 생성

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

- double DA[N_ROW][N_COL];



- 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(N_ROW, sizeof(double *));**
- To make the 2-dimensional array
  - **ppDA[i] = (double *)calloc(N_COL, sizeof(double));**

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

```
double **ppDA = (double **)calloc (N_ROW, sizeof(double*));
for (int i=0; i<N_ROW; i++)
{
   ppDA[i] = (double *) calloc (N_COL, sizeof(double));
}
ppDA[2][3] = 1.0;
```

```
typedef double* DblPtr

DblPtr *ppDB = (DblPtr *)calloc( N_ROW, sizeof(DblPtr));
for (int i=0; i<N_ROW; i++)
{
   ppDB[i] = (DblPtr) calloc (N_COL, sizeof (double));
}
ppDB[2][3] = 1.0;
```

class Mtrx

```
/**  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);
    Mtrx(double *dA, int num_data, int num_row, int num_col);
    Mtrx(istream& fin);
    ~Mtrx(); // destructor
    int getN_row() { return n_row; }
    int getN_col() { return n_col; }
    void fprintMtrx(ostream& fout);
    void setName(string nm) { name = nm;};
    string getName() { return name;};
    Mtrx add(const Mtrx&);
    Mtrx sub(const Mtrx&);
    Mtrx multiply(const Mtrx&);
private:
    string name;
    int n_row;
    int n_col;
    double **dM;
};
#endif
```

class Mtrx 멤버함수 구현

```
/**   Matrix.cpp  (1) */
#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";
}
```

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

Mtrx::~Mtrx()
{
    // cout << "destructor of Mtrx ("
    //          << name << ")" << endl;
    /*
    for (int i=0; i<n_row; i++)
        delete [] dM[i];
    delete [] dM;
    */
}
```

```
/**   Matrix.cpp  (3) */
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";
}
```

```
/**   Matrix.cpp  (4) */

#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(2);
             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;
}
```

출력 결과	확장 완성형 코드
─	0xA6, 0xA1
│	0xA6, 0xA2
┌	0xA6, 0xA3
┐	0xA6, 0xA4
┘	0xA6, 0xA5
└	0xA6, 0xA6

```
┌  1.00  2.00  3.00  4.00  5.00┐
│  2.00  3.00  4.00  5.00  1.00│
│  3.00  2.00  5.00  3.00  2.00│
│  4.00  3.00  2.00  7.00  2.00│
└  5.00  4.00  3.00  2.00  9.00┘
```

```
/**   Matrix.cpp  (5) */

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;
}
```

```
/**   Matrix.cpp  (6) */
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;
}
```

```
/**   Matrix.cpp  (7) */

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;
}
```

class Mtrx 응용 프로그램

```
/**  main.c (1) */

#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;
        }
```

```
/**  main.c (2) */

        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);

        Mtrx mtrxD(mtrxA.getN_row(), mtrxB.getN_col());
        mtrxD = mtrxA.add(mtrxB);
        mtrxD.setName("MtrxD = mtrxA.add(mtrxB)");
        mtrxD.fprintMtrx(fout);

        Mtrx mtrxE(mtrxA.getN_row(), mtrxB.getN_col());
        mtrxE = mtrxA.sub(mtrxB);
        mtrxE.setName("MtrxE = mtrxA.sub(mtrxB)");
        mtrxE.fprintMtrx(fout);
```

```
/**  main.c (3) */

            Mtrx mtrxF(mtrxA.getN_row(), mtrxC.getN_col());
            mtrxF = mtrxA.multiply(mtrxC);
            mtrxF.setName("MtrxF = mtrxA.multiply(mtrxC)");
            mtrxF.fprintMtrx(fout);

            fout.close();

} // end of main()
```

실행 결과

```
5  7
1.0 2.0 3.0 4.0 5.0 6.0 7.0
2.0 3.0 4.0 5.0 1.0 7.0 8.0
3.0 2.0 5.0 3.0 2.0 4.0 6.0
4.0 3.0 2.0 7.0 2.0 1.0 9.0
5.0 4.0 3.0 2.0 9.0 6.0 9.0

5  7
1.0 0.0 0.0 0.0 0.0 1.0 2.0
0.0 1.0 0.0 0.0 0.0 2.0 3.0
0.0 0.0 1.0 0.0 0.0 3.0 4.0
0.0 0.0 0.0 1.0 0.0 4.0 5.0
0.0 0.0 0.0 0.0 1.0 5.0 6.0

7  5
1.0 2.0 3.0 4.0 5.0
6.0 7.0 2.0 3.0 4.0
5.0 1.0 7.0 8.0 3.0
2.0 5.0 3.0 2.0 4.0
6.0 4.0 3.0 2.0 7.0
2.0 1.0 9.0 5.0 4.0
3.0 2.0 9.0 6.0 9.0
```

(Input data)

```
MtrxA =
┌                       ┐
|   1.0   2.0   3.0   4.0   5.0   6.0   7.0 |
|   2.0   3.0   4.0   5.0   1.0   7.0   8.0 |
|   3.0   2.0   5.0   3.0   2.0   4.0   6.0 |
|   4.0   3.0   2.0   7.0   2.0   1.0   9.0 |
└   5.0   4.0   3.0   2.0   9.0   6.0   9.0 ┘

MtrxB =
┌                       ┐
|   1.0   0.0   0.0   0.0   0.0   1.0   2.0 |
|   0.0   1.0   0.0   0.0   0.0   2.0   3.0 |
|   0.0   0.0   1.0   0.0   0.0   3.0   4.0 |
|   0.0   0.0   0.0   1.0   0.0   4.0   5.0 |
└   0.0   0.0   0.0   0.0   1.0   5.0   6.0 ┘

MtrxC =
┌                   ┐
|   1.0   2.0   3.0   4.0   5.0 |
|   6.0   7.0   2.0   3.0   4.0 |
|   5.0   1.0   7.0   8.0   3.0 |
|   2.0   5.0   3.0   2.0   4.0 |
|   6.0   4.0   3.0   2.0   7.0 |
|   2.0   1.0   9.0   5.0   4.0 |
└   3.0   2.0   9.0   6.0   9.0 ┘

MtrxD = mtrxA.add(mtrxB) =
┌                       ┐
|   2.0   2.0   3.0   4.0   5.0   7.0   9.0 |
|   2.0   4.0   4.0   5.0   1.0   9.0  11.0 |
|   3.0   2.0   6.0   3.0   2.0   7.0  10.0 |
|   4.0   3.0   2.0   8.0   2.0   5.0  14.0 |
└   5.0   4.0   3.0   2.0  10.0  11.0  15.0 ┘

MtrxE = mtrxA.sub(mtrxB) =
┌                       ┐
|   0.0   2.0   3.0   4.0   5.0   5.0   5.0 |
|   2.0   2.0   4.0   5.0   1.0   5.0   5.0 |
|   3.0   2.0   4.0   3.0   2.0   1.0   2.0 |
|   4.0   3.0   2.0   6.0   2.0  -3.0   4.0 |
└   5.0   4.0   3.0   2.0   8.0   1.0   3.0 ┘

MtrxF = mtrxA.multiply(mtrxC) =
┌                       ┐
|  99.0  79.0 172.0 124.0 160.0 |
|  94.0  81.0 193.0 144.0 161.0 |
|  84.0  64.0 153.0 124.0 134.0 |
|  87.0  93.0 149.0 118.0 165.0 |
└ 141.0 111.0 212.0 162.0 226.0 ┘
```

(Output result)

Bonus 구현 문제 -
전치행렬(transposed matrix) 생성 및 반환

```
/**  Class_Mtrx.h */
#include <iostream>
#include <fstream>
using namespace std;
#define MAX_SIZE 100

class Mtrx {
public:
        Mtrx(int num_row, int num_col);
        Mtrx(double *dA, int num_data, int num_row, int num_col);
        Mtrx(istream& fin);
        ~Mtrx(); // destructor
        int getN_row() { return n_row; }
        int getN_col() { return n_col; }
        void fprintMtrx(ostream& fout);
        void setName(string nm) { name = nm;};
        string getName() { return name;};
        Mtrx add(const Mtrx&);
        Mtrx sub(const Mtrx&);
        Mtrx multiply(const Mtrx&);
        Mtrx transpose(); // 전치행렬을 생성 및 반환
private:
        string name;
        int n_row;
        int n_col;
        double **dM;
};
```

전치행렬 생성 및 출력 결과

```
/**  main.c */

#include <iostream>
#include <fstream>
#include "Class_Mtrx.h"
using namespace std;

void main()
{
        ifstream fin;
        ofstream fout;

        . . . .

        Mtrx mtrxA(fin);
        mtrxA.setName("MtrxA");
        int n_row = mtrxA.getN_row();
        int n_col = mtrxA.getN_col();
        mtrxA.fprintMtrx(fout);

        . . . .

        Mtrx mtrxG(mtrxA.getN_col(), mtrxC.getN_row());
        mtrxG = mtrxA.transpose();
        mtrxG.setName("MtrxG = mtrxA.transpose()");
        mtrxG.fprintMtrx(fout);

        fout.close();

} // end of main()
```

```
1   MtrxA =
2   ┌     1.0     2.0     3.0     4.0     5.0     6.0     7.0┐
3   │     2.0     3.0     4.0     5.0     1.0     7.0     8.0│
4   │     3.0     2.0     5.0     3.0     2.0     4.0     6.0│
5   │     4.0     3.0     2.0     7.0     2.0     1.0     9.0│
6   └     5.0     4.0     3.0     2.0     9.0     6.0     9.0┘
7
```

```
45  MtrxG = mtrxA.transpose() =
46  ┌     1.0     2.0     3.0     4.0     5.0┐
47  │     2.0     3.0     2.0     3.0     4.0│
48  │     3.0     4.0     5.0     2.0     3.0│
49  │     4.0     5.0     3.0     7.0     2.0│
50  │     5.0     1.0     2.0     2.0     9.0│
51  │     6.0     7.0     4.0     1.0     6.0│
52  └     7.0     8.0     6.0     9.0     9.0┘
53
54
```

Lab. 3 Oral Test

Q 3.1 C/C++ 프로그래밍에서 사용되는 포인터 (pointer)의 사용 용도에 대하여 설명하라. (핵심포인트: 함수의 호출과 반환에서 사용, 동적 메모리 할당, 자기참조 구조체/클래스)

Q 3.2 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)

Q 3.3 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)

Q 3.4 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)