객체지향형 프로그래밍과 자료구조

Ch 3. C++ 클래스에서의 포인터와 동적 메모리 사용

정보통신공학과

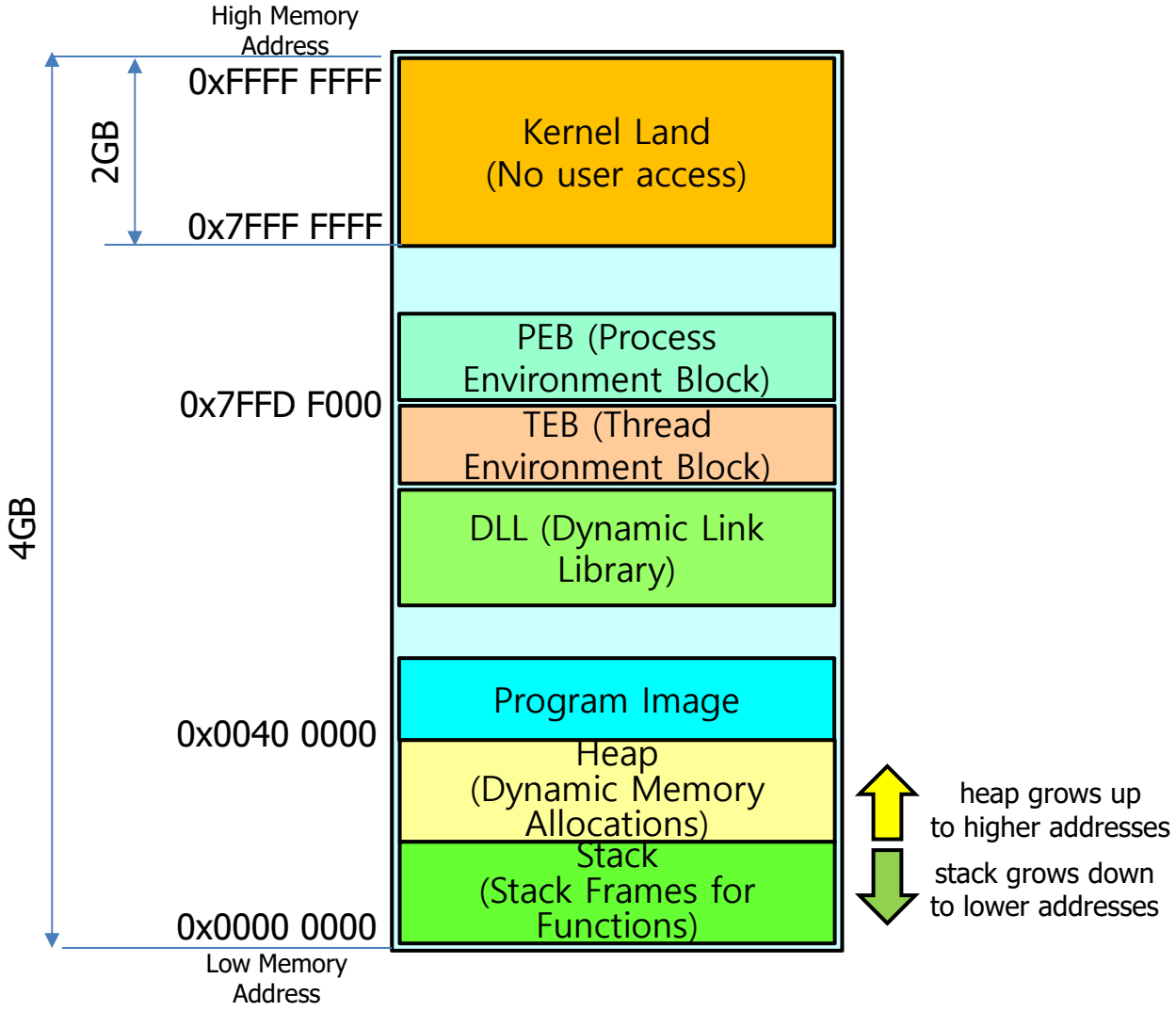
교수 김 영 탁

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

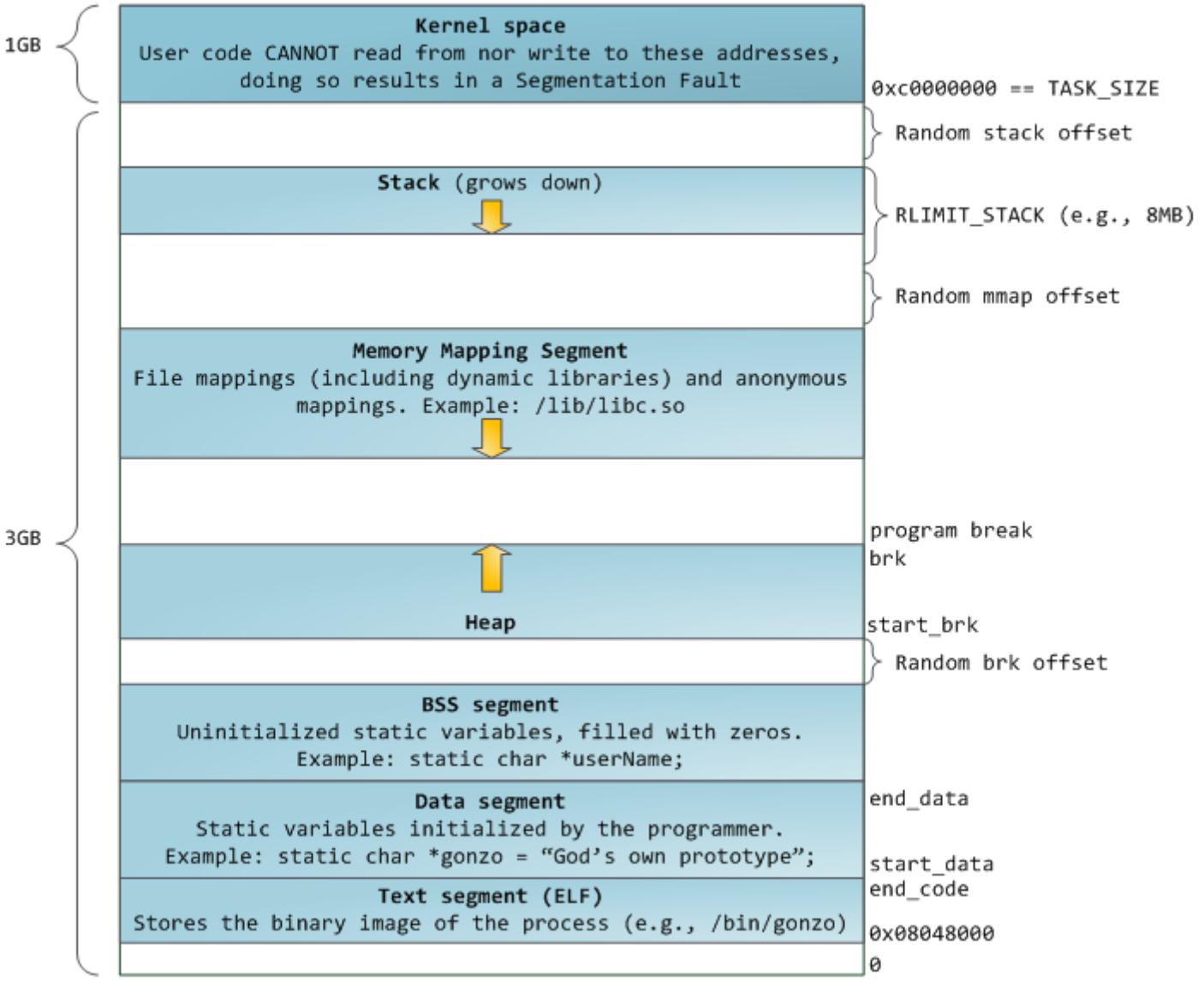
Outline

- 실행중인 프로그램의 가상메모리 맵과 주소
- 포인터 변수
- 포인터 연산
- 클래스 멤버 함수 호출과 반환에서의 포인터 및 참조
- C++ 클래스에서의 동적 메모리 사용
- C++ 클래스에서의 2차원 동적 배열 생성 및 활용 – Class Mtrx
- 자료구조 구성을 위한 자기참조 클래스

Windows 환경에서의 실행중인 프로그램 (Process) 가상 메모리 맵



Linux 운영체제 환경에서의 가상 메모리 맵



포인터, 포인터 연산,
함수 호출과 반환에서의 포인터 사용

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

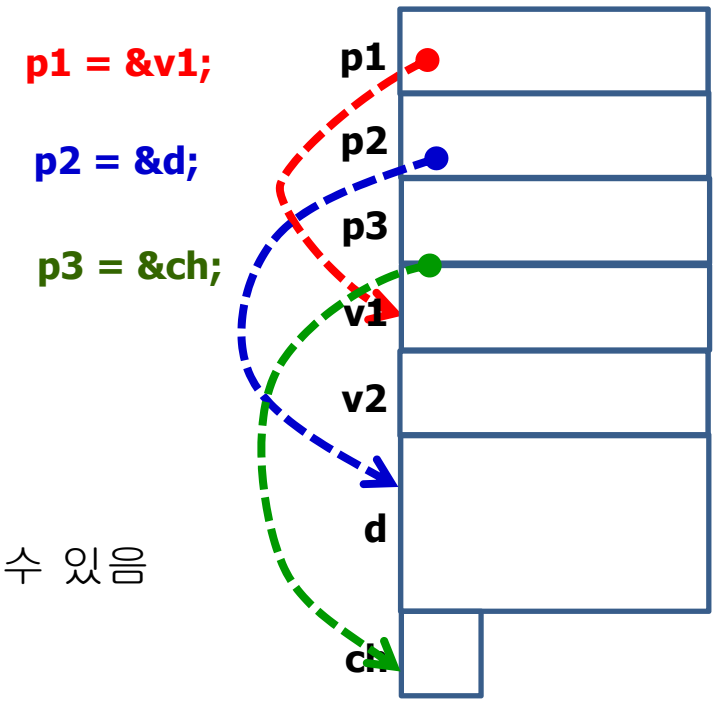
포인터의 자료형, 주소 연산자 (&), 간접참조 연산자 (*)

◆ 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

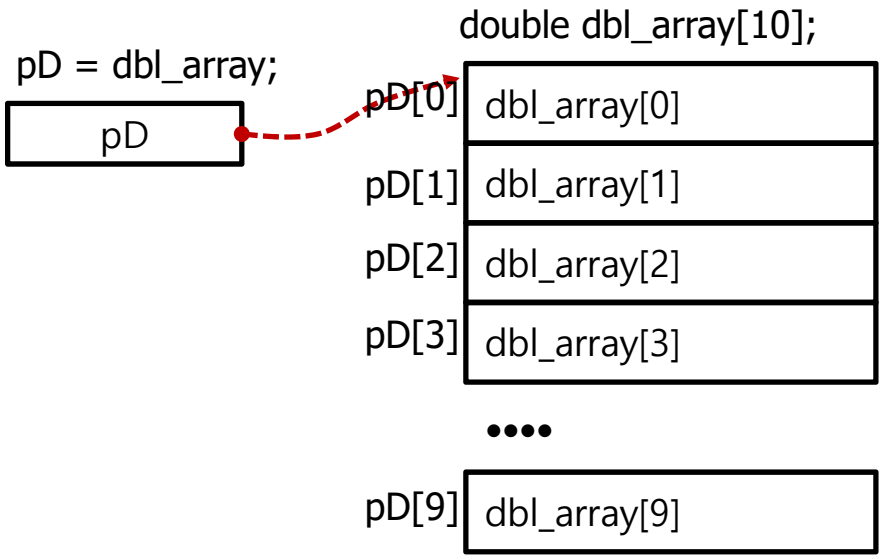
  v1 = 100;
- 포인터와 간접참조 연산자에 의한 접근

  ***p1 = 200;**

포인터 연산

◆ 포인터 연산

- 포인터에 1을 더하는 것: 그 포인터의 자료형 크기만큼 앞으로 이동
- 포인터에 1을 빼는 것: 그 포인터의 자료형 크기만큼 뒤로 이동



포인터 연산 시험

```
void test_pointer_arithmetics()
{
	char *pc = (char *)0x1000;
	int *pi = (int *)0x1000;
	double *pd = (double *)0x1000;
	printf("Incrementing pointers: \n");
	for (int i = 0; i < 5; i++)
	{
		printf("%d: pc (%p), pi(%p), pd(%p)\n", i, pc, pi, pd);
		pc = pc + 1;
		pi = pi + 1;
		pd = pd + 1;
	}
	printf("Decrementing pointers: \n");
	for (int i = 0; i < 5; i++)
	{
		--pc;
		--pi;
		--pd;
		printf("%d: pc (%p), pi(%p), pd(%p)\n", i, pc, pi, pd);
	}
}
```

```
Incrementing pointers:
0: pc (00001000), pi(00001000), pd(00001000)
1: pc (00001001), pi(00001004), pd(00001008)
2: pc (00001002), pi(00001008), pd(00001010)
3: pc (00001003), pi(0000100C), pd(00001018)
4: pc (00001004), pi(00001010), pd(00001020)
Decrementing pointers:
0: pc (00001004), pi(00001010), pd(00001020)
1: pc (00001003), pi(0000100C), pd(00001018)
2: pc (00001002), pi(00001008), pd(00001010)
3: pc (00001001), pi(00001004), pd(00001008)
4: pc (00001000), pi(00001000), pd(00001000)
계속하려면 아무 키나 누르십시오 . . .
```

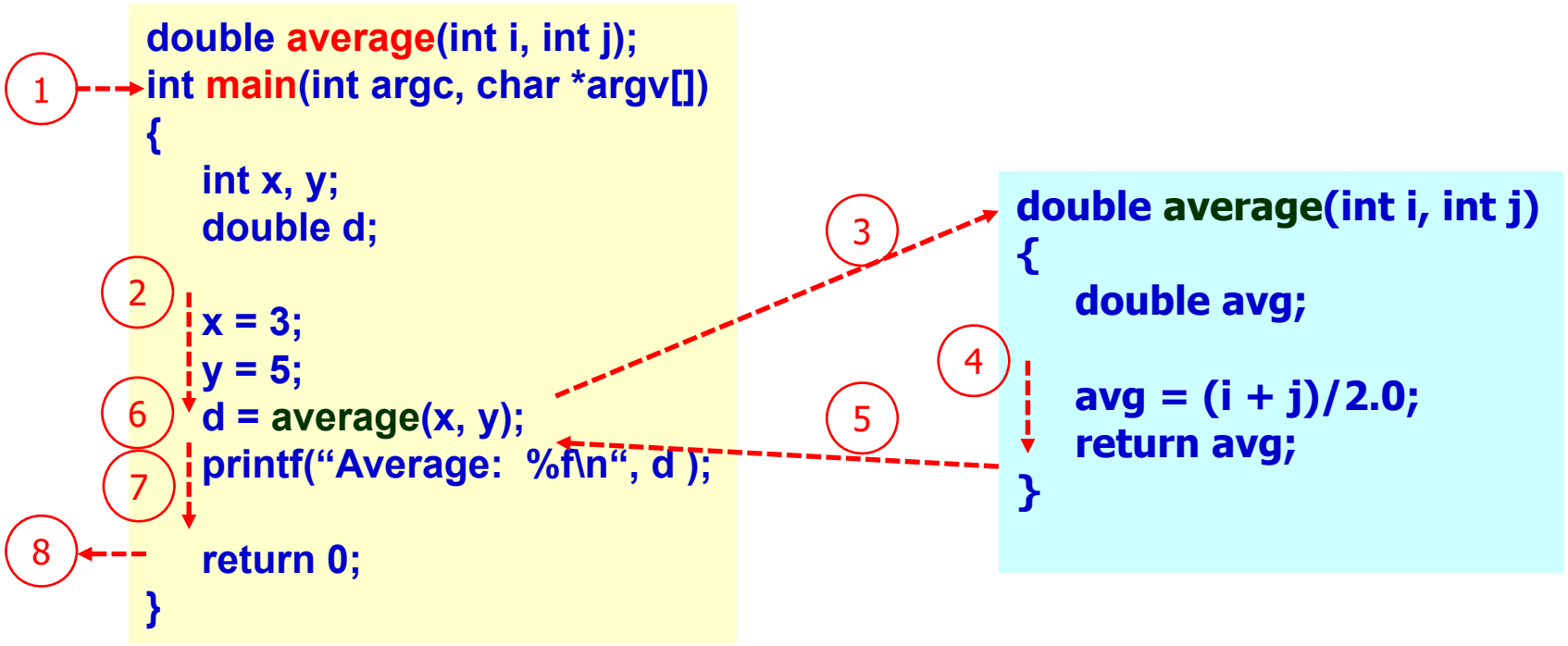
C++ 클래스 멤버함수 호출과 반환에서의 포인터와 참조

◆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.

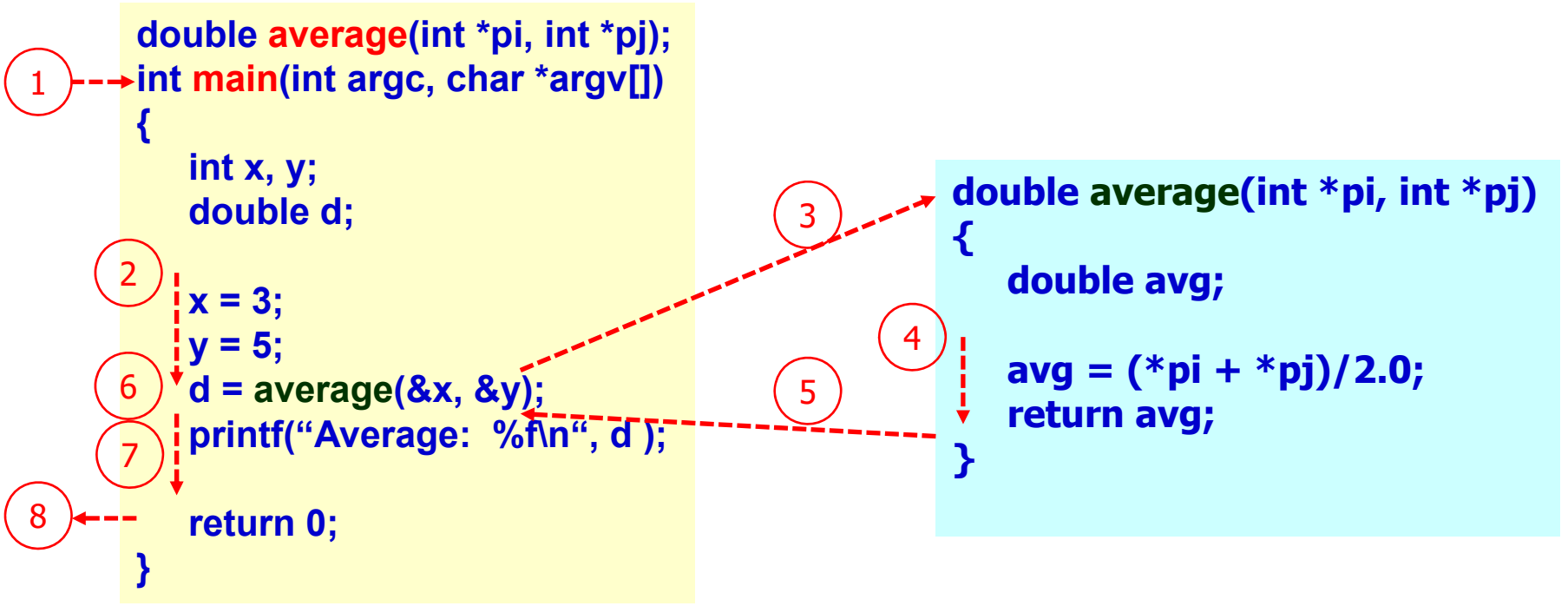
Call-by-Value, Return-by-Value

◆ 인수 (argument)의 값을 복사하여 전달



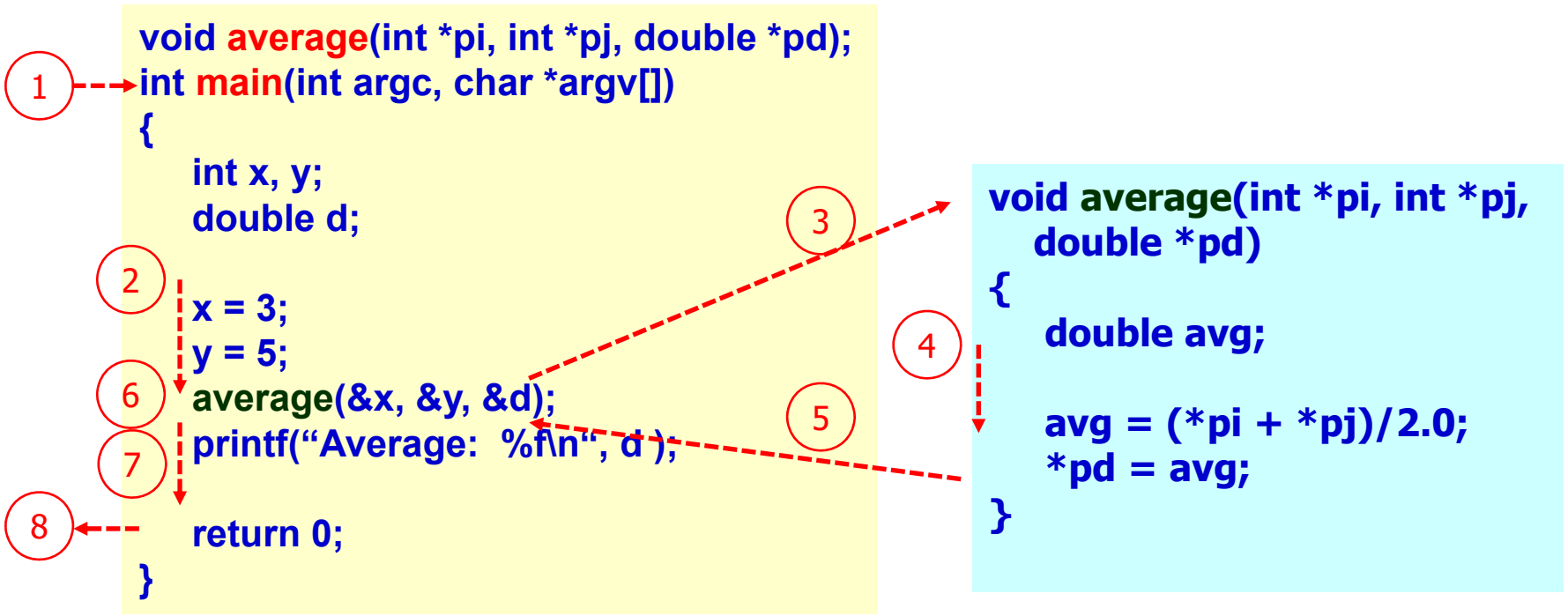
Call-by-Pointer, Return-by-Value

◆ 인수 (argument)의 주소를 전달



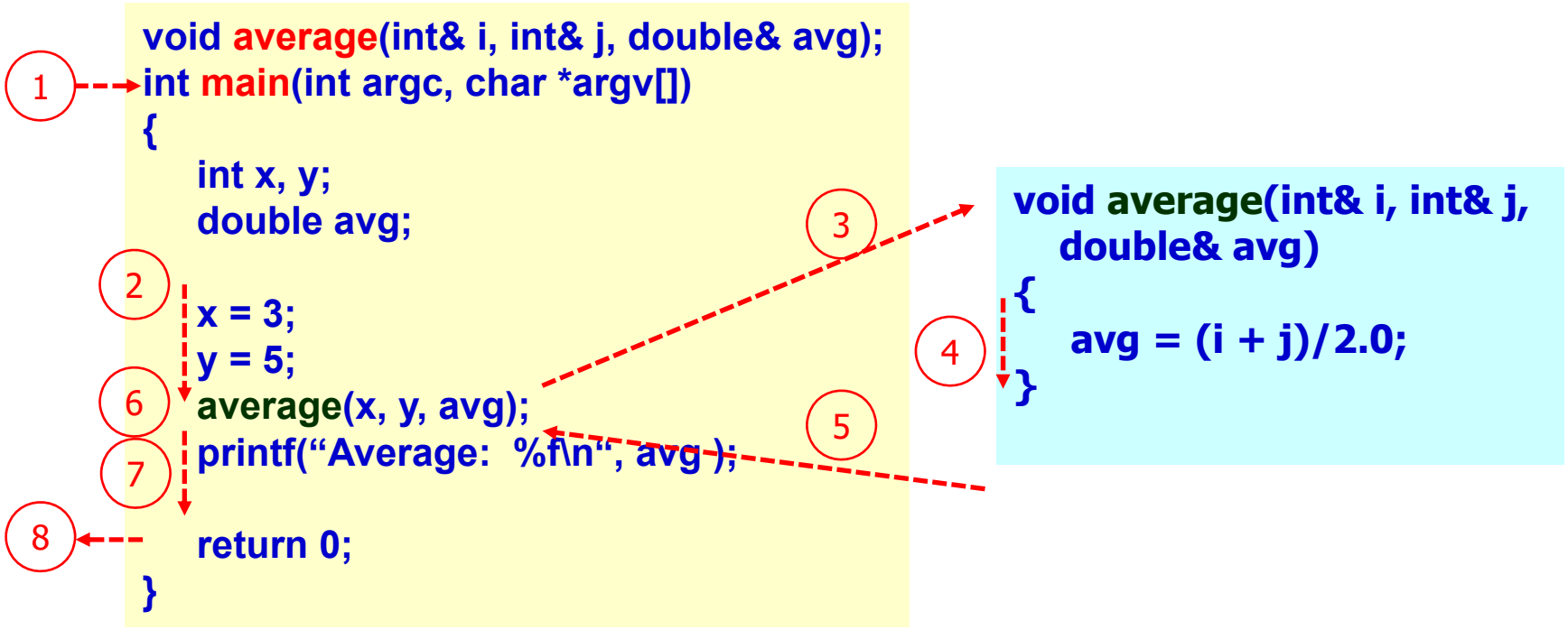
Call-by-Pointer, Return-by-Pointer

◆ 인수(argument)의 주소를 전달



Call-by-Reference

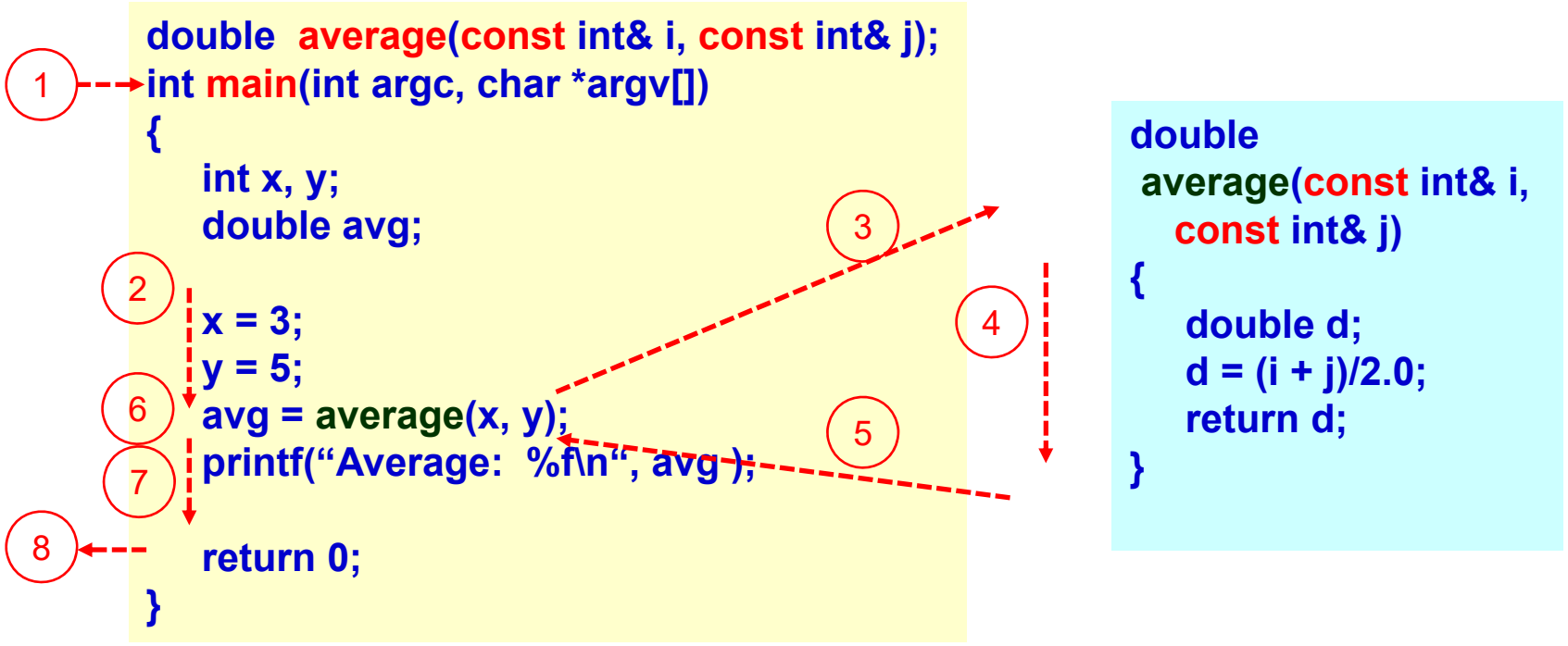
◆ 인수 (argument)의 참조 (reference)를 전달



함수의 호출에서의 const

◆ const

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



Class에서의 동적 메모리 할당,
class BigArray

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

class BigArray

```
/* BigArray.h (1) */
#ifndef BIG_ARRAY_H
#define BIG_ARRAY_H
#include <iostream>
#include <fstream>
using namespace std;

typedef struct
{
    int min;
    int max;
    double avg; // average
    double var; // variance
    double std_dev; // standard deviation
} ArrayStatistics;
```

```
/* BigArray.h (2) */

class BigArray
{
public:
    BigArray(int n); // constructor
    ~BigArray(); // destructor
    void genBigRandArray(int base_offset);
    int size() { return num_elements; }
    void suffle();
    void selection_sort();
    void quick_sort();
    void getStatistics(ArrayStatistics &);
    void fprintStatistics(ostream& fout);
    void fprintBigArray(ostream& fout, int
        elements_per_line);
    void fprintSample(ostream& fout, int
        elements_per_line, int num_sample_lines);
private:
    int *big_array;
    int num_elements;
};
#endif
```

```
/* BigArray.cpp (1) */
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <iomanip>
#include "BigArray.h"
using namespace std;

BigArray::BigArray(int n) // constructor
:num_elements(n)
{
    big_array = (int *) new int[num_elements];
    if (big_array == NULL)
    {
        cout << "Error in creation of dynamic
        array of size (" << num_elements
        << ") !!" << endl;
        exit;
    }
}

BigArray::~BigArray() // destructor
{
    if (big_array != NULL)
       delete[] big_array;
}
```

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

void BigArray::genBigRandArray(int base_offset)
{
    char *flag;
    int count = 0;
    int rand_h, rand_l, big_rand, biased_big_rand;
    srand(time(0));
    flag = (char *) new char[num_elements];
    while (count < num_elements)
    {
        rand_h = rand();
        rand_l = rand();
        big_rand = ((long)rand_h << 15) | rand_l;
        big_rand = big_rand % num_elements;
        if (flag[big_rand] == 1)
           continue;
        else
        {
            flag[big_rand] = 1;
            biased_big_rand = big_rand + base_offset;
            big_array[count] = biased_big_rand;
            count++;
        }
    }
    delete[] flag;
}
```

32,767보다 더 큰 난수의 생성

◆ rand() 함수의 한계

- rand() randomly generates 0 ~ RAND_MAX (32,767) integer value
- if big random numbers (e.g., 0 ~ 500,000) are necessary, rand() cannot be used

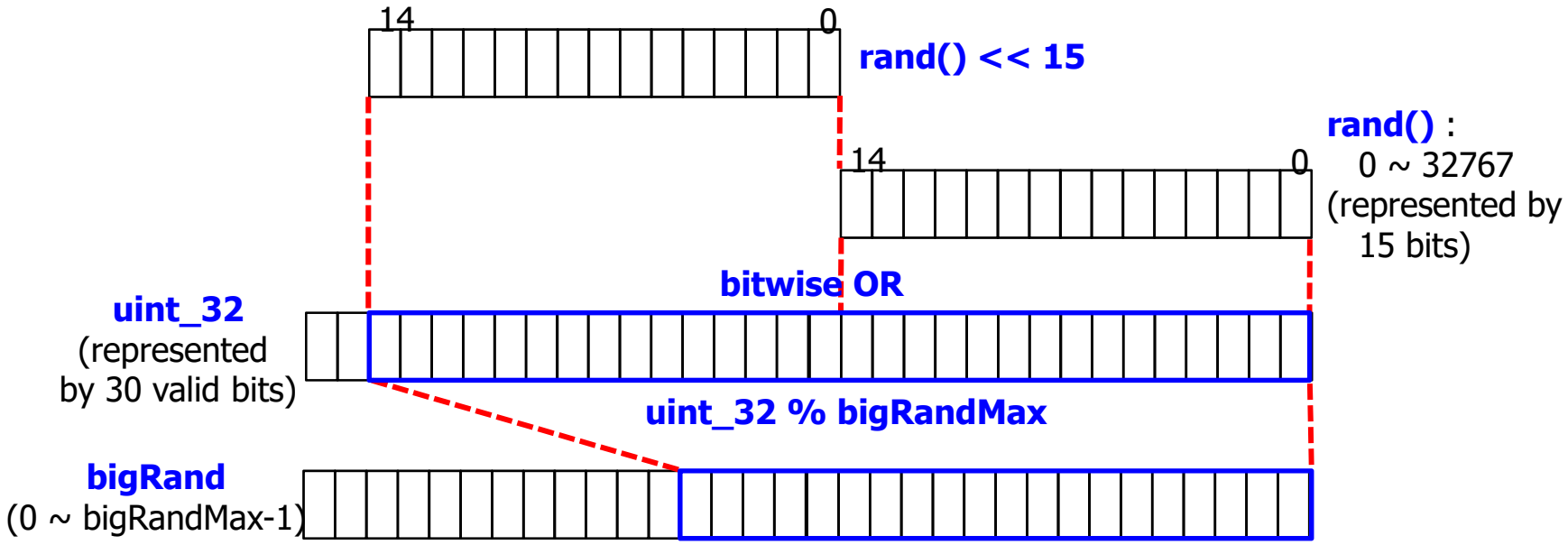
◆ 32,767보다 더 큰 난수로 구성된 배열 생성

- **genBigRandArray(int mA[], int bigRandMax)**
- generates *non-duplicated* big random numbers
  in the range of 0 ~ bigRandMax-1,
  where bigRandMax can be bigger than RAND_MAX (32,767)
- as result, the non-duplicated random numbers are contained in mA[]

genBigRandArray()

◆ Generation of random numbers with bigRandMax (up to 2^{30})

```
unsigned int uint_32, bigRand;

uint_32 = ((unsigned int)rand() << 15) | rand();  // bitwise left shift, bitwise OR
bigRand = uint_32 % bigRandMax;
```



/* BigArray.cpp (3) */

```
void BigArray::getStatistics(ArrayStatistics
     & stats)
{
    int min = INT_MAX;
    int max = INT_MIN;
    double mean = 0.0;
    double sum = 0.0;
    double sq_sum_avg = 0.0;
    double diff_sq_sum = 0.0;
    double var, std_dev;
    int element;

    for (int i = 0; i < num_elements; i++)
    {
        element = big_array[i];
        sum += element;
        if (element > max)
           max = element;
        if (element < min)
           min = element;
    }
    mean = sum / (double)num_elements;
```

/* BigArray.cpp (4) */

```
    diff_sq_sum = 0.0;
    for (int i = 0; i < num_elements; i++)
    {
        element = big_array[i];
        diff_sq_sum +=
            (element - mean)*(element - mean);
    }
    var = diff_sq_sum / (double)num_elements;
    std_dev = sqrt(var);

    stats.min = min;
    stats.max = max;
    stats.avg = mean;
    stats.var = var;
    stats.std_dev = std_dev;
}
```

/* BigArray.cpp (5) */

```
void BigArray::fprintStatistics(ostream& fout)
{
    ArrayStatistics stats;

    fout.setf(ios::fixed);
    fout.setf(ios::showpoint);
    fout.precision(2);

    getStatistics(stats);
    fout << "Statistics: " << endl;
    fout << "  min (" << stats.min << "), max ("
         << stats.max << "), avg (" << stats.avg;
    fout << "), var (" << stats.var << "), std_dev ("
         << stats.std_dev << ")" << endl;
}
```

/* BigArray.cpp (6) */

```
void BigArray::suffle()
{
    srand(time(0));
    int index1, index2;
    int rand_1, rand_2;
    int temp;


    for (int i = 0; i < num_elements; i++)
    {
        rand_1 = rand();
        rand_2 = rand();
        index1 = ((rand_1 << 15) | rand_2)
            % num_elements;
        rand_1 = rand();
        rand_2 = rand();
        index2 = ((rand_1 << 15) | rand_2)
            % num_elements;

        temp = big_array[index1];
        big_array[index1] = big_array[index2];
        big_array[index2] = temp;
    }
}
```

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

void BigArray::selection_sort()
{
    int min; // index of the element with minimum value
    double minValue; // minimum value

    for (int i = 0; i < num_elements - 1; i++)
    {
        min = i;
        minValue = big_array[i];
        for (int j = i + 1; j < num_elements; j++)
        {
            if (minValue > big_array[j])
            {
                min = j;
                minValue = big_array[j];
            }
        }
        if (min != i) // if a smaller element is found, then swap
        {
            /* minValue is dA[min] */
            big_array[min] = big_array[i];
            big_array[i] = minValue;
        }
    }  // end for
}
```

```
/* BigArray.cpp (8) */

int _partition(int *array, int size, int left, int right,
    int pivotIndex, int level)
{
        int pivotValue; // pivot value
        int newPI; // store index
        double temp;
        int i;

#ifdef DEBUG_QUICKSORT
        for (i = 0; i<level; i++) // put indentation
        fout << "  ";
        fout << " Partition (left " << left << ", right "
                << right << ", pivotIdex "
                << pivotIndex << "(pV:"
                 << array[pivotIndex]
                 << ") )  =>";
#endif

        pivotValue = array[pivotIndex];

        temp = array[pivotIndex];
        array[pivotIndex] = array[right];
        array[right] = temp; // Move pivot to end
```

```
/* BigArray.cpp (9) */

        newPI = left;
        for (i = left; i <= (right - 1); i++) {
                if (array[i] <= pivotValue) {
                        temp = array[i];
                        array[i] = array[newPI];
                        array[newPI] = temp;
                        newPI = newPI + 1;
                }
        }

        // swap array[newPI] and array[right];
        // Move pivot to its final place
        temp = array[newPI];
        array[newPI] = array[right];
        array[right] = temp;

        return newPI;
}
```

```
/* BigArray.cpp (10) */

void _quick_sort(int *array, int size, int left, int right, int level)
{
    int pI, newPI; // pivot index

    if (left >= right) {
        return;
    }
    else if (left < right)
    { // subarray of 0 or 1 elements already sorted
        //select a pI (pivotIndex) in the range left ≤ pI ≤ right
        pI = (left + right) / 2;
    }
    newPI = _partition(array, size, left, right, pI, level);

    if (left < (newPI - 1))
    {
        _quick_sort(array, size, left, newPI - 1, level + 1);
        // recursively sort elements on the left of pivotNewIndex
    }
    if ((newPI + 1) < right)
    {
        _quick_sort(array, size, newPI + 1, right, level + 1);
        // recursively sort elements on the right of pivotNewIndex
    }
} // end _quick_sort()
```

```
/* BigArray.cpp (11) */

void BigArray::quick_sort()
{
    int pI, newPI; // pivot index

    _quick_sort(big_array, num_elements, 0, num_elements - 1, 0);

}

void BigArray::fprintBigArray(ostream& fout, int elements_per_line)
{
    int count = 0;
    while (count < num_elements)
    {
            fout << setw(5) << big_array[count];
            count++;
            if (count % elements_per_line == 0)
                  fout << endl;
    }
    fout << endl;
}
```

```
/* BigArray.cpp (12) */

void BigArray::fprintSample(ostream& fout,
     int elements_per_line,
     int num_sample_lines)
{
      int last_block_start;
      int count = 0;

      for (int i = 0; i<num_sample_lines; i++)
      {
            for (int j = 0; j < elements_per_line; j++)
            {
                  if (count > num_elements)
                  {
                        fout << endl;
                        return;
                  }
                  fout << setw(10) << big_array[count];
                  count++;
            }
            fout << endl;
      }

      if (count < (num_elements –
            elements_per_line * num_sample_lines))
          count = num_elements –
            elements_per_line * num_sample_lines;
```

```
/* BigArray.cpp (12) */

      fout << endl << "     . . . . .  " << endl;

      for (int i = 0; i < num_sample_lines; i++)
      {
            for (int j = 0; j< elements_per_line; j++)
            {
                  if (count > num_elements)
                  {
                        fout << endl;
                        return;
                  }
                  fout << setw(10)
                        << big_array[count];
                  count++;
            }
            fout << endl;
      }
      fout << endl;
}
```

```
/* main_BigArray.cpp (1) */

#include <iostream>
#include <fstream>
#include "BigArray.h"

using namespace std;
#define ELEMENTS_PER_LINE 10
#define SAMPLE_LINES 5

void main()
{
      ofstream fout;

      fout.open("output.txt");
      if (fout.fail())
      {
           cout << "Error in opening output.txt !!"
           << endl;
           exit;
      }

      int base_offset = 0;
      int big_rand_size = 5000;
      BigArray ba_1(big_rand_size);
      fout << "Generating big rand array of "
         << ba_1.size() << " elements with
            base_offset " << base_offset << " ... "
         << endl;
```

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

      ba_1.genBigRandArray(base_offset);
      ba_1.fprintSample(fout,
         ELEMENTS_PER_LINE, SAMPLE_LINES);
      ba_1.fprintStatistics(fout);
      ba_1.selection_sort();
      ba_1.fprintSample(fout,
         ELEMENTS_PER_LINE, SAMPLE_LINES);
      cout << endl;

      big_rand_size = 500000;
      base_offset = -big_rand_size / 2;
      BigArray ba_2(big_rand_size);
      fout << endl << "Generating big rand array of "
         << ba_2.size() << " elements with
            base_offset " << base_offset << " ... "
         << endl;
      ba_2.genBigRandArray(base_offset);
      ba_2.fprintSample(fout,
         ELEMENTS_PER_LINE, SAMPLE_LINES);
      ba_2.fprintStatistics(fout);
      ba_2.quick_sort();
      ba_2.fprintSample(fout,
         ELEMENTS_PER_LINE, SAMPLE_LINES);

      fout.close();
}
```

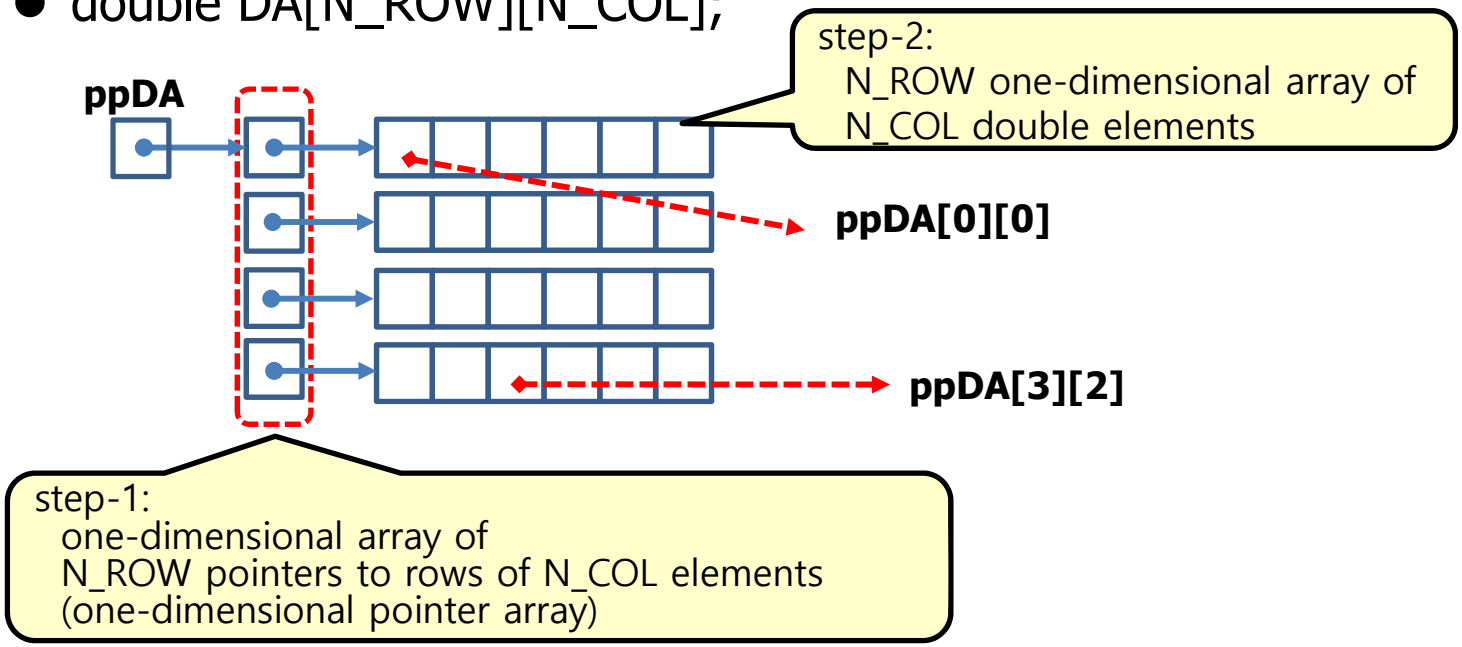
```
Generating big rand array of 5000 elements with base_offset 0 ...
      3996      4867      2452      1096      1514      1102      4151      4460      3079      4155
      2926      2316       564       440       787        96      4454      4766      4756      2004
       933      1860      4188       295      4182      1154      3618        45      4841      4038
      2656      3558      4987      3522       381      2213       182       196      4763      1306
      4060      2708      1866      4448      4136      2850      1683      4759      4245        22

     . . . . .
      3090      4630      2331      1420      1026      2960      3759      2274      3970      4872
       490      3490      3979      4374       448      3315      1279      1816      3773      3955
      4926       316      2618      3633       425      1226       447       997      2585       914
       501      3139      1528      3689      1955      2285      2623      3044      3067      2680
      4890       705      3399      3195      2575      2742      1208      3194        89        57

Statistics:
  min (0), max (4999), avg (2499.50), var (2083333.25), std_dev (1443.38)
         0         1         2         3         4         5         6         7         8         9
        10        11        12        13        14        15        16        17        18        19
        20        21        22        23        24        25        26        27        28        29
        30        31        32        33        34        35        36        37        38        39
        40        41        42        43        44        45        46        47        48        49

     . . . . .
      4950      4951      4952      4953      4954      4955      4956      4957      4958      4959
      4960      4961      4962      4963      4964      4965      4966      4967      4968      4969
      4970      4971      4972      4973      4974      4975      4976      4977      4978      4979
      4980      4981      4982      4983      4984      4985      4986      4987      4988      4989
      4990      4991      4992      4993      4994      4995      4996      4997      4998      4999


Generating big rand array of 500000 elements with base_offset -250000 ...
    168996   -110133   -197548    221096    191514      6102    239151   -165540    173079   -245845
      2926    -32684     90564    220440     35787    220096     29454    214766    -65244   -237996
   -199067    141860   -195812    -19705     29182    191154    118618    -14955    109841     74038
   -112344     58558     84987   -166478   -164619     42213    205182   -134804   -200237   -163694
    244060   -202292   -148134    234448    139136    182850     11683     59759      4245     45022

     . . . . .
   -124959    208366    150559    -63890   -100736    230848     21928    206806    153482     35856
   -114761    -39835   -206293     86156    235761    -16869   -188145    114308   -160619    249689
     41341    -28346   -180795     52447    237011   -151020    190537    103662    111030    -53250
    142320    155181    180389    170009    164304   -248980    206091    198813    109472    201119
     67386    -56752   -138522    130465     94798   -202944   -232586     89387    -56405    -72946

Statistics:
  min (-250000), max (249999), avg (-0.50), var (20833333333.05), std_dev (144337.57)
   -250000   -249999   -249998   -249997   -249996   -249995   -249994   -249993   -249992   -249991
   -249990   -249989   -249988   -249987   -249986   -249985   -249984   -249983   -249982   -249981
   -249980   -249979   -249978   -249977   -249976   -249975   -249974   -249973   -249972   -249971
   -249970   -249969   -249968   -249967   -249966   -249965   -249964   -249963   -249962   -249961
   -249960   -249959   -249958   -249957   -249956   -249955   -249954   -249953   -249952   -249951

     . . . . .
    249950    249951    249952    249953    249954    249955    249956    249957    249958    249959
    249960    249961    249962    249963    249964    249965    249966    249967    249968    249969
    249970    249971    249972    249973    249974    249975    249976    249977    249978    249979
    249980    249981    249982    249983    249984    249985    249986    249987    249988    249989
    249990    249991    249992    249993    249994    249995    249996    249997    249998    249999
```

Class에서의 2차원 동적 메모리 할당, class Mtrx

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;

      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_5x5_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("A");
    fout <<" MtrxA:\n";
    mtrxA.fprintMtrx(fout);

    Mtrx mtrxB(fin);
    mtrxB.setName("B");
    fout <<" MtrxB:\n";
    mtrxB.fprintMtrx(fout);

    int n_row = mtrxA.getN_row();
    int n_col  = mtrxB.getN_col();

    Mtrx mtrxC(n_row, n_col);
    mtrxC.setName("C");
    mtrxC = mtrxA.add(mtrxB);
    fout <<" MtrxC = mtrxA.add(mtrxB) :\n";
    mtrxC.fprintMtrx(fout);

    Mtrx mtrxD(n_row, n_col);
    mtrxD = mtrxA.sub(mtrxB);
    mtrxD.setName("D");
    fout <<" MtrxD = mtrxA.sub(mtrxB) :\n";
    mtrxD.fprintMtrx(fout);
```

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

    Mtrx mtrxE(n_row, n_col);
    mtrxE = mtrxA.multiply(mtrxB);
    fout <<" MtrxE = mtrxA.multiply(mtrxB) :\n";
    mtrxE.fprintMtrx(fout);

    fout.close();
} // end of main()
```

실행 결과

```
5  5
1.0 2.0 3.0 4.0 5.0
2.0 3.0 4.0 5.0 1.0
3.0 2.0 5.0 3.0 2.0
4.0 3.0 2.0 7.0 2.0
5.0 4.0 3.0 2.0 9.0

5  5
1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 1.0
```

(Input data)

```
MtrxA:
┌  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┘

MtrxB:
┌  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┘

MtrxC = mtrxA.add(mtrxB) :
┌  2.00  2.00  3.00  4.00  5.00┐
|  2.00  4.00  4.00  5.00  1.00|
|  3.00  2.00  6.00  3.00  2.00|
|  4.00  3.00  2.00  8.00  2.00|
└  5.00  4.00  3.00  2.00 10.00┘

MtrxD = mtrxA.sub(mtrxB) :
┌  0.00  2.00  3.00  4.00  5.00┐
|  2.00  2.00  4.00  5.00  1.00|
|  3.00  2.00  4.00  3.00  2.00|
|  4.00  3.00  2.00  6.00  2.00|
└  5.00  4.00  3.00  2.00  8.00┘

MtrxE = mtrxA.multiply(mtrxB) :
┌  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┘
```

(Output result)

Class의 정적 (Static) 멤버 함수

정적(Static) 멤버 변수

◆ 정적 멤버 변수 (static member variables)

- 클래스로 부터 파생된 모든 객체 인스탄스들이 글로벌 변수처럼 공유
- 하나의 객체에서 정적 멤버 변수를 변경하면 다른 모든 객체들이 변경된 값을 사용하게 됨

◆ 하나의 클래스로 부터 생성된 객체 인스탄스 들을 관리하는 것에 유용함

- 정적 멤버 함수가 몇 번이나 호출되었는지 관리
- 하나의 클래스로부터 몇 개나 되는 객체들이 생성되었는지 파악

◆ 데이터 멤버와 멤버함수 앞에 키워드 ***static*** 를 표시

정적 멤버 함수 (Static Functions)

◆Member functions can be static

- If no access to object data needed
- And still "must" be member of the class
- Make it a static function

◆Can then be called *outside* the class

- From non-class objects:
  - E.g., Server::getTurn();
- As well as via class objects
  - Standard method: myObject.getTurn();

◆Can only use static data, functions!

Static Members Example

```
/* Person.h */
#include <iostream>
#include <string>
#include "Date.h"
class Person
{
public:
    Person();
    Person(string n, Date bd);
    void setName(string n) { name = n; }
    void setBirthDate(Date bd) { birthDate = bd;}
    string getName() { return name; }
    Date getBirthDate() { return birthDate; }
    void print();
    static void incrementCountPerson() { count_persons++;  }
    static int getCountPersons() { return count_persons; }
private:
    static int count_persons;
    Date birthDate;
    string name;
};
```

```
/* Person.cpp */

#include "Person.h"

int Person::count_persons = 0;

Person:: Person()
: name(string("nobody")), birthDate(Date(1, 1, 1))
{
    incrementCountPerson();
}

Person:: Person(string n, Date bd)
: name(n), birthDate(bd)
{
    incrementCountPerson();
}

void Person::print()
{
   cout << " Person [name: " << name << "; birth date: ";
   birthDate.print();
   cout << "]";
   cout << ", total " << count_persons << " persons are created until now.";
}
```

```
/* main.cpp */

#include <iostream>
#include "Date.h"
#include "Person.h"

using namespace std;

int main()
{
    Person **pPersons;

    pPersons = (Person **)new Person*[NUM_PERSON];
    for (int i = 0; i < NUM_PERSON; i++)
    {
        pPersons[i] = new Person;
        cout << "Person[" << i << "] : ";
        pPersons[i]->print();
        cout << endl;
    }

    cout << "Person[0] : ";
    pPersons[0]->print();
    cout << endl;

    Delete [] pPersons;
    return 0;
}
```

```
Person[0] :  Person [name: nobody; birth date: January 1, 1 (Monday)], total 1 persons are created until now.
Person[1] :  Person [name: nobody; birth date: January 1, 1 (Monday)], total 2 persons are created until now.
Person[2] :  Person [name: nobody; birth date: January 1, 1 (Monday)], total 3 persons are created until now.
Person[3] :  Person [name: nobody; birth date: January 1, 1 (Monday)], total 4 persons are created until now.
Person[4] :  Person [name: nobody; birth date: January 1, 1 (Monday)], total 5 persons are created until now.
Person[0] :  Person [name: nobody; birth date: January 1, 1 (Monday)], total 5 persons are created until now.
계속하려면 아무 키나 누르십시오 . . .
```

자기참조 Class와
C++ 기반 자료구조 개요

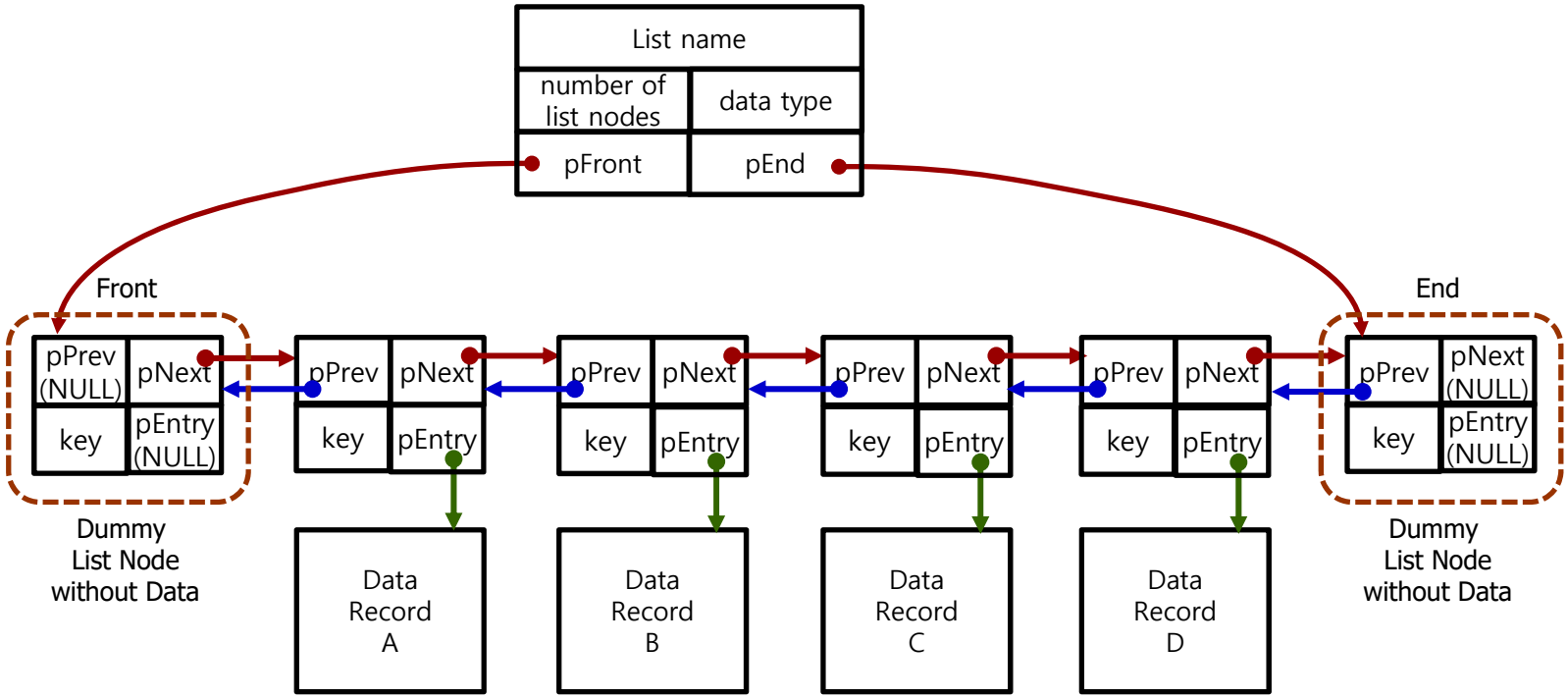
자료구조 구성을 위한 자기참조 클래스

◆ 자기참조 클래스 (self-referential class)

- 클래스의 데이터 멤버 중에 자기 자신과 동일한 구조의 클래스 객체를 가리키는 포인터가 포함되어 있는 클래스
- 예: 연결형 리스트 노드, 이진 트리노드, 스킵 리스트의 쿼드노드

연결형 리스트와 리스트 노드 (1)

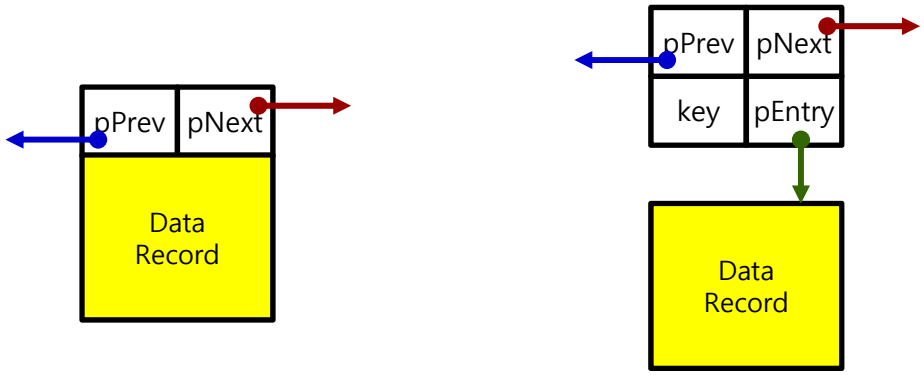
◆ Doubly Linked List (DLL)



연결형 리스트와 리스트 노드 (2)

◆ DLL Node (DLLN)

- pointer to next node (pNext)
- pointer to previous node (pPrev)
- Data record
- or search key and pointer to Data record



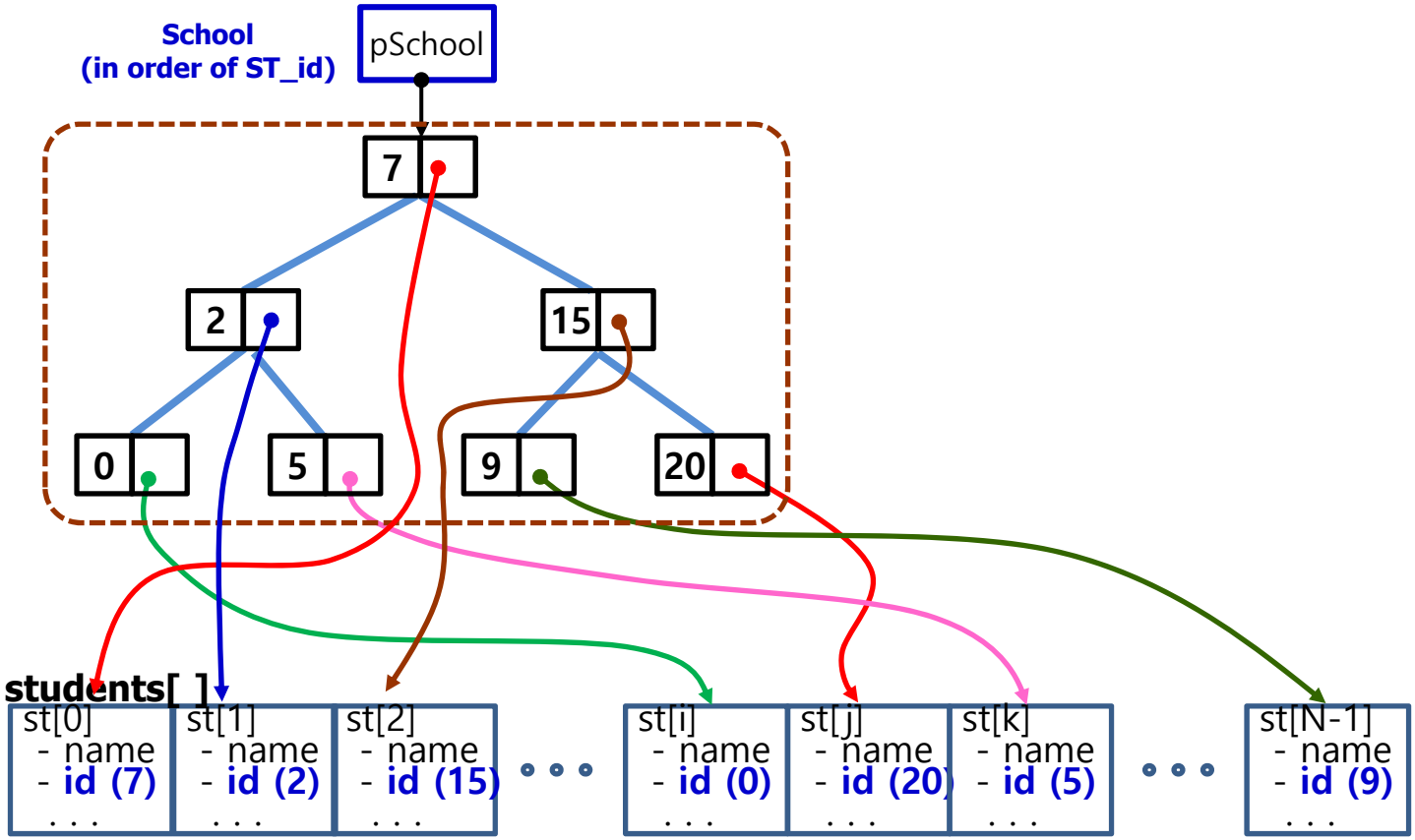
class DLLN

```
/** DLLN.h */
class DLLN // Doubly Linkd List Node
{
public:
    DLLN() {}; // default constructor
    ~DLLN() {}; // destructor
    int getKey() { return key; }
    Entry* getpEntry() { return pEntry; }
    void setpEntry(Entry* pE) {pEntry = pE; }
    DLLN* getPrev() { return prev; }
    DLLN* getNext() { return next; }
    void setPrev(DLLN* pr) { prev = pr; }
    void setNext(DLLN* nx) { next = nx; }
private:
    int key;
    Entry* pEntry;
    DLLN* prev;
    DLLN* next;
};
```

이진 탐색 트리와 트리노드 (1)

◆ Binary tree (이진트리)

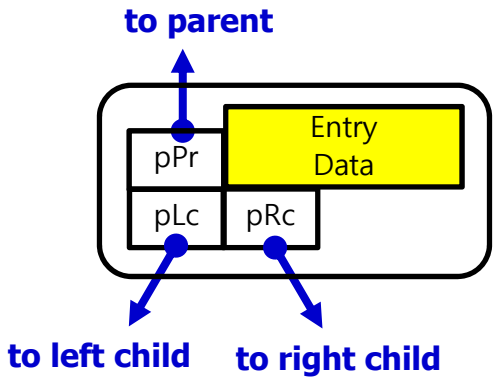
- 비 선형 자료 구조이며, 각 노드는 0, 1, 또는 2개의 다른 노드를 가리킴



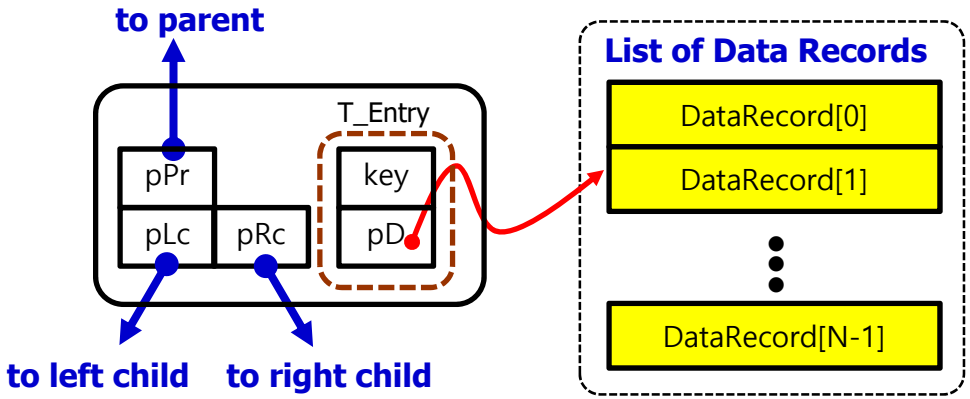
이진 탐색 트리와 트리노드 (2)

◆ Binary Search Tree Node (BSTN)

- pointer to parent (pPr)
- pointer to left child (pLc)
- pointer to right child (pRc)
- Entry data, or
- key and pointer to data record



(a) Binary Tree Node with Data



(b) Binary Tree Node with T_Entry (key, pointer data record)

class BSTN

```
/** Binary Search Tree Node. h */
class BSTN { // a node of the tree
public:
    BSTN() : pEntry(NULL), pPr(NULL), pLc(NULL), pRc(NULL) { }
    // default constructor
    BSTN(Entry *pE) : pEntry(pE), pPr(NULL), pLc(NULL), pRc(NULL) { }
    // constructor
    int getKey() { return key; }
    Entry* getpEntry() { return pEntry; }
    void setpEntry(Entry* pE) { pEntry = pE; }
    BSTN* getpPr() { return pPr; }
    BSTN* getpLc() { return pLc; }
    BSTN* getpRc() { return pRc; }
    BSTN** getppLc() { return &pLc; }
    BSTN** getppRc() { return &pRc; }
    void setpPr(BSTN* pTN) { pPr = pTN; }
    void setpLc(BSTN* pTN) { pLc = pTN; }
    void setpRc(BSTN* pTN) { pRc = pTN; }
private:
    int key; // element value
    Entry *pEntry; // element value
    BSTN* pPr; // parent
    BSTN* pLc; // left child
    BSTN* pRc; // right child
};
```

Homework 3

Homework 3

3.1 정수 데이터를 1,000,000개 담을 수 있는 동적 배열 생성 및 정렬 알고리즘 구현.

1) 강의자료에서 설명한 class BigArray를 사용하여 정수형 데이터 N = 1000000개를 담을 수 있도록 준비하라.
2) class BigArray의 getBigRandArray() 멤버함수를 사용하여 중복 되지 않는 정수형 난수를 1,000,000개 생성하라. 이 난수 생성에서 base_offset은 -500,000으로 설정할 것.
3) class BigArray의 getStatistics() 멤버함수를 사용하여 정수형 난수 1,000,000개의 최소, 최대, 평균, 중위수(median), 분산, 표준편차 값을 계산하라.
4) class BigArray의 quick_sort() 함수를 사용하여 동적으로 생성된 정수형 난수 배열을 정렬하고, 이 때 걸린 시간을 측정하고, 출력하라. 정렬된 정수 배열의 첫 부분과 마지막 부분의 샘플을 fprintSample() 함수를 사용하여 파일로 출력하라. elements_per_line는 10으로, num_sample_lines는 3으로 설정할 것.
5) class BigArray의 shuffle() 함수를 사용하여 정수형 난수 배열에 포함된 배열 원소들을 뒤섞을 것.
6) class BigArray의 selection_sort() 함수를 사용하여 동적으로 생성된 정수형 난수 배열을 정렬하고, 이 때 걸린 시간을 측정하고, 출력하라. 정렬된 정수 배열의 첫 부분과 마지막 부분의 샘플을 fprintSample() 함수를 사용하여 파일로 출력하라. elements_per_line는 10으로, num_sample_lines는 3으로 설정할 것.

Homework 3

3.2 임의의 크기의 행렬 데이터 (더블 형 실수)를 파일 input.txt로부터 세 번 입력받아 class Mtrx 객체인 mtrx_A, mtrx_B, mtrx_C에 저장하고, 첫 두 행렬의 덧셈, 뺄셈 계산을 실행하고, 첫째와 셋째 행렬의 곱셈 계산을 각각 실행 한 후, 결과 출력 파일 output.txt에 출력하는 C++ 프로그램 설계 및 구현.

1) 5 x 7 크기의 더블 형 행렬 2개와 7 x 5크기의 데이터를 파일에 준비할 것. 각 데이터 파일의 첫 부분에는 행렬의 크기 (즉, 행과 열의 갯수)를 지정하는 정수 2개를 각각 둘 것.

2) 데이터 파일 (input.txt)을 읽기 및 추가가 가능하도록 열고, 3개의 행렬의 데이터를 파일로 부터의 데이터로 초기화 할 것. class Mtrx의 생성자에서 파일로부터 입력 받은 행렬 크기에 따라 동적으로 2차원 배열을 생성한 후 파일 입력 데이터로 초기화할 것.

3) class Mtrx의 add() 함수를 사용하여 첫 두 행렬의 덧셈을 계산하여, 그 결과를 파일에 추가할 것.

4) class Mtrx의 sub() 함수를 사용하여 첫 두 행렬의 뺄셈을 계산하여, 그 결과를 파일에 추가할 것.

5) class Mtrx의 multiply() 함수를 사용하여 첫번째와 세번째 두 행렬의 곱셈을 계산하여, 그 결과를 파일에 추가할 것.