# 2022-2 객체지향 프로그래밍과 자료구조 Exam1B

**2022. 9. 24.**
**담당교수 김영탁**

## Section 1B. Class BigArray (30점)

### 1B.1 class BigArray

```
/* BigArray.h */
. . . .  // 필요한 전처리기 설정

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

class BigArray
{
public:
       BigArray(int size); // constructor
       ~BigArray(); // destructor
       int get_size() { return array_size; }
       void shuffle();
       void genBigRandArray(int offset);
       void selection_sort();
       void quick_sort();
       void getStatistics();
       void fprintStatistics(ostream& fout);
       void fprintSample(ostream& fout, int elements_per_line, int num_sample_lines);
private:
       int *big_array;
       int array_size;
       ArrayStatistics stats;
};
```

- class BigArray는 데이터 멤버로 정수 (integer) pointer인 big_array, 배열의 크기인 array_size, 배열에 포함된 데이터의 통계분석결과 데이터를 저장하는 ArrayStaticstics 구조체 변수를 포함
- 큰 규모의 배열을 사용하기 위한 class BigArray를 Class_BigArray.h에 구현

### 1B.2 class BigArray 멤버함수 구현

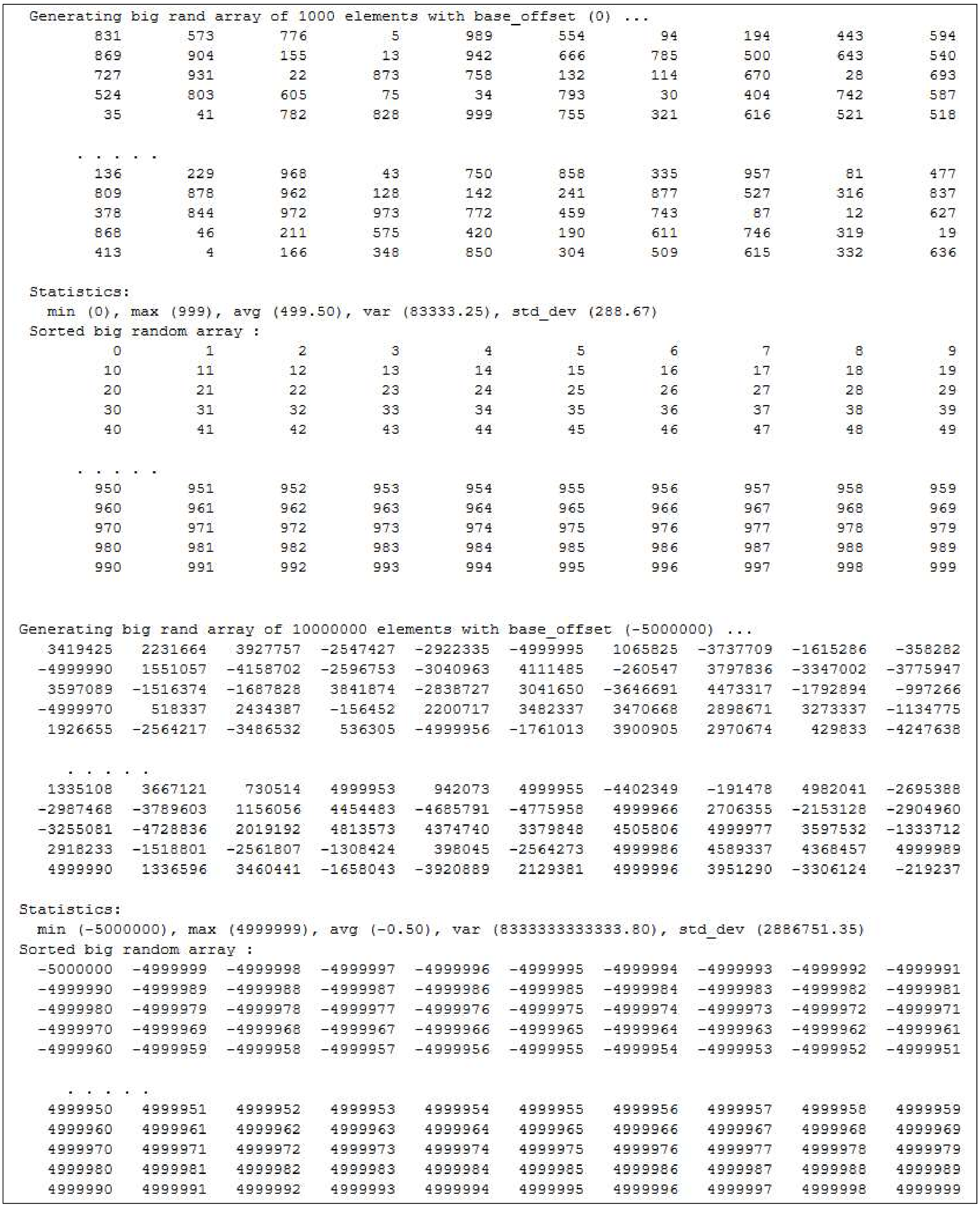
- 생성자 (constructor)는 배열의 크기 (size)를 전달받아, 동적으로 정수 배열을 생성
- 소멸자 (destructor)는 동적으로 생성한 배열을 반환
- get_size() 멤버함수는 array_size를 반환
- shuffle() 멤버함수는 big_array에 저장된 데이터들을 뒤섞어 줌
- genBigRandArray(int offset)은 class BigArray에 설정된 array_size 개수의 정수형 난수를 (0 ~ array_size-1) + offset 구간에서 중복되지 않는 정수형 난수 배열로 구성
- selection_sort()는 big_array에 저장된 데이터들을 선택 정렬 방식으로 오름 차순 정렬
- quick_sort() 멤버함수는 big_array에 저장된 데이터들을 퀵 정렬 방식으로 오름 차순 정렬
- getStatistics() 멤버함수는 big_array에 저장된 데이터들의 통계분석 자료(최소, 최대, 평균, 분산, 표준편차)를 산출하여 데이터 멤버인 stats에 기록
- fprintStatistics() 멤버함수는 big_array에 저장된 데이터들의 통계분석 자료(최소, 최대, 평균, 분산, 표준편차)를 산출하며, 그 결과를 지정된 파일 fout에 출력
- fprintSample() 멤버함수는 big_array에 저장된 데이터들 중 첫 부분과 끝 부분에서 한 줄에 elements_per_line 개 씩 num_sample_lines 줄의 데이터를 출력

- class BigArray의 멤버 함수들은 BigArray.cpp에 구현할 것

### 1B.3 main() 함수 구현

```
/* main_BigArray.cpp */

. . . // 필요한 헤더파일 추가, 전처리기 설정

#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 big_rand_size = 1000;
	int base_offset = 0;
	BigArray ba_1(big_rand_size);
	fout << "Generating big rand array of " << ba_1.get_size()
	     << " elements with base_offset (" << base_offset << ") ... " << endl;
	ba_1.genBigRandArray(base_offset);
	ba_1.fprintSample(fout, ELEMENTS_PER_LINE, SAMPLE_LINES);
	ba_1.fprintStatistics(fout);
	ba_1.selection_sort();
	fout << "Sorted big random array :" << endl;
	ba_1.fprintSample(fout, ELEMENTS_PER_LINE, SAMPLE_LINES);
	cout << endl;

	big_rand_size = 10000000;
	base_offset = -big_rand_size / 2;
	BigArray ba_2(big_rand_size);
	fout << endl << "Generating big rand array of " << ba_2.get_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();
	fout << "Sorted big random array :" << endl;
	ba_2.fprintSample(fout, ELEMENTS_PER_LINE, SAMPLE_LINES);

	fout.close();
}
```

## 1B.4 실행 결과 화면출력



```
 Generating big rand array of 1000 elements with base_offset (0) ...
       831       573       776         5       989       554        94       194       443       594
       869       904       155        13       942       666       785       500       643       540
       727       931        22       873       758       132       114       670        28       693
       524       803       605        75        34       793        30       404       742       587
        35        41       782       828       999       755       321       616       521       518

      . . . . .
       136       229       968        43       750       858       335       957        81       477
       809       878       962       128       142       241       877       527       316       837
       378       844       972       973       772       459       743        87        12       627
       868        46       211       575       420       190       611       746       319        19
       413         4       166       348       850       304       509       615       332       636

 Statistics:
   min (0), max (999), avg (499.50), var (83333.25), std_dev (288.67)
 Sorted big random array :
         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

      . . . . .
       950       951       952       953       954       955       956       957       958       959
       960       961       962       963       964       965       966       967       968       969
       970       971       972       973       974       975       976       977       978       979
       980       981       982       983       984       985       986       987       988       989
       990       991       992       993       994       995       996       997       998       999


Generating big rand array of 10000000 elements with base_offset (-5000000) ...
   3419425   2231664   3927757  -2547427  -2922335  -4999995   1065825  -3737709  -1615286   -358282
  -4999990   1551057  -4158702  -2596753  -3040963   4111485   -260547   3797836  -3347002  -3775947
   3597089  -1516374  -1687828   3841874  -2838727   3041650  -3646691   4473317  -1792894   -997266
  -4999970    518337   2434387   -156452   2200717   3482337   3470668   2898671   3273337  -1134775
   1926655  -2564217  -3486532    536305  -4999956  -1761013   3900905   2970674    429833  -4247638

     . . . . .
   1335108   3667121    730514   4999953    942073   4999955  -4402349   -191478   4982041  -2695388
  -2987468  -3789603   1156056   4454483  -4685791  -4775958   4999966   2706355  -2153128  -2904960
  -3255081  -4728836   2019192   4813573   4374740   3379848   4505806   4999977   3597532  -1333712
   2918233  -1518801  -2561807  -1308424    398045  -2564273   4999986   4589337   4368457   4999989
   4999990   1336596   3460441  -1658043  -3920889   2129381   4999996   3951290  -3306124   -219237

Statistics:
  min (-5000000), max (4999999), avg (-0.50), var (8333333333333.80), std_dev (2886751.35)
Sorted big random array :
  -5000000  -4999999  -4999998  -4999997  -4999996  -4999995  -4999994  -4999993  -4999992  -4999991
  -4999990  -4999989  -4999988  -4999987  -4999986  -4999985  -4999984  -4999983  -4999982  -4999981
  -4999980  -4999979  -4999978  -4999977  -4999976  -4999975  -4999974  -4999973  -4999972  -4999971
  -4999970  -4999969  -4999968  -4999967  -4999966  -4999965  -4999964  -4999963  -4999962  -4999961
  -4999960  -4999959  -4999958  -4999957  -4999956  -4999955  -4999954  -4999953  -4999952  -4999951

     . . . . .
   4999950   4999951   4999952   4999953   4999954   4999955   4999956   4999957   4999958   4999959
   4999960   4999961   4999962   4999963   4999964   4999965   4999966   4999967   4999968   4999969
   4999970   4999971   4999972   4999973   4999974   4999975   4999976   4999977   4999978   4999979
   4999980   4999981   4999982   4999983   4999984   4999985   4999986   4999987   4999988   4999989
   4999990   4999991   4999992   4999993   4999994   4999995   4999996   4999997   4999998   4999999
```