컴퓨터 프로그래밍 2
-hw08-

학번 : 201602038
제출일 : 2017.4.30.
이름 : 이 미 진

1. 함수 설명

1) AppIO

1-1) AppIO* AppIO_new (void) : AppIO 객체 생성

1-2) void AppIO_delete (AppIO* _this) : 주어진 AppIO 객체 소멸시킴

1-3) void AppIO_out_message (AppIO* _this, char* aMessage) : 주 어진 메시지 문자열 aMessage출력

2) AppController

2-1) AppController* AppController_new(void) : AppController 객체 생성

2-2) void AppController_delete(AppController* _this) : AppController 객체 소멸

2-3) void AppController_run(AppController* _this) : App 실행

2-4) void AppController_generateTestDataByRandomNumbers(AppController* _this, int aTestData[], int aMaxSize) : 난수 생성

2-5) long AppController_timeForUnsortedArrayList_add(AppController* _this, UnsortedArrayList* aList, Element aTestData[], int aTestSize) : 무순 리스트 삽입 시 시간 측정

2-6) long AppController_timeForSortedArrayList_add(AppController* _this, SortedArrayList* aList, Element aTestData[], int aTestSize) : 순서 리스트 삽입 시 시간 측정

2-7) long AppController_timeForUnsortedArrayList_removeMax(AppController* _this, UnsortedArrayList* aList, int aTestSize) : 무순 리스트 최대값 삭제 시 시간 측정

2-8) long AppController_timeForSortedArrayList_removeMax(AppController* _this, SortedArrayList* aList, int aTestSize) : 순서 리스트 최대값 삭제 시 시간 측정

2-9) long AppController_measureTimeOfUnsortedArrayList_min(AppController* _this, UnsortedArrayList* aList, Element aTestData[], int aTestSize) : 무순 리스트 최소값 얻을 때의 시간 측정

2-10) long AppController_measureTimeOfSortedArrayList_min (AppController* _this, SortedArrayList* aList, Element aTestData[], int aTestSize) : 순서 리스트 최소값 얻을 때의 시간 측정

2-11) void AppController_showResults(AppController* _this, int aTestSize, long aTimeForAdd, long aTimeForMin, long aTimeForRemoveMax) : 결과 출력

3) UnsortedArrayList

3-1) UnsortedArrayList* UnsortedArrayList_new (int aMaxSize) : UnsortedArrayList 객체 생성

3-2) void UnsortedArrayList_delete (UnsortedArrayList* _this) : UnsortedArrayList 객체 소멸

3-3) Boolean UnsortedArrayList_isEmpty (UnsortedArrayList* _this) : 배열이 비었을 경우

3-4) Boolean UnsortedArrayList_isFull (UnsortedArrayList* _this) : 배열이 꽉찼을 경우

3-5) Boolean UnsortedArrayList_add (UnsortedArrayList* _this, Element anElement) : 배열의 맨 끝에 삽입

3-6) Element UnsortedArrayList_min (UnsortedArrayList* _this) : 최소값 얻기

3-7) Element UnsortedArrayList_removeMax (UnsortedArrayList* _this): 최대값 삭제

3-8) Element UnsortedArrayList_removeAt(UnsortedArrayList* _this, int aPosition) : 주어진 위치의 원소를 삭제

3-9) int UnsortedArrayList_minPositionRecursively(UnsortedArrayList* _this, int left, int right) : 최소값의 위치를 얻음

3-10) int UnsortedArrayList_maxPositionRecursively (UnsortedArrayList* _this, int left, int right) : 최대값의 위치를 얻음

4) Timer

4-1) Timer* Timer_new(void); // 타이머 생성

4-2) void Timer_delete(Timer* _this); // 타이머 소멸

4-3) void Timer_start(Timer* _this); // 타이머 작동 시작

4-4) void Timer_stop(Timer* _this); // 타이머 작동 중지

4-5) long Timer_duration(Timer* _this); // 타이머 작동 시작부터 중지까지의 시간

5) SortedArrayList

5-1) SortedArrayList* SortedArrayList_new(int aMaxSize) : SortedArrayList 객체 생성

5-2) void SortedArrayList_delete (SortedArrayList* _this) : SortedArrayList 객체 소멸

5-3) Boolean SortedArrayList_isEmpty(SortedArrayList* _this) : 배열이 비었을 경우

5-4) Boolean SortedArrayList_isFull(SortedArrayList* _this) : 배열이 꽉찼을 경우

5-5) Boolean SortedArrayList_add(SortedArrayList* _this, Element anElement) : 배열의 맨 끝에 삽입

5-6) Element SortedArrayList_removeMax(SortedArrayList* _this) : 최대값 삭제

5-7) Element SortedArrayList_min(SortedArrayList* _this) : 최소값 얻기

5-7) int SortedArrayList_positionUsingBinarySearch(SortedArrayList* _this, Element anElement) : 삽입 위치 찾기

5-8) void SortedArrayList_addAt(SortedArrayList* _this, Element anElement, int aPosition) : 주어진 위치에 원소 삽입

5-9) Element SortedArrayList_removeAt(SortedArrayList* _this, int aPosition) : 주어진 위치에서 원소 삭제

2. 전체 코드

1) 8주차_1

1-1) main.c

```
//
//  main.c
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include <stdio.h>
#include "AppController.h"

int main(void) {
    AppController* appController = AppController_new(); //
appController 객체 생성
    AppController_run(appController);   // 프로그램 실행
    AppController_delete(appController); // appController 객체 소멸

    return 0;
}
```

1-2) AppIO.c

```
//
//  AppIO.c
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "AppIO.h"
#include "Common.h"

struct _AppIO{
    // 이곳에 선언할 감추어진 속성이 없음
};

AppIO* AppIO_new(){    // AppIO 객체 생성
    AppIO* _this = NewObject(AppIO);

    return _this;
}
```

```
void AppIO_delete(AppIO* _this){  // 주어진 AppIO 객체 소멸시킴
    free(_this);
}

void AppIO_out_message(AppIO* _this, char* aMessage){  // 주어진 메시지 문자열 aMessage출력
    printf("%s", aMessage);
}
```

1-3) AppIO.h

```
//
//  AppIO.h
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppIO_h
#define AppIO_h

#include <stdio.h>

typedef struct _AppIO AppIO;

AppIO* AppIO_new(); //AppIO 객체 생성
void AppIO_delete(AppIO* _this);   // 주어진 AppIO 객체 소멸시킴
void AppIO_out_message(AppIO* _this, char* aMessage);   //주어진 메시지 문자열 aMessage출력

#endif /* AppIO_h */
```

1-4) AppController.c

```
//
//  AppController.c
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//


#include "AppController.h"
```

```c
#include "Common.h"
#include "AppIO.h"
#include "UnsortedArrayList.h"
#include "Message.h"
#include "Timer.h"
#include <time.h>

#define MIN_TEST_SIZE   1000
#define NUMBER_OF_TESTS 5
#define TEST_SIZE_INTERVAL  1000
#define MAX_DATA_SIZE   MIN_TEST_SIZE +
(TEST_SIZE_INTERVAL*(NUMBER_OF_TESTS-1))



void AppController_generateTestDataByRandomNumbers(AppController*
_this, int aTestData[], int aMaxSize);    //난수 생성
long AppController_timeForUnsortedArrayList_add(AppController*
_this, UnsortedArrayList* aList, Element aTestData[], int
aTestSize);    //무순 리스트 삽입 시 시간 측정

long
AppController_timeForUnsortedArrayList_removeMax(AppController*
_this, UnsortedArrayList* aList, int aTestSize);   //무순 리스트 최대값
삭제 시 시간 측정

long
AppController_measureTimeOfUnsortedArrayList_min(AppController*
_this, UnsortedArrayList* aList, Element aTestData[], int
aTestSize);  // 무순 리스트 최소값 얻을 때의 시간 측정

void AppController_showResults(AppController* _this, int
aTestSize, long aTimeForAdd, long aTimeForMin, long
aTimeForRemoveMax);    // 결과 출력


struct _AppController{
    AppIO* _appIO;
};

AppController* AppController_new(){    // AppController 객체 생성
    AppController* _this = NewObject(AppController);
    _this->_appIO = AppIO_new();
    return _this;
}

void AppController_delete(AppController* _this){   //
AppController 객체 소멸
    AppIO_delete(_this->_appIO);
    free(_this);
```

```
}

void AppController_run(AppController* _this){   // 프로그램 실행
    int testData[MAX_DATA_SIZE];
    int testSize = MIN_TEST_SIZE;
    double timeForAdd, timeForMin, timeForRemoveMax;

    AppIO_out_message(_this->_appIO,
MSG_StartPerformanceMeasuring);
    AppController_generateTestDataByRandomNumbers(_this, testData,
MAX_DATA_SIZE);

    AppIO_out_message(_this->_appIO,
MSG_TitleForUnsortedArrayList);

    for(int i = 0 ; i < NUMBER_OF_TESTS ; i++){
        UnsortedArrayList* listForTest =
UnsortedArrayList_new(MAX_DATA_SIZE);
        timeForAdd =
AppController_timeForUnsortedArrayList_add(_this, listForTest,
testData, testSize);
        timeForMin =
AppController_measureTimeOfUnsortedArrayList_min(_this,
listForTest, testData, testSize);
        timeForRemoveMax =
AppController_timeForUnsortedArrayList_removeMax(_this,
listForTest, testSize);
        AppController_showResults(_this, testSize, timeForAdd,
timeForMin, timeForRemoveMax);
        UnsortedArrayList_delete(listForTest);
        testSize += TEST_SIZE_INTERVAL;
    }

    AppIO_out_message(_this->_appIO, MSG_EndPerformanceMeasuring);
}

void AppController_generateTestDataByRandomNumbers(AppController*
_this, int aTestData[], int aMaxSize){    // 난수 생성
    srand((unsigned)time(NULL));
    for(int i = 0;i < aMaxSize;i++){
        aTestData[i] = rand();
    }
}

long AppController_timeForUnsortedArrayList_add(AppController*
_this, UnsortedArrayList* aList, Element aTestData[], int
aTestSize){    //  무순 리스트 삽입 시 시간 측정
    long duration = 0;
    Timer* timer = Timer_new();

    for(int i = 0 ; i <aTestSize ; i++){
```

```
        Timer_start(timer);
        if(!UnsortedArrayList_isFull(aList)){
            UnsortedArrayList_add(aList, aTestData[i]);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);

    return duration;
}



long AppController_measureTimeOfUnsortedArrayList_min
(AppController* _this, UnsortedArrayList* aList, Element
aTestData[],int aTestSize){  // 최소값 얻을 때의 시간 측정
    int i;
    Element min;
    long duration = 0;

    Timer* timer = Timer_new();

    for(i = 0 ; i < aTestSize; i++){
        //이곳의 측정에서는 리스트의 내용이 변하지 않은 상태에서 동일한 행위를 반복한
다
        Timer_start(timer);
        if(!UnsortedArrayList_isEmpty(aList)){
            min = UnsortedArrayList_min(aList);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);
    return duration;
}


long
AppController_timeForUnsortedArrayList_removeMax(AppController*
_this, UnsortedArrayList* aList, int aTestSize){
    Element max;
    long duration = 0;

    Timer* timer = Timer_new();
    //리스트의 내용이 변하지 않은 상태에서 동일한 행위를 반복
    for(int i = 0; i <aTestSize ; i++){
        Timer_start(timer);
        if(!UnsortedArrayList_isEmpty(aList)){
```

```
            max = UnsortedArrayList_removeMax(aList);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);

    return duration;
}



void AppController_showResults(AppController* _this, int
aTestSize, long aTimeForAdd, long aTimeForMin, long
aTimeForRemoveMax){
    char results[255];
    sprintf(results, "크기: %4d, 삽입: %ld, 최소값얻기:%ld, 최대값삭제:
%ld\n", aTestSize, aTimeForAdd, aTimeForMin, aTimeForRemoveMax);
    AppIO_out_message(_this->_appIO, results);
}
```

1-5) AppController.h

```
//
//  AppController.h
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppController_h
#define AppController_h

typedef struct _AppController AppController;

AppController* AppController_new(); //AppController 객체 생성
void AppController_delete(AppController* _this);    //
AppController 객체 소멸
void AppController_run(AppController* _this);   // 프로그램 실행

#endif /* AppController_h */
```

1-6) Message.h

```
//
```

```
//  Message.h
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Messages_h
#define Messages_h


#define MSG_StartPerformanceMeasuring "<성능 측정을 시작합니다>\n"
#define MSG_EndPerformanceMeasuring "<성능 측정을 종료합니다>\n"
#define MSG_TitleForUnsortedArrayList "\n\"Unsorted Array List\"의
성능 (단위:마이크로 초)\n"

#endif /* Messages_h */
```

1-7) UnsortedArrayList.c

```
//
//  UnsortedArrayList.c
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//


#include "UnsortedArrayList.h"
int UnsortedArrayList_minPositionRecursively(UnsortedArrayList*
_this, int left, int right);    // 최소값 위치 찾기
int UnsortedArrayList_maxPositionRecursively (UnsortedArrayList*
_this, int left, int right);   // 최대값 위치 찾기
Element UnsortedArrayList_removeAt(UnsortedArrayList* _this, int
aPosition);    // 주어진 위치에서 원소 삭제

struct _UnsortedArrayList{
    int _maxSize;
    int _size;
    Element* _elements;
};

UnsortedArrayList* UnsortedArrayList_new (int aMaxSize){   //
UnsortedArrayList 객체 생성
    UnsortedArrayList* _this = NewObject(UnsortedArrayList);
    _this->_maxSize = aMaxSize;
```

```
    _this->_elements = NewVector(Element, _this->_maxSize);

    _this->_size = 0;
    return _this;
}

void UnsortedArrayList_delete(UnsortedArrayList* _this){  //
UnsortedArrayList 객체 소멸    free(_this);
}

Boolean UnsortedArrayList_isEmpty(UnsortedArrayList* _this){    //
배열이 비었을 경우
    return (_this->_size == 0);
}

Boolean UnsortedArrayList_isFull(UnsortedArrayList* _this){    //
배열이 꽉찼을 경우
    return  (_this->_size == _this->_maxSize);
}

Boolean UnsortedArrayList_add(UnsortedArrayList* _this, Element
anElement){    // 배열의 맨 끝에 삽입
    if (UnsortedArrayList_isFull(_this)) {
        return FALSE;
    }
    else{
        _this->_elements[_this->_size] = anElement;
        (_this->_size)++;
        return TRUE;
    }
}

Element UnsortedArrayList_removeMax(UnsortedArrayList* _this){ //
최대값 삭제
    int maxPosition;
    Element max;

    maxPosition = UnsortedArrayList_maxPositionRecursively(_this,
0,_this->_size-1);
    max = UnsortedArrayList_removeAt(_this, maxPosition);

    return max;
}

int UnsortedArrayList_maxPositionRecursively (UnsortedArrayList*
_this, int left, int right){
    // 최대값 위치를 찾음

    if ( left == right ) {
        return left ;
    }
```

```
    else {
        int mid = (left+right) / 2 ;
        int maxPositionOfLeftPart =
UnsortedArrayList_maxPositionRecursively (_this, left, mid);
        int maxPositionOfRightPart =
UnsortedArrayList_maxPositionRecursively (_this, mid+1, right);
        if ( _this->_elements[maxPositionOfLeftPart] >= _this-
>_elements[maxPositionOfRightPart] ) {
            return maxPositionOfLeftPart;
        }
        else {
            return maxPositionOfRightPart;
        }
    }
}

Element UnsortedArrayList_removeAt(UnsortedArrayList* _this, int
aPosition){    // 주어진 위치의 원소를 삭제
    // aPosition의 값은 반드시 _this->_size 값보다 작아야 함.
    Element removedElement = _this->_elements[aPosition];
    for( int i = (aPosition+1); i < (_this->_size); i++ ){
        _this->_elements[i-1] = _this->_elements[i];
    }
    _this->_size--;

    return removedElement;  // 삭제된 값은 return
}


Element UnsortedArrayList_min(UnsortedArrayList* _this){  // 최소값
얻기
    int minPosition;
    minPosition = UnsortedArrayList_minPositionRecursively (_this,
0,_this->_size-1);

    return _this->_elements[minPosition];
}

int UnsortedArrayList_minPositionRecursively(UnsortedArrayList*
_this, int left, int right){
    // 최소값의 위치를 얻음
    int minPosition;

    if ( left == right ) {
        return left ;
    }
    else {
        int mid = (left+right) / 2 ;
        int minPositionOfLeftPart =
UnsortedArrayList_maxPositionRecursively (_this, left, mid);
```

```
        int minPositionOfRightPart =
UnsortedArrayList_maxPositionRecursively (_this, mid+1, right);
        if ( _this->_elements[minPositionOfLeftPart] <= _this-
>_elements[minPositionOfRightPart] ) {
            minPosition = minPositionOfLeftPart;
        }
        else {
            minPosition = minPositionOfRightPart;
        }
    }

    return minPosition;

}
```

1-8) UnsortedArrayList.h

```
//  UnsortedArrayList.h
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef UnsortedArrayList_h
#define UnsortedArrayList_h
#include "Common.h"

typedef struct _UnsortedArrayList UnsortedArrayList;

UnsortedArrayList* UnsortedArrayList_new (int aMaxSize);    //
UnsortedArrayList 객체 생성
void UnsortedArrayList_delete(UnsortedArrayList* _this);    //
UnsortedArrayList 객체 소멸
Boolean UnsortedArrayList_isEmpty(UnsortedArrayList* _this);   //
배열이 비었을 경우
Boolean UnsortedArrayList_isFull(UnsortedArrayList* _this);    //
배열이 꽉찼을 경우
Boolean UnsortedArrayList_add(UnsortedArrayList* _this, Element
anElement);   // 배열의 맨 끝에 삽입
Element UnsortedArrayList_min(UnsortedArrayList* _this);  // 최소값
얻기
Element UnsortedArrayList_removeMax(UnsortedArrayList* _this); //
최대값 삭제
```

```
#endif /* UnsortedArrayList_h */
```

1-9) Timer.c

```
//
//  Timer.c
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Timer.h"
#include <time.h>

Timer* Timer_new(){//타이머 생성
    Timer* _this = NewObject(Timer);
    return _this;
}

void Timer_delete(Timer* _this){    //타이머 소멸
    free(_this);
}

void Timer_start(Timer* _this){ // 타이머 작동 시작
    _this->startCounter = clock();  // 실행 전 카운터 값을 얻음
}

void Timer_stop(Timer* _this){  //타이머 작동 중지
    _this->stopCounter = clock();//실행 후 카운터 값을 얻음
}

long Timer_duration(Timer* _this){//타이머 작동 시작부터 중지까지의 시간
    return (long)(_this->stopCounter - _this-
>startCounter)*(long)1000000 / (long)CLOCKS_PER_SEC;
}
```

1-10) Timer.h

```
//
//  Timer.h
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//
```

```h
#ifndef Timer_h
#define Timer_h
#include "Common.h"

typedef struct {
    clock_t startCounter;
    clock_t stopCounter;
}Timer;

Timer* Timer_new(); //타이머 생성
void Timer_delete(Timer* _this);    //타이머 소멸
void Timer_start(Timer* _this); //타이머 작동 시작
void Timer_stop(Timer* _this);  //타이머 작동 중지
long Timer_duration(Timer* _this);  //타이머 작동 시작부터 중지까지의 시간

#endif /* Timer_h */
```

1-11) Common.h

```h
//
//  Common.h
//  CP2_WEEK8_1
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Common_h
#define Common_h
#include <stdlib.h>

typedef int Element ;

#define NewVector(TYPE,SIZE) (TYPE*)malloc(sizeof(TYPE)*SIZE)
#define NewObject(TYPE) (TYPE*)malloc(sizeof(TYPE))
typedef enum {FALSE,TRUE} Boolean;  // FALSE와 TRUE 값을 갖는 Boolean 선언

#endif /* Common_h */
```

2) 8주차_2

2-1) main.c

```c
//
//  main.c
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include <stdio.h>
#include "AppController.h"

int main(void) {
    AppController* appController = AppController_new(); //
appController 객체 생성
    AppController_run(appController);   // 프로그램 실행
    AppController_delete(appController); // appController 객체 소멸

    return 0;
}
```

2-2) AppController.c

```c
//
//  AppController.c
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "AppController.h"
#include "Common.h"
#include "AppIO.h"
#include "UnsortedArrayList.h"
#include "Message.h"
#include "Timer.h"
#include "SortedArrayList.h"
#include <time.h>

#define MIN_TEST_SIZE   1000
#define NUMBER_OF_TESTS 5
#define TEST_SIZE_INTERVAL  1000
#define MAX_DATA_SIZE   MIN_TEST_SIZE +
(TEST_SIZE_INTERVAL*(NUMBER_OF_TESTS-1))
```

```
void AppController_generateTestDataByRandomNumbers(AppController*
_this, int aTestData[], int aMaxSize);    //난수 생성
long AppController_timeForUnsortedArrayList_add(AppController*
_this, UnsortedArrayList* aList, Element aTestData[], int
aTestSize);    //무순 리스트 삽입 시 시간 측정
long AppController_timeForSortedArrayList_add(AppController*
_this, SortedArrayList* aList, Element aTestData[], int
aTestSize);    // 순서 리스트 삽입 시 시간 측정
long
AppController_timeForUnsortedArrayList_removeMax(AppController*
_this, UnsortedArrayList* aList, int aTestSize);   //무순 리스트 최대값
삭제 시 시간 측정
long AppController_timeForSortedArrayList_removeMax(AppController*
_this, SortedArrayList* aList, int aTestSize);   //순서 리스트 최대값
삭제 시 시간 측정
long
AppController_measureTimeOfUnsortedArrayList_min(AppController*
_this, UnsortedArrayList* aList, Element aTestData[], int
aTestSize);  // 무순 리스트 최소값 얻을 때의 시간 측정
long AppController_measureTimeOfSortedArrayList_min
(AppController* _this, SortedArrayList* aList, Element
aTestData[], int aTestSize); // 순서 리스트 최소값 얻을 때의 시간 측정
void AppController_showResults(AppController* _this, int
aTestSize, long aTimeForAdd, long aTimeForMin, long
aTimeForRemoveMax);    // 결과 출력


struct _AppController{
    AppIO* _appIO;
};

AppController* AppController_new(){    // AppController 객체 생성
    AppController* _this = NewObject(AppController);
    _this->_appIO = AppIO_new();
    return _this;
}

void AppController_delete(AppController* _this){   //
AppController 객체 소멸
    AppIO_delete(_this->_appIO);
    free(_this);
}

void AppController_run(AppController* _this){   // 프로그램 실행
    int testData[MAX_DATA_SIZE];
    int testSize = MIN_TEST_SIZE;
    double timeForAdd, timeForMin, timeForRemoveMax;

    AppIO_out_message(_this->_appIO,
MSG_StartPerformanceMeasuring);
```

```
    AppController_generateTestDataByRandomNumbers(_this, testData,
MAX_DATA_SIZE);

    AppIO_out_message(_this->_appIO,
MSG_TitleForUnsortedArrayList);

    for(int i = 0 ; i < NUMBER_OF_TESTS ; i++){
        UnsortedArrayList* listForTest =
UnsortedArrayList_new(MAX_DATA_SIZE);
        timeForAdd =
AppController_timeForUnsortedArrayList_add(_this, listForTest,
testData, testSize);
        timeForMin =
AppController_measureTimeOfUnsortedArrayList_min(_this,
listForTest, testData, testSize);
        timeForRemoveMax =
AppController_timeForUnsortedArrayList_removeMax(_this,
listForTest, testSize);
        AppController_showResults(_this, testSize, timeForAdd,
timeForMin, timeForRemoveMax);
        UnsortedArrayList_delete(listForTest);
        testSize += TEST_SIZE_INTERVAL;
    }

    testSize = MIN_TEST_SIZE;

    AppIO_out_message(_this->_appIO, MSG_TitleForSortedArrayList);
    for(int i = 0 ; i <NUMBER_OF_TESTS;i++){
        SortedArrayList* listForTest = SortedArrayList_new
(MAX_DATA_SIZE);
        timeForAdd =
AppController_timeForSortedArrayList_add(_this, listForTest,
testData, testSize);
        timeForMin =
AppController_measureTimeOfSortedArrayList_min(_this, listForTest,
testData, testSize);
        timeForRemoveMax =
AppController_timeForSortedArrayList_removeMax(_this, listForTest,
testSize);
        AppController_showResults(_this, testSize, timeForAdd,
timeForMin, timeForRemoveMax);
        SortedArrayList_delete(listForTest);
        testSize += TEST_SIZE_INTERVAL;
    }

    AppIO_out_message(_this->_appIO, MSG_EndPerformanceMeasuring);
}

void AppController_generateTestDataByRandomNumbers(AppController*
_this, int aTestData[], int aMaxSize){    // 난수 생성
    srand((unsigned)time(NULL));
```

```
    for(int i = 0;i < aMaxSize;i++){
        aTestData[i] = rand();
    }
}

long AppController_timeForUnsortedArrayList_add(AppController*
_this, UnsortedArrayList* aList, Element aTestData[], int
aTestSize){    //  무순 리스트 삽입 시 시간 측정
    long duration = 0;
    Timer* timer = Timer_new();

    for(int i = 0 ; i <aTestSize ; i++){
        Timer_start(timer);
        if(!UnsortedArrayList_isFull(aList)){
            UnsortedArrayList_add(aList, aTestData[i]);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);

    return duration;
}

long AppController_timeForSortedArrayList_add(AppController*
_this, SortedArrayList* aList, Element aTestData[], int aTestSize)
{    // 순서 리스트 삽입 시 시간 측정
    long duration = 0;
    Timer* timer = Timer_new();

    for(int i = 0 ; i <aTestSize ; i++){
        Timer_start(timer);
        if(!SortedArrayList_isFull(aList)){
            SortedArrayList_add(aList, aTestData[i]);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);

    return duration;
}

long AppController_measureTimeOfUnsortedArrayList_min
(AppController* _this, UnsortedArrayList* aList, Element
aTestData[],int aTestSize){  // 최소값 얻을 때의 시간 측정
    int i;
    Element min;
    long duration = 0;
```

```
    Timer* timer = Timer_new();

    for(i = 0 ; i < aTestSize; i++){
        //이곳의 측정에서는 리스트의 내용이 변하지 않은 상태에서 동일한 행위를 반복한
다
        Timer_start(timer);
        if(!UnsortedArrayList_isEmpty(aList)){
            min = UnsortedArrayList_min(aList);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);
    return duration;
}

long AppController_measureTimeOfSortedArrayList_min
(AppController* _this, SortedArrayList* aList, Element
aTestData[], int aTestSize){
    int i;
    Element min;
    long duration = 0;

    Timer* timer = Timer_new();

    for(i = 0 ; i < aTestSize; i++){
        //이곳의 측정에서는 리스트의 내용이 변하지 않은 상태에서 동일한 행위를 반복한
다
        Timer_start(timer);
        if(!SortedArrayList_isEmpty(aList)){
            min = SortedArrayList_min(aList);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);
    return duration;
}


long
AppController_timeForUnsortedArrayList_removeMax(AppController*
_this, UnsortedArrayList* aList, int aTestSize){
    Element max;
    long duration = 0;

    Timer* timer = Timer_new();
    //리스트의 내용이 변하지 않은 상태에서 동일한 행위를 반복
```

```
    for(int i = 0; i <aTestSize ; i++){
        Timer_start(timer);
        if(!UnsortedArrayList_isEmpty(aList)){
            max = UnsortedArrayList_removeMax(aList);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);

    return duration;
}


long AppController_timeForSortedArrayList_removeMax(AppController*
_this, SortedArrayList* aList, int aTestSize){
    Element max;
    long duration = 0;

    Timer* timer = Timer_new();
    //리스트의 내용이 변하지 않은 상태에서 동일한 행위를 반복
    for(int i = 0; i <aTestSize ; i++){
        Timer_start(timer);
        if(!SortedArrayList_isEmpty(aList)){
            max = SortedArrayList_removeMax(aList);
        }
        Timer_stop(timer);
        duration += Timer_duration(timer);
    }

    Timer_delete(timer);

    return duration;
}

void AppController_showResults(AppController* _this, int
aTestSize, long aTimeForAdd, long aTimeForMin, long
aTimeForRemoveMax){
    char results[255];
    sprintf(results, "크기: %4d, 삽입: %ld, 최소값얻기:%ld, 최대값삭제:
%ld\n", aTestSize, aTimeForAdd, aTimeForMin, aTimeForRemoveMax);
    AppIO_out_message(_this->_appIO, results);
}
```

2-3) AppController.h

```
//
//  AppController.h
```

```
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppController_h
#define AppController_h

typedef struct _AppController AppController;

AppController* AppController_new(); //AppController 객체 생성
void AppController_delete(AppController* _this);    //
AppController 객체 소멸
void AppController_run(AppController* _this);   // 프로그램 실행

#endif /* AppController_h */
```

2-4) AppIO.c

```
//
//  AppIO.c
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "AppIO.h"
#include "Common.h"

struct _AppIO{
        // 이곳에 선언할 감추어진 속성이 없음
};

AppIO* AppIO_new(){    // AppIO 객체 생성
    AppIO* _this = NewObject(AppIO);

    return _this;
}


void AppIO_delete(AppIO* _this){  // 주어진 AppIO 객체 소멸시킴
    free(_this);
}

void AppIO_out_message(AppIO* _this, char* aMessage){  // 주어진 메시
지 문자열 aMessage출력
    printf("%s", aMessage);
```

```
}
```

2-5) AppIO.h

```
//
//  AppIO.h
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppIO_h
#define AppIO_h

#include <stdio.h>

typedef struct _AppIO AppIO;

AppIO* AppIO_new(); //AppIO 객체 생성
void AppIO_delete(AppIO* _this);   // 주어진 AppIO 객체 소멸시킴
void AppIO_out_message(AppIO* _this, char* aMessage);   //주어진 메시지 문자열 aMessage출력

#endif /* AppIO_h */
```

2-6) Common.h

```
//
//  Common.h
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Common_h
#define Common_h
#include <stdlib.h>

typedef int Element ;

#define NewVector(TYPE,SIZE) (TYPE*)malloc(sizeof(TYPE)*SIZE)
#define NewObject(TYPE) (TYPE*)malloc(sizeof(TYPE))
typedef enum {FALSE,TRUE} Boolean;  // FALSE와 TRUE 값을 갖는 Boolean 선언
```

```
#endif /* Common_h */
```

2-7) Message.h

```
//
//  Message.h
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Messages_h
#define Messages_h


#define MSG_StartPerformanceMeasuring "<성능 측정을 시작합니다>\n"
#define MSG_EndPerformanceMeasuring "<성능 측정을 종료합니다>\n"
#define MSG_TitleForUnsortedArrayList "\n\"Unsorted Array List\"의
성능 (단위:마이크로 초)\n"
#define MSG_TitleForSortedArrayList "\n\"Sorted Array List\"의 성능
(단위:마이크로 초)\n"

#endif /* Messages_h */
```

2-8) UnsortedArrayList.c

```
//
//  UnsortedArrayList.c
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "UnsortedArrayList.h"
int UnsortedArrayList_minPositionRecursively(UnsortedArrayList*
_this, int left, int right);    // 최소값 위치 찾기
int UnsortedArrayList_maxPositionRecursively (UnsortedArrayList*
_this, int left, int right);   // 최대값 위치 찾기
Element UnsortedArrayList_removeAt(UnsortedArrayList* _this, int
aPosition);    // 주어진 위치에서 원소 삭제

struct _UnsortedArrayList{
    int _maxSize;
    int _size;
```

```
    Element* _elements;
};

UnsortedArrayList* UnsortedArrayList_new (int aMaxSize){   //
UnsortedArrayList 객체 생성
    UnsortedArrayList* _this = NewObject(UnsortedArrayList);
    _this->_maxSize = aMaxSize;
    _this->_elements = NewVector(Element, _this->_maxSize);

    _this->_size = 0;
    return _this;
}

void UnsortedArrayList_delete(UnsortedArrayList* _this){  //
UnsortedArrayList 객체 소멸    free(_this);
}

Boolean UnsortedArrayList_isEmpty(UnsortedArrayList* _this){    //
배열이 비었을 경우
    return (_this->_size == 0);
}

Boolean UnsortedArrayList_isFull(UnsortedArrayList* _this){    //
배열이 꽉찼을 경우
    return  (_this->_size == _this->_maxSize);
}

Boolean UnsortedArrayList_add(UnsortedArrayList* _this, Element
anElement){    // 배열의 맨 끝에 삽입
    if (UnsortedArrayList_isFull(_this)) {
        return FALSE;
    }
    else{
        _this->_elements[_this->_size] = anElement;
        (_this->_size)++;
        return TRUE;
    }
}

Element UnsortedArrayList_removeMax(UnsortedArrayList* _this){ //
최대값 삭제
    int maxPosition;
    Element max;

    maxPosition = UnsortedArrayList_maxPositionRecursively(_this,
0,_this->_size-1);
    max = UnsortedArrayList_removeAt(_this, maxPosition);

    return max;
}
```

```
int UnsortedArrayList_maxPositionRecursively (UnsortedArrayList*
_this, int left, int right){
      // 최대값 위치를 찾음

    if ( left == right ) {
        return left ;
    }
    else {
        int mid = (left+right) / 2 ;
        int maxPositionOfLeftPart =
UnsortedArrayList_maxPositionRecursively (_this, left, mid);
        int maxPositionOfRightPart =
UnsortedArrayList_maxPositionRecursively (_this, mid+1, right);
        if ( _this->_elements[maxPositionOfLeftPart] >= _this-
>_elements[maxPositionOfRightPart] ) {
            return maxPositionOfLeftPart;
        }
        else {
            return maxPositionOfRightPart;
        }
    }
}

Element UnsortedArrayList_removeAt(UnsortedArrayList* _this, int
aPosition){    // 주어진 위치의 원소를 삭제
    // aPosition의 값은 반드시 _this->_size 값보다 작아야 함.
    Element removedElement = _this->_elements[aPosition];
    for( int i = (aPosition+1); i < (_this->_size); i++ ){
        _this->_elements[i-1] = _this->_elements[i];
    }
    _this->_size--;

    return removedElement;  // 삭제된 값은 return
}


Element UnsortedArrayList_min(UnsortedArrayList* _this){  // 최소값
얻기
    int minPosition;
    minPosition = UnsortedArrayList_minPositionRecursively (_this,
0,_this->_size-1);

    return _this->_elements[minPosition];
}

int UnsortedArrayList_minPositionRecursively(UnsortedArrayList*
_this, int left, int right){
    // 최소값의 위치를 얻음
    int minPosition;
```

```
    if ( left == right ) {
        return left ;
    }
    else {
        int mid = (left+right) / 2 ;
        int minPositionOfLeftPart =
UnsortedArrayList_maxPositionRecursively (_this, left, mid);
        int minPositionOfRightPart =
UnsortedArrayList_maxPositionRecursively (_this, mid+1, right);
        if ( _this->_elements[minPositionOfLeftPart] <= _this-
>_elements[minPositionOfRightPart] ) {
            minPosition = minPositionOfLeftPart;
        }
        else {
            minPosition = minPositionOfRightPart;
        }
    }

    return minPosition;

}
```

2-9) UnsortedArrayList.h

```
//
//  UnsortedArrayList.h
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef UnsortedArrayList_h
#define UnsortedArrayList_h
#include "Common.h"

typedef struct _UnsortedArrayList UnsortedArrayList;

UnsortedArrayList* UnsortedArrayList_new (int aMaxSize);    //
UnsortedArrayList 객체 생성
void UnsortedArrayList_delete(UnsortedArrayList* _this);    //
UnsortedArrayList 객체 소멸
Boolean UnsortedArrayList_isEmpty(UnsortedArrayList* _this);   //
배열이 비었을 경우
Boolean UnsortedArrayList_isFull(UnsortedArrayList* _this);    //
배열이 꽉찼을 경우
Boolean UnsortedArrayList_add(UnsortedArrayList* _this, Element
anElement);   // 배열의 맨 끝에 삽입
```

```
Element UnsortedArrayList_min(UnsortedArrayList* _this);  // 최소값 얻기
Element UnsortedArrayList_removeMax(UnsortedArrayList* _this); // 최대값 삭제


#endif /* UnsortedArrayList_h */
```

2-10) SortedArrayList.c

```
//
//  SortedArrayList.c
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "SortedArrayList.h"

struct _SortedArrayList{
    int _maxSize;
    int _size;
    Element* _elements;
};

int SortedArrayList_positionUsingBinarySearch(SortedArrayList* _this, Element anElement);   // 삽입 위치 찾기
void SortedArrayList_addAt(SortedArrayList* _this, Element anElement, int aPosition);   // 주어진 위치에 삽입
Element SortedArrayList_removeAt(SortedArrayList* _this, int aPosition);    // 주어진 위치에서 원소 삭제

SortedArrayList* SortedArrayList_new(int aMaxSize){  // SortedArrayList 객체 생성
    SortedArrayList* _this = NewObject(SortedArrayList);
    _this->_maxSize = aMaxSize;
    _this->_elements = NewVector(Element, _this->_maxSize);

    _this->_size = 0;
    return _this;
}
void SortedArrayList_delete(SortedArrayList* _this){  // SortedArrayList 객체 소멸
    free(_this);
}

//삽입 위치 찾기
```

```
int SortedArrayList_positionUsingBinarySearch(SortedArrayList*
_this, Element anElement){
    int left = 0;
    int right = _this->_size -1;
    int mid;
    while (left <= right){
        mid = (left + right) / 2;
        if(anElement == _this->_elements[mid])
            return mid;
        else if(anElement < _this->_elements[mid])
            return mid -1;
        else if(anElement > _this->_elements[mid])
            left = mid + 1;
    }
    return  left;
}

Boolean SortedArrayList_isEmpty(SortedArrayList* _this){    // 배열
이 비었을 경우
    return (_this->_size == 0);
}

Boolean SortedArrayList_isFull(SortedArrayList* _this){ // 배열이 꽉
찼을 경우
    return  (_this->_size == _this->_maxSize);
}

// 배열의 맨 끝에 삽입
Boolean SortedArrayList_add(SortedArrayList* _this, Element
anElement){
    if (SortedArrayList_isFull(_this))
        return FALSE ;
    else {
        //삽입할 위치를 결정한다.
        int positionForAdd =
SortedArrayList_positionUsingBinarySearch (_this, anElement);
        // 찾아진 삽입 위치에 주어진 원소를 삽입한다.
        SortedArrayList_addAt (_this, anElement, positionForAdd);
        return TRUE ;
    }
}
//최대값 삭제
Element SortedArrayList_removeMax(SortedArrayList* _this){
    int maxPosition;
    Element max;

    maxPosition = _this->_size-1;
    max = SortedArrayList_removeAt(_this, maxPosition);

    return max;
```

```
}

// 최소값 얻기
Element SortedArrayList_min(SortedArrayList* _this){
    int min;
    min = 0;

    return min;
}

void SortedArrayList_addAt(SortedArrayList* _this, Element
anElement, int aPosition){
    for(int i = (_this->_size-1);i>aPosition;i--){
        _this->_elements[i+1] = _this->_elements[i];
    }
    _this->_elements[aPosition] = anElement;
    (_this->_size)++;
}
// 주어진 위치의 원소를 삭제
Element SortedArrayList_removeAt(SortedArrayList* _this, int
aPosition){
    Element removedElement = _this->_elements[aPosition];

    for(int i = (aPosition+1); i<(_this->_size); i++){
        _this->_elements[i-1] = _this->_elements[i];
    }

    _this->_size--;

    return removedElement;
}
```

2-11) SortedArrayList.h

```
//
//  SortedArrayList.h
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef SortedArrayList_h
#define SortedArrayList_h
#include "Common.h"
#include "AppController.h"

typedef struct _SortedArrayList SortedArrayList;
```

```
SortedArrayList* SortedArrayList_new(int aMaxSize); //
SortedArrayList 객체 생성
void SortedArrayList_delete(SortedArrayList* _this);   //
SortedArrayList 객체 소멸
Boolean SortedArrayList_isEmpty(SortedArrayList* _this);    // 배열
이 비었을 경우
Boolean SortedArrayList_isFull(SortedArrayList* _this); // 배열이 꽉
찼을 경우
Boolean SortedArrayList_add(SortedArrayList* _this, Element
anElement);     // 배열의 맨 끝에 삽입
Element SortedArrayList_removeMax(SortedArrayList* _this);  // 최대
값 삭제
Element SortedArrayList_min(SortedArrayList* _this);    // 최소값 얻
기

#endif /* SortedArrayList_h */
```

2-12) Timer.c

```
//
//  Timer.c
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Timer.h"
#include <time.h>

Timer* Timer_new(){//타이머 생성
    Timer* _this = NewObject(Timer);
    return _this;
}

void Timer_delete(Timer* _this){    //타이머 소멸
    free(_this);
}

void Timer_start(Timer* _this){ // 타이머 작동 시작
    _this->startCounter = clock();  // 실행 전 카운터 값을 얻음
}

void Timer_stop(Timer* _this){  //타이머 작동 중지
    _this->stopCounter = clock();//실행 후 카운터 값을 얻음
}

long Timer_duration(Timer* _this){//타이머 작동 시작부터 중지까지의 시간
```

```
    return (long)(_this->stopCounter - _this-
>startCounter)*(long)1000000 / (long)CLOCKS_PER_SEC;
}
```

2-13) Timer.h

```
//
//  Timer.h
//  CP2_WEEK8_2
//
//  Created by stu2017s10 on 2017. 4. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Timer_h
#define Timer_h
#include "Common.h"

typedef struct {
    clock_t startCounter;
    clock_t stopCounter;
}Timer;

Timer* Timer_new(); //타이머 생성
void Timer_delete(Timer* _this);    //타이머 소멸
void Timer_start(Timer* _this); //타이머 작동 시작
void Timer_stop(Timer* _this);  //타이머 작동 중지
long Timer_duration(Timer* _this);  //타이머 작동 시작부터 중지까지의 시간

#endif /* Timer_h */
```

3. 전체 설명

1) 8주차_1

1-1) AppController_new() 함수로 appController 객체를 생성한다.

1-2) AppController_run() 함수로 프로그램을 실행한다.

1-3) AppController_run() 함수에서는 AppIO_out_message() 함수에서 MSG_StartPerformanceMeasuring 의 <성능 측정을 시작합니다> 메시지를 출력한다.

1-4) AppController_generateTestDataByRandomNumbers() 함수를 통해 난수를 생성한다.

1-5) AppIO_out_message()함수에서 MSG_TitleForUnsortedArrayList의 "Unsorted Array List \"의 성능 (단위:마이크로 초) 메시지를 출력한다.

1-6) for문을 통해 i=0이고 i가 NUMBER_OF_TESTS(5)가 될 때까지 i를 1씩 증가시키며 함수들을 반복한다.

1-7) UnsortedArrayList_new() 함수를 통해 listForTest 객체를 생성한다.

1-8) AppController_timeForUnsortedArrayList_add() 함수를 통해 무순 리스트에서 삽입 시 시간 측정을 한 값을 timeForAdd에 저장한다.

1-9) AppController_measureTimeOfUnsortedArrayList_min() 함수를 통해 무순 리스트에서 최소값을 얻을 때의 시간 측정을 한 값을 timeForMin에 저장한다.

1-10) AppController_timeForUnsortedArrayList_removeMax() 함수를 통해 무순 리스트에서 최대값을 삭제할 때의 시간 측정을 한 값을 timeForRemoveMax에 저장한다.

1-11) AppController_showResults() 함수를 통해 결과값을 보여준다.

1-12) UnsortedArrayList_delete() 함수를 통해 객체를 소멸시킨다.

1-13) 반복이 끝나면 AppIO_out_message() 함수에서 MSG_EndPerformanceMeasuring의 <성능 측정을 종료합니다> 메시지를 출력한다.

2) 8주차_2

2-1) AppController_new() 함수로 appController 객체를 생성한다.

2-2) AppController_run() 함수로 프로그램을 실행한다.

2-3) AppController_run() 함수에서는 AppIO_out_message() 함수에서 MSG_StartPerformanceMeasuring 의 <성능 측정을 시작합니다> 메시지를 출력한다.

2-4) AppController_generateTestDataByRandomNumbers() 함수를 통해 난수를 생성한다.

2-5) AppIO_out_message()함수에서 MSG_TitleForUnsortedArrayList의 "Unsorted Array List \"의 성능 (단위:마이크로 초) 메시지를 출력한다.

2-6) for문을 통해 i=0이고 i가 NUMBER_OF_TESTS(5)가 될 때까지 i를 1씩 증가시키며 함수들을 반복한다.

2-7) UnsortedArrayList_new() 함수를 통해 listForTest 객체를 생성한다.

2-8) AppController_timeForUnsortedArrayList_add() 함수를 통해 무순 리스트에서 삽입 시 시간 측정을 한 값을 timeForAdd에 저장한다.

2-9) AppController_measureTimeOfUnsortedArrayList_min() 함수를 통해 무순 리스트에서 최소값을 얻을 때의 시간 측정을 한 값을 timeForMin에 저장한다.

2-10) AppController_timeForUnsortedArrayList_removeMax() 함수를 통해 무순 리스트에서 최대값을 삭제할 때의 시간 측정을 한 값을 timeForRemoveMax에 저장한다.

2-11) AppController_showResults() 함수를 통해 결과값을 보여준다.

2-12) UnsortedArrayList_delete() 함수를 통해 객체를 소멸시킨다.

2-13) testSize 의 값을 MIN_TEST_SIZE(5) 로 초기화 시킨다.

2-14) AppIO_out_message() 함수에서 MSG_TitleForSortedArrayList의"Sorted Array List\"의 성능 (단위:마이크로 초) 메시지를 출력

2-15) for문을 통해 i=0이고 i가 NUMBER_OF_TESTS(5)가 될 때까지 i를 1씩 증가시키며 함수들을 반복한다.

2-16) SortedArrayList_new() 함수를 통해 listForTest 객체를 생성한다.

2-17) AppController_timeForSortedArrayList_add() 함수를 통해 순서 리스트에서 삽입 시 시간 측정을 한 값을 timeForAdd에 저장한다.

2-18) AppController_measureTimeOfSortedArrayList_min() 함수를 통해 순서 리스트에서 최소값을 얻을 때의 시간 측정을 한 값을 timeForMin에 저장한다.

2-19) AppController_timeForSortedArrayList_removeMax() 함수를 통해 순서 리스트에서 최대값 삭제 시의 시간 측정을 한 값을 timeForRemoveMax에 저장한다.

2-20) AppController_showResults() 함수를 통해 결과값을 보여준다.

2-21) SortedArrayList_delete() 함수를 통해 객체를 소멸시킨다.

2-22) 모든 반복이 끝나면 AppIO_out_message() 함수에서 MSG_EndPerformanceMeasuring의 <성능 측정을 종료합니다>메시지를 출력한다

4. 실행 결과

1) 8주차_1

```
<성능 측정을 시작합니다>

"Unsorted Array List"의 성능 (단위:마이크로 초)
크기 : 1000, 삽입 : 434, 최소값 얻기 : 17846, 최대값 삭제 : 10169
크기 : 2000, 삽입 : 800, 최소값 얻기 : 72566, 최대값 삭제 : 44483
크기 : 3000, 삽입 : 1223, 최소값 얻기 : 192175, 최대값 삭제 : 97018
크기 : 4000, 삽입 : 1590, 최소값 얻기 : 329548, 최대값 삭제 : 188031
크기 : 5000, 삽입 : 1966, 최소값 얻기 : 527541, 최대값 삭제 : 314647
<성능 측정을 종료합니다>
Program ended with exit code: 0
```

2) 8주차_2

```
<성능 측정을 시작합니다>

"Unsorted Array List"의 성능 (단위:마이크로 초)
크기 : 1000, 삽입 : 421, 최소값 얻기 : 17562, 최대값 삭제 : 10176
크기 : 2000, 삽입 : 826, 최소값 얻기 : 73326, 최대값 삭제 : 44827
크기 : 3000, 삽입 : 1225, 최소값 얻기 : 184790, 최대값 삭제 : 96141
크기 : 4000, 삽입 : 1635, 최소값 얻기 : 327628, 최대값 삭제 : 182191
크기 : 5000, 삽입 : 1978, 최소값 얻기 : 513823, 최대값 삭제 : 272048

"Sorted Array List"의 성능 (단위:마이크로 초)
크기 : 1000, 삽입 : 1128, 최소값 얻기 : 386, 최대값 삭제 : 394
크기 : 2000, 삽입 : 3389, 최소값 얻기 : 652, 최대값 삭제 : 674
크기 : 3000, 삽입 : 6561, 최소값 얻기 : 973, 최대값 삭제 : 1014
크기 : 4000, 삽입 : 10692, 최소값 얻기 : 1307, 최대값 삭제 : 1351
크기 : 5000, 삽입 : 16054, 최소값 얻기 : 1953, 최대값 삭제 : 1846
<성능 측정을 종료합니다>
Program ended with exit code: 0
```

3) 결과 분석

-삽입 : 무순 리스트가 순서 리스트보다 삽입 시 더 적은 시간이 걸린다.
-최소값 얻기 : 순서 리스트가 무순 리스트보다 최소값을 얻는 데 더 적은 시간이 걸린다.
-최대값 삭제 : 순서 리스트가 무순 리스트보다 최대값을 삭제하는 데 더 적은 시간이 걸린다.