자료구조 실습 보고서
[제 6 주] 재귀 : 성적처리

제출일 : 2015 – 04 – 08

201402395 이승희

1. 프로그램 설명서
   1) 주요 알고리즘 / 자료구조 / 기타
      : 한 과목의 성적을 입력 받아 성적순으로 정렬하고, 입력받은 성적들을 통해 평균 ,평균 이상인 학생 수, 최고점, 최저점을 계산한다. 퀵 정렬을 사용할 때 재귀의 개념을 활용하며, 최고점과 최저점도 마찬가지로 재귀의 개념을 활용하여 출력한다.
   2) 함수 설명서
      (1)GradeCounter 의 public member function
      - public GradeCounter() // 학점을 셀 수 있도록 준비를 한다
      - public void count(char aGrade) // 주어진 학점을 받아서 해당 학점의 학생수를 증가시키게 한다.
      - public int numberOfA() // 'A'학점의 학생수를 얻는다
      - public int numberOfB() // 'B' 학점의 학생수를 얻는다
      - public int numberOfC() // 'C' 학점의 학생수를 얻는다
      - public int numberOfD() // 'D' 학점의 학생수를 얻는다
      - public int numberOfF() // 'F' 학점의 학생수를 얻는다

      (2) Ban 의 public member function
      - public Ban(int givenMaxNumOfStudents) // 생성자
      - public int maxSize() // 학급 객체가 가질 수 있는 최대 학생 수를 얻어 알아낸다.
      - public int size() // 학급 객체가 가지고 있는 학생 수를 얻어 알아낸다.
      - public Boolean isEmpty() // 현재 학급이 비어있는지 확인하며 비어 있으면 true 를 반환
      - public Boolean isFull() // 현재 학급이 가득 차 있는지 확인하며 가득 차 있으면 true 를 반환
      - public Boolean add(Student aScore) // 주어진 학생을 Ban 에 저장
      - public void sortStudentsByScore() // 주어진 위치의 학생 객체를 얻는다
      - public void sortStudentsByScore() // 객체에 저장된 학생들의 성적을 성적 순으로 정렬
      - public int minScore() // 학급의 최저점을 얻는다
      - public int maxScore() // 학급의 최고점을 얻는다
      - public float averageScore() // 학급의 평균 점수를 얻는다
      - public int numberOfStudentsAboveAverage() // 평균 이상인 학생 수를 센다
      - public GradeCounter countGrades() // 학점 별 학생수를 센다.
      - private void swap(int a, int b) // a 와 b 를 바꾼다
      - private void quickSortRecursively(int left, int right) // partition 을 수행하여 만들어진 특정 값을 기준으로 크기가 작아진 각각의 부분을 quicksort 한다.

- private int partition(int left, int right) // 작은 원소들은 왼쪽에, 큰 원소들은 오른쪽에 오게 하고, 그 사이에 특정 값(pivot)이 놓이게 한 후 pivot 값을 반환한다.

- private int maxScoreRecursively(int left, int right) // 최고점을 찾아서 return 값으로 돌려준다. 두 개의 구간으로 나누는 재귀 함수로 작성한다.

- private int miinScoreRecursively(int left, int right) // 최저점을 찾아서 return 값으로 돌려준다. 크기를 (N-1)로 줄이는 재귀함수로 작성한다.

- private char scoreToGrade(int aScore) // 점수별 학점을 반환한다.

3) 종합 설명서
: 성적을 입력할 것인가의 여부를 묻고, 입력할 경우 한생의 점수를 입력받는다. 이 때, 성적이 0 보다 작거나 100 보다 크면 오류 메시지를 내보내고 입력은 무시한다. 입력하지 않을 경우 입력을 종료하며, 최대 학생 수 100 명 이상 입력되면, 공간 부족 메시지를 내보내고 종료한다.
성적이 입력되었을 때, 평균점과 평균 이상인 학생의 수 , 최고점과 최저점 및 성적 별 학점을 학생수와 함께 출력한다. 마지막에는 입력된 성적을 성적순으로 출력한다. 성적이 하나도 입력되지 않았을 때는 아무 일도 하지 않고 종료한다.

2. 프로그램 장단점 분석
: 입력받은 성적들을 성적순으로 정렬할 때 퀵 정렬을 사용하였으며 이 때 partition 하여 정렬하였기 때문에 주어진 크기의 배열을 quicksort 하는 문제가 크기가 작아진 두 개의 quicksort 문제로 바뀌었다. 또한 재귀의 특성상 크기를 줄여가다 보면 쉽게 답을 얻을 수 있는 상태에 도달하므로 코드의 양이 확실하게 줄어들었음을 확인할 수 있었다.

3. 실행 결과 분석

1) 입력과 출력

```
<< 성적 처리를 시작합니다 >>
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : 88
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : 63
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : -1
ERROR : 0보다 작거나 100보다 커서, 정상적인 점수가 아닙니다.
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : 30
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : 45
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : 96
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : 102
ERROR : 0보다 작거나 100보다 커서, 정상적인 점수가 아닙니다.
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : Y
- 점수를 입력하시오 : 98
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : N
[ 성적 입력을 종료합니다. ]
평균 점수는 70.0 입니다.
평균 이상인 학생은 모두 3 명 입니다.
최고점은 98점 입니다.
최저점은 30점 입니다.
A 학점은 모두 2 명 입니다.
B 학점은 모두 1 명 입니다.
C 학점은 모두 0 명 입니다.
D 학점은 모두 1 명 입니다.
F 학점은 모두 2 명 입니다.
학생들의 성적순 목록입니다.
점수 : 98
점수 : 96
점수 : 88
점수 : 63
점수 : 45
점수 : 30
<< 성적 처리를 종료합니다 >>
```

2) 결과 분석

: 기본적인 입출력 화면은 위와 같으며, 이 외에도 Y/N 이외의 다른 문자를 입력할 때 다음과 같이 ERROR 메시지와 함께 프로그램이 종료된다.

```
<< 성적 처리를 시작합니다 >>
성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : j
[ 성적 입력을 종료합니다. ]
잘못된 입력입니다.
<< 성적 처리를 종료합니다 >>
```

4. 소스코드

(1) AppController

```
public class AppController {
    private AppView _appView;
    private Ban _ban;

    public AppController() {
        this._appView = new AppView();

    }

    private boolean inputAndStoreStudents() {
        this.showMessage(MessageID.Notice_StartProgram);
        int score;
        boolean storingAStudentWasSuccessful = true;
        this._ban = new Ban();
        while (storingAStudentWasSuccessful
                && this._appView.inputDoesContinueToInputNextStudent()) {
            score = this._appView.inputScore();
            if (score < 0 || score > 100)
                this.showMessage(MessageID.Error_InvalidScore);
            else {
                Student aStudent = new Student(score);
                this._ban.add(aStudent);
            }
        }
        this.showMessage(MessageID.Notice_EndMenu);
        return storingAStudentWasSuccessful;

    }
```

```
    private void showStatistics() {
        this._appView.outputAverageScore(this._ban.averageScore());
        this._appView.outputNumberOfStudentsAboveAverage(this._ban
                .numberOfStudentsAboveAverage());
        this._appView.outputMaxScore(this._ban.maxScore());
        this._appView.outputMinScore(this._ban.minScore());

        GradeCounter gradeCounter = this._ban.countGrades();
        this._appView.outputGradeCountFor('A', gradeCounter.numberOfA());
        this._appView.outputGradeCountFor('B', gradeCounter.numberOfB());
        this._appView.outputGradeCountFor('C', gradeCounter.numberOfC());
        this._appView.outputGradeCountFor('D', gradeCounter.numberOfD());
        this._appView.outputGradeCountFor('F', gradeCounter.numberOfF());
    }

    private void showStudentsSortedByScore() {
        this.showMessage(MessageID.Show_SortedStudentList);
        for (int index = 0; index < this._ban.size(); index++) {
            this._appView.outputStudentInfo(this._ban.elementAt(index).score());
        }
    }

    public void run() {

        this.inputAndStoreStudents();
        if (this._ban.isEmpty() == true)
            this.showMessage(MessageID.Error_Input);
        else {
            this.showStatistics();
            this._ban.sortStudentsByScore();
            this.showStudentsSortedByScore();
        }
        this.showMessage(MessageID.Notice_EndProgram);

    }
     private void showMessage(MessageID aMessageID) {
         switch (aMessageID) {
         case Notice_StartProgram:
             System.out.println("<< 성적 처리를 시작합니다 >>");
             break;
         case Notice_EndProgram:
             System.out.println("<< 성적 처리를 종료합니다 >>");
             break;
         case Notice_StartMenu:
             System.out.print("- 점수를 입력하시오 : ");
             break;
         case Notice_EndMenu:
             System.out.println("[ 성적 입력을 종료합니다. ]");
             break;
         case Error_WrongMenu:
             System.out.println("잘못된 입력입니다.");
             break;
         case Show_SortedStudentList:
             System.out.println("학생들의 성적순 목록입니다.");
             break;
         case Error_InvalidScore:
             System.out.println("ERROR : 0보다 작거나 100보다 커서, 정상적인 점수가 아닙니다.");
             break;
         case Error_Input:
             System.out.println("잘못된 입력입니다.");
             break;

         }

     }
}
```

(2) AppView

```java
import java.util.*;

public class AppView {
    private Scanner _scanner;

    public AppView() {
        this._scanner = new Scanner(System.in);

    }

    public int inputInt() {
        return Integer.parseInt(this._scanner.nextLine());
    }

    public String inputString() {
        return this._scanner.nextLine();
    }

    public boolean inputDoesContinueToInputNextStudent() {
        char answer;
        System.out.print("성적을 입력하려면 'Y'를, 입력을 종료하려면 'N'을 입력하시오 : ");
        answer = this.inputString().charAt(0);
        if (answer == 'Y' || answer == 'y')
            return true;
        else
            return false;

    }

    public int inputScore() {
        int score;
        System.out.print("- 점수를 입력하시오 : ");
        score = this.inputInt();
        return score;
    }

    public void outputAverageScore(float anAverageScore) {
        System.out.println("평균 점수는 " + anAverageScore + " 입니다.");
    }

    public void outputNumberOfStudentsAboveAverage(int aNumber) {
        System.out.println("평균 이상인 학생은 모두 " + aNumber + " 명 입니다.");
    }

    public void outputMaxScore(int aMaxScore) {
        System.out.println("최고점은 " + aMaxScore + "점 입니다.");
    }

    public void outputMinScore(int aMinScore) {
        System.out.println("최저점은 " + aMinScore + "점 입니다.");
    }

    public void outputGradeCountFor(char aGrade, int aCount) {
        System.out.println(aGrade + " 학점은 모두 " + aCount + " 명 입니다.");
    }

    public void outputStudentInfo(int aScore) {
        System.out.println("점수 : " + aScore);
    }

}
```

(3) DS1_06_201402395_이승희

```java
public class DS1_06_201402395_이승희 {
    public static void main(String[] args){
        AppController appController = new AppController();
        appController.run();
    }
}
```

(4) Ban

```java
public class Ban {
    private static final int DEFAULT_MAX_SIZE = 100;
    private int _maxSize;
    private int _size;
    private Student[] _elements;
```

```
public Ban() {
    this._maxSize = DEFAULT_MAX_SIZE;
    this._size = 0;
    this._elements = new Student[this._maxSize];

}

public Ban(int givenMaxNumOfStudents) {
    this._maxSize = givenMaxNumOfStudents;
    this._size = 0;
    this._elements = new Student[this._maxSize];

}

public int maxSize() {
    return this._maxSize;
}

public int size() {
    return this._size;
}

public boolean isEmpty() {
    if (this._size == 0)
        return true;
    else
        return false;

}

public boolean isFull() {
    if (this._size >= this._maxSize)
        return true;
    else
        return false;

}

 public boolean add(Student aScore) {
     if (this.isFull() == true)
         return false;
     else {
         this._elements[this._size] = aScore;
         this._size++;
         return true;
     }
 }

 public Student elementAt(int aPosition) {
     return this._elements[aPosition];
 }

 public void sortStudentsByScore() {
     int size = this._size;
     if (size >= 2) {
         int minLoc = 0;
         for (int i = 1; i < size; i++) {
             if (this._elements[i].score() < this._elements[minLoc].score())
                 minLoc = i;
         }
         swap(minLoc, size - 1);
         quickSortRecursively(0, size - 2);
     }

 }

 public int numberOfStudentsAboveAverage() {
     float average = averageScore();
     float score;
     int numberOfStudentsAboveAverage = 0;

     for (int i = 0; i < this._size; i++) {
         score = this._elements[i].score();
         if (score >= average)
             numberOfStudentsAboveAverage++;
     }
     return numberOfStudentsAboveAverage;

 }
```

```
public float averageScore() {
    float sumOfScores = (float) sumOfScoresRecursively(0, this._size - 1);
    float average = sumOfScores / this._size;
    return average;
}

public GradeCounter countGrades() {
    char currentGrade;
    GradeCounter gradeCounter = new GradeCounter();
    for (int i = 0; i < this._size; i++) {
        currentGrade = this.scoreToGraer(this._elements[i].score());
        gradeCounter.count(currentGrade);
    }
    return gradeCounter;
}

private void quickSortRecursively(int left, int right) {
    int mid;
    if (left < right) {
        mid = this.partition(left, right);
        this.quickSortRecursively(left, mid - 1);
        this.quickSortRecursively(mid + 1, right);
    }

}

private void swap(int positionA, int positionB) {
    Student temp = this._elements[positionA];
    this._elements[positionA] = this._elements[positionB];
    this._elements[positionB] = temp;

}
private int partition(int left, int right) {
    int pivot = left;
    int pivotScore = this._elements[pivot].score();
    right++;
    do {
        do {
            left++;
        } while (this._elements[left].score() > pivotScore);
        do {
            right--;
        } while (this._elements[right].score() < pivotScore);
        if (left < right)
            this.swap(left, right);
    } while (left < right);
    this.swap(pivot, right);
    return right;
}

private float sumOfScoresRecursively(int left, int right) {
    if (left > right) {
        return 0;

    } else {
        return (this._elements[left].score() + this.sumOfScoresRecursively(
                left + 1, right));
    }
}
```

```
    private int maxScoreRecursively(int left, int right) {
        int leftMax;
        int rightMax;

        if (left == right)
            return this._elements[left].score();
        else {
            int mid = (left + right) / 2;
            leftMax = this.maxScoreRecursively(left, mid);
            rightMax = this.maxScoreRecursively(mid + 1, right);
            if (leftMax >= rightMax)
                return leftMax;
            else
                return rightMax;
        }
    }

    private int minScoreRecursively(int left, int right) {
        if (left == right)
            return this._elements[left].score();
        else {
            int leftMin=this._elements[left].score();
            int nextMin=this.minScoreRecursively(left+1, right);
            if(leftMin<=nextMin)
                return leftMin;
            else
                return nextMin;

        }
    }

     private char scoreToGraer(int aScore) {
         if (aScore >= 90)
             return 'A';
         else if (aScore >= 80)
             return 'B';
         else if (aScore >= 70)
             return 'C';
         else if (aScore >= 60)
             return 'D';
         else
             return 'F';
     }

     public int minScore() {
         return this.minScoreRecursively(0, this._size - 1);
     }

     public int maxScore() {
         return this.maxScoreRecursively(0, this._size - 1);
     }
}
```

(5) Student

```
public class Student {
    private int _score;

    public Student() {
        this._score=0;

    }

    public Student(int aScore) {
        this._score = aScore;

    }

    public int score() {
        return this._score;
    }

    public void setScore(int aScore) {
        this._score = aScore;
    }


}
```

(6) GradeCounter

```
public class GradeCounter {
    private int _numberOfA;
    private int _numberOfB;
    private int _numberOfC;
    private int _numberOfD;
    private int _numberOfF;

    public GradeCounter(){

    }

    public int numberOfA(){
        return this._numberOfA;
    }

    public int numberOfB(){
        return this._numberOfB;
    }
    public int numberOfC(){
        return this._numberOfC;
    }
    public int numberOfD(){
        return this._numberOfD;
    }
    public int numberOfF(){
        return this._numberOfF;
    }
    public void count(char aGrade){
        switch(aGrade){
        case 'A':
            this._numberOfA++;
            break;
        case 'B':
            this._numberOfB++;
            break;
        case 'C':
            this._numberOfC++;
            break;
        case 'D':
            this._numberOfD++;
            break;
        default:
            this._numberOfF++;
        }
    }
}
```

(7) MessageID

```
public enum MessageID {
    Notice_StartProgram,
    Notice_EndProgram,
    Notice_StartMenu,
    Notice_EndMenu,

    Show_SortedStudentList,

    Error_WrongMenu,
    Error_InvalidScore,
    Error_Input

}
```

5. 정리
   1) 실습 자료의 초록색으로 되어 있는 부분의 구현 방법과 그렇게 프로그램을 작성한 이유
   2) [생각해 볼 점]
      (1) 학생의 정보를 더 같이 입력 할 경우(학번 등) 바뀌어야 하는 부분은?
      : 처음에 입력받을 때 학번과 점수를 각각 입력받아야 하며, 출력할 때 경우에 맞는 각 학생의 성적과 함께 학번도 함께 출력해야 한다.

(2) 재귀는 항상 좋은 성능 향상을 할까?
: Fibonacci Number 문제를 예로 들었을 때, 항의 수가 많아질수록 수의 크기가 기하급수적으로 커지게 되어 프로그램이 다운될 수 있다. 이와 같이 문제 상황을 재귀적으로 해결할 수 있다고 해서 반드시 효율적이지 않다.