10.8 Case Study: Processing Test Scores

PROBLEM

Professor von Neuperson has a file containing the class roster for each class she teaches. To help with processing the grades for her classes, she would like a program that will allow her to: enter a test score for each student in the file; calculate the mean of the scores, excluding the lowest and the highest score; and then display each person's name and score and the difference between that score and the mean. For example, if the mean score is 75, the program should display something like

```
... 82 (+7)
```

for a score of 82, and

```
... 69 (-6)
```

for a score of 69.

OBJECT-CENTERED DESIGN

BEHAVIOR. The program should display on the screen a prompt for the name of the class roster file and read its name from the keyboard. It should then read a sequence of names from the input file. The program should then read a sequence of test scores from the keyboard, prompting for them using the sequence of names. It should eliminate the outliers of this sequence—the high and low scores—and then compute the mean of the sequence of test scores. Finally, the program should display this mean value, and then display the sequences of names and test scores, along with the difference of each test score from the mean.

OBJECTS. Although most of the objects in our behavioral description are familiar, two of them require special attention: the sequence of names and the sequence of scores. Each of these sequences can be stored in a `vector<T>` for an appropriate type `T`.

	Software Objects		
Problem Objects	**Type**	**Kind**	**Name**
name of the class roster file	`string`	varying	*inputFileName*
the sequence of names	`vector<string>`	varying	*names*
the sequence of scores	`vector<double>`	varying	*scores*
the outliers of the sequence	`double, double`	varying	none
the mean score	`double`	varying	*meanScore*
the difference of a score and the mean score	`double`	varying	none

OPERATIONS. The operations specified in our behavioral description are as follows:

i. Read a `string` from the keyboard
ii. Open a stream to an input file
iii. Read a sequence of `string` values from the file stream into a `vector<string>`
iv. Read a sequence of `double` values from the keyboard into a `vector<double>`, using a sequence of `string` values stored in a `vector<string>` as prompts
v. Eliminate the outliers of a sequence of `double` values
vi. Compute the mean of a sequence of `double` values stored in a `vector<double>`
vii. Display a `double`
viii. Display the sequence of `string` values stored in a `vector<string>`, a corresponding sequence of `double` values stored in a `vector<double>`, and the difference of two `double` values

Operations, i, ii, and viii are predefined, and a straightforward modification of the `read()` function of Figure 10-4 can be used for operation iii. In the last section we saw that operation v can be done with two statements, and the `mean()` function from Figure 10-6 in the text provides operation vi. We will need to write a function `promptAndRead()` to perform operation iv and another function `printResults()` for operation viii.

BEHAVIOR OF `promptAndRead()`. This function should receive a sequence of names from its caller. For each name in that sequence, it should display on the screen a prompt for that name's score, read the score from the keyboard, and append it to a sequence of `double` values. The resulting sequence of `double` values should be passed back to the caller.

OBJECTS FOR `promptAndRead()`. The objects in this subproblem are as follows:

	Software Objects			
Problem Objects	**Type**	**Kind**	**Movement**	**Name**
a sequence of names	`vector<string>`	constant	received (in)	*names*
a name in the sequence	`string`	varying	none	*names*[*i*]
a double value	`double`	varying	none	*aScore*
a sequence of doubles	`vector<double>`	varying	passed back (out)	*scores*
for-loop-control variable	`int`	varying	none	*i*

OPERATIONS FOR `promptAndRead()`. The operations for this function are:

i. Receive a sequence of names from the caller
ii. Access one `string` from a sequence of `string` values
iii. Display a `string` on the screen
iv. Read a `double` from the keyboard
v. Append a `double` to a sequence of `double` values
vi. Repeat operations ii–v once for each `string` in a sequence of `string` values
vii. Pass a sequence of `double` values back to the caller

Each of these operations is predefined.

ALGORITHM FOR `promptAndRead()`. We can organize these operations into the following algorithm for the function `promptAndRead()`:

Algorithm for `promptAndRead()`

1. Receive *names*, a sequence of names from the caller.
2. For each index *i* in *names*:
    a. Display *names*[*i*] in a prompt for a score.
    b. Read a `double` from `cin` into *aScore.*
    c. Append *aScore* to a sequence of `double` values named *scores.*
3. Pass *scores* back to the caller.

CODING AND TESTING OF `promptAndRead()`. The following function encodes the preceding algorithm in C++.

Case Study 10.8-1 Prompting and Reading Test Scores.

```
/* promptAndRead reads a sequence of test scores from the keyboard.
 *
 * Receive:      names, a vector of strings,
 *               scores, a vector of doubles
 * Precondition: names is not empty AND scores is empty
 * Output:       prompts for test scores, using names
 * Input:        a sequence of test scores
 * Pass back:    scores containing the input values
 ************************************************************/

void promptAndRead(const vector<string>& names, vector<double>& scores)
{
   double aScore;                        // input variable
   for (int i = 0; i < names.size(); i++) // for each index in sequence
   {
      cout << "Enter the score for "
           << names[i] << ": ";          //   prompt,
```

```
      cin >> aScore;                    //   read, and
      scores.push_back(aScore);         //   append
   }
}
```

Once `promptAndRead()` has been thoroughly tested, we continue to the function `printResults`, which is to perform the following operation in the original problem:

viii. Display the sequence of `string` values stored in a `vector<string>`, a corresponding sequence of `double` values stored in a `vector<double>`, and the difference of two `double` values

BEHAVIOR OF `printResults()`. This function should receive from its caller a sequence of string values (names), a sequence of scores, and a mean score. It should display the mean value on the screen, and then for each name in the sequence of names, it should display that name, the corresponding entry in the sequence of scores, and the difference of that score and the mean.

OBJECTS FOR `printResults()`. The objects in this subproblem are as follows:

	Software Objects			
Problem Objects	**Type**	**Kind**	**Movement**	**Name**
a sequence of names	`vector<string>`	constant	received (in)	*names*
a sequence of scores	`vector<double>`	constant	received (in)	*scores*
the mean score	`double`	constant	received (in)	*meanScore*
a name in the sequence	`string`	varying	none	*names*[*i*]
the corresponding score	`double`	varying	none	*scores*[*i*]
the loop-control variable	`int`	varying	none	*i*

OPERATIONS FOR `printResults()`). The operations in this function are

i. Receive a `double`, a sequence of `string` values, and a sequence of `double` values from the caller
ii. Display a `double` on the screen, with appropriate formatting for a test score
iii. Access a `string` in a sequence of `string` values
iv. Access the corresponding `double` in a sequence of `double` values
v. Display the accessed `string` and `double` on the screen

vi. Display the difference of two `double`s, showing the sign
vii. Repeat operations iii–vi once for each `string` in a sequence of `string` values

Each of these operations is predefined in C++.

ALGORITHM FOR `printResults()`. We can organize these operations into the following algorithm:

Algorithm for printResults()

1. Receive *meanScore*, a `double`; *names*, a sequence of `string` values; and *scores*, a sequence of `double` values from the caller.
2. Display *meanScore* via `cout`.
3. For each index *i* in *names*:
    a. Display *names*[*i*], *scores*[*i*], and *scores*[*i*] – *meanScore*, with appropriate formatting.

CODING AND TESTING OF `printResults()`. The following function encodes the preceding algorithm in C++.

Case Study 10.8-2 Displaying Names, Test Scores, and Mean Difference.

```
/* printResults() displays names, test scores, and differences
 * between the scores and the mean score.
 *
 *  Receive: out, an ostream,
 *           meanScore, a double,
 *           names, a vector of strings,
 *           scores, a vector of doubles
 *  Output:  each name in names, each score in scores
 *           and the difference of each score and meanScore
 ************************************************************/

#include <iostream>                     // ostream
#include <iomanip>                      // setprecision()
using namespace std;

void printResults(ostream& out, double meanScore,
                  const vector<string> & names,
                  const vector<double> & scores)
{
  out << "\nThe mean score is "
      << right << fixed << showpoint    // format for test scores
      << setprecision(1) << meanScore   // show mean score
      << " (ignoring max and min).\n"
      << endl;
```

```
  for (int i = 0; i < names.size(); i++)  // for each index in names:
    out << setw(20) << names[i] << left   //   display name,
        << noshowpos << setw(5) << right
        << scores[i] << "\t("             //    score, and
        << showpos << setw(5)
        << scores[i] - meanScore          //    difference from mean
        << ')' << endl;
}
```

ALGORITHM FOR ORIGINAL PROBLEM. Once we have functions to perform each of the operations needed for our problem, we are ready to organize those operations into an algorithm.

Algorithm for Score-Processing Problem

1. Prompt for and read the name of the class roster file into *inputFileName.*
2. Open an `ifstream` named *inStream* to the file whose name is in *inputFileName.* (If this fails, display an error message and terminate the algorithm.)
3. Read a sequence of names from *inputFileName* into *roster.*
4. Using *roster* to prompt, read a sequence of test scores into *scores.*
5. Save a copy of *scores* in *originalScores.*
6. Eliminate the outliers from *scores.*
7. Compute the mean of the values in *scores,* and store it in *meanScore.*
8. Via `cout`, display *meanScore,* each name in *names,* the corresponding score from *originalScores,* and the difference of that score from *meanScore.*

CODING. The following program implements the preceding algorithm. It uses the `vector<T>` class template.

Case Study 10.8-3 Test Score Processing.

```
/* testScores.cpp processes a sequence of test scores, using a class
 * roster stored in a file.
 * Input(keyboard): the name of the roster file and
 *                  a sequence of test scores
 *  Input(file):    a sequence of names
 *  Precondition:   the sequence of names is not empty
 *  Output:         the mean of the sequence of test scores,
 *                    each student's name, the score for that student,
 *                   and the difference between the score and the mean
*********************************************************************/
```

```
#include <iostream>                          // cout, cin, <<, >>
#include <fstream>                           // ifstream
#include <string>                            // string
#include <vector>                            // vector<T>
#include <algorithm>                         // max, min_element()
using namespace std;
#include "myVector.h"                        // read(), mean()

void promptAndRead(const vector<string>& names, vector<double>& scores);

void printResults(ostream& out, double meanScore,
                  const vector<string>& names,
                  const vector<double>& scores);

int main()
{
   cout << "This program requires a roster of student names.\n"
        << "Enter the name of the roster file: ";
   string inputFileName;
   cin >> inputFileName;

   ifstream fin( inputFileName.data() );    // stream to roster
   vector<string> roster;                   // the class roster
   read(inputFileName, roster);             // -- read it

   vector<double> scores;                   // the score sequence
   promptAndRead(roster, scores);           // -- read it

   vector<double> originalScores = scores;  // save a copy
                                            // remove extreme values
   scores.erase( min_element( scores.begin(), scores.end() ) );
   scores.erase( max_element( scores.begin(), scores.end() ) );

   double meanScore = mean(scores);         // find mean w/o extremes
                                            // output w/ extremes
 printResults(cout, meanScore, roster, originalScores);
}


/*** Insert the definitions of:
       PromptAndRead() from Case Study 10.8-1, and
       PrintResults() from Case Study 10.8-2 here. ***/
```

Listing of file `names.txt` used in sample run:

```
Jack_Sprat
Jill_Tumbling
Mary_HattaLamb
Peter_Pumpkin
Jack_B_Nimble
Cinderella_Slipper
Prince_Charming
```

Sample run:

```
This program requires a roster of student names.
Enter the name of the roster file: names.txt
Enter the score for Jack_Sprat: 100
Enter the score for Jill_Tumbling: 5
Enter the score for Mary_HattaLamb: 70
Enter the score for Peter_Pumpkin: 75
Enter the score for Jack_B_Nimble: 80
Enter the score for Cinderella_Slipper: 73
Enter the score for Prince_Charming: 77

The mean score is 75.0 (ignoring max and min).

Jack_Sprat          100.0   (+25.0)
Jill_Tumbling         5.0   (-70.0)
Mary_HattaLamb       70.0   ( -5.0)
Peter_Pumpkin        75.0   ( +0.0)
Jack_B_Nimble        80.0   ( +5.0)
Cinderella_Slipper   73.0   ( -2.0)
Prince_Charming      77.0   ( +2.0)
```