Programming Design, Spring 2014
Homework 2

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

Submission. To submit your work, please upload the following two files to the online grading system at http://lckung.im.ntu.edu.tw/PD/.

1. A .pdf for Problems 1 to 2.
2. Your .cpp file(s) for Problems 3 and 4 (optional).

Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***8:00am, March 3, 2014***. Please answer in either English or Chinese.

Problem 0

(0 points) Please do the following two things:

(a) Please try to form study groups to study or work on homework together. If you have been included in a group, please write down the names of your group members and indicate who is the tutor; otherwise, please skip this problem. This problem is just a reminder. Regarding giving free grades to tutors, only the answers on the midterm exam counts.

(b) Please read Sections 3.6, 3.10–3.12, and 4.1–4.11 of the textbook.[1] In any case, I strongly suggest you to read the textbook thoroughly before you start to do this homework.

Problem 1

(10 points) When the instructor was ten years old, he played an RPG game on his PC a lot. In that game, a player controls a soldier (and his friends) to explore the world and try to kill a devil and save the world. Once the soldier kills a monster during the trip, he collects some experience points and some dollars. He may get to a higher level and become stronger if he has more experience points and buy better weapons when more dollars are earned. The process must be repeated for a long time until the soldier can kill the devil.

While playing the game, one thing confused the instructor a lot. No matter how hard he played the game, the number of dollars carried by the soldier can never be greater than \$65,535. For example, if the soldier currently carries \$65,530 and then kills a monster which should leave \$10 to the soldier, the soldier will only have \$65,535 instead of \$65,540 after the battle, which seems to be very weird. Interestingly, many games at that time have this similar feature.

Eight years later, after the instructor studied "Programming Design" in the university, he understood the reason. As you are also studying "Programming Design", you should also know the reason. Use your professional knowledge to explain why the soldier can only carry at most \$65,535 in that game. Moreover, explain why it was common for those old game but uncommon for games developed in recent years. Explain in Chinese if it is your mother language or English otherwise. Use your own words and limit your answer to 500 words.

[1] The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

Problem 2

(10 points) Consider the following C++ program

```
char c1 = 0, c2 = 0;

cin >> c1;
cin >> c2;

if ((c1 == 'a' && c2 == 'b') || (c1 == 'b' && c2 == 'a'))
  cout << "Here.\n";
else
  cout << "There.\n";
```

(a) (3 points) Explain what does this program do.

(b) (3 points) What will happen if that `||` is replaced by `&&`?

(c) (4 points) Rewrite this program with a nested selection. Do not use logic operators.

Problem 3

(80 points) You are running a retail store by selling fresh apples. Every evening before you go to bed, you need to order from your supplier for apples. Those apples will be shipped to you the next day before you open the store. One difficulty for you to run a successful business is certainly demend uncertainty. If you prepare too many apples (i.e., more than the demand), unsold apples must be thrown away and you incur an overage cost. On the contrary, if you prepare too few, you will face lost sales and incur an underage cost. In short, you must make a forecast for the demand of apples, and you want to forecast to be as accurate as possible.[2]

Among many different forecasting methods, a simple but widely adopted one is the *moving average* method. The method generates a forecast for future according to the data collected in the past. Suppose at period n we are given a series of historical values (in our example, these values are daily demands you observed in the past) x_1, x_2, ..., and x_n, where x_t is the value we observed in period t. If we adopt an *k-period* moving average method, then for period $t+1$ our forecast will be

$$f_{n+1} = \frac{x_{n-k+1} + x_{n-k+2} + \cdots + x_n}{k}.$$

The idea is simple: The next-period is assumed to be the average of the values in the previous k periods (including the current period). Note that if $k > n$, the method cannot be applied and no forecast should be generated. Also note that if $k < n$, values too far from this period will not be used. In an extreme case in which $k = 1$, we naively predict the next-period value to be the current-period one.

Suppose you have decided to use moving average to forecast future demands, a natural question regarding the number of past periods to consider (i.e., the value of k) then arises. To decide which k best fits our demand pattern, we need to somehow compare the "performances" of different values of k. One way to evaluate the accuracy of a forecasting method is through the *mean absolute deviation* (MAD). Suppose we have a series of past values $\{x_t\}_{t=1,\ldots,n}$ and another series of forecast values $\{f_t\}_{t=k+1,\ldots,n}$, the absolute deviation (AD) of period t is $|x_t - f_t|$ and the MAD is[3]

$$\frac{\sum_{t=k+1}^{n} |x_t - f_t|}{n-k}.$$

[2]Forecasting and inventory decisions will be further discussed in the courses "Operations Research" and "Statistics" in the second year in the Department of Information Management.

[3]In C++, try to include the library `<cmath>` and use the function `abs()`. There are many other mathematical functions implemented in `<cmath>`. Try to search for the full list online!

The larger the MAD is, the worse the forecasting method is. In our example, suppose we want to compare the performances of setting $k = 2$ and $k = 3$ for a given series $(4, 6, 3, 7, 8, 12, 4, 7, 9, 10)$, the relevant calculations may be summarized in the table.[4] The two MADs, 3.06 and 3.14, are the averages of the third and fifth rows, respectively. As $3.06 < 3.14$, choosing $k = 2$ is believed to be better than choosing $k = 3$. We may then use $k = 2$ to generate our forecast $\frac{9+10}{2} = 9.5$ for the next period.

t	1	2	3	4	5	6	7	8	9	10
x_t	4	6	3	7	8	12	4	7	9	10
f_t $(k = 2)$	–	–	5	4.5	5	7.5	10	8	5.5	8
AD $(k = 2)$	–	–	2	2.5	3	4.5	6	1	3.5	2
f_t $(k = 3)$	–	–	–	4.33	5.33	6	9	8	7.67	6.67
AD $(k = 3)$	–	–	–	2.67	2.67	6	5	1	1.33	3.33

Given the large volume of past sales data, you now want to write a program to help you find the best moving average parameter k for you to do future forecasts.

Input/output formats

The input contains 35 lines of nonnegative integer values. In each line, there are $n + 1$ values, which are n, x_1, x_2, ..., x_n, and then a new line character. Values are separated by white spaces. The first value $n \in \mathbb{Z} \cap [1, 10000]$ indicates the number of historical sales quantities you have recorded. Then $x_t \in \mathbb{Z} \cap [0, 10000]$ is the tth-period sales quantity. For each line, your program should run the three-period moving average method and output the MAD based on the input, a white space, the forecast for the next period, then a new line character. For both output values, be precise to the second digit by throwing away the remaining digits.[5] If $n = 3$, no MAD may be calculated and your program should simply output the forecast and then a new line character. If $n < 3$, your program should simply output a new line character.

As an example, if a line contains

```
9 1 2 3 1 2 3 1 2 6
```

as a set of $n = 9$ historical values, your program should use $k = 3$ as the parameter to run the moving average and obtain 1.16666 as the MAD and 3 as the forecast. Your program should then output

```
1.16 3
```

and then change to a new line for the next output. More examples are provided below:

- Input: `3 9 8 0`; output: `5.66`.
- Input: `5 9 8 0 6 7`; output: `1.33 4.33`
- Input: `8 110 180 234 157 196 48 112 76`; output: `47.06 78.66`.

What should be in your source file

For this problem, you will be give a template C++ file that repeatedly reads the testing data prepared on PDOGS for you. In other words, you only need to implement the task specified above and do not need to worry about how to read testing data. Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are NOT allowed to use techniques not covered in lectures. Finally, you should write relevant comments for your codes.

[4] If we choose k to be the moving average parameter, we will have only $n-k$ forecasts (where n is the number of observed values) and thus $n - k$ ADs.

[5] For example, if you get 1.2385, output 1.23; if you get 3.2, output 3.2 rather than 3.20. To convert 1.2385 to 1.23, one why of doing so in C++ is to use the function `floor()` defined in `<cmath>`: First multiply a fractional number by 100, then apply `floor()` on it, and then divide it by 100.

Grading criteria

- 70% of your grades for this program will be based on the correctness of your output. PDGOS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each fully correct line of output gives you 2 points.
- 30% of your grades for this program will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.

Bonus: Problem 4

(20 points) For exactly the same inputs of Problem 4, now your task is to identify the best parameter k^* within 1 and $\min\{n-1, 6\}$, report such k^*, the associated MAD, and the forecast. As an example, if a line contains

```
9 1 2 3 1 2 3 1 2 6
```

as a set of $n = 9$ historical values, your program should run the moving average method six times, one for each $k \in \{1, 2, ..., 6\}$. The six MADs are 1.625, 1.5, 1.16667, 1.35, 1.65, and 1.66667, respectively, and thus the best value is $k^* = 3$. As the forecast is $\frac{1+2+6}{3} = 3$, your program should output

```
3 1.16 3
```

and then change to a new line. Note that the output should contain first the best k^*, then a white space, then the associated MAD, then a white space, then the forecast, and finally a new line character. If multiple values of k all give the least MAD, the smallest k wins and should be the only one included in the output. All the values should be precise to the second digits with remaining digits thrown away. If $n \leq 6$, only values within 1 and $n-1$ should be tried as candidates of k^*. Finally, if $n = 1$, a new line character should be output directly