Computer Programming and Applications (ENGG1002A)
Assignment 3

Deadline :	**on or before 14th Nov., 2012 (Wed) 5pm**
Weighting :	12.5% (of the whole course)

The problem

In Assignment 2, you have written a program for some touring company to compute, for any tourist guide, how much this guide has earned on a working day. The company stores these data in a text file called `record.txt` as follows:

```
John    100  12/30/2012
Ronald  250  1/1/2012
Francis   300  8/13/2012
Poon    89   1/3/2012
Teddy   120  1/3/2012
Francis   400  1/3/2012
Todd    321      1/3/2012
Cliff   34   1/3/2012
Ronald  353  1/4/2012
Joe     179  1/4/2012
John    167  1/4/2012
...
```

As can be seen, the file contains a sequence of records of the following format:

Guide_name   money_earned   month/day/year

For example, the first line

```
John 100 12/30/2012
```

in the above file shows that the guide John worked on December 30, 2012, and he earned $100 on that day. When writing your program, you may assume that

- the company has no more than 100 guides,
- each guide's name is a single word (i.e., there is no space in a name),
- money_earned is a non-negative integer,
- month, day and year are integers; year is always 2012,
- all the three fields always exist and are valid, e.g. no invalid date
- fields are separated by one or more spaces or tabs
- the records are not in any particular order

At the end of this year, the company will need to extract some useful information from `record.txt.` As an experienced programmer, you are asked to write a C++ program to help the company extract them. In addition to `record.txt,` your program should read another file `command.txt,` which stores the commands for processing `record.txt`. After executing each command, you should write the result to an output file called `report.txt`.

Before giving details of these commands, we would suggest some array variables to summarize the data in `record.txt`. Later, you will find that they are very useful for executing the commands. Note that this is only a suggestion. You may have other implementation.

Hint: How to organize the data

Your program may declare twelve arrays of `int`, each of size 100, namely, Jan[100], Feb[100], …, Nov[100] and Dec[100]. You should also declare an array of `string` name[100]. The following is the purpose of these arrays:

- The array name[100] stores the name of the guides.
- The arrays Jan[100], Feb[100], …, Dec[100] store the total amount of money earned by the guides in those months.

For example, suppose that the company has only three guides Tom, John and Peter. Then, after processing all the records in `record.txt`, the content of name[100] may be as follows:

0	1	2	3	…
Peter	John	Tom	-	-

(Note that we don't require the names to be stored in any specific order.) And if John, who is occupying name[1], has earned

- 12, 14, 54, 95, 321, 42, 52, 41, 92, 120, 32, 23 in January, February, March, April, May, June, July, August, September, October, November and December, respectively, then,
- Jan[1], Feb[1], Mar[1], Apr[1], May[1], Jun[1], Jul[1], Aug[1], Sep[1], Oct[1], Nov[1] and Dec[1] should be storing 12, 14, 54, 95, 321, 42, 52, 41, 92, 120, 32, 23 respectively.

The following figure shows the overall structure of these arrays.

	name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	Peter	…											
1	John	12	14	54	95	321	42	52	41	92	120	32	23
2	Tom	…							…				
3	-	-	-	…									-

These arrays are very useful. For example, if you want to find out how much Tom has earned in August, you can first search the array name[100] to see at which entry Tom is occupied. Using the above example, Tom is occupying name[2]. Then, Aug[2] is storing the money Tom has earned in August.

Your first challenge is to write the program codes that read the records from `record.txt` one by one, and for each record read, update the 12 *month* arrays and *name* array accordingly.

Note: You may notice that the month arrays can be represented by a 2D array. In fact, using 2D array will make the program more compact and simpler, though the logic may not be as straightforward.
You are also advised to test thoroughly if the arrays are correctly filled (by using debugger or adding debugging statements to dump the content out) before proceeding to the next step.

We now give the details of the commands. We only require you to implement two simple commands.

Command summarize

The format of this command is:

summarize *guidename endmonth*

"summarize" is the name of the command, *guidename* is the name of a guide in the company, and *endmonth* is an integer from 1 to 12. The command asks the program to print, for each month from the first to the *endmonth*-th month, the money earned by the guide in that month. It also prints the total amount of money earned by the guide during this period. The words in the command line are separated by one or more spaces or tabs.

For example, suppose that lazy John has only worked four days in the whole year, and his records in `record.txt` appear as follows:

```
..
John 123 2/11/2012
...
John 200 2/24/2012
...
John 304 5/13/2012
...
John 225 12/7/2012
...
```

Then, for the command

```
summarize John 5
```

Your program should print the following lines in `report.txt`:

```
Result for summarize John 5:
	January: 0 dollars
	February: 323 dollars
	March: 0 dollars
	April: 0 dollars
	May: 304 dollars
	Total: 627 dollars

```

In all output, one blank line should be added at the bottom

Note that the leading spaces of the lines with January, February, March, April, May and Total are generated by a single tab, and the words/numbers are separated by a single space.

For all output, one blank line should be added to the end.

Your program should also check for errors in the command. There are two types of errors.

(1) No record found: Suppose that there is not any record for King in `record.txt`. Then for the command “summarize King 4”, your program should output the following to `report.txt`:

```
Result for summarize King 4:
      There is no record for King.
```

If King has records in `record.txt` but just doesn't have any in the first 4 months, this error message should NOT be printed.

(2) Invalid *endmonth*: Note that *endmonth* must be between 1 and 12. Thus, for the command “summarize John 18”, if John has some records in `record.txt,` your program should output the following (otherwise you should print the “no record” message as above):

```
Result for summarize John 18:
      Endmonth 18 is invalid.
```

Command Find-stat

The format of this command is:

Find-stat *quarter*

where “Find-stat” is the name of the command, and *quarter* can be one of the four strings: “Q1”, “Q2”, “Q3” and “Q4”, corresponding to the four quarters of a year. To be more precise, Q1 covers the first three months, from January to March, Q2 covers April to June, Q3 covers July to September, and Q4 covers October to December.
In response to this command, your program should compute, for each guide, the total amount of money earned by him in *quarter*. Then, write to `report.txt` the maximum, minimum and average of these earnings. For example, suppose that the company has only three guides Peter, John and Tom. In the first quarter, Q1 (i.e., from January to March), Peter, John and Tom have earned totally 1245, 2812, and 980 dollars, respectively. Then, in response to the command

```
Find-stat Q1
```

your program should print the following lines in `report.txt.`

```
Result for Find-stat Q1:
       Maximum: 2812 dollars
       Minimum: 980 dollars
       Average: 1679 dollars
```

If average is a real number, your program should print the integral part only. (e.g. print "123" for 123.78)
Note that the quarter must be either Q1, Q2, Q3 or Q4. For other value, your program should output an error message. For example, for the command “Find-stat X1”, your program should output to `report.txt`

```
Result for Find-stat X1:
       Quarter X1 is invalid.
```

Note: You may assume that there is no other command in `command.txt.`

Sample input and output

Below is a set of sample input and output files for helping you understand the assignment requirements.

Input:

(1) “`record.txt`”

```
John      100   1/1/2012
Ronald    250   1/1/2012
Ronald    350   1/4/2012
John      140   1/4/2012
John      110   2/11/2012
John      200   2/24/2012
John      300   5/13/2012
Ronald    330   5/17/2012
Shara     500   5/22/2012
Hebe      450   6/23/2012
Ronald    100   8/18/2012
```

(2) “`command.txt`”

```
summarize John 1
summarize Ronald 12
Find-stat Q1
summarize nobody 5
Find-stat Q2
summarize John 5
```

(3) Output: “`report.txt`”

```
Result for summarize John 1:
    January: 240 dollars
    Total: 240 dollars

Result for summarize Ronald 12:
    January: 600 dollars
    February: 0 dollars
    March: 0 dollars
    April: 0 dollars
    May: 330 dollars
    June: 0 dollars
    July: 0 dollars
    August: 100 dollars
    September: 0 dollars
    October: 0 dollars
    November: 0 dollars
    December: 0 dollars
    Total: 1030 dollars

Result for Find-stat Q1:
    Maximum: 600 dollars
    Minimum: 0 dollars
    Average: 287 dollars

Result for summarize nobody 5:
    There is no record for nobody.

Result for Find-stat Q2:
    Maximum: 500 dollars
    Minimum: 300 dollars
    Average: 395 dollars

Result for summarize John 5:
    January: 240 dollars
    February: 310 dollars
    March: 0 dollars
    April: 0 dollars
    May: 300 dollars
    Total: 850 dollars
```

The following will be the corresponding sample run:

```
C:\work> assign3

C:\work> type report.txt
Result for summarize John 1:
   January: 240 dollars
   Total: 240 dollars

Result for summarize Ronald 12:
   January: 600 dollars
   February: 0 dollars
...
```

Advice

- This assignment demonstrates the importance of data structure. With a well-designed data structure, the program will be much easily to write. So before proceeding to the actual coding, please think carefully what types of arrays, variables etc. are required.
- Don't jump directly to the coding phase! Think and write down the program logic first. It will eventually save you many hours of debugging time.
- Learn how to use debugger! Don't just write the program and hope that the output will be correct. It seldom happens. Your program will likely to have all sorts of problems. By setting breakpoints at appropriate lines and printing the intermediate data out, you can narrow down the source of errors.
- Before you use a certain language feature or standard function, try it out with a small testing program first. Make sure you understand how it works before putting it in your actual program.

Program style

Besides program correctness, marks might be deducted if the following is not followed:

- I/O format - your program will be marked by a program, therefore the wordings, format and order of input / output must follow <u>exactly</u> that of the above examples.
- Program style - your program should be easy to read and understand. Make use of proper indentation, comments and meaningful variable names to achieve this.

Marking scheme

The assignment will be marked in the following way:

- [100%] Your program will be tested against a number of cases. Each case will be worth a certain number of marks. All output of the case must be correct to score any marks.

[**Plagiarism is strictly prohibited**] Zero marks will be given to both the source and copy if discovered. You must not copy or let others copy your work. It's your own responsibility to prevent others from copying your program directly or indirectly (e.g. obtain your program through another person).

This assignment is an individual work. You can discuss with others in the design phase only. Coding must be done separately and group program is NOT accepted.

Submission (Late submission will not be marked)

- Please submit through the Moodle assignment 3 submission function.
- Please submit only the **source program** (i.e. **a3.cpp**).
- Marking will be done using Dev-C++ version 4.9.9.2.