IN2029 Programming in C++
Coursework

There is a single coursework in this module, counting for 30% of the overall module mark. This coursework is due at 5pm on Sunday 1st December. As with all modules, this deadline is hard. (See "Submission" below for the procedure for applying for extensions.)

This coursework simulates a simple database operation, and provides practice with classes and standard containers. It does not require any features covered after session 6.

Description

The file `payments.txt` contains a simple textual table. Each line lists three pieces of information: a person's name, an object name, and the price of the object purchased by the person, separated by one space, e.g.:

```
Bob eggs 2.5
Alice apple 1.5
Ava pen 2
Bob pen 3.5
Alice eggs 2
```

(The meaning of the units depends on the objects, and need not concern us.) You may assume that person and object names do not contain spaces. The records are not in any special order. You may assume that any combination of person and object (e.g. `Bob eggs`) occurs at most once.

To read this file, you will need to use the class `ifstream`, which is derived from `istream`, like so:

```
#include <fstream>
...
    ifstream in("payments.txt");
    ... istream operations ...
```

The constructor opens the file; you can then test **`in`** to see whether it succeeded.

The file `people.txt` contains a series of lines, each representing a person's name, e.g.:

```
Bob
Alice
```

Again the lines are in no special order. In addition, these names are a subset of names in the `payments.txt`.

Your program should generate a statement for each person listed in `people.txt` and print

it to the standard output. For each statement, the first line should display the person's name. The following lines should list the objects they purchased along with the amount paid for each object (one object per line). Finally, the last line should print "`Total: value`", where `value` is the sum of all payments. For example, for the given inputs (`payments.txt` and `people.txt`), it should print:

```
Bob
eggs 2.5
pen 3.5
Total: 6
Alice
apple 1.5
eggs 2
Total: 3.5
```

Your program should validate the structure of the input files and display an appropriate error message if any issues are found. For example, if `people.txt` is empty, it should print a message such as: '`The people.txt file is empty`.' Similarly, if a line is missing an item, or if the payment is not a valid integer or floating-point number, the program should generate an error message. For instance, if `payments.txt` contains a line like this:

```
Alice 2.5
```

Or if we have something like this:

```
Alice pen two
```

It should generate an appropriate error message. We assume there are no empty lines between the data, all people's names start with a capital letter, and all other letters (for both names and objects) are lowercase. Additionally, we assume that a single space separates names and payments. Finally, the numbers in `payments.txt` are assumed to have at most two digits after the decimal point. Therefore, you do not need to validate these aspects.

Please follow this format precisely, with no additional output, as I will be running your programs through automated tests. The tests will use my own files `payments.txt` and `people.txt` with the same format as the above examples, but may involve different data.

Your solution should make appropriate use of at least one class and at least one container.

Marking

70% of the marks will be for correctness and efficiency, including avoiding unnecessary copying. 10% will be for managing error messages. The remaining 20% of the marks will be for clean programming style, including consistent layout, sensible identifier names, useful comments, avoidance of superfluous variables and data members, and general clarity of code. You may use language features not covered in the lectures, but your code must be standard C++. (If using Visual Studio, set the Disable Language Extensions property under C/C++ Language to Yes.)

Submission

Submit a ZIP file containing your source files only, to Moodle.

If you believe that you have extenuating circumstances that justify an extension, you should

- submit your work to Moodle (there will be a different submission area for late submissions) within one week of the deadline (i.e. by 5pm Sunday 8th December), and then send an email to `smcse-extension@city.ac.uk` including your student number, course, module code and the original due date, **and**
- apply for an extension through the Extenuating Circumstances process.

As not all claims for extenuating circumstances meet the criteria, you can also submit on Moodle what you have by the deadline, as a fallback in case your claim is rejected.

Caution

We expect that your submissions should be your own work. You must not discuss the details of this coursework with other students, nor can you ever share your code (in any form) with them.

You must not use generative AI tools, or code found online. Any of these this things (including giving your code to another student) may be treated as academic misconduct.