Trade Parser

Input Data

You are presented with an input file which records the trading history of multiple financial instruments in a binary format. In this file, each trade will consist of the following data: instrument name, price, and quantity.

Field description	Data type	Size (bytes)	Description
Instrument name	Characters	8	The instrument name is 8 Latin-1 chars or less. For names which are shorter than 8 bytes, field is null-padded. Note, when all 8 characters are used, field is not null-terminated.
Quantity	Integer	4	Quantity of instrument traded (in native host endianness)
Price	Double	8	Price of a trade, expressed as floating-point number with double precision (IEEE 754 in native host endianness).

Program Requirements

Your job is to write a program, which will:

- Receive the name of the input binary file via its single argument
- Read the file and **for each instrument** calculate:
    - Minimum price of trade
    - Maximum price of trade
    - Volume-weighted average price (VWAP). Volume-weighted average price for each instrument is calculated using following formula:

      $$VWAP = \frac{\sum_{i=1}^{K} (P_i * Q_i)}{\sum_{i=1}^{K} Q_i}$$

      Where P is a price of a single trade, Q is a quantity of a single trade and K is a total number of trades in the instrument
- Print calculated data to standard output, lexicographically sorted by instrument name in ascending order, in following format (one line per instrument, each instrument name from the input file must appear once in the output, regardless of the number of trades in the instrument):
  `<InstrumentName><_tab_><MinPrice><_tab_><MaxPrice><_tab_><VWAP>\n`

  All floating-point numbers should be rounded to **2** decimal places. There should be no padding for instrument name, even if it was present in the input file.

- Exit with error code 0 (normal exit)

Sample input and output

We have prepared sample data with which you can test. Please note, it doesn't cover all possibilities and is used for illustration only. The data sets we will run with your submitted program will be much larger.

The sample data is a binary file which could be represented by following .csv file:

```
BBB,20,3.4
AAA,40,21
BBB,30,5.1
2354,15,10.5
BBB,10,7
AAA,20,25.5
```

We expect the following output to be produced with the sample:

```
2354\t10.50\t10.50\t10.50
AAA\t21.00\t25.50\t22.50
BBB\t3.40\t7.00\t4.85
```

We output the instruments in this order because lexicographically "2354" is less than "AAA" which is less than "BBB".

Please note how we used trailing 0s for the output.

For 2345, there is a single trade, and thus, minimum price, maximum price, and VWAP are all the same.

For AAA, there are two trades. 21 is smallest price and 25.5 is highest. VWAP is calculates as `(40 * 21 + 20 * 25.5) / (40 + 20)` which is equal to 22.5.

For BBB, there are three trades. The minimum price of 3.4 and the maximum price is 7. VWAP is calculated as `(20 * 3.4 + 30 * 5.1 + 10 * 7) / (20 + 30 + 10)` which is equal to 4.85.

Assumptions

You can assume that the total number of trades in the file will be high, but the total number of distinct instruments (as defined by instrument name field) in the file will always be less than 10,000,000. Please note, records in the file can appear in random order.

Handling exceptional data

You should treat the file as **untrusted** input and be ready to deal with pathological input files. Your program should not crash on such cases or exhibit other undefined behavior. Instead, it should print clear diagnostic message on **standard error** and terminate with an error code **1**.

Technical Requirements

When evaluating your program, it will be compiled and run using a recent version of GCC on a Linux host that has 64GB of available physical RAM memory, 1TB of disk space and swap enabled. If our version of GCC doesn't like your submission, we'll make reasonable attempts to get it to work. The less esoteric you make your submission, the easier that will be for us.

You can use any C++ dialect up to and including C++17, any STL or core language features. Please avoid any third-party libraries or APIs, including boost.

Make sure to provide a way to build your program. We like simplicity so requiring a single command to build is nice. Make or CMake are strongly preferred.

If you're writing code on another platform, make sure it's standards-compliant enough to run in this environment, and doesn't have any non-C++ dependencies. If you encounter any issues with that, or have any questions about it then please reach out to us straight away.

Evaluation

We want to get an idea of what your code might look like if you were to come and work with us - albeit on a small scale. To that end, we'd like to see a correct, performant, maintainable application.

More specifically we'll be looking for these things, in roughly this order of importance:

- Correctness. We expect to see correct output for all input files with which we run your code. For cases where the input data is bad or if the program can't perform the task for any other reason, you should print a clear diagnostic message to **standard error** and return error code 1.
- Clarity and conciseness. We value clear and expressive code that doesn't require too many comments in order for someone else to understand it. Don't let that stop you from adding comments describing the decisions you made along the way though. We like to know what you were thinking when you wrote the code. Let us know what assumptions you made.
- Performance. The code should run as efficiently and quickly as possible. If you need to tradeoff memory and performance then go with using more memory to make it faster (within the limits above).
- Standardization. We like to see the use of established programming idioms and patterns, as well as evidence of familiarity with standard language features. We prefer usage of standard STL structures and algorithms instead of using your own code where an algorithm or structure is readily available.

Effort expectations

You will be given 48 hours to return the test once you receive the instructions. We expect this exercise to take about 2 hours of your time but there are no limits to the time you spend on it other than the 48 hour deadline. Please return the completed code (preferably in a .tar.gz gzipped tar archive) which has all source code files and any build configurations. Make sure to provide commands to build the program, as well as the command to run the compiled program on the provided sample input. You do not need to include any compiled binaries or program output.

Please do not hesitate to reach to us if you feel these requirements are not clear, have a particular question, or would like to clarify any detail.

Good luck!