Highway I

Submission deadline:	**2018-12-09 23:59:59**
Late submission with malus:	**2019-01-06 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**1.4667**
Max. assessment:	**3.0000** (Without bonus points)
Submissions:	10 / 20 Free retries + 10 Penalized retries (-10 % penalty each retry)
Advices:	2 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to develop a program to compute the toll.

Assume a toll collection system. The toll is always computed as the number of kilometers times the toll rate. However, the rates are different, e.g. the use of the highway, ecology fee, road taxes, ... For the sake of simplicity, we assume there is 26 such rates, denoted by upper case letters A to Z.

The rates may vary for different sections of the highway. The input defines the length of the highway section and the list of toll rates. The list does not have to be complete, the unlisted toll rates will be copied from the previous highway section. For example:

```
{
  [ 50: A=10.5, E=80 ],
  [ 30: Z=20, A=7.5, X=130 ],
  [ 200: A=0, E=300 ]
}
```

is interpreted as:

- the first highway section is 50 km long (from 0 km to 50 km). Toll rate A is 10.5 per km and toll rate E is 80 per km. The other rates are zero (the first highway section does not have any previous section to copy the remaining toll rates from),
- the second section is 30 km long (from 50 km to 80 km). Toll rate A is changed to 7.5 per km, toll rate E is copied from the predecessor as 80 per km, rate X is 130 per km, and rate Z is 20 per km. The other toll rates remain zero,
- the third section is 200 km long (from 80 km to 280 km). Rate A is zero, rate E is 300 per km, X 130 per km, and Z is 20 per km (both X and Z are copied from the previous section). The other rates are zero.

Having the toll rates set, the program is to compute the toll to pay. The program is given a list of pairs of integers. The integers represent the entry/exit point on the highway (in kilometers). Given that interval, the program displays all nonzero toll fees A to Z to pay.

The input of the program is the list of highway sections and the toll rates in the sections. The list is enclosed in curly braces, each section is enclosed in square brackets. The length of the section is a positive integer, the toll rates are identified by uppercase letters A to Z, the toll rate is a decimal number (positive or zero). Following the sections, there is a list of queries. Each query consists of two integers, the entry/exit kilometer of the highway. The order of the entry/exit kilometer may be either `entry < exit` or `entry > exit` (driving in the opposite direction). The queries are processed until end-of-file is reached.

If the entry/exit kilometer is exactly equal to the boundary of a highway section (with different toll fees), then there apply only the toll rates from the section relevant to the direction of travel. In the example highway sections above, the queries:

```
40 50
50 10
```

are computed using the toll rates only from the first section. On the other hand, queries:

```
50 60
70 50
```

are computed using the toll rates only from the second section. If the query covers more than one highway section, the individual sections contribute to the toll based on the number of kilometers, e.g.:

```
30 75    20km in the first section and 25km in the second section
100 20   30km in the first section, 30km in the second section (whole section) and 20km in the third section
```

The program must detect an invalid input. The following is considered invalid:

- the length of the highway section is not a positive integer (zero is invalid too),
- the toll rate is not identified by an uppercase letter A to Z,
- the toll rate is not a decimal or is negative (i.e., toll rate must be either positive or zero),
- there is no section defined for the highway (valid highways must define at least one section),
- a query is not formed by two integers,
- highway entry and exit kilometer are the same,
- highway entry/exit kilometer are outside of the highway (smaller than zero, or greater than the overall length of the highway),
- there are missing or redundant some separators (square braces, curly braces, colons, commas).

The program is tested in a limited environment. Both time and memory is limited. The limits are set such that a correct implementation of a naive algorithm passes all tests. The problem needs to design a reasonable memory representation of the highway sections. The available memory is proportional to the input problem size. Therefore, a static allocation (of e.g. 100000 highway sections) may fail. Be prepared for many highway sections and long highways (highway length of millions or billions km).

Sample program run:

```
Toll:
{ [ 50: A=10.5, E=80 ], [ 30: Z=20, A=7.5, X=130 ], [ 200: A=0, E=300 ] }
Search:
10 70
10 - 70: A=570.000000, E=4800.000000, X=2600.000000, Z=400.000000
100 200
100 - 200: E=30000.000000, X=13000.000000, Z=2000.000000
55 166
55 - 166: A=187.500000, E=27800.000000, X=14430.000000, Z=2220.000000
166 55
166 - 55: A=187.500000, E=27800.000000, X=14430.000000, Z=2220.000000
0 280
0 - 280: A=750.000000, E=66400.000000, X=29900.000000, Z=4600.000000
49 50
49 - 50: A=10.500000, E=80.000000
49 51
49 - 51: A=18.000000, E=160.000000, X=130.000000, Z=20.000000
50 51
50 - 51: A=7.500000, E=80.000000, X=130.000000, Z=20.000000
50 50
Invalid input.
```

```
Toll:
{ [ 1000000 : A = 3.25 , C = 1 ] , [ 10000000 : B = 1.75 , D = 2 ], [1000000000:X=7] }
Search:
500000 3000000
500000 - 3000000: A=8125000.000000, B=3500000.000000, C=2500000.000000, D=4000000.000000
20000000 2000000000
Invalid input.
```

```
Toll:
{[1:A=3.25,C=1],[1:B=1.75,D=2]}
Search:
1 2
1 - 2: A=3.250000, B=1.750000, C=1.000000, D=2.000000
0 2
0 - 2: A=6.500000, B=1.750000, C=2.000000, D=2.000000
0 1
0 - 1: A=3.250000, C=1.000000
```

```
Toll:
{[5:A=10],[6:B=10],
[7:A=0],[8:B=0]}
Search:
3 5
3 - 5: A=20.000000
7 9
7 - 9: A=20.000000, B=20.000000
12 14
12 - 14: B=20.000000
20 23
20 - 23:
```

```
Toll:
{ [ 30: a=10 ] }
Invalid input.
```

```
Toll:
{ [ 30: A=20 ]
Invalid input.
```

Advice:

- The sample runs above list both the output of your program (bold face text) and user input (regular text). The bold/regular formatting is included here, in the problem statement page, to increase readability of the listing. Your program must output the text without any additional markup.
- There is no explicit upper limit on the number of highway sections, the problem assumes dynamic allocation. A statically allocated storage either fails the basic test (statically allocated array is too big), or fails the borderline test (the array is too small).
- Use structures to store the highway sections.
- Use C language to implement the program. Do not use C++, do not use STU (`vector` ...). If STE is used, the program will not compile.
- There are decimal numbers on the output, thus, your program may result in numbers a bit different from the reference. The testing environment accepts small differences up to 1 ‰.
- The bonus test inputs many highway sections and many queries. An efficient algorithm and some input data pre-processing is required to pass the bonus test.

Sample data: **Download**

☐ **Reference**