Billboards I

Submission deadline:	**2018-12-02 23:59:59**
Late submission with malus:	**2019-01-06 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**3.3000**
Max. assessment:	**3.0000** (Without bonus points)
Submissions:	11 / 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 that plans the positions of billboards along a highway.

Even though billboards positioned along highways impose some threat, they are still used as an efficient form of advertising. The efficiency of the billboards may be increased if the layout is well thought up. The task is to develop a program to solve the planning. We assume there is a highway of length `len`. There are certain places along the highway suitable to place the billboards (conversely, the billboards cannot be placed anywhere else). The suitable places are identified by their position from the beginning of the highway; the position is an integer in the range `[1;len-1]`. There is at most 1000000 such places.

We are about to use some of the places for our advertising. Our concern is to use as few billboards as possible. On the other hand, we require that the distance between two consecutive billboards is at most `dist`. The first billboard must be placed within `dist` from the beginning of the highway. Similarly, the last billboard must be placed within `dist` from the end of the highway. Based on these criteria, the program shall find the required number of billboards.

The input of the program is an integer - the length of the highway `len`. The length is followed by the list of suitable places. Each place is described by an integer; the integers are enclosed in curly braces and separated by commas. Following the places, there is a list of queries. Each query is formed by an integer - the distance `dist`. The queries are read from the input until the end-of-file is reached.

The output of the program is the required number of billboards for each input query. The billboards cannot be placed for certain combinations of positions and distances. A different output is printed out in these case. The exact output format is shown in the sample runs below.

The program must validate input data. If the input is invalid, the program must detect it, it shall output an error message (see below), and terminate. If displayed, the error message must be sent to the standard output (do not send it to the error output) and the error message must be terminated by a newline (\n). The input is considered invalid, if:

- the distances are not valid integers,
- highway length is zero or negative,
- suitable billboard positions do not fit the range `<1;len-1>`,
- the list of suitable billboard positions is empty,
- there is more than 1000000 billboard positions in the input,
- the distance `dist` is negative or zero,
- there are missing/redundant separators (curly brace, comma, colon).

Sample program runs:

```
Suitable positions:
1000: { 250, 500, 750 }
Distances:
800
Billboards: 1
500
Billboards: 1
300
Billboards: 3
250
Billboards: 3
200
N/A
```

```
Suitable positions:
1000 : { 250 , 300 , 550 , 750 }
Distances:
371
Billboards: 3
507
Billboards: 2
273
Billboards: 4
561
Billboards: 1
```

```
Suitable positions:
1000:{481,692,159,843,921,315}
Distances:
1000
Billboards: 0
999
Billboards: 1
```

519
Billboards: 1
518
Billboards: 2
377
Billboards: 2
376
Billboards: 3
315
Billboards: 3
314
Billboards: 4
308
Billboards: 4
307
Billboards: 5
211
Billboards: 5
210
N/A

Suitable positions:
3:{1,2,1,2}
Distances:
1
Billboards: 2
10
Billboards: 0

Suitable positions:
500:{250,830}
Invalid input.

Suitable positions:
330:{15,240 310
Invalid input.

Advice:

- The sample runs above list both the output of your program (boldface font) and user input (regular font). 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.
- Do not forget the newline (\n) after the last output line.
- The program is to store the suitable billboard positions. There is up to 1000000 such positions. Therefore, the positions may be stored in a statically allocated array. This task does not require dynamic memory allocation.
- A reasonable implementation of the naive algorithm passes all tests except the bonus tests. The naive algorithm just passes the suitable positions an allocates them with respect to the maximal allowed distance.
- Speed tests #1 and #2 are extra bonus tests. The tests input long highways with many suitable places for the billboards. Moreover, there is a lot of queries with various distances. The naive algorithm is too slow to finish in the given time limit. An efficient algorithm must be used, moreover, it may be advantageous to pre-process the input data.
- The inputs are chosen such that the input values fit into the `int` data type.
- Your program will be tested in a restricted environment. The testing environment limits running time and available memory. The exact time and memory limits are shown in the reference solution testing log. The memory limit is approx. 500MiB, i.e., more than enough to store the input data in a statically allocated array.
- Textual description of valid input data structure is not 100% exact. Therefore we provide a formal specification of the input language in EBNF:

```
input      ::= { whiteSpace } integer { whiteSpace } ':' { whiteSpace } '{' posList '}'
               queryList { whiteSpace }
whiteSpace ::= ' ' | '\t' | '\n' | '\r'
posList    ::= { { whiteSpace } integer { whiteSpace } ',' } { whiteSpace } integer { whiteSpace }
queryList  ::= { { whiteSpace } integer }
integer    ::= digit { digit }
digit      ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

Sample data: **Download**

Reference

11	**2018-11-24 20:02:05**	**Download**
Submission status:	Evaluated	
Evaluation:	3.3000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %