Sequence #2

Submission deadline:	**2018-11-18 23:59:59**
Late submission with malus:	**2019-01-06 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**6.6000**
Max. assessment:	**5.0000** (Without bonus points)
Submissions:	16 / 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 analyzes numbers from a given interval. The problem is an extension of the simpler variant. The analyzed numbers will not always be binary. Instead, the program shall analyze the sequences of numbers written in different bases.

An instance of the problem is defined by the radix `R`, a lower boundary `LO`, and an upper boundary `HI`. Assume all integers from the interval `[LO;HI]` written as radix-`R` numbers. The numbers form a sequence of digits. The program shall:

- compute the overall length of the sequence (i.e., the total number of digits),
- compute the number of zeros in the sequence, or
- compute the longest contiguous run of zeros in the sequence.

For example, input problem instance `r4:[10;20]` defines sequence of numbers from 10 to 20 written as base-4 numbers. The sequence is:

```
  22 23 30 31 32 33 100 101 102 103 110
```

The sequence is 27 digits long, out of which, there are 7 zeros. The longest contiguous run of zeros is of length 2. Similarly, input input problem instance `r13:[10;20]` defines a sequence of numbers from 10 to 20 written as base-13 numbers. The sequence is:

```
  A B C 10 11 12 13 14 15 16 17
```

The sequence is 19 digits long, out of which, there is 1 zero, and the longest zero sequence is just 1.

The input of the program is a list of problem instances to solve. In general, the problem list may be of any length. Each problem instance is defined as a radix, a closed interval of numbers, and a code of the desired computation. The input may have two forms:

```
rR: <LO;HI> OP
<LO;HI> OP
```

where `R` is a decimal integer that defines the radix. If the radix is omitted (the second short form), we assume radix `R=10`. Values `LO` and `HI` are decimal integers that define a closed interval of numbers to analyze. Finally, `OP` is the code of the desired computation. It is a single character:

- `l` to compute the overall length of the sequence,
- `z` to compute the number of zeros in the sequence, or
- `s` to compute the longest run of zeros in the sequence.

There is a list of problem instances in the program's input. The program shall read the individual problems instances, process them, and display the desired result. The processing of the input ends when the program reaches the end of file (EOF).

The output of the program is the result of the desired computation; the result is shown for each input problem instance. 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 radix or the interval boundaries are invalid (are not valid decimal integers),
- the boundaries are negative,
- the lower bound is greater than the upper bound,
- the radix is not in the range 2 to 36,
- the desired computation is not any of `l`, `z`, and `s`, or
- there are missing/redundant separators (angle brace, semicolon).

Sample program runs:

```
Enter the intervals:
<0;100> l
Digits: 193
<0;100> z
Zeros: 12
<0;100> s
Sequence: 2
r2:<11;21> l
Digits: 50
r2:<11;21> z
Zeros: 22
r2:<11;21> s
Sequence: 4
r4:<10;20> l
Digits: 27
```

```
r4:<10;20> z
Zeros: 7
r4:<10;20> s
Sequence: 2
r 3 : < 70 ; 112 > l
Digits: 204
r 3 : < 70 ; 112 > z
Zeros: 70
r 3 : < 70 ; 112 > s
Sequence: 4
r10: <6; 100> l
Digits: 187
r10: <6; 100> z
Zeros: 11
r10: <6; 100> s
Sequence: 2
r36 :<44;144> l
Digits: 202
r36 :<44;144> z
Zeros: 3
r36 :<44;144> s
Sequence: 1
```

```
Enter the intervals:
<10;10> l
Digits: 2
<10;10> z
Zeros: 1
<10;10> s
Sequence: 1
r19:<61;61> l
Digits: 2
r19:<61;61> z
Zeros: 0
r19:<61;61> s
Sequence: 0
r44:<3,16> l
Invalid input.
```

```
Enter the intervals:
<31;27> l
Invalid input.
```

```
Enter the intervals:
33;41 z
Invalid input.
```

```
Enter the intervals:
<33;asdf> s
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 can be developed without additional functions (i.e., in one big `main`). However, if divided into functions, the program is readable and easier to debug.
- A reasonable implementation of the naive algorithm passes all tests except the bonus tests. The naive algorithm passes all numbers in the input interval and analyzes their binary representation. Loops, conditions, and a few integer variables are needed in the implementation. There is no need for arrays or strings. Do not construct the entire binary sequence in the program's memory. Such implementation is clumsy, error-prone, laborious, and slow.
- Speed test #1 is a bonus test. The test inputs long intervals, the total number of digits is to be computed. The number of digits must be computed using an efficient algorithm to pass the time limits. Such an algorithm does not scan all numbers in the input interval. Instead, it establishes the result based just on the interval boundaries. The other computations (count of zeros, the longest run of zeros) are not tested in speed test #1, thus efficient algorithms are not needed to pass. If a program passes the bonus test, it is awarded some extra points (above the nominal 100%).
- Speed test #2 is similar to the speed test #1. All three computations (total number of digits, count of zeros, the longest run of zeros) are tested, moreover, the input intervals are long. All computations must be done by efficient algorithms to pass the test. The implementation may need a few small arrays (up to 100 elements) to store some intermediate results.
- The inputs are chosen such that the input values fit into the `int` data type. The same applies to the resulting values in the mandatory and optional tests. The intervals in the bonus tests are very long, thus the results may exceed the limits of `int` data type. It is recommended to use the `long long int` data type when developing a program that is supposed to pass the bonus tests.
- Please strictly adhere to the format of the output. The output must exactly match the output of the reference program. The comparison is completed by a machine, meaning an exact match is required. If your program provides output different from the reference, the program is considered malfunctioning. Be very careful since the comparison is sensitive even to whitespace characters (spaces, newlines, tabulators). Please note that all output lines are followed by a newline character (\n). The new line character must be present after the last line of the output and after the error messages. Download the enclosed archive. The archive contains a set of testing inputs and the expected outputs. Read Progtest FAQ to learn how to use input/output redirection and how to simplify testing of your programs.

- 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. However, neither time nor memory limit could cause a problem in this simple program. Next, the testing environment prohibits the use of functions which are considered "dangerous" (functions to execute other processes, functions to access the network, ...). If your program uses such functions, the testing environment refuses to execute the program. Your program may use something like the code below:

```
int main ( int argc, char * argv [] )
{

  ...

  system ( "pause" ); /* prevent program window from closing */
  return 0;
}
```

  This will not work properly in the testing environment - it is prohibited to execute other programs. (Even if the function were allowed, this would not work properly. The program would infinitely wait for a key to be pressed, however, no one will press any key in the automated testing environment. Thus, the program would be terminated on exceeded time limit.) If you want to keep the pause for your debugging and you want the program to be accepted by the Progtest, use the following trick:

```
int main ( int argc, char * argv [] )
{

  ...

#ifndef __PROGTEST__
  system ( "pause" ); /* this is ignored by Progtest */
#endif /* __PROGTEST__ */
  return 0;
}
```

- Textual description of valid input data structure is not 100% exact. Therefore we provide a formal specification of the input language in EBNF:

```
input      ::= full | short
full       ::= { whiteSpace } 'r' { whiteSpace } integer { whiteSpace } ':' short
short      ::= { whiteSpace } '<' { whiteSpace } integer { whiteSpace } ';'
               { whiteSpace } integer { whiteSpace } '>' { whiteSpace } computation { whiteSpace }
whiteSpace ::= ' ' | '\t' | '\n' | '\r'
computation::= 'l' | 'z' | 's'
integer    ::= digit { digit }
digit      ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

Sample data:	**Download**

Reference

16	**2019-01-02 15:20:33**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Max. run time: 0.006 s (limit: 0.500 s)
    - Total run time: 0.026 s
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test': success
    - result: 100.00 %, required: 50.00 %
    - Max. run time: 0.006 s (limit: 0.500 s)
    - Total run time: 0.024 s
    - Optional test success, evaluation: 100.00 %
  - Test 'Invalid input test': success
    - result: 100.00 %, required: 50.00 %
    - Max. run time: 0.006 s (limit: 0.100 s)
    - Total run time: 0.152 s
    - Optional test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 100.00 %, required: 50.00 %
    - Max. run time: 0.006 s (limit: 1.000 s)
    - Total run time: 0.088 s
    - Optional test success, evaluation: 100.00 %
  - Test 'Speed test #1': success
    - result: 100.00 %, required: 100.00 %
    - Max. run time: 0.005 s (limit: 0.100 s)