University of Arkansas
Department of Computer Science and Computer Engineering
2019 High School Programming Contest
Problems

Wing Ning Li Adam Higgins Paul Martin Tyler Moore Matthew Luther
John Shell

Revision Date: March 6, 2019

Please carefully read the output specification. Submitted solutions are graded by a differential program, i.e., the output produced by a submission will be compared *byte-for-byte* with the output of the judges' solutions. Thus, even though submitted solutions may produce output that is *technically* correct, it will be counted as incorrect if these specifications are not followed exactly.

For all problems, solutions receive input from standard input (*stdin*) and produce output on standard output (*stdout*).

Restricting input and output to stdin and stdout does not prevent a program from processing files. Let `Main` be the program name, `Input` be the input file name, and `Output` be the output file name. The following command at the terminal allows the program to read and write the input and output files respectively:

```
./Main < Input > Output      # for C or C++, assuming Main is the executable
java Main < Input > Output   # for Java, assuming Main is a class file
```

For all problems, an *integer* is defined as an optional minus sign followed by one digit from the set 1..9 followed by zero or more digits from the set 0..9. `3`, `-21`, and `309` are integers; `0123`, `- 4`, and `+38` are not.

A *positive integer* is an integer whose value is greater than 0. You may assume the value of a positive integer is no more than $2^{32} - 1$, or $4,294,967,295$.

A *string*'s maximum length is 16,384.

For all problems, a *line* is defined as zero or more characters followed by a newline ('\n') character. Unless otherwise directed, a line of output should have no leading or trailing whitespace. A program producing the integer 27 on one line of output should consist of the character '2' followed by the character '7' followed by the newline ('\n') character.

Input for a problem consists of multiple test cases; each test case is followed by a blank line. You must read input until the *end-of-file*. Refer to the code for the Warm Up problem for hints and ideas for handling input properly.

For each test case, your program must output the result following the specified format in each problem. Output a blank line after the output of each test case. Refer to the code for the Warm Up problem again for hints and ideas for handling output properly.

1 Add them up

In the warmup problem, You are asked to write a program to add a sequence of integers in an input line and to write the sum to the standard output. For this problem, you are asked to write a program to add all the positive integers together and all the negative integers together. Then write the two sums to the standard output.

input each input instance consists of two lines. The first line is a positive integer, n, indicating the number of integers in the second line. The second line contains n integers separated by one or more spaces. The input may have multiple instances and each instance is followed by a blank line; therefore, your program must process the input until the *end-of-file.*

output for each input instance, there is a line of output containing: `The positive sum of the input is x and the negative sum is y.` The x stands for the sum of positive integers in the input instance and y stands for the sum of negative integers in the input. *There must also be a blank line following your answer output.*

Example:

Input

```
2
6 9

5
-1 -2 -3 4 5
```

Output

```
The positive sum of the input is 15 and the negative sum is 0.

The positive sum of the input is 9 and the negative sum is -6.
```

2 Analyze and display

Given a sequence of integers, if all the values are positive, output the values in increasing order (more precisely nondecreasing); If all the values are negative, output the values in decreasing order (more precisely non increasing); If the values are positive and negative values only, output the positive and negative values alternatingly, when it is possible, such that all positive values are decreasing and all negative values are increasing; If all the values do not satisfy the above, output `The problem does not specify how to output!` (Note that 0 is neither a positive nor a negative integer.)

input each input instance consists of two lines. The first line is an integer n, $0 < n \leq 1000$, indicating the number of integers in the second line. The second line contains n integers separated by one or more spaces. The input may have multiple instances and each instance is followed by a blank line; therefore, your program must process the input until the *end-of-file.*

output for each input instance, there is a line of output. The actual output depends on the analysis of the input according the instructions provided in the problem. *There must also be a blank line following your answer output.*

Example:

Input

```
5
4 1 3 8 7

5
-4 -1 -3 -8 -7

6
-4 1 -3 8 7 2

5
-4 1 -3 8 0
```

Output

```
1 3 4 7 8

-1 -3 -4 -7 -8

8 -4 7 -3 2 1

The problem does not specify how to output!
```

3 Extract digits

A decimal number system has ten digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and a decimal number is a sequence of digits where the left most digit is not 0 if the number has two or more digits. A Hexadecimal number system has sixteen digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (where A stands 10 in decimal, B stands 11 in decimal, C stands 12 in decimal, D stands 13 in decimal, E stands 14 in decimal, and F stands 15 in decimal). A hexadecimal number is similarly defined as the decimal number with additional digits A to F.

Write a program to extract each hexadecimal digits. Output the hexadecimal digits with a + separating each digit if the number is a hexadecimal number, and output `This is not a hexadecimal number.`, otherwise.

input Each test case contains a sequence of characters, where the number of characters is between 1 and 100. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output the required text as shown in the examples *and a blank line.*

Example:

Input

```
9876543210ABCDEF

EFG
```

Output

```
9+8+7+6+5+4+3+2+1+0+A+B+C+D+E+F

This is not a hexadecimal number.
```

4 Determine carry in hexadecimal addition

Children learned multi-digit addition in decimal numbers from right to left, one digit at a time. Adding two hexadecimal numbers is similar, where if the sum of the two digits is 16 or more a carry is generated to the next position and the value in the current position is the sum minus 16. For example, if the two digits of a position is B (11) and 7, the sum is 18. Hence 1 is carried to the next position and the current position has value 2.

Write a program to count the number of carry operations for each hexadecimal addition.

input Each test case contains two valid hexadecimal numbers separated by a space, where the number of digits of each number is between 1 and 100. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output the number of carry operations as shown in the examples *and a blank line.*

Example:

Input

```
123 456

ABC 754A

ABC 743
```

Output

```
No carry operation.

3 carry operations.

1 carry operation.
```

5 Finding a closer point

A point in the plane geometry is represented by (x,y) coordinates, where x and y are real numbers. The distance between the two points is based on right triangles and the Pythagorean Theorem.

Given three points of the plane one after another in an input line, write a program to determine if the second point or the third point has a shorter distance to the first point and output the point with a shorter distance. Output both second and third points if their distances to the first point are identical.

input Each test case has 6 real numbers separated by a space, where the first pair is the (x,y) coordinates of the first point, the second pair is the second point, the third pair the third point. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output the point(s) closer to the first point as shown in the example *and a blank line as second*

Example:

Input

```
0 0 1 1 2 2

0 0 1 2 2 1

1.2 1.1 0 0 2.2 1.1
```

Output

```
1 1

1 2 2 1

2.2 1.1
```

6 Minimum distance wiring

In the previous problem, we know how to compute the distance between two points. Given a set of distinct points, we would like to connect all the points together by pairwise connections such that the wire length used to connect all the points is minimized. Note that we assume the wire length to connect a pair of points is the same as the distance between the pair of points.

Write a program to computer the minimum wire length rounded to the nearest integer (3.1 to 4 and 3.6 to 4 as well) to connect a given set of points.

input Each test case has 1 + 2n real numbers. The first number is an integer n, $1 \leq n \leq 1000$, which is followed by n pairs of points as 2n real numbers. The 1+2n numbers are separated by spaces. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output the smallest integer that is grater than or equal to the minimum wire length *and a blank line as second.*

Example:

Input

```
3 1.0 1.0 2.0 2.0 2.0 4.0
```

Output

```
4
```

7 Counting steps

Given two positive integers x, y, where x is no greater than y. The process of going from integer x to integer y involves a sequence of steps. The length of each step must be non-negative and can be one bigger than, equal to, or one smaller than the length of the previous step. The length of the first step and the last step must be 1.

Write a program to determine the minimum number of steps in order to get from x to y.

input Each test case has two integers x y separated by a space, where $0 < x \leq y < 2^{31}$. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output the minimum number of steps to get from x to y as shown *and a blank line as second*

Example:

Input

```
45 48

45 49

45 50
```

Output

```
3

3

4
```

8 Counting steps again

This problem is similar to the previous problem with some adjustment to the length of the allowed steps as the following. The length of each step must be non-negative and can be two bigger than, equal to, or two smaller than the length of the previous step. The length of the first step must be 1 and the length of the last step may be any positive value.

Write a program to determine the minimum number of steps in order to get from x to y.

input Each test case has two integers x y separated by a space, where $0 < x \leq y < 2^{31}$. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output the minimum number of steps to get from x to y as shown *and a blank line as second*

Example:

Input

```
45 48

45 49

45 50
```

Output

```
3

2

3
```