acm International Collegiate Programming Contest

UC Programming Contest 2004
Warm Up Session
University of Carabobo, Venezuela

October 29, 2004

This problem set should contain a rules section and three (3) problems on five (5) numbered pages. Please inform a runner immediately if something is missing from your problem set.

Rules

- There are three (3) problems for each team to be completed in two (2) hours.
- You may use any of the following programming languages: Pascal, C, C++ or Java. You may use different programming languages for different problems and even for different submissions of a problem.
- All problems require that you read test data from the standard input and write results to the standard output.
- All problems have a source file name shown below the title of the problem. The extensions you must use for your source files are: *.pas* for programs written in Pascal, *.c* for those written in C, *.cpp* for those in C++, and *.java* for those in Java.
- You may use any of the standard libraries that your chosen programming language provides. You may not use any other library that requires an extra flag to be passed to the compiler command. If you do this, judges will probably get a compilation-linking error in your program.
- Output corresponds exactly to the provided sample output format, including (mis)spelling and spacing. Multiple spaces will not be used in any of the judges' output, except where explicitly stated.
- Your solution to any problem should be submitted for judging using PC^2 software only. Once you have submitted a solution, it will reach the judges. The time it takes for your problem to be judged will depend on how busy the judges are. Once your submission has been judged, you will receive a message through PC^2 indicating the judgment. If your solution is accepted the message will be "Yes", if it's not then the message will be "No" along with the type of error the judges encountered, which may be: "Compilation Error", "Run-time Error", "Time-limit Exceeded", "Wrong Answer", "Excessive Output", "Incomplete Output" or "Output Format Error".

Problem A: Reverse

Source File: reverse.*

Statement of the Problem

As you may know, in most languages, texts are written from left to right. However, exceptions exist, in which case people read from right to left (e.g. Hebrew).

Having heard about this phenomenon, and always willing to make a few bucks, John believes that he can aid the translation business by making a preprocessor for translation from left-to-right languages to right-to-left languages. The way John sees it, if we would reverse the normal text line by line, half of the work has been done.

Unfortunately, John never got around actually writing the program, since he got involved in an even more promising project involving Unicorns.

Maybe you can write this program for him.

Input Specification

The first line of input is an integer N specifying the number of test cases. Each test case consists of a single line, where each line is no longer than 80 characters. The newline is not considered to be part of the line itself.

Output Specification

For each test case, print on a single line the characters of the input line in reverse order.

Sample Input

```
3
Frankly, I don't think we'll make much
money out of this scheme.
madam I'm adam
```

Expected Output for the Sample Input

```
hcum ekam ll'ew kniht t'nod I ,ylknarF
.emehcs siht fo tuo yenom
mada m'I madam
```

Problem B: Recognizing Good ISBNs

Source File: isbn.*

Statement of the Problem

Most books now published are assigned a code which uniquely identifies the book. The International Standard Book Number, or ISBN, is normally a sequence of 10 decimal digits, but in some cases, the capital letter X may also appear as the tenth digit. Hyphens are included at various places in the ISBN to make them easier to read, but have no other significance. The sample input and expected output shown below illustrate many valid, and a few invalid, forms for ISBNs.

Actually, only the first nine digits in an ISBN are used to identify a book. The tenth character serves as a check digit to verify that the preceding 9 digits are correctly formed. This check digit is selected so that the value computed as shown in the following algorithm is evenly divisible by 11. Since the check digit may sometimes need to be as large as 10 to guarantee divisibility by 11, a special symbol was selected by the ISBN designers to represent 10, and that is the role played by X.

The algorithm used to check an ISBN is relatively simple. Two sums, $s1$ and $s2$, are computed over the digits of the ISBN, with $s2$ being the sum of the partial sums in $s1$ after each digit of the ISBN is added to it. The ISBN is correct if the final value of $s2$ is evenly divisible by 11.

An example will clarify the procedure. Consider the (correct) ISBN 0-13-162959-X (for Tanenbaum's Computer Networks). First look at the calculation of $s1$:

```
-------------------------------------------------------------
digits in the ISBN    0   1   3   1   6   2   9   5   9   10(X)
partial sums          0   1   4   5  11  13  22  27  36   46
-------------------------------------------------------------
```

The calculation of $s2$ is done by computing the total of the partial sums in the calculation of $s1$:

```
-------------------------------------------------------------
s2 (running totals)   0   1   5  10  21  34  56  83  119  165
-------------------------------------------------------------
```

We now verify the correctness of the ISBN by noting that 165 is, indeed, evenly divisible by 11.

Input Specification

The first line of input is an integer N specifying the number of test cases. The input data for this problem will contain one candidate ISBN per line of input, perhaps preceded and/or followed by additional spaces. No line will contain more than 80 characters, but the candidate ISBN may contain illegal characters, and more or fewer than the required 10 digits. Valid ISBNs may include hyphens at arbitrary locations.

Output Specification

The output should include a display of the candidate ISBN and a statement of whether it is legal or illegal. The expected output shown below illustrates the expected form.

Sample Input

```
15
0-89237-010-6
0-8306-3637-4
  0-06-017758-6
    This_is_garbage
1-56884-030-6
    0-8230-2571-3
    0-345-31386-0
    0-671-88858-7
    0-8104-5687-7
    0-671-74119-5
    0-812-52030-0
    0-345-24865-1-150
0-452-26740-4
    0-13-139072-4
    0-1315-2447-X
```

Expected Output for the Sample Input

```
0-89237-010-6 is correct.
0-8306-3637-4 is correct.
0-06-017758-6 is correct.
This_is_garbage is incorrect.
1-56884-030-6 is correct.
0-8230-2571-3 is correct.
0-345-31386-0 is correct.
0-671-88858-7 is correct.
0-8104-5687-7 is correct.
0-671-74119-5 is correct.
0-812-52030-0 is correct.
0-345-24865-1-150 is incorrect.
0-452-26740-4 is correct.
0-13-139072-4 is correct.
0-1315-2447-X is correct.
```

Problem C: Your Punishment

Source File: punish.*

Statement of the Problem

You have been behaving very bad at school, bragging all the time because you think you're the smartest of the class. Because of this, your math teacher has decided to punish you every time you misbehave by making you calculate the sum of a sequence of numbers. If you're so smart as you say you are, you may be able to write a program to do several sequence sums at a time. That way, you can get rid of your punishment.

Input Specification

The input contains several lines of pairs of integers A and B ($1 \leq A \leq B \leq 70000$). After those lines, there's a line with two integers equal to zero. All pairs of integers are separated by an arbitrary amount of whitespace.

Output Specification

For each pair of integers A and B, write the sum of all the numbers between them (not including A and B). If there are no numbers between them, write the message "No numbers to add". Use the format specified in the sample output. On each of the lines, after specifying the sums, leave some space and write the message "My program rocks!" so that it gets right justified on the 80th column of the text, i.e., the '!' symbol must appear exactly on the 80th column.

Sample Input

```
9 15
317 318
5497  5499
3 8
0 0
```

Expected Output for the Sample Input

```
10 + ... + 14 = 60                                                  My program rocks!
No numbers to add.                                                  My program rocks!
5498 = 5498                                                         My program rocks!
4 + ... + 7 = 22                                                    My program rocks!
```