ICPC North America Regionals

icpc international collegiate programming contest

The 2021 ICPC Southern California Regional Contest

Official Problem Set

icpc global sponsor
programming tools

icpc international collegiate programming contest

icpc.foundation

upsilon pi epsilon
honor society

aws educate

icpc EdTech
gold sponsor

2021/2022 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
Ultimate Binary Watch

The *Ultimate Binary Watch* is a "maker" project that uses small LEDs to display the time on a watch face. The display uses four columns of four LEDs each, with each column representing a digit of the current time in hours and minutes. Time is displayed in 24-hour format, with the left-most column displaying the tens position for hours, the next column displaying the ones position for hours, the next column displaying the tens position for minutes, and the right-most column displaying the ones position for minutes. The bottom LED of each column is the low-order bit, with the bit positions increasing moving up the column. For example, the time 1615 would be displayed as shown in Figure 1.

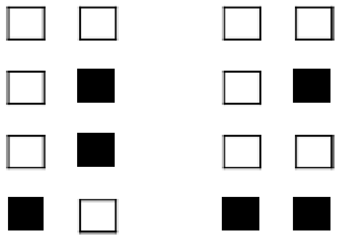

Figure 1. Watch face showing a 24-hour time of 1615.

Your team is to write a program that will take a series of 24-hour times and print the corresponding watch faces. Input is a list of one to sixty valid 24-hour times, each on a separate line beginning in the first column. The input list ends with the end-of-file.

For each time in the input, your program is to print the time on a line starting in the first column, followed by four lines with a representation of the watch face displaying that time. The tens of hours is to be in the first column, the single hours in the third, the tens of minutes in the seventh, and the single minutes in the ninth. Use asterisks to represent bits that are set and periods to represent bits that are clear. Columns not used are to be filled with spaces. No extra whitespace is to appear at the beginning or end of any output line.

Sample Input

```
1615
1900
0830
```

Problem 1
Ultimate Binary Watch (continued)

Output for the Sample Input

```
1615
. .   . .
. *   . *
. *   . .
* .   * *
1900
. *   . .
. .   . .
. .   . .
* *   . .
0830
. *   . .
. .   . .
. .   * .
. .   * .
```

2021/2022 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Bracket Pairing

There are four types of brackets: round `()`, square `[]`, curly {}, and angle `<>`. A bracket sequence is defined to be *valid* as follows:

- An empty sequence is valid.
- If X is a valid bracket sequence, then pXq is a valid bracket sequence, where p is an open bracket, q is a close bracket, and p, q are of the same type.
- If X and Y are valid bracket sequences, then the concatenation of X and Y, $Z = XY$, is a valid bracket sequence.

You have a bracket sequence in which some brackets are given, but the others are unknown and represented by question marks (‘`?`’). You shall fill in each unknown bracket using the four types of brackets described above and obtain a valid bracket sequence. How many different valid bracket sequences can you obtain?

The input is a single line giving a non-empty bracket sequence. The length of the sequence is even and no larger than 20. All sequence characters are either one of the four types of open or close brackets, or a question mark denoting an unknown bracket. There is at least one question mark in the sequence.

Output the number of different valid bracket sequences you can obtain.

Sample Input 1

```
(??)
```

Output for Sample Input 1

```
5
```

Sample Input 2

```
(<{}>??]
```

Output for Sample Input 2

```
1
```

Problem 2
Bracket Pairing (continued)

Sample Input 3

```
(?]]
```

Output for Sample Input 3

```
0
```

2021/2022 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
Exoplanet Lighthouses

As humanity travels to extraterrestrial bodies, the need for human or robotic explorers to determine their location on the surface will be critical. Lighthouses offer a simple solution as line-of-sight navigational beacons. An important measurement for lighthouses is geographic range, the maximum visible, line-of-sight distance between an observer and the lighthouse before the lighthouse is obscured by the horizon due to the curvature of the surface. The surface *traveling* distance corresponding to the geographic range is crucial for planning fuel and oxygen supplies to ensure an accurate—and safe—return to base stations.

Your team is to write a program that computes the maximum surface distance that allows for a line-of-sight between an observer and a lighthouse. Your program is to assume that exoplanets are spherical with a known radius.

Input is a series of one to twenty test cases, one per line, ending with end-of-file. Each line has three real values separated by whitespace: R, h_1, and h_2. R is the radius of the spherical body in kilometers, $(100 \leq R \leq 20000)$. h_1 is the height of the observer above the surface R in meters, $(1 \leq h_1 \leq 1000)$. h_2 is the height of the lighthouse beacon above the surface R in meters, $(1 \leq h_2 \leq 1000)$. The observer is a point-receiver, which can be a sensor or a human eye, and the beacon is a point-source, such as a lamp or laser. See Figure 1.

For each test case, print a line with a single real value for the maximum surface distance D in kilometers between the observer and lighthouse. The result must be within one meter of the judges' reference value.

Sample Input

```
6371.0 1.0 27.0
6371.0 1.0 26.99
6371   1.0  1.0
1737.4 1   27
```

Output for the Sample Input

```
22.117714375394343
22.114279232261655
7.139187161948777
11.550070205049764
```

Problem 3
Exoplanet Lighthouses (continued)

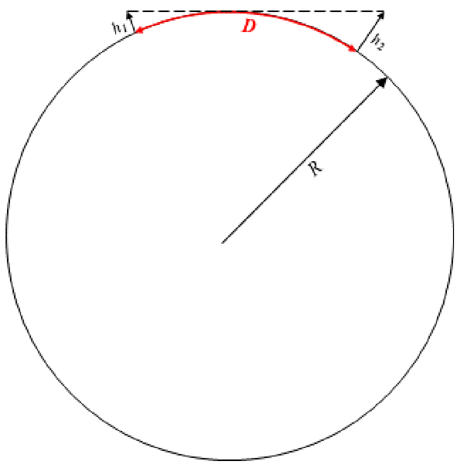

Figure 1. "Line of Sight" between source and receiver on an exoplanet's surface. D is the curved surface distance on the exoplanet. h_1 and h_2 are perpendicular to the local surface.

Problem 4
Lone Rook

On a chess board of r rows and c columns there is a lone white rook surrounded by a group of opponent's black knights. Each knight attacks up to 8 squares as in a typical chess game, which are shown in Figure 1. The knight on the circled square attacks the 8 squares highlighted by dots. The rook can move horizontally and vertically by any number of squares. The rook can safely pass through an empty square that is attacked by a knight, but it must move to a square that is not attacked by any knight. The rook cannot jump over a knight while moving. If the rook moves to a square that contains a knight, it may capture it and remove it from the board. The black knights never move. Can the rook eventually safely move to the designated target square?

Figure 1 illustrates how the white rook can move to the target square at the top-right corner in the first sample case. The rook captures one black knight at the bottom-right of the board on its way.

The first line of input contains two integers r and c ($2 \leq r, c \leq 750$). Each of the next r lines describes one row of the board using c characters: the letter '`R`' represents the white rook, a '`K`' represents a black knight, a dot '`.`' represents an empty square, and the letter '`T`' represents the white rook's target square. There is exactly one '`R`', exactly one '`T`', and at least one '`K`' on the board. It is guaranteed that the white rook starts in a square that is not attacked by any knight. The target square may be attacked by a knight, in which case the knight must be captured before the rook can safely move to the target square.

Print a line contiaining the string "`yes`" if the white rook can move to the target square, or "`no`" otherwise.

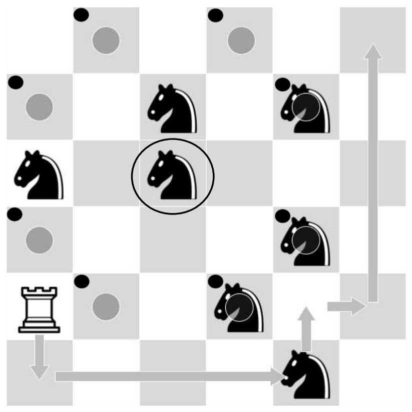

Figure 1. Diagram for Sample Input 1.

Sample Input 1

```
6 6
.....T
..K.K.
K.K...
....K.
R..K..
....K.
```

Output for Sample Input 1

```
yes
```

Sample Input 2

```
3 4
RK..
KK..
...T
```

Output for Sample Input 2

```
yes
```

Sample Input 3

```
4 4
.K..
KR..
K...
.K.T
```

Output for Sample Input 3

```
no
```

2021/2022 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Land Equality

There is a kingdom where the old King wants to divide his land into two pieces and give them to his two descendants. The King's land is a grid of r rows and c columns. Each cell in the grid has an integer value representing the prosperity of the cell, which can be 0 (deserted), 1 (regular), or 2 (fertile). Two cells are connected if they share a side horizontally or vertically.

Each descendant shall receive a single connected piece of land with at least one cell, in which all cells must be directly connected or indirectly connected via other cells. There shall be no leftover cells, which means that each cell must be given to one descendant. The *prosperity* of a piece of land is the product of all the prosperity values of its cells. The King wants the absolute difference between the prosperity of the two descendants' land to be as small as possible. He has asked his best counselor to devise a land division plan between the two descendants.

The first line of input contains two positive integers r and c ($2 \leq r \times c \leq 64$). The next r lines each have c integers giving the prosperity values of the King's land. All those integers are 0, 1, or 2.

Print a line containing the smallest absolute difference between the prosperity of the two descendants' land.

Sample Input 1

```
3 4
1 2 1 1
2 2 1 2
1 2 2 2
```

Output for Sample Input 1

```
8
```

Sample Input 2

```
2 3
0 1 2
0 1 2
```

Output for Sample Input 2

```
0
```

Sample Input 3

```
1 3
2 0 2
```

Output for Sample Input 3

```
2
```

2021/2022 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
Subprime

There is an open math problem: Is every non-negative integer a substring of at least one prime number when expressed in base ten?

Integer a is a substring of integer b if it is equal to an integer derived from b by deleting zero or more consecutive digits of the most and least significant digits of b. For example, 123 is a substring of: $123, 56123, 123789, 50182312365, 41237912123$.

Your team's job is to see how many primes, in a given range, contain a substring of a given integer. We are interested in integer substrings that may include significant leading zeroes.

The input is a series of 1 to 25 lines terminated by end-of-file. Each line is a test case, with two positive integers i, j in base ten and a string k separated by spaces. i and j are indexes into the list of primes in ascending order, with 2 being the first prime, and k is a string of at most six digits which is the integer substring to be searched for. k may be zero or have significant leading zeroes. A prime shall be counted only once even if the digit substring occurs more than once in the prime.

$$i \leq j$$

$$1 \leq i, j \leq 100000$$

For example, consider the input 1 10 9. This is a search from the first (2) through the tenth (29) primes for any containing the substring 9. The answer is 2 (19 and 29).

For each test case print on one line the count of the number of primes containing the substring.

Sample Input

```
1 10 9
1 10000 389
500 1000 43
1 100 8
8000 9000 395
50000 80000 572
90000 100000 9999
1 100000 37502
1 1000 000
1 1000 00
4509 12345 9999
```

Problem 6
Subprime (continued)

Output for the Sample Input

```
2
45
26
8
0
63
5
1
0
10
4
```

2021/2022 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Time of Use

Time-of-Use is a billing technique that charges a consumer for electricity based upon the time of day that the electricity is consumed, instead of charging a flat rate. Owners of electric vehicles (EVs), either fully electric or plug-in hybrids, can reduce their electric bills by choosing when to charge. The introduction of home-based renewable energy sources that provide "free" power, such as rooftop solar, can provide additional savings. However, the generation of renewable power often overlaps with the most expensive time-of-use intervals, making the choice of when to charge for minimum incurred costs no longer a simple matter of charging when the utility rates are least expensive.

Your team will assist EV owners by writing a program that combines average electricity consumption, time-of-use billing intervals, and projected renewable electricity generation to determine the lowest-cost time to begin charging their EVs.

Your program will be provided two days (48 hours) of projected electricity consumption (without plugging in the EV), two days (48 hours) of projected renewable electricity generation, and a series of charging characteristics of the EV. Average consumption and projected electricity generation are reported in 15-minute intervals.

Input to your program is as follows:

- 192 integer values, separated from each other by spaces and/or new-lines, specifying forty-eight hours of predicted consumption in integer watt-hours, in 15-minute intervals, corresponding to interval start times of 00:00, 00:15, 00:30, . . ., through 47:45. Each value will be in the range 0 to 3,000 inclusive.
- 192 integer values, separated from each other by spaces and/or new-lines, specifying forty-eight hours of predicted power generation in integer watt-hours, in 15-minute intervals, corresponding to interval start times of 00:00, 00:15, 00:30, . . ., through 47:45. Each value will be in the range 0 to 2,500 inclusive.
- Time-of-use intervals with cost per kilowatt-hour in integer cents. This input begins with a line containing t, the number of time-of-use intervals ($1 \leq t \leq 24$). Each time-of-use interval is expressed as $hhmm_1$ $hhmm_2$ c, where $hhmm_1$ and $hhmm_2$ are integers expressing hours in the thousands and hundreds place, minutes in the tens and units places. Cost, c, is an integer ($1 \leq c \leq 200$). The first interval starts at time 0000, and the last interval ends at time 4800. Other than the first interval, $hhmm_1$ for a given interval always equals $hhmm_2$ for the previous interval. Interval times are in ascending order and always fall on 15-minute boundaries.
- One to twenty charging configurations, ending with end-of-file. Each configuration contains five integer values separated by whitespace: battery capacity in watt-hours (range 1,000 to 150,000 inclusive), charging power draw in watts (range 500 to 6,000 inclusive), current charge as an integer percentage from 0 to 100, the current time in *hhmm* format, and the desired time in *hhmm* format to achieve full charge. The battery capacity is always a multiple of 100 watt-hours.

For each charging configuration, print the earliest start time on or after the current time to achieve full charge at the lowest cost. Print the time on a line by itself in *hhmm* format.

For the purposes of your program, charging occurs in 15-minute intervals. Any "final" 15-minute interval can either meet or exceed the full charge, however, the charger continues to draw power for the entire interval. Charging is done in a single continuous session. If full charge is not possible by the desired time, display the current time as the start time. If no charging is necessary (battery already at 100 percent charge), print the desired time as the start time.

Problem 7
Time of Use (continued)

Sample Input

```
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
600  600  600  600  600  600  600  600  600  600  600  600  600  600  600  600
600  600  600  600  600  600  600  600  600  600  600  600  600  600  600  600
300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
600  600  600  600  600  600  600  600  600  600  600  600  600  600  600  600
600  600  600  600  600  600  600  600  600  600  600  600  600  600  600  600
600  600  600  600  300  300  300  300  300  300  300  300  300  300  300  300
300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
   300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
   200  200  200  200  200  200  200  200  200  200  200  200  200  200  200  200
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   300  300  300  300  300  300  300  300  300  300  300  300  300  300  300  300
   400  400  400  400  400  400  400  400  400  400  400  400  400  400  400  400
   400  400  400  400  400  400  400  400  400  400  400  400  400  400  400  400
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
5
0000 0730 10
0730 1500 20
1500 2000 40
2000 3200 10
3200 4800 20
12000 1200 0 1600 2000
12000 6000 0 3800 4200
12000 6000 0 3200 4200
12000 2000 0 3200 4200
12000 2000 59 3200 4200
12000 2000 59 1600 4200
```

Output for the Sample Input

```
1600
3800
3700
3600
3700
2000
```

2021/2022 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Culvert

A culvert is a drain or pipe that allows water to flow under a barrier such as a road. Your job is to find the diameter of the pipe needed to carry the discharge from rain on a specified area.

Here are the empirical formulas to use:

$$Q = R \times M \times c \times (S/M)^{0.25}$$

and

$$Q = 3.10 \times D^{2.31} \times H^{0.5}$$

Where:

- Q is the discharge in cubic feet per second,
- R is the maximum rainfall intensity in inches per hour ($0.01 \leq R \leq 12$),
- M is the area in acres ($0.01 \leq M \leq 10^5$),
- c is the drainage coefficient of the area (for example: 0.25 for a farming area, 0.75 for a built up business area with paved streets) ($0.2 \leq c \leq 0.8$),
- S is the average slope of the area in feet per 1000 feet ($0.1 \leq S \leq 1000$),
- D is the diameter of the pipe in feet, and
- H is the minimum head differential of water in feet. That is, the difference in the height of the water above the inlet of the pipe to the height above the outlet. ($0.1 \leq H \leq 25$)

Culvert pipes come in standard diameters of 12, 15, 18, 21, 24, 30, 36, 42, 48, 54, 60, 66, 72, 75, and 84 inches. There are 12 inches in a foot.

The input is a series of 1 to 25 lines terminated by end of file. Each line has the data for one test case in this order:

R M c S H

The numbers are separated by spaces. All values are greater than zero.

For each test case, print a line with the diameter in inches of the smallest sized standard culvert pipe that will carry the flow. Should the flow exactly match the maximum flow of a pipe, use the next highest size. Should the flow exceed what an 84 inch pipe can carry, print a line containing only the string "`non-standard`".

Sample Input

```
1 350 .25 20 1.18
5.8 350 .25 20.0 1.18
10 350.0 .25 20.0 1.18
.35   45.7 .75 5 2
.35 45.7  .75  5 .3
.01 1000 .2 2 .1
10 1000 .2 2 5
```

Problem 8
Culvert (continued)

Output for the Sample Input

```
42
84
non-standard
15
24
12
72
```

2021/2022 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Secret Sauce

Restaurant Secret is a popular restuarant in town. Its key to success is its Secret sauce—in almost every dish, the Secret sauce turns an ordinary dish into a best-seller.

Despite its name, the Secret sauce is just a combination of some ingredients (plus their amounts, which are always integers). For example,

`SecretSauce = garlic × 2 + sugar × 2 + salt × 4 + water × 3`

Today, the Secret sauce has run out. Fortunately, the restuarant has some ingredients (not necessarily all) and some other sauces in stock, each with unlimited supply, which can be used to compose Secret sauce. For simplicity, the amount of each ingredient or sauce are integers. The recipes of all sauces are known and given as a combination of ingredients or other sauces along with their integer amounts. For example,

```
GarlicSauce = garlic × 1 + salt × 2
SweetSauce = sugar × 2 + water × 3
SweetGarlicSauce = SweetSauce × 1 + GarlicSauce × 1
```

Suppose that all three sauces (Garlic sauce, Sweet sauce and Sweet Garlic sauce) are in stock. One unit of Secret sauce can be made from: (a) 2 units of Garlic sauce and 1 unit of Sweet sauce, or (b) 1 unit of Sweet Garlic sauce and 1 unit of Garlic sauce.

The manager of Restaurant Secret hired you to compute in how many ways one unit of Secret sauce can be made.

Input to your program is in the following format:

M
I_1
I_2
...
I_M
N
S_1 K_1 $C_{1,1}$ $U_{1,1}$ $C_{1,2}$ $U_{1,2}$ ... C_{1,K_1} U_{1,K_1}
S_2 K_2 $C_{2,1}$ $U_{2,1}$ $C_{2,2}$ $U_{2,2}$ ... C_{2,K_2} U_{2,K_2}
...
S_N K_N $C_{N,1}$ $U_{N,1}$ $C_{N,2}$ $U_{N,2}$ ... C_{N,K_N} U_{N,K_N}
L
C_1
C_2
...
C_L
K_T $C_{T,1}$ $U_{T,1}$ $C_{T,2}$ $U_{T,2}$ ... C_{T,K_T} U_{T,K_T}

The first line consists of an integer M ($1 \leq M \leq 7$), the number of all ingredients. The following M lines contain the names of ingredients, one per line. Each name is an alphabetical string starting with a lowercase letter, and the length is no more than 20 characters. All ingredient names are unique.

Problem 9
Secret Sauce (continued)

The following line consists of an integer N ($1 \leq N \leq 20$), the number of all known sauces. The following N lines describe the recipes of sauces. The i-th line starts with the name of the i-th sauce S_i (an alphabetical string starting with an uppercase letter, and the length is no more than 20 characters). The rest of the line describes the recipe to make 1 unit of this sauce: an integer K_i ($1 \leq K_i \leq M$), the number of different sauces or ingredients, followed by K_i pairs of strings and integers ($C_{i,j}$ and $U_{i,j}$), where $C_{i,j}$ is the name of a sauce or ingredient, and $U_{i,j}$ is the number of units ($1 \leq U_{i,j} \leq 4$). All sauce names are unique. All $C_{i,j}$ must be unique within this recipe and must be one of $I_1, I_2, \ldots, I_M, S_1, S_2, \ldots, S_N$. There are no cyclic references in the recipes. No sauce will ever aggregate more than 65536 units of any ingredient.

The following line consists of an integer L ($1 \leq L \leq M+N$), the number of different sauces or ingredients in stock. The following L lines contain $C_1, C_2, \ldots, C_L$, the names of sauces or ingredients in stock, one per line. All C_i must be one of $I_1, I_2, \ldots, I_M, S_1, S_2, \ldots, S_N$. Each sauce or ingredient in stock has unlimited supply.

The final line describes how to make 1 unit of the Secret sauce: an integer K_T ($1 \leq K_T \leq M$), the number of different ingredients, followed by K_T pairs of strings and integers ($C_{T,i}$ and $U_{T,i}$), where $C_{T,i}$ is the name of an ingredient, and $U_{T,i}$ is the number of units ($1 \leq U_{T,i} \leq 4$). All $C_{T,i}$ must be one of $I_1, I_2, \ldots, I_M$.

Your program is to print a line with the number of ways to make 1 unit of the Secret sauce. The number is guaranteed to be less than 2^{63}.

Sample Input

```
2
sugar
garlic
2
SauceA 1 sugar 1
SauceB 2 garlic 1 SauceA 1
3
garlic
SauceA
SauceB
2 garlic 2 sugar 3
```

Output for the Sample Input

```
3
```

Explanation for the Sample

There are three ways to make 1 unit of the Secret sauce: (1) 2 units of garlic and 3 units of SauceA; (2) 1 unit of garlic, 2 units of SauceA and 1 unit of SauceB; (3) 1 unit of SauceA and 2 units of SauceB.

2021/2022 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 10
Stringy

Your job is to reimplement an ancient string processing language. See the attached manual page.

The input consists of a stringy program followed by the input to that program.

The program consists of a series of lines of at most 80 characters terminated by a line starting with "/*". That program's input, if any, follows. Input is a series of lines of at most 80 characters terminated by the end of file.

The judged data is guaranteed to be valid, with no syntax or run-time errors.

Sample Input

```
  set count '0'
readLines input aLine /f finish
  add count '1'
  rpl aLine ',' ';' /s
  output aLine / readLines
finish set mess count
  append mess ' lines processed'
  output mess
/*
  foo = 1, 2, 3, 4, 5
  bar = a, b, c, d, e, f
  spam = 37
```

Output for the Sample Input

```
  foo = 1; 2; 3; 4; 5
  bar = a; b; c; d; e; f
  spam = 37
3 lines processed
```

STRINGY

Stringy is a string processing language. All variables and constants are strings.

A program is made up of command lines which consist of an optional label, an operation, an optional list of arguments, and an optional goto section. An operation is required. The individual tokens on a command line are separated by whitespace.

A line starting with '*' is a comment line.

Variable names and labels start with an alphabetic character and consist of an alphanumeric string of at most 30 characters. Case is significant. Variables and labels are in different name spaces but it is considered poor practice to use variables with the same name as a label. It is a compile time error to have lines with the same label. It is also a compile time error if a label is referenced but not defined.

All variables are set to the null string, '', at the start of program execution.

Constants are string literals enclosed in single or double quotes. Within a literal the enclosing quote is represented by doubling the quote. These literals represent the same strings: "don't enter" and 'don''t enter'.

A label is defined by starting in the first character of a command line. If the first character is a blank, the command has no label.

The goto section determines which line of the program is executed next:

```
  '/ foo' is an unconditional branch to the line with the label 'foo'
  '/s foo' goes to that line if the command was successful
  '/f foo' goes to that line if the command failed
```

It is a compile time error to mix a conditional goto with an unconditional one. Both types of conditional gotos can appear, but only one of each. There can be only one unconditional goto on a line. If there is no goto or no conditional goto is taken, execution continues with the next line.

As a shortcut, if no label appears after a conditional goto, control continues on the same command line. Thus

```
  rpl a ',' ';' /s
```

is the same as

```
foo rpl a ',' ';' /s foo
```

For numeric operations it is a run time error if any of the strings are not optionally signed decimal integers. The maximum value is implementation dependent but should be at least

```
  -10000 <= n <= 10000
```

Results of numeric operations will start with a '-' if needed, but not a '+'.

```
operations:

  add a b
    a is a variable, b is a variable or a literal
    result: the contents of a are set to the old contents of a plus the
      contents of b
    is always successful

  sub a b
    a is a variable, b is a variable or a literal
    result: the contents of a are set to the old contents of a minus the
      contents of b
    is always successful

  eq a b
    a and b can be variables or literals
    successful if the contents of a and b are numerically equal
    otherwise fails

  lt a b
    a and b can be variables or literals
    successful if the contents of a are numerically less than the contents of b
    otherwise fails

  gt a b
    a and be can be variables or literals
    successful if the contents of a are numerically greater than the contents of b
    otherwise fails

  isnum a
    a can be a variable or a literal
    successful if the content of a is only a number
    otherwise fails

  set a b
    a is a variable, b is a variable or literal
    result: the contents of a are set to the contents of b
    always successful

  len a b
    a is a variable, b is a variable or literal
    result: the content of a is set to the length of the content of b
    is always successful

  rpl a b c
    a is a variable, b and c are variables or literals
    result: if the content of b is a substring of the contents of a
      its first occurrence in a is replaced by the contents of c
    successful if the replacement happens otherwise fails
    always fails if b is the null string

  seq a b
    a and b can be variables or literals
    successful if the contents of a are equal to the contents of b
    otherwise fails

  split a b c d
    a, c, and d are variables, b can be a variable or literal
    result: if the content of b is a substring of a, c is set to the
      content of a before the first occurrence of that substring and
      d to the content after.
    successful it the string is split else fails and the content of c and d
      are not changed
    always fails if b is the null string
    c or d may be the null string if the match is at the beginning or end
      of a

  append a b
    a is a variable and b can be a variable or literal
    the content of a becomes the content of the old a concatenated with
      the contents of b
    is always successful

  prepend a b
    a is a variable and b can be a variable or literal
    the content of a becomes the content of b concatenated with
      the old content of a
    is always successful

  input a
    a is a variable
    set the content of a to the next line of standard input
    does not include any line terminating character
    fails if beyond the end of the input file otherwise successful

  output a
    a can be a variable or literal
    output the content of a to standard output followed by a new line
    is always successful

  exit
    terminate the program

notes
  There is an implied exit at the end of every program

  Compile time and run time error messages are implementation dependent.
```