

SRU Area High School Programming Contest
February 21, 2006
Official Problem Set

General Directions:

- Assume that all data is valid.
- No data line will be longer than 80 characters.
- You must program for all valid special cases.
- Do not open any files for reading or writing.
Read from the keyboard (*stdin*) and write to the monitor (*stdout*).
RockTest will automatically redirect your data file so that your program reads it instead of the keyboard.
- Only output counts, not the solution used.
- Real number output: Unless the problem specifically says otherwise, the exact number of decimal places to be displayed is not important. However, your answers should show at least one decimal place unless the answer happens to be an integer value.
- Comma separate values. Examples of the test data with comma separated values is shown in a text box adjacent to the standard data input.

1. Prime number

A prime number is a positive integer that has only two factors: 1 and itself. For example, 13 is a prime number. However, 10 is not a prime number because its factors include 1, 2, 5, and 10. Given a positive integer, determine if it is prime number or not. If it is a prime number, display "... is a prime number"; otherwise, display "... is not a prime number". The list ends with 0.

Test data:

```
11  
15  
19  
0
```

Output from test data:

```
11 is a prime number  
15 is not a prime number  
19 is a prime number
```

2. Duplicate characters

Given a string, remove all duplicate, adjacent characters from the string:

Test data:

aaabbcccd effe

Output from test data:

abcdefe

3. Reverse order

Given a list of non-zero integers, display the digits in reverse order for each number (but omit the leading zeros). The list ends with 0.

Test data:

18352
10
-23
0

Output from test data:

25381
1
-32

4. Columnar transposition cipher

In simple columnar transposition cipher, the plaintext is written horizontally onto a piece of graph paper with fixed width. The ciphertext is then read vertically.

For example, the sentence of “THE WEATHER IS SO NICE THAT WE WANT TO PLAY” is written as the following using a width of six.

T	H	E	W	E	A
T	H	E	R	I	S
S	O	N	I	C	E
T	H	A	T	W	E
W	A	N	T	T	O
P	L	A	Y		

Resulting in the output “TTSTWPHHOHALEENANAWRITTYEICWTASEEO”

Given a string and a positive integer width, output the ciphertext.

Test data:

THEWEATHERISSONICETHATWEWANTTOPLAY 6

Output from test data:

TTSTWPHHOHALEENANAWRITTYEICWTASEEO

Test data 2:

THEWEATHERISSONICETHATWEWANTTOPLAY 1

Output from test data 2:

THEWEATHERISSONICETHATWEWANTTOPLAY

5. Appointment Schedule

A very busy executive has a set of seven back-to-back meetings. To help her arrange her schedule you are going read the starting time in standard format (12:34 AM or 1:02 PM) followed by the duration of the meetings in hh:mm format. You will then print a schedule of all of the meetings.

Rules for output (and input):

- Times should print in the standard format (as shown), not 12:0 PM.
- AM or PM is always capitalized.
- 12:00 AM is midnight, 12:00 PM is noon.

Test data:

12:30 AM 4:30

Output from test data:

Meeting starts at 12:30 AM

Meeting ends at 5:00 AM

Meeting starts at 5:00 AM

Meeting ends at 9:30 AM

Meeting starts at 9:30 AM

Meeting ends at 2:00 PM

Meeting starts at 2:00 PM

Meeting ends at 6:30 PM

Meeting starts at 6:30 PM

Meeting ends at 11:00 PM

Meeting starts at 11:00 PM

Meeting ends at 3:30 AM

Meeting starts at 3:30 AM

Meeting ends at 8:00 AM

6. SuDoKu Solution Checker

A currently popular game called SuDoKu requires the player to fill in the digits 1 through 9 in a 9 by 9 square following the simple rules that each row, each column and each of the nine 3 by 3 sub-squares (indicated with shading) contain each of the digits one through nine exactly once. Two examples are:

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Write a program that reads nine lines, each of which contains nine numbers, and determines if the 81 numbers represent a solution to a SuDoKu puzzle.

If the numbers do not represent a solution, print the message **Not a solution**

If the numbers do represent a solution, print the message **This is a solution**

Test Data 1:	Test Data 2:	Test Data 3:
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 99	9 2 3 4 5 6 7 8 1
4 5 6 7 8 9 1 2 3	4 5 6 7 8 9 1 2 3	4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6	7 8 9 1 2 3 4 5 6	7 8 9 1 2 3 4 5 6
2 3 4 5 6 7 8 9 1	2 3 4 5 6 7 8 9 1	2 3 4 5 6 7 8 9 1
5 6 7 8 9 1 2 3 4	5 6 7 8 9 1 2 3 4	5 6 7 8 9 1 2 3 4
8 9 1 2 3 4 5 6 7	8 9 1 2 3 4 5 6 7	8 9 1 2 3 4 5 6 7
3 4 5 6 7 8 9 1 2	3 4 5 6 7 8 9 1 2	3 4 5 6 7 8 9 1 2
6 7 8 9 1 2 3 4 5	6 7 8 9 1 2 3 4 5	6 7 8 9 1 2 3 4 5
9 1 2 3 4 5 6 7 8	9 1 2 3 4 5 6 7 0	9 1 2 3 4 5 6 7 8
Output from test data 1: This is a solution	Output from test data 2: Not a solution	Output from test data 3: Not a solution

Each line will contain exactly nine integer numbers. The numbers are supposed to be single digit integers, but you will note that **Example 2** contains two bad values, either of which is sufficient to force the “Not a solution” determination.

Test data for CSV files will be separated by commas:

1, 2, 3, 4, 5, 6, 7, 8, 9

4, 5, 6, 7, 8, 9, 1, 2, 3 *et cetera*

7. Dinner Cost

A short order (fast food) restaurant offers the following selections at the price indicated:

Cost	Item	Cost	Item	Cost	Item	Cost	Item
\$0.85	fries	\$1.50	burger	\$1.00	soda	\$1.75	wings
\$0.89	salad	\$1.11	shake	\$1.65	chicken	\$1.70	fish
\$2.00	pasta	\$0.99	coffee	\$0.85	tea	\$2.25	dessert

There is one dinner special, \$2.75 for a burger, fries and soda combination.

Read a series of one line transactions. Each transaction consists of a list of numbers and items followed by a zero. An item is always preceded by a single positive integer number that represents the quantity of that item to be purchased. Items are in no particular order and items may be repeated on a line.

The names of the items must match the names given above. Items not on the list will result in an error message that specifies the (bad) item name – see below. For simplicity, **Fries** (capitalized) or **burgers** (pluralized) would not be considered valid items while **fries** and **burger** are.

There are an unknown number of lines. For each line, simply print the total cost (without tax). Whenever possible the price of a special should be used rather than the individual prices for the burger/fries/soda combination.

Test data:

1 burger 0

1 Fries 0
1 fries 2 pasta 3 soda 1 fries 2 burger 0
1 fries 1 burger 3 fish 0
3 tea 1 fries 2 chips 1 burger 2 dessert 1 soda 1 burger 0

Output from test data:

\$1.50
We don't sell Fries
\$0.00
\$10.50
\$7.45
We don't sell chips
\$11.30

8. Frequency Count

Write a program to compute the frequency of characters in each input line. An input line of text may contain both upper and lower case letters. Your frequency counting program is case-insensitive.

Input Format:

Each line of input text may contain up to 80 letters. The end of input lines is denoted by a line containing only the # character.

Output Format:

Copy the input line to the output, then display or print one line for each letter that appears in the previous line of input, showing the letter and its frequency count, separated by one or more spaces.

Test data 1:

```
ABBA
xyzzY
#
```

Output from test data 1:

```
ABBA
a  2
b  2
xyzzY
x  1
y  2
z  2
```

Test data 2:

```
Write a program to compute the frequency of alphabets
SRU Programming Contest
#
```

Output from test data 2:

```
Write a program to compute the frequency of alphabets
a  4
b  1
c  2
```

e 6
f 2
g 1
h 2
i 1
l 1
m 2
n 1
o 4
p 3
q 1
r 4
s 1
t 5
u 2
w 1
y 1

SRU Programming Contest

a 1
c 1
e 1
g 2
i 1
m 2
n 2
o 2
p 1
r 3
s 2
t 2
u 1

10. Word Match

You have been given the task to monitor all computer email traffic, searching for keywords that suggest terrorist activities. However, some terrorist elements in society have started scrambling the letters in some words of their email messages in an attempt to evade detection.

You, undeterred by such transparent tricks, decide to write a program that searches sentences for suspicious words whose letters have been rearranged.

Input:

The input is a sequence of pairs, where each pair consists of a list followed by a sentence. The list gives the words to search for in the following sentence. All letters in the input are in lower case. Words are at most 20 letters long and are separated by a single space. Each line is at most 80 characters long. Each list fits on a single line and contains at most 20 words. Sentences may occupy more than one line, and are terminated by a full stop (period).

The end of input is denoted by a list containing only a #.

Output:

For each pair of list and sentence, find the words from the list that appear in the sentence with their letters shuffled. Print these words in the order they appear in the list, separated by a single space. If no words from the list match the words in the sentence, just print a blank line.

Test data:

```
bomb
we twan ot mobb mericaa.
guns mines missiles
aameric ssell snug
dan iimsssle ot sit neeemis.
cat sat mat
cat sat tam mat.
#
```

Output from test data:

```
bomb
guns missiles
cat sat mat
```

11. String Processor

Write a simple string processor. This string processor needs to read in a single line of text followed by a line of commands that will modify the string. Your program should output the line of processed text.

The commands are as follows:

0	(zero) Move cursor to the start of the line
\$	(currency symbol) Move cursor to the position after the last character (end of line).
x	Delete the character at the cursor position if not at end of line.
s	Swap the character at the cursor with that to the right of it, as long as the cursor is not at or just before the end of the line.
i x	Insert the character 'x' at the cursor position, and move the cursor along one space.
u	Make the character go into upper case if a letter, and move the cursor along one space.
+	Move the cursor right one space.
-	Move the cursor left one space.

The cursor can be positioned at any character, from the first character, all the way through one position after the last character. The input commands are in a single line following the input text line, not separated by spaces. The cursor starts at the first character in the line for each text input line. The input is terminated by an input text line that contains only a single character, '#'.

Test data:

```
Hello, I am a frog.  
$-----xxxxipieirisioin  
Needle nardle noo.  
+++xizuu+++xxips  
#
```

Output from test data:

```
Hello, I am a person.  
NeezLE napel noo.
```

12. Symbolic Artificial Intelligence

Write a simple symbolic AI processor, which will be told some facts, and asked some questions. The AI processor must then answer the questions.

Fortunately, these facts and questions relate only to **specific individuals**, who are in **groups** (collective nouns) which perform **actions** (verbs) on other groups.

The grammar used is very simple: there are two types of statements and one type of question, and all nouns are made plural through the expedient of adding an s.

Input

Each line of the input contains either a simple sentence or a blank. The end of the input is indicated by a line whose first character is a #. Each sentence has one of the following forms:

- A statement of the form "*a*s *v* *bs*.", where *a* and *b* are nouns and *v* is a verb. This means that everything of type *a* does *v* to everything of type *b*.
- A statement of the form "*a* is a *b*.", where *a* and *b* are nouns. This means that the particular individual *a* is of type *b*.
- A question of the form "Does *a* *v* *b*?", where *a* and *b* are nouns and *v* is a verb. This asks whether the individual *a* does the action *v* to the individual *b*.

Lines are up to 80 characters long. Words are up to 40 characters long and may consist of upper and lower case letters, although all searching is done case-insensitively. There may be more than one space between words, but there are never any spaces between words and the punctuation marks at the end of each sentence. Allow for up to 100 facts in the input.

Output:

For each question in the input, print a single line, which contains "Yes ." if the preceding input implies that the question is true, and "No ." otherwise.

Test data:

Dogs chase cats.
Sheeps ignore glasss.

Fido is a dog.
Fluffles is a cat.
Does Fido chase Fluffles?
Does Fluffles chase Fido?
#

Output from test data:

Yes.

No.