TCEA
HIGH SCHOOL
PROGRAMMING CONTEST

AREA PROBLEM SET

FEBRUARY 2005

Texas Computer Education Association
2005 High School Programming Contest
Area Problem Set

Problem 2.1 When You're Hot, You're HOT

General Statement: Temperatures may be given using Celsius or Fahrenheit. Your task is to write a program that will input a temperature using one of those scales and convert it to the other. The formula for converting a Fahrenheit temperature to Celsius is $C = \frac{5}{9}(F - 32)$. The formula for converting a Celsius temperature to Fahrenheit is $F = \frac{9}{5}C + 32$.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a temperature represented as an integer. Each integer is followed by a single space and a single character indicating if it is a Celsius temperature or a Fahrenheit temperature. A `C` represents a Celsius temperature, and an `F` represents a Fahrenheit temperature.

Name of Data File: pr21.dat

Output: Your program should produce *n* lines of output (one for each line of input text). Each output line should contain a floating-point value and a character that represents the converted temperature. The floating-point number should be displayed to two decimal places, even if it represents a whole number. You must also display at least one digit before the decimal point. In addition, there should be exactly one space between each floating-point number and its corresponding character.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Irrespective of scale, each input temperature will be between –100 and 500, inclusive.

Sample Input:

```
3
100 C
212 F
0 C
```

Sample Output:

```
212.00 F
100.00 C
32.00 F
```

Problem 2.2 Beat the Clock

General Statement: A programming contest (not unlike this one) starts at 9:30 a.m. The contest organizers need a program that can read a submission time and determine the number of minutes between that time and the beginning of the contest. Your task for this problem is to write that program.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a time of day represented as a string in the form of `hh:mm xx`, where hh is the hours portion of the time, mm is the minutes portion of the time, and xx is either `am` or `pm` depending on whether it is the morning or afternoon.

Name of Data File: pr22.dat

Output: Your program should produce *n* lines of output (one for each line of input text). Each output line should be an integer representing the number of minutes between 9:30 am and the time given in the corresponding input data collection.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The hours and minutes portion of the input times will always be given using two digits.
Each input time will be between 09:31 am and 11:59 pm, inclusive.
12:00 noon will not be used as an input time.

Sample Input:

```
3
09:35 am
10:09 am
02:00 pm
```

Sample Output:

```
5
39
270
```

Problem 2.3 I'd Like to Count a Vowel, Please

General Statement: A word consists of a collection of letters. Some letters are consonants, and some are vowels. The vowels are the letters a, e, i, o, and u. Your task is to count the number of vowels in an input word.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a single word.

Name of Data File: pr23.dat

Output: Your program should produce *n* lines of output (one for each line of input text). Each output line should be an integer representing the number of vowels in the corresponding input word.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each input word is composed of only lowercase letters.
The length of each input word is between 1 and 80 letters, inclusive.
The only vowels are a, e, i, o, and u. For the purpose of this problem, the letter y is not a vowel.

Sample Input:

```
3
hello
programming
xyzzy
```

Sample Output:

```
2
3
0
```

Problem 2.4 Hashing It Out

General Statement: As an alternative to using an index, system designers may choose to use hashing functions to organize their data for access. A hash function maps data values to memory locations. The idea is that when given the value of a search key, *K*, for a data record, the system can apply the hashing function, *h*, on that value to determine where its corresponding data record is stored. Thus, the system can search for the data record with key value *K* by simply computing *h(K)*.

A common hash function evenly splits the digits in a data value in half and adds the two halves together. The even splitting is performed based on the number of digits in the data value. For example, given a data value 4127, the hash function would first split it into 41 and 27 and then add those values together to obtain a final value of 68. Thus, key value 4127 would get mapped to location 68. Your task is to write a program that computes the memory location of a data value using the above hashing function.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of a data value with an even number of digits.

Name of Data File: pr24.dat

Output: Your program should produce *n* lines of output (one for each input data collection). The output for each data collection should simply contain the value produced by applying the above hashing function.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each input data value will be between 10 and 99999999, inclusive.

Sample Input:

```
3
4005
12345678
20
```

Sample Output:

```
45
6912
2
```

Problem 2.5 Analyzing Anagrams

General Statement: An anagram of a word can be created by rearranging the letters of that word. Each letter must be used in the rearrangement, and every letter can only be used once. Your task is to write a program that can determine if two different words are anagrams of each other.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of two words *A* and *B*. The words are separated by a single space.

Name of Data File: pr25.dat

Output: Your program should produce *n* lines of output (one for each input data collection). If the input words are anagrams, your program should simply print `yes`. If they are not anagrams, your program should simply print `no`.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each word contains only lowercase characters.
The length of each word is between 1 and 35 characters, inclusive.

Sample Input:

```
3
car race
earth heart
xyzzy xyzxy
```

Sample Output:

```
no
yes
no
```

Problem 2.6 Nothing But Net

General Statement: When evaluating how a basketball player performed in a game, people often look at the number of points the player scored. As an alternative, others consider the number of points the player scored per shot taken. So, instead of considering that player A scored 40 points, they also consider that player A took 40 shots as well. Thus, player A averaged 1 point per shot attempt.

Your task is to write a program that computes the points per shot attempt for a player based on that player's statistics for the game. Each made basket may be worth 1, 2, or 3 points.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection represents the statistics for a given player and is of the form $A\ x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ with each component separated by a single space. Component *A* represents the player's name and is a string of characters. Each x_j represents the number of made baskets of point value *j*. Each y_j represents the total number of shots attempted by the player to score a basket of point value *j*.

Name of Data File: pr26.dat

Output: Your program should produce *n* lines of output (one for each input data collection). The output for each data collection should contain the corresponding input player's name followed by a single space and a floating-point value representing that player's points per shot attempt. All of the printed floating-point values generated by your program should be rounded to two digits after the decimal point. You must also display at least one digit before the decimal point.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Players' names will consist of only uppercase letters.
There will be exactly five letters in each player's name.

Sample Input:

```
3
SMITH 1 1 2 2 3 3
BROWN 0 2 4 6 1 1
JONES 4 5 4 5 4 5
```

Sample Output:

```
SMITH 2.33
BROWN 1.22
JONES 1.60
```

Problem 5.1 Password Possibilities

General Statement: After unsuccessfully trying to login to your account, you begin to consider how many different possible passwords there could be. You know that your password is 8 characters long, and it can contain only lowercase letters, uppercase letters, and digits. Since there are 26 possible lowercase letters, 26 uppercase letters, and 10 digits, you know that each symbol in your password could be 1 of 62 possible characters (26+26+10 = 62). Thus, there are 62^8 possible 8-character passwords.

To understand the magnitude of the above number, you estimate it as a power of 2. Since the *closest power of 2* to 62 is 64, you know 62^8 is approximately 64^8. Since 64 equals 2^6, you compute that 64^8 equals $(2^6)^8$, which is 2^{48}. Thus, 2^{48} is an approximation of 62^8.

Instead of approximating 62^8, your task is to write a program that can approximate any x^y.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of two integers *x* and *y*. Your program should estimate x^y as 2^k by finding ***the closest power of 2 to x.***

Name of Data File: pr51.dat

Output: Your program should produce *n* lines of output (one for each input data collection). For each data collection, your program will compute its estimation as 2^k. The output for each data collection should be the value of *k* (the exponent).

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of each x is between 1 and 1,000,000, inclusive.
The value of each y is between 0 and 1,000,000, inclusive.
If x is equally close to two powers of 2, use the larger of the powers.

Sample Input:

```
3
62 8
26 1000
100 0
```

Sample Output:

```
48
5000
0
```

Problem 5.2 Backspace to the Future

General Statement: While eating and drinking near your computer, you spilled a glass of water on your keyboard. After quickly cleaning it up, you noticed that the backspace key no longer works. Instead of erasing the previous letter typed, the backspace key generates a ~ symbol each time it is struck. This is a problem because you need your computer to write a paper for tomorrow.

You realize that you will never use the ~ symbol in your paper. So, every occurrence of the ~ symbol in your paper will represent one time that you struck the backspace key. You realize that you could simply write a program that will read the text generated with your defective keyboard and convert it to the paper that you intended to write. Your task for this problem is to write that program.

Input: The first line of the input is an integer *n* that represents the number of lines of text in the paper. Each line will contain a maximum of 80 characters including the ~ symbols.

Name of Data File: pr52.dat

Output: Your program should produce *n* lines of output (one for each line of input text). Each output line should contain the text of its corresponding input line where each ~ symbol is treated as a backspace key.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: There is a character to erase each time the backspace key is used.

Sample Input:

```
3
Te~ricks~   ~~and~~~or Tree~at?~!
TCEA State~~~~~Area Con~~~Programing~~~ming Contests~
xyz~~~xx~yzyz~~zz~yy~ ~xyz~~~
```

Sample Output:

```
Trick or Treat!
TCEA Area Programming Contest
xyzzy
```

Problem 5.3 Colliding Circles

General Statement: While writing a flying saucer video game, it didn't take long for you to realize that a significant portion of this program would be devoted to detecting when objects collide in a scene. To simplify the problem, you represent all of the flying saucers in your game as two-dimensional circles. This means that to detect if a scene has a collision, you simply need to determine if any of the circles intersect.

Each circle in the game is represented as a combination of a two-dimensional center position *(x, y)* and a radius *r*. Given a collection of these circles as input, your program needs to identify if any pair of those circles intersect and print an appropriate message. Your task is to write a program that will perform those functions.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection represents a set of input circles. Each circle in the set is represented by three integers *x*, *y*, and *r* separated by spaces. The center of the circle is the point *(x, y)*, and the radius of the circle is *r*.

Name of Data File: pr53.dat

Output: Your program should produce *n* lines of output (one for each input data collection). If no two circles intersect within an input collection, the corresponding output line should contain the message `VALID SCENE`. Alternatively, if any two circles intersect in a collection, the corresponding output line should contain the message `INVALID SCENE`.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: All of the values *x*, *y*, and *r* are between 0 and 20, inclusive.
The number of circles in each collection is between 2 and 8, inclusive.
Circles intersect if their borders share at least one common point.

Sample Input:

```
3
10 10 5 1 1 5
0 1 1 4 5 1 9 10 1 14 15 1 18 19 1
0 0 5 20 20 5 7 7 5
```

Sample Output:

```
VALID SCENE
VALID SCENE
INVALID SCENE
```

Problem 5.4 Digital Dates

General Statement: When users enter dates into a computer program, they will enter them in various formats. Even if we instruct the users to enter the date in a month day year format, they may still enter it using 1 or 2 digits for the month, 1 or 2 digits for the day of the month, and 2 or 4 digits for the year. They may also use spaces, slashes, or dashes to separate the month from the day and the day from the year.

Your task is to write a program that will interpret the date that is entered by the user and print the date in yyyymmdd format. The input date may be given using any of the options described earlier.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of a date in month day year format.

Name of Data File: pr54.dat

Output: Your program should produce *n* lines of output (one for each input data collection). The output for each data collection should contain the input date in yyyymmdd format where yyyy means a 4-digit year, mm means a 2-digit month, and dd means a 2-digit day of the month.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: All dates given as input will be valid dates.
All entered dates will be between Jan 01, 1960 and Dec 31, 2059, inclusive.

Sample Input:

```
3
12/3/1975
1-15-41
7 8 19
```

Sample Output:

```
19751203
20410115
20190708
```

Problem 5.5 Breaking Points (page 1 of 2)

General Statement: You notice that your text editor always breaks lines of text so that the resulting text is as close to 80 characters long as possible without exceeding that limit (except for the last line). When a line exceeds this limit, the editor starts at the 80th character in the line and searches to the *left*. When it finds a space, it uses it to break the line. The portion of the line after that space is then concatenated to the beginning of the next line (with another space in between).

Your task is to write a program that uses the above procedure to display text using any arbitrary maximum length for a line.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow. Each data collection consists of two lines of input. The first line in a collection contains a single integer *k* which represents the maximum length of any line that can be in the output. The second line of the collection contains the input that should be broken into lines as close to length of *k* characters as possible. The breaks should be performed as described in the general statement.

Name of Data File: pr55.dat

Output: Your program should produce *n* sets of output data, one for each data collection. There should be one blank line between each set of output data. The only difference between the input text and the output text should be the line breaks. In other words, do not change the case of the input text.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value for each *k* will be between 10 and 80, inclusive.
A “word” in the input text is considered to be any continuous set of characters without spaces. Do not break lines in the middle of a word.
No word in the input text will be longer than *k* characters.
There will be only one space between each pair of words.

Sample Input:

```
3
20
Here is some sample text. Here is some more text.
10
This is another line of input.
50
This is the last sample input case.
```

Problem 5.5 Breaking Points (page 2 of 2)

Sample Output:

```
Here is some sample
text. Here is some
more text.

This is
another
line of
input.

This is the last sample input case.
```

Problem 5.6 Tug of War

General Statement:	You are throwing a party and want to organize a tug of war game. So, you need to create two teams out of your party guests. In order to make the game as fair as possible, both teams should have the same number of people. In addition, the total weight of the people on each team should be as close as possible. While you can ensure that each team has the same number of people, it is much harder selecting the team members so that the total team weights are as nearly equal as possible. Your task is to write a program that will define these teams.
Input:	The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of a set of *2k* integers separated by spaces where each integer represents the weight of one of the party guests. Thus, there should be *k* members on each team
Name of Data File:	pr56.dat
Output:	Your program should produce *n* lines of output (one for each input data collection). The output for each data collection should be the total weight for each team separated by a single space. The value for the team with the smaller total weight should be listed first. The output is to be formatted exactly like that for the sample output given below.
Assumptions:	The number of integers in each data collection will be even (*2k*). The value for each *k* will be between 1 and 10, inclusive. The value of each weight is between 100 and 300, inclusive.

Sample Input:

```
3
300 150 200 300
100 150 200 250 300 300
300 100
```

Sample Output:

```
450 500
650 650
100 300
```

Problem 9.1 When in Rome …

General Statement: Roman numerals are numbers expressed as sequences of characters. Each character stands for a particular value, and the total value of a Roman numeral is computed by adding the values of its characters. Reading a Roman numeral from left to right, the characters should be sorted in decreasing order in terms of their numeric values. If a character with a smaller value appears to the left of a character with a larger value, the value of the smaller character is subtracted from the total of the Roman numeral *instead* of added.

Your task is to write a program that reads in a set of Roman numerals and prints them out in sorted order. The order must be increasing in terms of the value of the Roman numerals.

The value of each character in a Roman numeral is as follows:

M = 1000	C = 100	X = 10	I = 1
D = 500	L = 50	V = 5	

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of a set of Roman numerals where each Roman numeral is separated by a single space.

Name of Data File: pr91.dat

Output: Your program should produce *n* lines of output (one for each input data collection). The output for each data collection should be the Roman numerals listed in sorted order. There should be one space between each Roman numeral.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each Roman numeral will be valid.
All characters will be given in uppercase.

Sample Input:

```
3
MCM XCV CIII
IV IX VI VIII VII V X XI
C D I L M V X
```

Sample Output:

```
XCV CIII MCM
IV V VI VII VIII IX X XI
I V X L C D M
```

Problem 9.2 Necessary Nesting

General Statement: A key task in compiling programs is identifying expressions with an incorrect syntax. Part of that ability is determining if symbols of grouping in an expression are properly nested. Every opening symbol must have a matching closing symbol (and vice-versa). In addition, a matching pair of grouping symbols may appear inside another pair of symbols, but they may not be interleaved.

Your task is to write a program that determines if an expression is properly nested. The matching pairs of opening and closing grouping symbols are as follows:

Opening Symbol	Matching Closing Symbol
`{`	`}`
`(`	`)`
`<`	`>`
`[`	`]`

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of a string of characters without spaces. There may be other characters besides the symbols of grouping.

Name of Data File: pr92.dat

Output: Your program should produce *n* lines of output (one for each input data collection). The output for each data collection should be `EXPRESSION IS PROPERLY NESTED` if the bracket symbols are properly nested or `SYNTAX ERROR` if the brackets are not properly nested.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The maximum number of characters in a single data collection is between 1 and 80, inclusive.

Sample Input:

```
3
(abc(*def)
{<start(true)$x[4+(3-4)]=10>}
[{<>()]}
```

Sample Output:

```
SYNTAX ERROR
EXPRESSION IS PROPERLY NESTED
SYNTAX ERROR
```

Problem 9.3 Chain, Chain, Chain (page 1 of 2)

General Statement: A simple game to play is to take two words of the same length and try to form a chain of words between them. For two words to be adjacent in the chain, they must differ in exactly one letter. For example, consider forming a chain between two seven letter words booster and roasted. A possible chain for going from booster to roasted is as follows:

booster
rooster
roaster
roasted

Your task is to write a program that reads in a dictionary of possible words, *D*, to use in a chain. Your program will then read in pairs of words and print the shortest chain that links them using only the words in *D*.

Input: The first line of the input contains two integers *m* and *n* that represent the number of words in the dictionary and the number of test cases, respectively. The integers will be separated by a single space. The next *m* lines contain the dictionary words with each line containing a single word. The following *n* lines represent the test cases where each test case contains two of the words in the dictionary. The two words in each case will be separated by a single space.

Name of Data File: pr93.dat

Output: Your program should display *n* sets of output (one for each test case). Each output set should list the shortest chain for the corresponding input test case. The chain should begin with the first word in the test case and end with the last word in the test case. Each pair of adjacent words must differ in only one letter. Your program should print only one dictionary word per line, and it should print a blank line between each output set.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: All of the dictionary words will be the same length.
Each word only consists of lower case letters.
There will be one minimum solution for each input test case.

Problem 9.3 Chain, Chain, Chain (page 2 of 2)

Sample Input:

```
8 3
that
this
what
then
when
than
thin
thus
this that
what when
when then
```

Sample Output:

```
this
thin
than
that

what
that
than
then
when

when
then
```

Problem 9.4 Checking Numbers (page 1 of 2)

General Statement: When writing a check, the value is written in English. So, to write a check for $100.35, it is necessary to write `one hundred and 35/100`. The above example indicates that when a floating-point value is specified, the portion after the decimal point is expressed as a numeric fraction with a denominator of 100. In addition, the word `and` is used to separate it from the portion before the decimal.

If the value being written is a whole number, such as writing a check for $100, there is no value after the decimal point that needs to be written. Thus, the amount is written with the phrase `and no/100` appended to the amount (with a space in between). Thus, a check for $100 is written as `one hundred and no/100`.

Your task is to write a program that will translate an input check value into its English equivalent.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a single value that may be an integer or a floating-point value.

Name of Data File: pr94.dat

Output: Your program should produce *n* lines of output (one for each input data collection). The output for each data collection should be the value of the input collection expressed in English.

The following is an exhaustive list of the words your program may print to represent the values to the left of the decimal point.
```
one, two, three, four, five, six, seven, eight, nine,
ten, eleven, twelve, thirteen, fourteen, fifteen,
sixteen, seventeen, eighteen, nineteen, twenty,
thirty, forty, fifty, sixty, seventy, eighty, ninety,
hundred.
```

To represent the values to the right of the decimal point, your program must print `and xx/100` after the above English portion, where `xx` is the value after the decimal point. If the value is a whole number, your program must print `and no/100` after the English portion.

Problem 9.4 Checking Numbers (page 2 of 2)

The output for each data collection should be printed on one line, and each word should be printed using only lowercase letters. Your program should not print any dashes meaning that whenever a dash should be printed, your program should print a space.

The output is to be formatted exactly like that for the sample output given below.

Assumptions:

Each input value will be between 1 and 999.99, inclusive.
If the input value is a floating point value, there will be exactly two digits given after the decimal point. If the input value is a whole number, no decimal point will be given (see the sample input).
The numerator of any printed fraction should always have two digits.
Thus, 1.05 must be written as `one and 05/100`.

Sample Input:

```
3
18.45
672.09
153
```

Sample Output:

```
eighteen and 45/100
six hundred seventy two and 09/100
one hundred fifty three and no/100
```

Problem 9.5 Common Crossings (page 1 of 2)

General Statement: A word cross is formed by printing a pair of words so that they share a common letter. The words are printed so that they form a large X-like shape where each word can be read from top to bottom (see sample data). If multiple letters can be used, the common letter should be as close to the middle of the first word as possible. If two letters can be used that are an equal distance from the middle of the word, the leftmost letter should be used.

Your task is to write a program that reads in a pair of words and forms a word cross.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of two words separated by a single space.

Name of Data File: pr95.dat

Output: Your program should print n sets of output (one for each data collection) where each set is separated by a single blank line. Each set should display the output of the corresponding data collection as a word cross. The word that is listed first in each data collection should be read from the top left to the bottom right. The word that is listed second should be read from the top right to the bottom left.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each word will be composed of only lowercase letters.
No word will have a letter that appears twice.
There will be at least one common letter between each pair of words in a data collection.

Sample Input:

```
3
two one
one two
computer education
```

Problem 9.5 Common Crossings (page 2 of 2)

Sample Output:

```
t
 w
  o
 n
e

  t
 w
o
 n
  e

  c
   o
    m   e
     p d
      u
     c t
    a   e
   t     r
  i
 o
n
```

Problem 9.6 Digital Calculator (page 1 of 2)

General Statement: Your new calculator displays each digit using a grid of characters containing 5 rows and 3 columns. Each character consists of only spaces and # symbols. The calculator uses the following grid patterns to display each digit 0-9.

```
 #  ### ### # # ###
 #    #   # # # #
 #  ### ### ### ###
 #  #     #   #   #
 #  ### ###   # ###

### ### ### ### ###
#     # # # # # # #
###  #  ### ### # #
# # #   # #   # # #
### #   ###   # ###
```

Given two numbers in the above format, write a program that prints their sum in the same grid format.

Input: The input consists of two integer values *x* and *y* displayed in the above format. Each value is displayed on a grid of 5 rows by *C* columns, where *C* is the number of digits in the number multiplied by 4. (There is an extra column *on the right* representing a grid space between digits). Thus, the digit can be read in the *leftmost* 3 columns of each 5x4 grid of characters. The value for x is given in the first five lines of the input, and the value for y is given in lines 6-10.

Name of Data File: pr96.dat

Output: Your program should output the sum of *x* + *y* in the grid format using # symbols and spaces. There should be one blank column between each pair of characters.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of x will be between 0 and 99999, inclusive.
The value of y will be between 0 and 99999, inclusive.
There is one column of spaces after each digit in x and y (see sample input).
There are no blank lines between x and y.
All characters in the input grids will be valid digits.

Problem 9.6 Digital Calculator (page 2 of 2)

Sample Input:

```
### ### ###
  #   # # #
###  #  # #
#   #   # #
### #   ###
 #  ### ### # # # #
 #    # # # # # # #
 #  ### ### ### ###
 #    # # #   #   #
 #  ### ###   #   #
```

Sample Output:

```
 #  # #  #   #  # #
 #  # #  #   #  # #
 #  ###  #   #  ###
 #    #  #   #    #
 #    #  #   #    #
```