Taylor High School

Advanced

Problem 2.1 **Halloween Night**

General Statement: Halloween is here! Convey your excitement by outputting the date of this illustrious holiday in the special format displayed below.

Time Allocation: 1 second

Name of data file: There will be no data file.

Input: There will be no input.

Output: The output should be formatted exactly like the sample output below.

Assumptions: There is no input.

Sample Input: None

Sample Output:

```
HAL LOW EEN HAL L
O W E    E    N H
A L L    O  WEE N
H A L    L    O W
EEN HAL  L  OWE E
```

Problem 2.2 **Phantom of the Mall**

General Statement: Christiaan the Phantom needs to buy candy to hand out to trick-or-treaters on Halloween. However, after being dormant in his underground lair for so many centuries, he has forgotten simple addition and multiplication. He does know, though, that a large bag of candy costs $2.50, a medium sized bag costs $1.25, and a small bag costs $0.75. Help Christiaan figure out the total amount he must pay for the candy.

Time Allocation: 1 second

Name of data file: adv22.dat

Input: The first line of input will be an integer *n* indicating the number of lines to follow. Each line will have 3 integers, each separated by a single space, representing the number of large, medium, and small candy bags, respectively, that Christiaan buys.

Output: Output the total amount of money that Christiaan the Phantom spends, to the nearest penny. The output should be formatted exactly like the sample output below.

Assumptions: *n* will be an integer between 1 and 1,000,000 inclusive.
The number of bags of candy of each size will be between 0 and 100,000 inclusive.
All input will be valid.

Sample Input:

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

Sample Output:

```
$4.50
$5.75
$0.75
```

Problem 2.3 **Lost in Translation**

General Statement: Fin the Incorporeal received an e-mail from his friend, Beard the Bombastic about an upcoming Halloween party. However, Fin is having a hard time reading it, for he is a 19th century ghost of Ernest Rutherford and is not fluent in the language of emoticons, which Beard has used liberally throughout his e-mail. Help Fin translate the emoticons into plain English.

Time Allocation: 1 second

Name of data file: adv23.dat

Input: The first line of input will be an integer *n* representing the number of lines to follow. Each line of input will have a single emoticon, *e*.

Output: Translate the emoticon using the following definitions:

Emoticon	Definition
=D	I am happy
=P	I pwn you
=(	I am sad
<3	I like
=O	I am shocked
=3	I am smug

If *e* has a repetition of the second character (e.g. `=DDDDDD`), then for every repetition of a character *x* after the first appearance of *x*, an exclamation mark should be added to the end of the translation. Therefore `=DDDDDD` is equivalent to `I am happy!!!!!`

Assumptions: *n* will be an integer between 1~~0~~ and 10,000 inclusive.
The number of characters in *e* will be between 2 and 1,000,000 inclusive.
All input will be valid.

Sample Input:

```
2
=PPPPPP
<3
```

Sample Output:

```
I pwn you!!!!!
I like
```

.

Problem 2.4 **House of Mirrors**

General Statement: Grumpy the Ghoul and Wendy the Witch are at the Halloween carnival, playing in the House of Mirrors. Grumpy cannot find Wendy, so he decides to write a message to her and reflect it in the multitude of mirrors so that she might see it no matter where she is. However, because the image is reflected, Wendy can only read Grumpy's message if the word is palindromic, (the same backwards and forwards), spaces removed, since she can read letters reflected in a mirror but not sentences. Help Grumpy determine whether or not Wendy can read his message.

Time Allocation: 1 second

Name of data file: adv24.dat

Input: The first line of input contains an integer *n* that represents the number of lines to follow. Each of the next *n* lines consists of a sentence *c* that Grumpy is considering.

Output: The output should consist of a single word `yes` or `no`, indicating whether or not the sentence *c* becomes a palindromic word after removing all spaces and converting every letter to lowercase.

Assumptions: *n* will be an integer between 1~~0~~ and 1,000,000 inclusive.
c will contain only uppercase and lower case letters and spaces.
c will begin and end with a letter.
c will have between 1 and 1,000,000 characters inclusive.
All input will be valid.

Sample Input:

```
2
Sup not on a pus
Must sell at tallest sum
```

Sample Output:

```
no
yes
```

Problem 2.5 **The Invited**

General Statement: George the Gnome is a very lazy typist. While typing out his invitations for his Halloween party, he was too lazy to capitalize his letters and add correct punctuation. His mother, annoyed at the grammatical atrocities George has committed, wants to correct his mistakes. Help his mother write a program to correct the capitalization and punctuation mistakes in George's sentences.

Time Allocation: 1 second

Name of data file: adv25.dat

Input: The first line of input contains an integer *n* that represents the number of lines to follow. The next *n* lines each have a single sentence *s*. *s* may contain any number of words.

Output: The output should consist of the sentence *s* with the correct punctuation and capitalization, according to the rules below:

1) The first letter of the first word of each sentence should be capitalized.
2) If adjacent words are separated by two spaces, a comma must be inserted and the number of spaces between the two words reduced to one. (Therefore `Hi  Bob` becomes `Hi, Bob`).
3) All sentences must end in a period and only a period.
4) The word "`I`" should always be capitalized.

Assumptions: *n* will be an integer between 1~~0~~ and 1,000,000 inclusive.
s will contain only words and spaces.
s may or may not contain uppercase letters.
s will have between 1 and 1,000,000 characters inclusive.
There will be no more than two spaces between adjacent words.
All input will be valid.

Sample Input:

```
2
you  mister vampire are invited to my party
do not forget i love sugar  so bring candy
```

Sample Output:

```
You, mister vampire are invited to my party.
Do not forget I love sugar, so bring candy.
```

Problem 2.6 **Sharing is Caring**

General Statement: Bob is very excited to go trick-or-treating. Unfortunately, his bully of a brother, Hubert, insists that he share his candy with him once Bob has finished, since Hubert is too lazy to go trick-or-treating himself. Unwilling to do so, Bob strikes a deal: he will only share his candy with Hubert if the number of pieces he obtains is a power of 2. Help Bob figure out whether the number of pieces of candy he has is an integral power of 2.

Time Allocation: 1 second

Name of data file: adv26.dat

Input: The first line of input contains an integer *n* that represents the number of lines to follow. The next *n* lines will have a single integer *p*, signifying the number of pieces of candy Bob has.

Output: The output should consist of a single lowercase word, either `yes` or `no`, indicating whether *p* is an integral power of two. The output is to be formatted exactly like the sample output given below.

Assumptions: *p* will be a positive base ten number between 1 and 1,000,000 inclusive.
n will be an integer between 1~~0~~ and 1,000,000 inclusive.
An integral power of two is defined as any integer *y* of the form 2□ , where *x* is an integer greater than or equal to 0.
All input will be valid.

Sample Input:
```
2
2
10
```

Sample Output:
```
yes
no
```

Problem 5.1 **Paranoia 0.0**

General Statement: Nefis the Zombie is a very paranoid zombie. When decorating her windows for Halloween, she insists that all designs must be vertically symmetrical. However, her eyesight is not very good, due to a mishap in one of her previous experiments. Help Nefis determine if her decoration is symmetrical or not.

Time Allocation: 1 second

Name of data file: adv51.dat

Input: The first line of input contains an integer *n* that represents the number of data collections to follow. The first line of each data collection consists of two integers *l* and *w*, indicating the length and width of Nefis' rectangular window. The next *l* lines will contain the design on the witch's window, which consist only of capital letters.

Output: The output should consist of a single word `yes` or `no`, indicating whether or not the design on Nefis' window is symmetrical. A design is considered symmetrical if all the characters on its vertical axis (the column that is in the exact middle of the design) are the same and the design can be reproduced by reflecting the characters on the left or right side of the axis to the opposite side. Therefore,

```
BCXCB
ACXCA
BCXCB
```

is symmetrical, while

```
 QOQ
TOP
POP
```

is not. Output should be formatted exactly like the sample output below.

Assumptions: *n* will be an integer between 1~~0~~ and 1,000,000 inclusive.
l will be an integer between 1 and 1000 inclusive.
w will be an odd integer between 1 and 999 inclusive.
The design will consist of only capital letters.
All input will be valid.

Sample Input:

```
2
3 3
AJA
IJO
FOF
4 5
ARLRA
WOLOW
RKLKR
CJLJC
```

Sample Output:

```
no
yes
```

Problem 5.2 **Final Destination**

General Statement: Nory the Ninja is participating in a Halloween treasure hunt around her neighborhood, and since the prize is an inordinate amount of candy, she is extremely motivated to win. The treasure hunt consists of a series of clues that lead to the final goal of candy heaven. Each clue leads to a place where there are multiple clues – the idea is that each contestant must test out every clue, for all but one leads to a dead end. However, Nory, being a ninja, can use her ninja skills to determine which clue is correct with the following method: At each clue location there will be a specific "key word" unbeknownst to the other contestants. If all the letters of a clue appear in order within that key word, then it is the correct clue. For example, `abd` is a correct clue for the key word `abcd`, but `cde` and `bca` are not. Help Nory determine whether or not a clue is correct.

Time Allocation: 1 second

Name of data file: adv52.dat

Input: The first line of input will be an integer *n* representing the number of data collections to follow. Each data collection will consist of a key word *s* followed by a clue in consideration, *t*. *s* and *t* will be on separate lines.

Output: The output should consist of a single lowercase word, either `yes` or `no`, indicating whether *t* is a correct clue or not. The output is to be formatted exactly like the sample output given below.

Assumptions: *s* and *t* will contain between 1 and 1000 characters, inclusive.
s will contain only lowercase letters and spaces.
t will contain only lowercase letters (no spaces).
s and *t* will each begin and end with a letter.
All input will be valid.

Sample Input:

```
2
boulevard
board
jack o lantern
cooler
```

Sample Output:

```
yes
no
```

Problem 5.3 **Candy Stacking**

General Statement: Will the Werewolf has bought large amounts of candy in anticipation of the trick-or-treaters that come visit him on Halloween night. Bored while waiting for his doorbell to ring, he decides to organize his candy in a peculiar manner. First, he takes an arbitrary number of buckets and stacks them up in a pyramid formation, with one more bucket in each successive row (going downwards). Then, he fills the buckets with candy according to the Fibonacci series. If the buckets are numbered and stacked like so:

```
1
2 3
4 5 6
7 8 9 10
…
```

, then the first bucket will have 1 piece of candy, the 2nd will have 2 pieces, the 3rd will have 3 pieces, the 4th will have 5 pieces, and so on, continuing with the Fibonacci sequence until the required number of buckets are filled. Help Will figure out how many pieces of candy he must put in each bucket.

Time Allocation: 1 second

Name of data file: adv53.dat

Input: The first line of input will be an integer *n* representing the number of lines to follow. Each line consists of a single integer *x* that represents the number of buckets to be filled with candy.

Output: The output should consist of Fibonacci numbers in a pyramid formation, representing the amount of candy in each bucket, which are stacked like a pyramid. If there are not enough buckets to make a perfect pyramid, then the pyramid should be constructed from the top downwards until there are no more buckets (yes this defies gravity). Each row is filled from left to right. A single space should separate each number in the pyramid. A blank line should separate each pyramid. The output should be formatted exactly like the sample output below.

Assumptions: *x* will be an integer between 1 and 90 inclusive.
(Warning – the *nth* Fibonacci number for *n*>45 is greater than 2^{32})
Trailing spaces at the end of rows will not be judged.
All input will be valid.

Sample Input:

```
2
8
15
```

Sample Output:

```
1
2 3
5 8 13
21 34

1
2 3
5 8 13
21 34 55 89
144 233 377 610 987
```

Problem 5.4 **Anaconda Doodling**

General Statement: Stewart the Studious Student is an unfortunate soul and has to take classes on the night of Halloween, so while dreaming earnestly about candy and costumes, he takes notes in a very strange manner. Every sentence he writes is in a snake-like formation (the concentration needed to do this keeps him focused). Given a sentence that his teacher says, output what it will look like in his notes.

Time Allocation: 1 second

Name of data file: adv54.dat

Input: The first line of input will be an integer *n* representing the number of data collections to follow. Each data collection will consist of two lines: the first line of input will be an integer *x*, representing the width of the snake. The next line of input will be a string *s*, which is to be reformatted and output.

Output: The output should consist of the string *s* output in the snake like formation, left-justified, with a width *x,* like those demonstrated in the sample output. A word snake is created by first writing *x* letters from left to right, 1 letter directly below the last letter, then x letters from right to left, then 1 letter directly below the last letter, and so on, making a snake like shape. There will be no spaces or punctuation in the word snake, and all letters will be capitalized. Each output should be separated by a line of space.

Assumptions: *n* will be an integer between 1~~0~~ and 1,000,000 inclusive.
s may contain punctuation, spaces, and upper and lower case letters.
Punctuation in *s* is limited to commas and ending punctuation.
s will have between 2 and 100,000 characters inclusive.
x will be an integer between 2 and 100 inclusive.
The output must contain only uppercase letters.
All input will be valid.

Sample Input:

```
2
3
Pumpkins
5
Frankenstein is crashing the party!
```

Sample Output:

```
PUM
  P
NIK
S

FRANK
    E
IETSN
N
ISCRA
    S
TGNIH
H
EPART
    Y
```

Problem 5.5 **Lucky Lucky!**

General Statement:

Casper decided he wants to dress up as a basketball player for Halloween. He tries to think of a number to put on the back of his jersey, and decides to put a random lucky number on his back to give him just that, luck. A lucky number in mathematics is a natural number obtained using the following method:
If you take the list of integers, beginning with 1:

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21…}

Look at the first number after 1. Since 2 is the first number after 1, then you must cross out every second number in the set:

{1, ~~2~~, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~, 15, ~~16~~, 17, ~~18~~, 19, ~~20~~, 21…}

Since 2 was the last number used, find the next number (not crossed out) after 2. Since that number is 3, cross out every third number remaining (do not count the numbers already crossed out).

{1, ~~2~~, 3, ~~4~~, ~~5~~, ~~6~~, 7, ~~8~~, 9, ~~10~~, ~~11~~, ~~12~~, 13, ~~14~~, 15, ~~16~~, ~~17~~, ~~18~~, 19, ~~20~~, 21…}

And repeat the process over and over again. The number *c* that determines which numbers are to be crossed off is the next number after the previous *c* that was used. For example, the next *c* in the example above would be 7, since it is the next number after 3, the number that was previously used. The numbers that are left after crossing numbers out from the list of integers are called lucky numbers. (Since they are lucky enough not to be crossed off) Help Casper find the *x*th lucky number to put on the back of his jersey.

Time Allocation: 1 second

Name of data file: adv55.dat

Input: The first line of input contains an integer *n* that represents the number of lines to follow. On each of the next *n* lines is an integer *x,* representing the *x*th lucky number you must output. Output should be formatted exactly like the sample output below.

Output: The *x*th lucky number.

Assumptions: *x* will be an integer between 1 and 1000 inclusive.
All input will be valid.

Sample Input:

```
2
3
5
```

Sample Output:

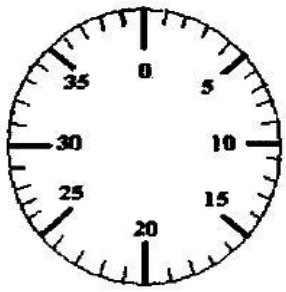
```
7
13
```

Problem 5.6 **Just the Right Combination**

General Statement: Finch the Dwarf, being paranoid, has locked up his share of candy obtained from his hard earned trick-or-treating in his trusty safe. Unfortunately, the next day, he forgot his 3-number combination, and he is devastated, for he really wants to eat his candy. Luckily, Finch remembers a trick to figuring out the correct locker combination by determining for which numbers the lock "clicks" when turned past those numbers. When he tries this, Finch discovers that four numbers click when he turns it clockwise, three numbers click when he turns it counterclockwise afterwards, and then only 2 numbers click when he turns it clockwise a final time.

Finch soon remembers some other rules. If a locker combination is represented as `A-B-C`, where `A, B, C` represent positive integers, then the 2nd number must be a minimum of 180º clockwise away from the 1st number, and the 3rd number must be a minimum of 90º counterclockwise and maximum of 180º counterclockwise away from the 2nd number. Also, the sum of `A, B,` and `C` cannot exceed 70.

If his lock looks like so:



with only the numbers 0 through 39 on it, help Finch determine which of the possible combinations is the right one. There are 9° between each tick on the lock. For example, it is 45 degrees from the digits 0 to 5 on the lock above.

Time Allocation: 1 second

Name of data file: adv56.dat

Input: The first line of input will be an integer *n* representing the number of data collections to follow. Each data collection consists of three lines: the first line will contain 4 integers, each separated by a single space, representing the possible numbers for `A`, the second line will contain 3 integers, each separated by a single space, representing the possible numbers for `B`, and the third line will contain 2 integers, each separated by a single space, representing the possible numbers for `C`.

Output: The correct combination of Finch's lock, according to the rules stated above, in the form `A-B-C`, where `A, B,` and `C` represent positive integers between 0 and 39 inclusive. Output should be formatted like the sample output below.

Assumptions:

n will be a number between 1~~0~~ and 100,000 inclusive.
Every number in each data collection will be between 0 and 40, exclusive.
There will be only one valid combination for each input.
All input will be valid.

Sample Input:

```
2
1 20 15 31
20 36 31
25 27
3 13 23 33
32 1 2
15 10
```

Sample Output:

```
1-36-25
3-32-15
```

Problem 9.1 **It's a secret!**

General Statement: Mad Scientists Snarvel and Mivel are communicating with each other about their secret formulas for creating the ultimate candy. However, to keep their formulas a secret, they decide to use the Vigenère cipher. In this special cipher, to encrypt the plaintext, or the message that the scientists want to send to each other, the following table of alphabets, consisting of the alphabet written out in 26 rows, each row shifting cyclically to the left in comparison to the previous row, is used:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Suppose the plaintext to be encrypted is `SECRETINGREDIENT` and that the key is `PUMPKIN`. Write the key over and over until the length of the resulting string matches that of the plaintext: `PUMPKINPUMPKINPU` Then, look at the first letter of the plain text, `S`, and the first letter of the repeated key string, `P`. The encrypted letter, `H`, is found in column `S` and row `P` of the table above. Use this method to find each letter of the ciphertext, or the coded message. Completing the encoding of the plaintext `SECRETINGREDIENT` will result in the following ciphertext:

Plaintext: `SECRETINGREDIENT`
Key: `PUMPKINPUMPKINPU`
Ciphertext: `HYOGOBVCADTNQRCN`

Time Allocation: 1 second

Name of data file: adv91.dat

Input: The first line of input will be an integer *n* that represents the number of data collections to follow. Each data collection will consist of 2 lines: the first line will be the plaintext that is to be encoded, and the second line is the key used to encrypt the plaintext.

Output: The ciphertext, in all caps, and no spaces. Output is to be formatted exactly like the sample output below.

Assumptions:

n will be an integer between 1 and 1,000,000 inclusive.
The plaintext may contain spaces, punctuation, and lowercase letters.
Punctuation and spaces should be ignored when encrypting the plaintext. (i.e. the plaintext should be changed to all caps, no spaces, no punctuation before being encrypted into ciphertext)
The ciphertext will not contain spaces, punctuation, or lowercase letters.
The key and plaintext will each contain between 5 and 1,000 characters inclusive.
The key will be given in all caps.
All input will be valid.

Sample Input:

```
1
secret ingredient...
PUMPKIN
```

Sample Output:

```
HYOGOBVCADTNQRCN
```

Problem 9.2 **Number Crunching**

General Statement: Mivel the Mad Scientist is doing some calculations on his calculator in order to perfect the ultimate chemical that will create the ultimate candy. However, his calculator operates on a very strange form of RPN (reverse polish notation): All the numbers must be inputted first (and they can only be single digit, positive numbers), and the list of operations must be inputted last. Then, the calculator carries out the operations (from the first operation to the last) on the given list of numbers. The regular order of operations is to be ignored – the first operation is carried out with the first two digits, the second operation is carried out with the result of the first operation and the next digit, and so on.
If digits are represented as follows:

```
###   #   ###  ###  # #  ###  ###  ###  ###  ###
# #   #     #    #  # #  #    #      #  # #  # #
# #   #   ###  ###  ###  ###  ###   #   ###  ###
# #   #   #      #    #    #  # #   #   # #    #
###,  # , ###, ###,   #, ###, ###,  # , ###,   #
```

Output what the result of the calculations would look like on Mivel's calculator.

Time Allocation: 1 second

Name of data file: adv92.dat

Input: The first line of input will be an integer *n* representing the number of data collections to follow. Each data collection consists of a list of digits in the format listed above, each digit separated by a single space. After each list of digits is a list of operations to be operated on the digits, all on one line with no spaces between them.

Output: The output should be the result of the operations on the given numbers, rounded to the nearest hundredth. The digits of the result should be formatted like the numbers above, and output should be formatted exactly like the sample output below. A decimal point is represented as `#`, with a space before and after the decimal point. A blank line should separate each answer.

Assumptions: *n* will be an integer between 1~~0~~ and 1,000,000 inclusive.
The only numbers given will be digits.
The width of each digit is three characters, including spaces.
The number of operations will be one less than the number of digits given.
The result will be a positive double.
The operations are as follows: `+`, add; `-`, subtract; `*`, multiply; `/`, divide.
The above operations are the only operations that will be given.
Each digit is separated by a space.
There will be one trailing space at the end of each line.
All input will be valid.

Sample Input:

```
2
### # # ###
  # # #   #
### ### ###
#     #   #
###   # ###
+/
### ### ### ### ###
# # # #   # #     #
### ###  #  ### ###
  # # #  #    #   #
  # ###  #  ### ###
+-*/
```

Sample Output:

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

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

Problem 9.3 **Tic Tac Toe, Three in a Row**

General Statement: Harry and Ron are fighting for the last piece of candy from their trick-or-treating rewards. Unable to decide on who gets it, they decide to play a game of Tic Tac Toe to determine the winner of the exalted piece of candy. Given a partially completed Tic Tac Toe board, determine the number of ways that Harry can win, Ron can win, or for there to be a draw (called a "Ghost's game").

Time Allocation: 1 second

Name of data file: adv93.dat

Input: The first line of input will be an integer *n* representing the number of data collections to follow. Each data collection consists of a tic tac toe board in the given form:

```
@@@
@@@
@@@
```

where `'@'` represents either an `X`, an `O`, or an unfilled space.

Output: Output the number of distinct ways there are for each possible outcome if one more move is permitted. Assume Harry is `X` and Ron is `O`. For example, in the board

```
XXO
1O2
XO3
```

there are 3 spaces where an `X` or an `O` could be placed, labeled `1`, `2`, and `3`. If an `X` is placed in space `1`, Harry would win. If an `X` is placed in spaces `2` or `3`, or an `O` is paced in spaces `1`, `2`, or `3`, then nobody would win, ending in a Ghost's Game. Thus there is one way for Harry to win, 0 ways for Ron to win, and 5 ways for a Ghost's game. Output should be formatted in the following manner:

```
# way(s) for Harry to win.
# way(s) for Ron to win.
# way(s) for Ghost's game.
```

, where `'#'` represents the number of possible ways for each outcome. A blank line should separate each output.

Assumptions: *n* will be an integer between 1~~0~~ and 10,000 inclusive.
Every game board will have an outcome of some sort
A "Ghost's game" in this context means that the result of one more move will not result in a win or loss for either player.
The total number of ways for each possible outcome should equal double the number of empty spaces.

Sample Input:

```
2
XXO
 O
XO
XOO
 XX
 XO
```

Sample Output:

```
1 way(s) for Harry to win.
0 way(s) for Ron to win.
5 way(s) for Ghost's game.

1 way(s) for Harry to win.
0 way(s) for Ron to win.
3 way(s) for Ghost's game.
```

Problem 9.4 **The Power of the Matrix**

General Statement: Millicent the Mathematician has derived a formula to determine which neighborhoods will yield the most candy. However, part of her formula involves multiplying matrices (representing 2 different neighborhoods) together. Despite being a mathematician, she is very lazy and doesn't want to do all this math by hand. Write a program to help Millicent multiply matrices.

Time Allocation: 1 second

Name of data file: adv94.dat

Input: The first line of input will be an integer *n* that represents the number of data collections to follow. On the first line of each data collection will be four integers, r1, c1, r2, c2, each separated by a single space. r1 and c1 represent the number of rows and columns, respectively, in the first matrix, and r2 and c2 represent the number of rows and columns, respectively in the second matrix. On the next r1 lines will be the matrix A; each entry (number) of A will be separated by a single space. On the next r2 lines will be the matrix B formatted in the same way as A.

Output: The product of matrices A and B; output should be formatted like the sample output below. The resulting matrix should have r1 rows and c2 columns, and each entry should be separated by a single space. A line of space should separate each output. Matrix multiplication is accomplished in the following manner: If 2 matrices, A and B are to be multiplied, then entry $AB_{i,j}$ (where i and j represent the row and column of the entry in AB) =

$$A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + \ldots + A_{i,n}B_{n,j}$$

,where n is the number of columns in A and the number of rows in B. So, if $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 4 & 6 \\ -2 & 1 & 3 \end{bmatrix}$ then $AB_{1,1}$ = (2)(1)+(1)(-2) = 0

Using the above method, $AB = \begin{bmatrix} 0 & 9 & 15 \\ -3 & 6 & 12 \end{bmatrix}$

Assumptions: *n* will be an integer between ~~0~~1 and 1,000,000 inclusive.
r1, c1, r2, c2 will be integers between 0 and 100 inclusive.
c1 will always equal r2.
All elements of the matrix will be integers.
All input will be valid.

Sample Input:

```
2
2 2 2 2
1 1
0 1
1 -1
0 1
2 3 3 2
2 5 7
-2 -5 -7
7 8
4 5
1 2
```

Sample Output:

```
1 0
0 1

41 55
-41 -55
```

Problem 9.5 **Giant troubles**

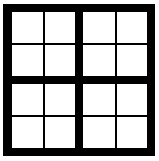
General Statement: Oh no! Baby Thor, son of Helga the Giant, is trapped in the Poltergeist's haunted maze. Luckily, Thor manages to stampede his way through the puny hedges toward the exit. However, the exit is blocked by a magical door, which can only be opened by solving the Sudoku puzzle inscribed on it. Help Thor solve the puzzle and escape back to his mommy.

Time Allocation: 1 second

Name of data file: adv95.dat

Input: The first line of input will be an integer *n* that represents the number of data collections to follow. Each data collection represents the Sudoku puzzle, which has 4 rows and 4 columns. The next 4 rows contain the puzzle, and each line will have four integers between 0 and 4 inclusive. '0' denotes an empty box that Thor must fill in with the correct number, and integers between 1 and 4 inclusive represent filled in squares that cannot be changed.

Output: Output the completed Sudoku board. To solve a Sudoku puzzle, view each puzzle as a 4x4 grid, separated into 4 2x2 squares as pictured below:



Every row, column, and outlined 2x2 square must be filled with the numbers 1 through 4 with no repeats. For example, if the puzzle given is:

```
1004
0200
3000
0100
```

where '0' represents an empty space, then the completed puzzle would be:

```
1324
4231
3412
2143
```

A blank line should separate each output. Output should be formatted exactly like the sample output below.

Assumptions: There will be only one correct solution for each input.
The only numbers given in each Sudoku board will be integers between 0 and 4 inclusive.
The only numbers in the solved Sudoku board should be integers between 1 and 4 inclusive.
All input will be valid.

Sample Input:

```
2
1004
0200
3000
0100
1004
0420
0302
0000
```

Sample Output:

```
1324
4231
3412
2143

1234
3421
4312
2143
```

Problem 9.6 **Candy Run**

General Statement: Tilly is attempting to find the path of greatest satisfaction. According to data collected from past trick-or-treating adventures, Tilly has created a maps of different neighborhoods detailing which houses give out what amounts of candy. She is trying to find the shortest path that will get her the most amount of candy. In other words, she needs to find the smallest number of houses that she has to visit to get from the starting point to the finishing point, while also trying to visit houses such that the amount of candy she gets is the maximum for the number of houses that she visits. Help her determine what is the maximum amount of candy that she can obtain.

Time Allocation: 1 second

Name of data file: adv96.dat

Input: The first line of input will be an integer *x* that represents the number of data collections to follow. The first line of each data collection will contain two integers *r* and *c,* separated by a single space, that represents the number of rows and columns, respectively, in the maze. The next *r* lines will contain the maze. Use the following key for the maze:

`#`	Empty lot – Tilly does not visit these
Digits `0-9`	A house; the number is the amount of candy the house offers
`S`	Start
`F`	Finish

Output: The maximum amount of money that can be obtained from a single path from start to finish.

Assumptions: Once a house has been visited, it may not be visited again.
Each free space (except for start and finish) will be represented by a single integer $0 \le n \le 9$.
Start and Finish have a value of 0
The number of houses she may visit is equal to the number of steps it takes to get from start to finish.

Sample Input:

```
1
5 6
S#457#
150###
48687#
1###9#
#F0156
```

Sample Output:

```
50
```