Problem 8: Foveated Rendering

Points: 15

Author: Gary Hoffmann, Denver, Colorado, United States

Problem Background

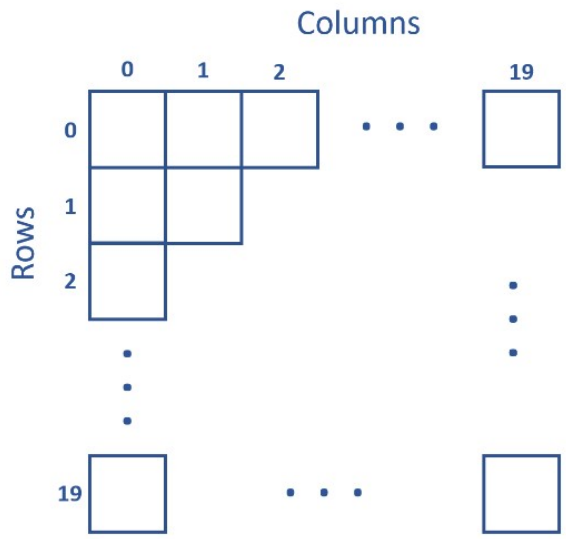
Virtual Reality has exploded into the market in the last five years, being used for everything from games and entertainment to product design and engineering. One of the more recent advances in VR headset design is the addition of eye tracking to increase performance.

The human eye has an extremely narrow field of view in which perfect 20/20 vision is attainable and fine detail can be distinguished. This clarity of vision is due to the fovea, a small depression in the inner retina specialized for this purpose. However, due to the size of the fovea, the human eye can only see clearly within a field of view of less than 10°. The rest of our vision comes from the brain piecing together imagery as we look around.

Due to this fact, a VR headset only needs to render the highest resolution imagery directly where the user is looking. Images outside of that field of view can be rendered at a lower quality, increasing the performance of the system.

Problem Description

You have been tasked with writing a module for a virtual reality application that determines the rendering quality for each section of the headset's screen. For simplicity, your module will only deal with a single eye on a single screen. The screen will be divided into a 20-by-20 grid of blocks.



Your program will be given the coordinates within the grid at which the user is currently focusing their sight, and will need to output the rendering level of each cell in the grid row by row.

The cell the user is looking directly at should be rendered at full quality - 100%. All cells around that cell should be rendered at half quality (50%), and all cells around those should be rendered at one-quarter quality (25%). All other cells should be rendered at the minimum level of 10%.

For example, if the user is looking at the block in row 7, column 10, the rendering quality for each block in the grid would be:

Row \ Col	0	⋯	7	8	9	10	11	12	13	⋯	19
0	10%	⋯	10%	10%	10%	10%	10%	10%	10%	⋯	10%
⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋰	⋮
4	10%	⋯	10%	10%	10%	10%	10%	10%	10%	⋯	10%
5	10%	⋯	10%	25%	25%	25%	25%	25%	10%	⋯	10%
6	10%	⋯	10%	25%	50%	50%	50%	25%	10%	⋯	10%
7	10%	⋯	10%	25%	50%	100%	50%	25%	10%	⋯	10%
8	10%	⋯	10%	25%	50%	50%	50%	25%	10%	⋯	10%
9	10%	⋯	10%	25%	25%	25%	25%	25%	10%	⋯	10%
10	10%	⋯	10%	10%	10%	10%	10%	10%	10%	⋯	10%
⋮	⋮	⋰	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
19	10%	⋯	10%	10%	10%	10%	10%	10%	10%	⋯	10%

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line of input containing two integers, separated by spaces, representing the row and column number of the eye position within the screen's grid, respectively. Row and column numbers will be between 0 and 19 inclusive.

```
2
7 10
0 0
```

Sample Output

For each test case, your program must output the rendering quality percentage for each block in the grid. Each row should be printed as a separate line, and columns should be separated by spaces.

```
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 25 25 25 25 25 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 25 50 50 50 25 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 25 50 100 50 25 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 25 50 50 50 25 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 25 25 25 25 25 10 10 10 10 10 10 10
```

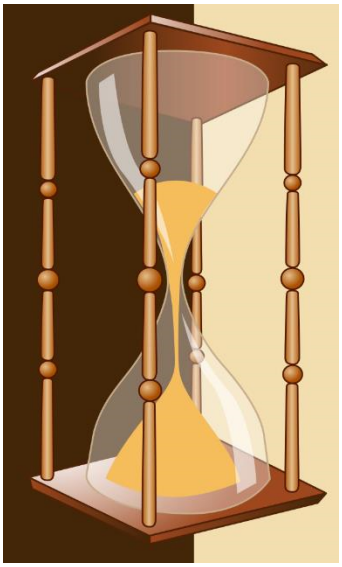
```
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
100 50 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
50 50 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
25 25 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
```

Problem 9: Time and Time Again

Points: 20

Author: Jonathan Brown, Fort Worth, Texas, United States

Problem Background



Times and periods of times can be expressed in many different ways. National and regional differences, and even personal preferences, have led to a wide range of formats for expressing times. This can lead to a great deal of confusion; does the date 01/03 refer to January 3rd or March 1st… or January 2003? Is the time 8:45 in the morning or the evening?

You have been asked to break through some of this confusion by converting a list of times to a new, consistent format.

Problem Description

Your program will receive a list of time durations that provide the number of hours, minutes, and/or seconds within the duration.

- Hours will be given as a non-negative integer followed by a lowercase letter 'h' (e.g. 2h). Hours will range from 0 to 99 inclusive.
- Minutes will be given as a non-negative integer followed by a lowercase letter 'm' (e.g. 2m). Minutes will range from 0 to 59 inclusive.
- Seconds will be given as a non-negative integer followed by a lowercase letter 's' (e.g. 2s). Seconds will range from 0 to 59 inclusive.

These values may not be presented in this order. Values may be separated by spaces, commas, and/or the word "and"; this text should be ignored. Some of these values may be missing; for example, an input may only give you minutes and seconds. Any omitted values should be assumed to be zero.

Regardless of what information is provided, your program will need to print the duration in a simpler, more consistent format:

HH:MM:SS

In this format, HH is a two-digit number representing the number of hours (including a leading zero, if necessary). MM is a two-digit number representing the number of minutes (including a leading zero, if necessary). SS is a two-digit number representing the number of seconds (including a leading zero, if necessary). Each number is

separated from the next with a colon, and they are always presented in the same order. All numbers must be included with the output, even if they are zero.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line of input containing a string describing a time duration in a variable format as noted above.

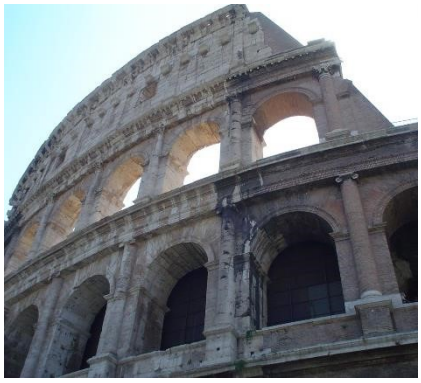
```
5
1m and 45s
10m,10s
32s, and 12h
76h
1s
```

Sample Output

For each test case, your program must output the same time interval on a single line in the HH:MM:SS format described above.

```
00:01:45
00:10:10
12:00:32
76:00:00
00:00:01
```

Problem 10: Caesar Cipher



Points: 20

Author: Steve Gerali, Denver, Colorado, United States

Problem Background

The Caesar Cipher is one of the earliest known ciphers, and among the simplest to learn. It is a "substitution cipher", in which each letter in the original message (the "plaintext") is shifted a certain number of places down the alphabet. For example, with a shift of 1, an A would be replaced with a B, a B would be replaced with a C, and so on. This method is named after Julius Caesar, who apparently used it to communicate with his generals.

To pass an encrypted message from one person to another, it is necessary that both parties have the "key" for the cipher, so that the sender can encrypt it and the recipient can decrypt it. For the Caesar Cipher, the key is the number of letters by which to shift the cipher alphabet.

Problem Description

You are working for the History Channel, who wants to decrypt all communications that Julius Caesar made to his generals in order to support a new documentary they're filming about the Roman emperor. You will be given a list of encrypted messages, and the key believed to be used to encrypt those messages. Your program must decrypt those messages.

For the purposes of this problem, we will be using the English alphabet, shown below in its standard order (with a shift of 0).

```
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
```

If encrypting a message with a shift of 1, each letter in the plaintext will be replaced with the respective letter shown in the 1-shifted alphabet below.

```
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
```

To decrypt a message, the process is reversed; a letter in the ciphertext would be replaced with the respective letter in the original English alphabet.

Spaces are not encrypted in this cipher and should remain in place when decrypting a message.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include two lines:

- A line with a single integer representing the message key - the number of letters by which to shift the alphabet when encrypting the message.
- A line containing lowercase letters and spaces, representing the encrypted message.

```
3
1
buubdl bu ebxo
3
ghvwurb wkh fdvwoh
6
yzkgr znk ynov
```

Sample Output

For each test case, your program must output the decrypted message. Messages should be printed in lowercase, and all spaces should be retained.

```
attack at dawn
destroy the castle
steal the ship
```

Problem 11: Count to 10

Points: 25

Author: Ryan Regensburger, Huntsville, Alabama, United States

Problem Background

When testing software or hardware, it's considered a "best practice" to test every possible situation to prove that the code or device is stable under any condition it might come across. For example, if we have a chip with eight LEDs, we might want to light up those LEDs in every combination to make sure they function properly. This is essentially an 8-bit binary counter, displaying each number from 0 to 255.

Problem Description

In this problem, you will need to generate test data for a binary counter like that described above. You will be provided with the number of bits to use for your counter, and will need to generate a list of all binary numbers with at most that number of bits in numerical order.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line with a positive integer, representing the number of bits to use.

```
1
3
```

Sample Output

For each test case, your program must output a list of binary numbers, ranging from 0 to the maximum value with the indicated number of bits, inclusive. Numbers must be listed one per line, in numerical order. Include any leading zeros up to the required bit length.

```
000
001
010
011
100
101
110
111
```