UNIVERSITY OF MANITOBA

DEPARTMENT OF COMPUTER SCIENCE

Manitoba High School Programming Contest 2017
2 June 2017 12:45 – 3:20

Contest Rules:

- Do not open this package until instructed to do so.
- All solutions must be entered completely during the contest. No electronic copies of pre-written code are permitted.
- You may submit as many solutions as you like to each problem; however, incorrect solutions will be assessed a time penalty.
- Contest score is based on the most problems successfully solved; ties are won by shortest total time taken, including any penalties.
- A correct submission must solve the given problem and produce correct output for the given test data within a reasonable time.
- Programming style will not be considered during judging.
- Any programming language resources and notes are allowed.
- No other Internet access is allowed during the contest.

Submission Requirements / Pre-submission Checklist:

A. All input must be read from standard input (in Java, `System.in`). Only Problem 1 does not require input. **Do not open input files.**
B. All output must be written to standard output (in Java, use `System.out.print` or `println`).
C. Your output format must follow the problem requirements **exactly**.
D. Submit the source code file (`.java`, `.c`, etc.–**NOT** `.class`, `.exe`, etc.).
E. Java programs must be in a single file and not placed in a package (no `package` statements; `import` statements are of course OK).
F. Java programs must be complete; **they require a `main` method**.

Problem 1 – Fuel Efficiency

Most metric to imperial conversions are simple conversions based on a multiplication or division, using facts like “there are 1.61 km in one mile” and “there are 3.79 L in one US gallon”. But fuel efficiency conversions (for cars) are a bit more complicated, as metric fuel efficiency measures how much fuel it takes to go a certain distance (L/100 km) and imperial fuel efficiency measures how far you can go on a certain volume of fuel (miles per gallon, abbreviated mpg). So as measurements in L/100 km goes up, mpg goes down.

Write a program that prints a table of imperial fuel efficiencies for a fixed number of metric fuel efficiency values. The mpg should be expressed to the next lowest whole number (i.e., your program should always round down). Use the conversion quantities given in the first paragraph to help with the conversions.

The table should show the conversion to mpg of measurements from 4 L/100 km (hybrid car fuel efficiency) to 15 L/100 km (minivan full of bricks fuel efficiency). The table should show all integer-valued metric fuel efficiencies.

Input

For this problem, there is no input. Produce the output based on the values above.

Output

For each metric fuel efficiency, write one line of output as “`X L/100 km is Y mpg`.” where `X` and `Y` are integer values. There is a single space between all words, as well as between the integer values and the units. There is a period at the end of each line.

Sample Output

`4 L/100 km is 58 mpg.`
`5 L/100 km is 47 mpg.`
(rows for 6-14 L/100 km deleted)
`15 L/100 km is 15 mpg.`

Problem 2 – Skipto Crypto

You and your friend have devised a cryptographic protocol – that is, a way to encode a text message so that it (hopefully) can't be deciphered by anyone except for the two of you. Your protocol works by including digits in the encoded message that tell you how many letters to skip to find the next character of the real message. The first character of the encoded message is always a digit to tell you how to start skipping characters in your message.

For instance, consider the encoded message `1h2he2al3aal312o0`. The first `1` indicates that you should skip to the next character to get the first letter (`h`). The next step is to move to the next character (which will always be a digit): in this case it is a `2`. We skip two characters to get the second letter (`e`), then repeat this process:

- Next digit is a `2`, skip to `l`
- Next digit is a `3`, skip to `l`
- Next digit is (again) a `3`, skip to `o`
- Next digit is a `0`, stop decoding.

Thus, the output for this message is "`hello`".

The digit that gives skipping information is always a single digit from 1-9 and it is always positive. A digit of zero indicates the end of the message to be decoded, however there may be extra, ignored characters after the zero on the same line. The junk characters that are skipped over can either be letters, numbers or punctuation (underscore, period or comma). The letters of the message can also be letters, numbers or punctuation. If an underscore appears as a non-junk character, it represents a space (and should be printed as a space in the output) but there are no true spaces in the input.

Input

Each line of the input (except the last) represents a case to decode. The inputs are all decodable messages. The last line is a single zero. You should not process this line.

Output

For each message, output "`Case #x`" where `x` is a digit (starting from 0), then a colon, a space and then the decoded message.

Sample Input	Sample Output
`1h2he2al3aal312o0` `20o399v6helloe1r1_91234567891010100abc31415042.000`	`Case #0: hello` `Case #1: over 9000`

Judging Data for Problem 2

*Note: because of line-wraps, four of the twelve test cases are displayed over multiple lines: however, these are each a **single line in the input files**. The test cases are described after the file.*

```
1h2he2al3aal312o0
20o399v6helloe1r1_91234567891010100abc31415042.00
5VOybh5sQn0i4.P4g6qgGghh0IVj6p
6xbV6rs45JSc8L6M.S59h6,xDrvo5PH4Uo3ZFl0ZPO.q
8KYhXBXUc3bho6ab,Vcm7sFs_T4p83HurzmQu71wnlwut1e2Ir0aCSH2K5f
47fBs5GtUyc3TAi5_TXSe6o4Yr3n7uXJiMvc2me0o5SNQ
4f_rd2Ka5Z50uy0VSoC
2b27ipzS.704aQs1170A
1t68domYh20i1s1_31Vi6O.3ZTs2h_6Dguzfa5lJb7_1s4UTEe21c4qM9r393e22t66piWU
_712DfT0m4pOTe53fWRs52WL7s1a2vg78_x8lAe02qS9Mz
4XA8I3oqt4e4S_3cMi5chLys6qgkQm_7Tq_3Von7bXNK13o8r4bejHrt4_0X_7sZyxx_v1e
64szA7r63pP_Ky1_435lw8oD6l5hfe2al5,o,ll7XNTemX_3aIh2mi2ld3rAd5IcKne1n0S
Vs8aNmsNI
7ZNHvj1t7lGLocKh5AR_Si3,Us5Bn5Y_2mm2Me3Rws5oAbns7sM,zlta4jrHg22e4T8G_4D
aEc6mxGJgo6BeShGn7rADcOKt7fZegdFa3Kyi1n36Js1_65UOXfn6USMr5u3Brm4q7kb1e8
uqousI4r8f7fZXPis1_6N,kI685yjJF64p8y75Q6wS_2s58TWCqDN335Jic0_66y0YD04Yl
v90BZyb5s7c
88G,s9sQO4nR.n1e5rx5S_7vVWTJbi5x2Jzf6jGqCv_2Ib8f9pGv4xy7o13kUf_4rlKl4IK
Ua36on6la0NTd54k6v,5ByFf_7zTUD,6a2On3wCd5_HnQ_82Oeb8Gqt3FIw3.Do8AovmTX2
_3hNi7Dv6Qbuf5TT58_6jqEdpb875Yq3WSy7A1cqum_6Cg0Xws8gSdJl1te6YPv.Ia1.06W
0
```

Lines that start test cases begin with:

- `1h2`
- `20o`
- `5VO`
- `6xb`
- `8KY`
- `47f`
- `4f_`
- `2b2`
- `1t6` (spans two lines above)
- `4XA` (spans three lines above)
- `7ZN` (spans four lines above)
- `88G` (spans three lines above)

Problem 3 – Radix sorting

Radix sorting is an algorithm that sorts numbers based on their digits in a certain position of the number. That is, given a position in the number (left-most digit, second digit from the right, etc.), one step of radix sort is to order the numbers by that digit, ignoring all the other digits for the moment. The overall radix sorting requires many steps of this basic sorting step, but we are only interested in the basic step. For instance, suppose we are radix sorting the numbers

167 258 349 430 521 642

and we are sorting based on the middle digit. Then the result would be the list

521 430 349 642 258 167

Notice that the numbers aren't sorted in increasing order at all, but they are sorted according to their second digit: 521 has the smallest second digit (2) and 167 has the largest second digit (6).

What do we do about breaking ties? For instance, in the example, both 349 and 642 have middle digit 4. To break the tie, the sorting should be "stable", which means that the number that appeared first in the original list should appear first in the new list if you need to break a tie. That's why 349 comes before 642 in the sorted list, since it was first in the unsorted list.

Write a program that accepts a list of integers, all of which have the same number of non-zero digits, as well as position that you should sort by. The program should produce the list sorted by that position. The sort must be stable.

Input

There are multiple test cases in the input. The first line of a test case has two integers, n and d. The number n is the number of integers to be sorted, and d is the position that we should be sorting. Positions are always counted from the right, starting at 1 (the right-most digit of the number).

The second line of a test case is n integers, all separated from the next in the list by a single space.

After the final test case, there are two integers 0 0 on a single line. This is not a test case: it indicates the end of the input. Do not produce any output for this line.

Output

For each test case, output the list of integers, sorted by the specified digit.

Sample Input	Sample Output
6 2 167 258 349 430 521 642 6 3 9119 5037 3370 3270 3275 1000 0 0	521 430 349 642 258 167 5037 1000 9119 3270 3275 3370

Judging Data for Problem 3

```
6 2
167 258 349 430 521 642
6 3
9119 5037 3370 3270 3275 1000
1 1
111
2 1
111 112
2 1
112 111
2 2
112 111
20 1
7096 7683 6125 3689 1379 7962 8749 5662 5545 4023 5383 7288 1919 5553 4772 3675 2399 7255 7216 2959
20 2
7096 7683 6125 3689 1379 7962 8749 5662 5545 4023 5383 7288 1919 5553 4772 3675 2399 7255 7216 2959
20 3
7096 7683 6125 3689 1379 7962 8749 5662 5545 4023 5383 7288 1919 5553 4772 3675 2399 7255 7216 2959
20 4
7096 7683 6125 3689 1379 7962 8749 5662 5545 4023 5383 7288 1919 5553 4772 3675 2399 7255 7216 2959
0 0
```

Problem 4 – Penniless Change

Since 2013, Canada hasn't had a penny (1¢ coin). Credit and debit card transactions were not affected by the removal of the penny, but the federal government developed "guidelines" to describe how cash transactions should be rounded. According to the rules, the cost of the item (**not** the amount of change) should be rounded to a 5-cent value. They rules are:

Last digit of cash value	**Rounding**
0,1,2	Down to 0
3,4,5,6,7	To 5
8,9	Up to next 0

So an item with a cash value of 1.03 would be rounded to 1.05 and an item with a cash value of 3.49 would be rounded to 3.50.

In this problem, you should write a tool that will take a cash value and payment value and return the amount of change provided. You should calculate the value based on the rounding guidelines, and calculate which coins should be returned. For the purposes of this question, you can assume that the amount of change will be between 0 and 499 cents, and that the coins are the toonie (200¢), loonie (100¢), quarter (25¢), dime (10¢) and nickel (5¢).

When calculating the coins to return, your tool should always report the smallest number of coins possible. For instance, if the amount of change to return is 35¢, your tool should say "1 quarter and 1 dime" and **not** (for instance) "3 dimes and 1 nickel".

Input

The first line of input is the number of transactions to consider. Every other line is the information on one transaction. Each transaction contains two integers: the cost of the items, and the amount of payment provided. The values are integers giving the number of cents, so $4.59 would be represented as the integer 459.

The value of the change provided will always be at least 0¢ and at most 495¢, however, some of the costs or payments may be greater than $4.95. No payments will involve pennies.

Output

For each transaction, provide an output in the form "`Case #x: a*200 b*100 c*25 d*10 e*5`" where `a`,`b`,`c`,`d` and `e` are integers giving the number of toonies, loonies, quarters, dimes and nickels that should be returned, and `x` is the case number, starting at 1. There is one space between each coin denomination and the next.

Sample Input	Sample Output
2 76 100 1034 1500	Case #1: 0*200 0*100 1*25 0*10 0*5 Case #2: 2*200 0*100 2*25 1*10 1*5

Judging Data for Problem 4

```
14
76 100
1034 1500
100 100
99 100
96 95
5 10
4 10
3 15
8 35
7 105
2 200
160 500
100 595
203 600
```

Problem 5 – Life

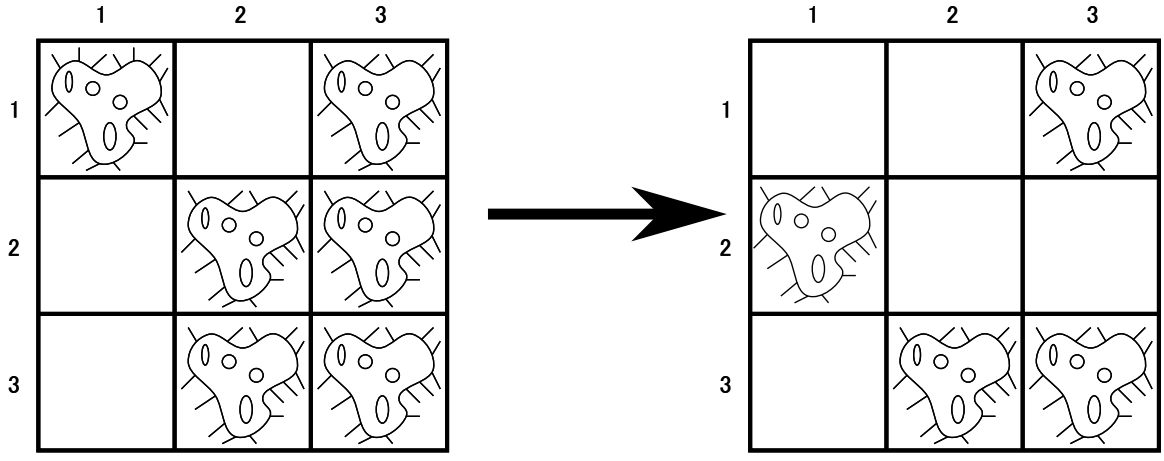
Conway's Game of Life is a zero-player game that allows "cells" living in spaces on a 2-dimensional board to live and die based on how many neighbours they have. The Game of Life is called a zero-player game because it depends only on the starting positions of the cells in the spaces of the board, and doesn't depend on randomness or any action by any players.

Each space in the 2-dimensional board has eight neighbours in each direction (including diagonals) and either contains a live cell or doesn't. At each time step, each cell counts the number of cells that are alive in all the neighbouring positions of the board and decides whether it should live or die in the next time step. The rules to decide whether the cell lives or die are:

- If a live cell has less than two live neighbours, the cell dies at the next time step (of loneliness, presumably).
- If a live cell has more than three live neighbours, the cell dies at the next time step (from starvation due to overpopulation).
- Otherwise, if the live cell has exactly two or three live neighbours, the cell remains alive for the next time step.

Additionally, there's a procreation rule: if an empty space (without a live cell in it) is surrounded by exactly three live cells, then a live cell should be inserted in that spot in the next step.

To see this, consider the following example which shows a 3x3 grid at two consecutive time steps (first time step on the left and the second on the right). In the grid on the left, the cell in row 1 and col 1 has only one live neighbour (the cell in row 2, col 2). Thus, the cell should die, which is why the same cell is empty in the grid on the right. Similarly, the cell in row 1, col 3 survives (it has two neighbours) and the cell in row 2, col 2 dies since it has too many neighbours.



In this question, you will write a program that reads a 2D grid of the two characters 'X' (capital X) and '_' (underscore) and a number of time steps, and

shows the status of the grid after the requested number of time steps. The above situation shows the situation after 1 time step.

Input

The input consists of several test cases. The first line of the input is the number of test cases.

Each test case begins with a line with two integers, n and t. The size of the grid is nxn and the number of time steps is t. The next n lines contain the grid: each line contains exactly n characters, each of which is an X or a _. There are no spaces within each line or at the beginning or end of each line of the grid. The value of n is at least 1 and the value of t is at least 0.

Output

For each test case, first output an identifier line "CASE X" where X is the case number (starting at zero). Then output the nxn grid, using the characters X and _.

Sample Input	Sample Output
3 3 1 X_X _XX _XX 5 2 _____ __X__ __X__ __X__ _____ 6 3 ______ _XX___ _XX___ ___XX_ ___XX_ ______	CASE 0 __X X__ _XX CASE 1 _____ __X__ __X__ __X__ _____ CASE 2 ______ _XX___ _X____ ____X_ ___XX_ ______

Judging Data for Problem 5

```
14
3 1
X_X
_XX
_XX
5 2
_____
__X__
__X__
__X__
_____
6 3
______
_XX___
_XX___
___XX_
___XX_
______
2 1
__
__
2 1
XX
XX
3 1
_X_
_XX
_X_
3 2
_X_
_XX
_X_
5 2
_____
___X_
__X__
_X___
_____
```

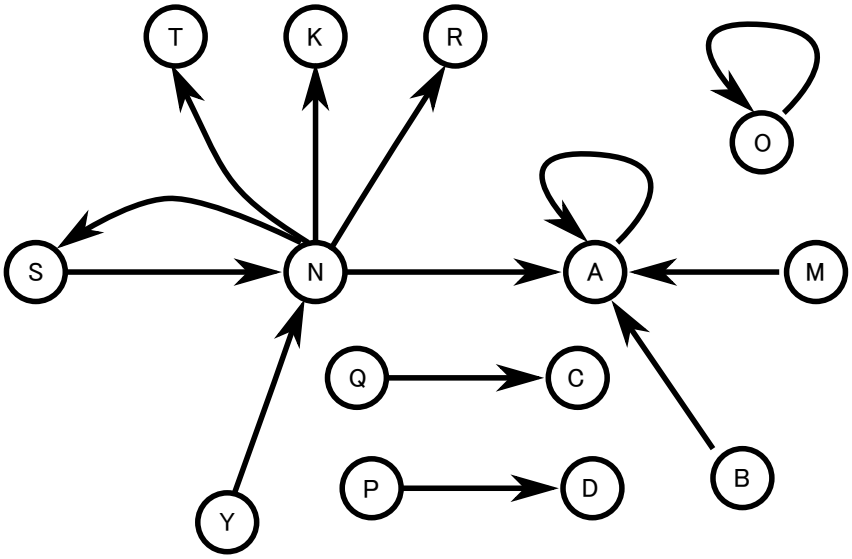
(data continues on next page)

```
16 16
________________
________________
________________
________________
________________
________________
________________
_____X____X_____
___XX_XXXX_XX___
_____X____X_____
________________
________________
________________
________________
________________
________________
5 6
X___X
___XX
_XX_X
X__X_
__XX_
3 16
XXX
___
___
4 14
_X_X
XX__
_XX_
_XX_
5 1
_XXX_
__X_X
XX__X
X__X_
_XX_X
5 15
__XXX
XX___
X_XX_
XXXXX
XXXX_
```

Problem 6 – Word Chain Game

The Word Chain game, also known as Geography, Last and First or The Name Game, is a game for children where a category of words or phrases is chosen, and players take turns choosing words so that the last letter of the previous word is the same as the first letter of the next words. Words also can't be repeated. So if the category was "Canadian provinces and territories" and one player says "Saskatchewan" then the next player could say "Nunavut", provided it hadn't been said before.

When a category with only a small number of words is chosen, this can lead to games with dead-ends, and separated words. For instance, suppose we chose the thirteen provinces and territories as the categories of words we were using for our game. Then there are only a small number of links that can be made, as illustrated in this diagram.



In this picture, the letters represent the first letters of words and the arrows denote that you can go from a word with one first letter to another word with a given first letter. For instance, the arrow from "S" to "N" represents "Saskatchewan" while the arrow from "N" to "S" represents "Northwest Territories".

As you can see, if a player says "Newfoundland and Labrador", the game leads to a dead-end, since no provinces or territories start with "R". But also, there are some games that are separated: If a player starts with "Quebec", there is no way to reach the other provinces.

Write a program that takes a list of words or phrases in a category, and finds how many separate games there are in the list. A game is separate from another game if it leads to a series of turns (possibly none) that does not involve any words from another set. For instance, in the above example, there are 4 separate

games (PEI, Quebec, Ontario and the list of all other provinces and territories). A separate game does not have to be able to play every word in one game; it just has to be made of words that are separate from the words of all the other separate games.

Input

The input consists of several test cases. Each test case starts with an integer n, which is the number of words or phrases in the category. The next n lines each have one word or phrase from the category. There may be spaces in the phrases, but each line should be taken as a single phrase (like "British Columbia", which starts with b and ends with a). Words may start and end with upper or lower case letters, but the case should be ignored in your computations. Only unaccented Latin letters (A-Z and a-z) and spaces will appear in the input. All words or phrases will have length greater than one.

The last line of the input is the number 0. This is not an input test case and should be ignored.

Output

For each test case, output "Case #X: Y games" where X is a case number (starting from 1) and Y is the number of separate games in the category.

Sample Input	Sample Output
5 adverb basic comma elf flame 13 British Columbia Alberta Saskatchewan Manitoba Ontario Quebec New Brunswick Nova Scotia Prince Edward Island Newfoundland and Labrador Yukon Northwest Territories Nunavut 3 alfalfa bacteria crumb 0	Case #1: 2 Case #2: 4 Case #3: 1

Judging Data for Problem 6

```
5
adverb
basic
comma
elf
flame
13
British Columbia
Alberta
Saskatchewan
Manitoba
Ontario
Quebec
New Brunswick
Nova Scotia
Prince Edward Island
Newfoundland and Labrador
Yukon
Northwest Territories
Nunavut
3
alfalfa
bacteria
crumb
17
abracadabra
bulb
cardiac
demand
eagle
fireproof
hairbrush
kayak
label
mushroom
nation
overdo
radar
secrets
tablet
widow
yesterday
4
speechless
twilight
raincoat
ruthlessness
10
diehard
```

(continued next column)

```
flagstaff
eastbound
elf
lime
microscope
speechless
twilight
raincoat
ruthlessness
99
Afghanistan
Albania
Bahamas
Bahrain
Bangladesh
Belgium
Belize
Benin
Bolivia
Brazil
Brunei
Cambodia
Cameroon
Chad
Chile
Comoros
Congo
Cyprus
Denmark
Djibouti
Dominica
Ecuador
Egypt
Eritrea
Fiji
Finland
France
Gabon
Gambia
Germany
Greece
Haiti
Honduras
Hungary
Iceland
India
Iran
Iraq
```

(continued next page)

Israel
Italy
Jamaica
Japan
Kazakhstan
Kenya
Kiribati
Kosovo
Kuwait
Laos
Latvia
Lebanon
Lesotho
Luxembourg
Macedonia
Madagascar
Malawi
Maldives
Mauritius
Mexico
Mozambique
Namibia
Nauru
Nepal
Netherlands
Niger
Norway
Oman
Pakistan
Palau
Panama
Paraguay
Peru
Philippines
Poland
Portugal
Qatar
Romania
Samoa
Senegal
Seychelles
Singapore
Spain
Swaziland
Taiwan
Tanzania
Thailand
Togo
Turkey
Turkmenistan
Tuvalu
Uganda
(continued next column)

Ukraine
Uruguay
Uzbekistan
Vanuatu
Venezuela
Vietnam
Yemen
Zambia
Zimbabwe
10
adverb
cold
elf
grinch
jayhawk
lukewarm
neutrino
pedometer
shirt
undertow
14
undertow
shirt
pedometer
neutrino
lukewarm
jayhawk
grinch
elf
cold
adverb
bioplastic
dome
fig
hi
0

There is no contest material on this page.