Eleventh Annual
University of Central Florida

ACM · UPE

High School Programming Tournament

Problems

Problem Name	Filename
Happiness Man	HAPPY
Princess's Escape Plan	PRINCESS
Counting Dashes	DASHES
Invoke the OrACKle	ORACKLE
Mike Likes to Climb Trees	TREES
Psychic Gypsy Battle	GYPSY
Smash the Copy Machine!	SMASH
Warcraft	WARCRAFT
Proper Penguin Speech	FEEP
The Cycle of Life	LIFE

Call your program file: *Filename*.PAS, *Filename*.C or *Filename*.CPP
Call your input file: *Filename*.IN

For example, if you are solving Smash the Copy Machine!:
Call your program file: SMASH.PAS, SMASH.C or SMASH.CPP
Call your input file: SMASH.IN

Happiness Man

Filename: HAPPY

Happiness Man, the super hero, was born with incredible super powers. His favorite power is his Ray of Happiness and Friendliness. This energy beam drains the evil from those it hits causing them to become polite, helpful members of society. Sometimes, however, his opponents are just too evil in which case he must use his other weapon, his Ray of Pulping and Juicing. (It was so named because he originally used this beam to drain the juice from oranges, but soon found that it could be used on humans, with grisly effects.)

Happiness Man can "cure" up to 50 points of evil per day and can Pulp up to 500 lb. of matter per day. He prefers to cure his enemies' evil, rather than pulping them, but since he often faces many enemies per day, he can't always use his Happiness Ray on them all. For instance, suppose he must fight three enemies today:

> Repulsive Man, who has an evilness rating of 49 and weighs 98 lb.,
> The Deadly Bulb, with an evilness rating of 15 and a weight of 350 lb., and
> Ostrich Woman, with an evilness rating of 23 and a weight of 200 lb.

If Happiness Man uses his Happiness Ray on Repulsive Man, he won't have enough power left to use that weapon on anyone else. He won't be able to Pulp the two other enemies because they weigh too much. His only hope is to Pulp Repulsive Man. Then, he can use his Happiness Ray on the other two villains. The trouble is how can he know when to use which weapon? Fortunately, he has another power, Psychic Precognition, which foresees all the enemies he'll have to fight that day. With the raw data that his Precognition power gives him, he can figure out which weapon to use for each enemy.

The Problem:

Your job is to help Happiness Man make sense of his precognition power. Given a list of the villains he'll fight on a given day, and the villains' evilness and weight, output a list of which weapon to use on whom. There will be up to five enemies per day. You can assume that it will be possible for Happiness Man to defeat all the enemies if he uses his powers in the correct fashion. If there are multiple successful configurations, choose the one that pulps the fewest people. If there are multiple successful configurations that pulp the same number of people, choose one of them.

The Input:

The input file will contain lists of Happiness Man's enemies for various days. The first line of the input for each day will be an integer *n*, representing the number of villains for that day, which will be between 1 and 5 inclusive. The next *n**2 lines contain information about the day's villains. The villains are in the order they will fight Happiness Man. Each villain is described on two lines. The first line of each villain's input will be the name of the villain (the name will not be more than 40 characters long). The second line of each villain's input will contain two integers. The first represents the villain's evilness, and will be between 1 and 100 inclusive. The next represents the villain's weight, and will be between 1 and 5000 inclusive. After all the day's

enemies are described, the data for the next day's enemies begins. Input ends with a value of 0 for *n*.

The Output:

Print which ray should be used for each enemy. If the Happiness Ray should be used, print a line in the form "`Use the Happiness Ray on <villain name>`". If the Pulping Ray should be used, print a line in the form "`Use the Pulping Ray on <villain name>`". (Replace "`<villain name>`" with the villain's actual name.) Print the enemies in the order they appear in the input. Print a blank line after each day's villains.

Sample Input:

```
3
Repulsive Man
49 98
The Deadly Bulb
15 350
Ostrich Woman
23 200
0
```

Sample Output:

```
Use the Pulping Ray on Repulsive Man
Use the Happiness Ray on The Deadly Bulb
Use the Happiness Ray on Ostrich Woman
```

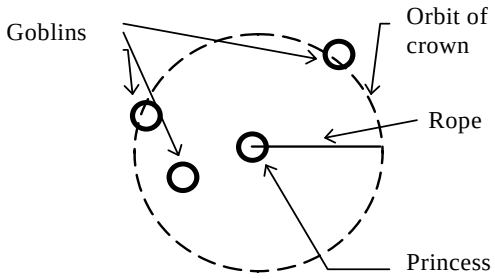
Princess's Escape Plan

Filename: `PRINCESS`

The princess of a small kingdom has been kidnapped by a band of vicious goblins and is being held in a cavern far below the surface. She would like to simply wait until a rescuer comes to save her, since that would be the safest thing to do, but she just doesn't have any faith in the local knights. Fortunately, she is prepared for just such an emergency. By pressing a secret button on her royal crown, she causes a razor to protrude over the crown's rim, making it into a dangerous weapon. She has also found a length of rope. Her plan is to use the crown as a weapon by attaching the rope to the crown and then spinning the crown in a circle around her.

The princess has found herself in a large open cavern, with many goblins in it. She has to act quickly. Her plan is to swing her razor-crown in a circle once, hitting as many goblins as possible, then run for the exit during the confusion.

The Problem:



Given a list of goblins' (x, y) positions, and the position of the princess, your program must decide the length of rope the princess should use (this determines the radius for the crown's orbit). Each goblin (and the princess) can be thought of as 1-unit-diameter circles, with the center of the circles at the given (x, y) coordinates.

Do not consider the cave walls in this problem. Just find the *integer* radius for the crown's orbit which hits the most goblins. If multiple orbits hit the same number of goblins, choose the smallest orbit. Since the coordinates for the goblins and the princess are given in integers, it is not possible for people to overlap. The crown always goes in a perfect circle, even when it hits a goblin. You may assume that any goblins on the inside of the orbit duck under the rope. For the purposes of this problem, the crown does not have any physical size (in other words, it is a single point). When calculating the length of the rope, assume that the rope extends from the center of the princess's circle.

The Input:

There will be multiple input sets. The first line of each input set will contain a single integer *n*, indicating the number of goblins in that input set. The next line of each input set will contain two integers, representing the x- and y-coordinates of the princess. After that, there will be *n* lines, each containing two integers, representing the x- and y-coordinates for a goblin. The value of *n* will be between 1 and 100 inclusive. When a value of zero is read for *n*, this indicates the end of the input. All x- and y-coordinates will be integers between 1 and 1000 inclusive. No two goblins in the same input set will have the same coordinates; nor will any goblin have the same coordinates as the princess.

The Output:

For each data set, print a line with a single integer, indicating the length of rope that the princess should use.

Sample Input:

```
3
50 50
58 52
38 46
46 54
0
```

Sample Output:

```
6
```

Counting Dashes

Filename: DASHES

Ali is trying to teach his daughter to count. As an exercise, he has her count the yellow dashes in the middle of the road on long road trips. But he can't verify her answers since he is watching the road and can't count them himself. To get around this problem, he notes the odometer reading and has her count for several miles. By knowing how far the car has traveled and some facts about the dashes, he can calculate how many dashes she should have counted.

The Problem:

Given the length of a dash, the distance between each dash, and the distance traveled, calculate how many complete dashes the car has traveled over. All of these are given in feet, greater than 0 and less than 500,000. You may assume that the road is perfectly straight and that Ali has his daughter start counting just after the car has passed over a dash (i.e. that dash would not be counted).

The Input:

There will be several data sets. Each one-line data set contains three numbers: the length of a dash, the distance between each dash, and the distance traveled. End of data is signaled by end-of-file.

The Output:

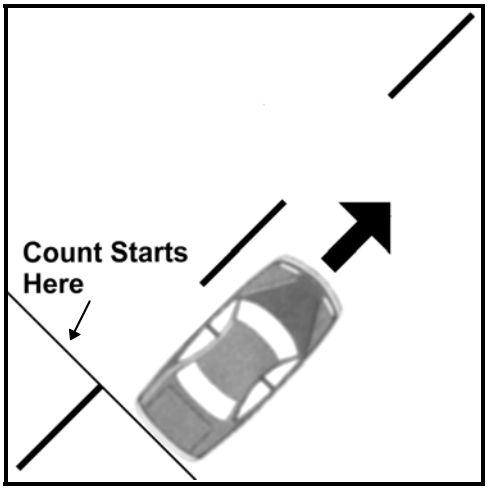
For each input set, you are to print the number of complete dashes the car has traveled over.

Sample Input:

```
4 6 10
4 6 19
4 6 5280
```

Sample Output:

```
1
1
528
```



Invoke the OrACKle

Filename: `ORACKLE`

The CEO of OrACKle computer corporation has decided to make the development of a NetPC the company's top priority. Since OrACKle normally deals with database products and the company's programmers are at a loss to write the software needed to run the computer, they have decided to contract out the work to you. Fortunately for you, you were prepaid and they only need you to write a display driver with a limited, simple syntax. Unfortunately for you, because of the low cost of the system, you were only paid in black and gold M&M's™.

The Problem:

The NetPC display driver is an alphanumeric display 70 columns wide and 24 rows long that provides for omnidirectional output. It also uses periods (".") for blank spaces rather than the space character. Therefore, a clear display will have 24 rows of 70 periods. The NetPC display driver understands five commands (always in upper-case): `ADD`, `VFLIP`, `HFLIP`, `REPAINT` and `CLEAR`.

`ADD` defines text to be placed on the display. This command is followed by two integers, x and y, representing the starting location for the text on the display. Therefore, the NetPC display driver must be able to handle starting locations both on and off the screen. In this case, you should process the command as usual but only those characters that would be written to the display should be shown. Note that (1, 1) is the top-left corner of the display. Following the starting location is a direction for the text. Possible values for the direction are:

Direction	**Meaning**
U	Up
UR	Diagonally up and to the right
R	Right
DR	Diagonally down and to the right
D	Down
DL	Diagonally down and to the left
L	Left
UL	Diagonally up and to the left

Following the direction is the text to be displayed (which will be no longer than 255 characters in length). Any spaces that exist in the input text should be converted to periods for output on the display. In addition, if there are more characters in the text than are needed for the display, then ignore the extra characters. In other words, do not wrap. Furthermore, `ADD` commands should be processed in the order they occur in the input file (in the case where two or more `ADD` commands write to the same display location, the last one takes priority). Note that the command, starting location, direction and text will be separated by a single space.

`VFLIP` flips the output about a horizontal line through the middle of the display. The text at the top of the display moves to the bottom and vice-versa.

`HFLIP` similarly flips the output about a vertical line. The text at the left of the display moves to the right and vice-versa.

`REPAINT` actually outputs the text written to the driver to the display itself (or in the case of this problem, to the screen).

`CLEAR` is used to clear the display back to all periods (as described above).

In this problem, you are asked to implement these five commands of the NetPC display driver.

The Input:

Each line of the input file will contain a command (`ADD`, `VFLIP`, `HFLIP`, `REPAINT` or `CLEAR`) beginning in column 1. The `ADD` command will be followed by additional parameters as described above.

The Output:

For each `REPAINT` command, output the state of the display. Leave a blank line between the outputs for each `REPAINT` command.

(Sample Input and Output are on the following page)

Sample Input:

```
ADD 2 1 R NetPC coming soon!
HFLIP
ADD 2 1 D from OrACKle Corporation
ADD 1 6 R Welcome to the UCF High School Programming Tournament
REPAINT
```

Sample Output:

```
.f.............................................!noos.gnimoc.CPteN.
.r..............................................................
.o..............................................................
.m..............................................................
..............................................................
Welcome.to.the.UCF.High.School.Programming.Tournament.........
.r..............................................................
.A..............................................................
.C..............................................................
.K..............................................................
.l..............................................................
.e..............................................................
..............................................................
.C..............................................................
.o..............................................................
.r..............................................................
.p..............................................................
.o..............................................................
.r..............................................................
.a..............................................................
.t..............................................................
.i..............................................................
.o..............................................................
.n..............................................................
```

Mike Likes to Climb Trees

Filename: TREES

Mike likes to climb trees. Whenever he finds himself in a new place, he investigates the various trees in order to climb one. Unfortunately, this causes him to sometimes climb trees in illegal locations such as in city parks. In this problem, you are asked to find out how high Mike has climbed in order to help the police search for him.

The Problem:

Given a series of values that represent the heights (above the ground) of tree branches that Mike used in his climb (in the order he used them), report the maximum height that he achieved so that the police can focus their search. Note that Mike may not always end his climb at the highest point.

The Input:

The first line of the input contains the number of data sets *n* (a positive integer) in the input file. The data sets will be on the next *2n* lines. The first line of each data set contains a single positive integer that is the number of branches *b* in Mike's climb. The second line contains *b* positive integers which are the heights of the branches (in the order Mike followed them) in his climb.

The Output:

For each input set, you are to print the message "`Look at or below height `" followed by the maximum height that Mike reached. Follow the format of the Sample Output below.

Sample Input:

```
2
10
2 4 7 10 15 20 22 23 26 30
9
4 5 6 7 8 9 10 13 13
```

Sample Output:

```
Look at or below height 30
Look at or below height 13
```

Psychic Gypsy Test

Filename: `GYPSY`

The princess of a small kingdom has escaped from her kidnappers and must find a way home. After days of searching the wilderness, she has stumbled into a gypsy caravan. The gypsies say they will help her if she can prove her identity (a princess would be worth a reward...). Knowing that the kingdom's royalty have strong psychic powers, the gypsies have prepared a "guessing game" to find out if the princess can foresee the future.

The princess and the eldest gypsy fortune-teller sit at a table with a bowl of M&M's™ on it. They take turns removing between 1 and 5 M&M's™ inclusive. The person who has to remove the last M&M™ is the loser. The princess does have psychic powers, and she knows the quantity of M&M's™ in the bowl and all the moves the gypsy is going to make. She still has to win the game using this information.

The Problem:

Your program must play the part of the princess and find a way to win the game, given the number of M&M's™ the gypsy will take each turn. Note that the princess goes first.

The Input:

The input file will have the data for several "games". Every line of the file represents a different game. Each line consists of a series of integers. The first integer on each line indicates the number of M&M's™ in the bowl (between 1 and 500 inclusive) at the beginning of the game. The rest of the integers on the line represent the gypsy's moves; each of these integers will be between 1 and 5 inclusive. Note that these are the moves the gypsy would *like* to make. If there aren't enough M&M's™ left to take as many as the gypsy would like, the gypsy takes all the M&M's™ remaining.

No line of the input will have more than 255 characters. You may assume that there will be at least enough moves on the input line to determine the game's winner.

The Output:

The output for each game will take one line. The line should begin with "`Game #`*n*`: `", where *n* indicates the game number (the first game is number 1). If the princess can win the game, print out the moves of one such winning game, in order. On the princess's turn, print "`P:`*n*", where *n* is the number of M&M's™ the princess takes. On the gypsy's turn, print "`G:`*n*", where *n* is the number of M&M's™ the gypsy takes. If the princess cannot win the game, do not print the moves for the game. Instead, print the message "`The princess cannot win!`".

(Sample Input and Output are on the following page)

Sample Input:

```
20 5 3 2 1 4
7 1 1 1 5
```

Sample Output:

```
Game #1: P:5 G:5 P:5 G:3 P:1 G:1
Game #2: P:3 G:1 P:2 G:1
```

Smash the Copy Machine!

Filename: SMASH

Every once in a while, copy machines do strange things. For example, there was once a copy machine that miscollated pages. The original had 3 pages and the machine was set to make 5 copies. The machine normally copies one page at a time (the first page is copied first) and puts the copies on different stacks for collating. This machine made 5 copies (one page at a time), but instead of putting the copies on 5 stacks, it put them on 7 stacks! That is, the first page was copied 5 times and put on the first 5 stacks. Then, copying of the second page started, but instead of putting the first copy of the second page on stack 1, the machine put the first copy of the second page on stack 6, the second copy of the page on stack 7, and then the third copy of the page on stack 1 (in other words, it wrapped around two steps late).

The Problem:

Given the number of pages in the original document to be copied, the number of copies made, and the number of stacks for collating, print the contents of each collating stack.

The Input:

The input consists of multiple data sets. Each data set contains three integers (all between 1 and 10 inclusive) on a single line each separated by spaces. The first integer represents the number of pages in the original, the second integer is the number of copies made, and the third integer is the number of stacks for collating. End of data is indicated by end-of-file.

The Output:

For each data set, print its data set number (beginning with data set 1). Then, on the following lines, print the contents (page numbers) of each stack. Leave a blank line after the output for each data set. Follow the format illustrated in the Sample Output.

(Sample Input and Output are on the following page)

Sample Input:

```
3 5 7
3 5 8
```

Sample Output:

```
Data set 1
   Stack 1: 1 2 3
   Stack 2: 1 2
   Stack 3: 1 2
   Stack 4: 1 3
   Stack 5: 1 3
   Stack 6: 2 3
   Stack 7: 2 3

Data set 2
   Stack 1: 1 2
   Stack 2: 1 2
   Stack 3: 1 3
   Stack 4: 1 3
   Stack 5: 1 3
   Stack 6: 2 3
   Stack 7: 2 3
   Stack 8: 2
```

Warcraft

Filename: WARCRAFT

What do you think the judges are doing while you are slaving away at your solutions? Our investigative reporting team has found that they have modified the submission program to generate random responses instead of judging your programs! Why? They are too busy playing a game called Warcraft. Maybe if you can get the judges' attention by writing a program that gives them strategy tips on playing, they will send you a 'correct' response.

The Problem:

The players are separated by terrain made up of grass, dirt, water, trees and mountains. To attack the opponent, a player can build ogres, shipyards, goblin sappers, or dragons. Each one of these requires successively more resources to build; that is, ogres are cheap but dragons are very expensive. The types of terrains that each unit allows travel over are:

- Ogres: Can only cross grass or dirt.
- Shipyards: Allow you to cross water, in addition to grass and dirt.
- Sappers: Allow you to cross mountains and trees, in addition to grass and dirt.
- Dragons: Flying, can cross anything.

In this problem, you are to help the judges by determining the single, cheapest resource that can be used and still have a path between Player 1's town and Player 2's within the terrain constraints above. For example, if Player 1's town is completely separated from Player 2's town by water, then the judges need to build shipyards (assuming there are no mountains or trees in the way; otherwise, they would need to build dragons because they have to cross both water, and mountains or trees).

The Input:

You will be given a section of the map to analyze and give advice about. For each data set, the first line will consist of two integers *n* and *m* indicating the width and height of the map region. Maps will be no larger than 64x64, and will contain at least two tiles. Terrain is represented as follows:

Grass	.
Dirt	:
Mountain	#
Water	~
Tree	T
Player 1's Town	1
Player 2's Town	2

No other characters will appear. There will always be exactly two players and each player will have exactly one town. End of data is indicated by end of file.

The Output:

For each data set, you are to analyze the map and give player 1 advice. Note that diagonal movement is not allowed. For each map, print:

`Map #n: <advice>`

Replace `n` with the data set number (beginning with 1). Replace `<advice>` with advice about how to reach the opponent's town. From player 1's town, if player 2's town is reachable following only dirt and/or grass, then the advice is "`Build Ogres.`" If only water must be crossed, print "`Build a Shipyard.`" If only mountains and/or trees must be overcome, print "`Build Sappers.`" Finally, if the way is blocked by both water, and mountains or trees, print "`Build Dragons.`" If more than one answer applies, choose the one with the lowest resource cost.

Sample Input:

```
20 10
.TT.2..........:##
#
TT...TTTTTT....#:##
#
TTTTTTT:::.~~.::#:~
~
.....:::~~~~~~~~~~~
~
~~~~~~~~~~~~~~~~~~~
~
~~~~~:::::::::::::.
.
#:.~~::.TTTTTTTTTTT
.
:#..TT..TT..TTTTTTT
T
#..TTT.............
.
#.................1
.
5 5
.#.2.
:##..
..###
.....
1..TT
7 7
#..2..#
..TTT..
TT.::TT
~~~~~~~
##.:.##
..###.T
T..1..T
```

Map #1 contains a river. There are trees and mountains, too, but there are ways around them. **Build a Shipyard.**

Sample Output:

```
Map #1: Build a Shipyard.
Map #2: Build Sappers.
Map #3: Build Dragons.
```

Proper Penguin Speech

Filename: FEEP

Biologists from the Flightless Nesting Organization and Research Division published the results of their research on the effect of jet engine noise on the growth of penguins. Their findings suggest that penguins raised in the presence of jet engine noise are 23 times more likely to have problems correctly pronouncing "feep," the most important word in the penguin language. To prevent this tragic problem, the International Penguin Safety Department of the United Nations has banned all flights over penguin nesting sites.

The Problem:

It is your job to generate a list of planes which would cross over these nesting areas from input describing the nesting sites and the planned paths of planes. The fate of the entire penguin language is in your hands!

The Input:

The first line of the input file will be a number n indicating the number of nesting sites. Directly following will be n lines describing the n nesting sites. Each nesting site will be listed as four integers, (x_1, y_1) and (x_2, y_2), separated by a single space which are opposite corners of a rectangle describing the site. All nesting sites will have an area greater than 0. The line immediately following the nesting sites will contain a number p which is the number of planes to be tracked. The next p lines will contain four integers, (x_s, y_s) and (x_d, y_d), separated by a single space where (x_s, y_s) is the plane's starting position and (x_d, y_d) is the plane's destination. All x and y values will be integers between -10000 and 10000 inclusive. The number of nesting sites will not exceed 20 and the number of planes will not exceed 500.

The Output:

Planes which fly over penguin nesting sites should be listed, one per line, in the form:

```
Plane #a would fly over nesting site #b.
```

where a is the plane's number in the input file and b is the nesting site's number in the input file. Output should be listed sorted first by plane number and second by nesting site number in the case of multiple sites crossings for a single plane. Planes not actually flying through the sites but which fly on the border should be included in the offending plane list.

(Sample Input and Output are on the following page)

Sample Input:

```
2
-10 -10 10 10
15 -5 17 10
3
-20 -20 -5 -5
-20 0 20 5
30 71 18 24
```

Sample Output:

```
Plane #1 would fly over nesting site #1.
Plane #2 would fly over nesting site #1.
Plane #2 would fly over nesting site #2.
```

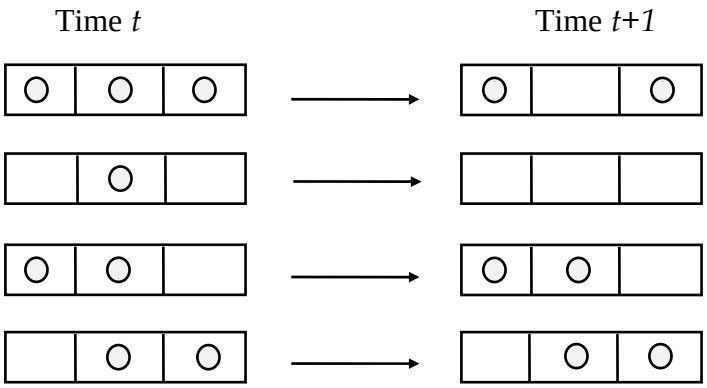
The Cycle of Life

Filename: `LIFE`

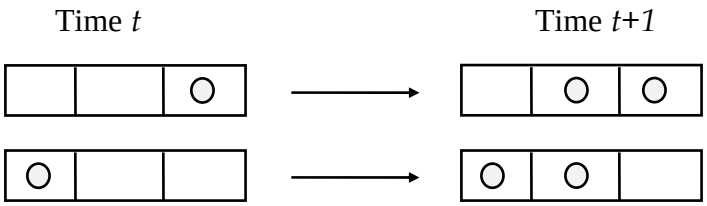
This is a one dimensional version of John Conway's "Game of Life". There are n slots (n is between 1 and 14 inclusive). Each slot may contain one living cell. There will be an initial configuration of cells at time $t = 0$. The colony of cells evolves according to the following rules.

- No live cell in the first or last slot will die for any reason. Also, no new cell will evolve at the first or last cell.

- If there is a cell in a given slot and if it has two neighbors, it dies at the next moment because of over population. It also dies if it has no neighbors, because of starvation. If a cell has only one neighbor, it continues to live at the next moment. An illustration for n=3 is given below.

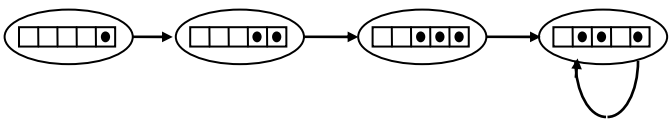


- If there is no cell in a given slot, a new cell evolves at the next moment if and only if it has only one cell in its neighborhood. An illustration for n=3 is given below.



The Problem:

It can be observed that the same pattern of colonies evolve with time. For instance:



These colonies have an evolution period of 1 time unit since the same patterns remain after the fourth step. Note that the cycle need not include the first few states. Some colonies die out after some time (they have no living cells). In this problem, you are to compute the period of evolution (if it exists) given an initial configuration.

The Input:

You will be given several strings of 0's and 1's, each representing a colony at time *t*=0 and beginning in column 1 of a new line. A 1 indicates the presence of a cell in that slot and a '0', its absence. End of data is indicated by the end of file.

The Output:

You have to compute and report the period of evolution if the population does not die out. In this case, output the message "`<c> has a period of <n> units.`" where `<c>` is the colony from the input and `<n>` is the period of evolution that you found. If the colony dies out, output the message "`<c> dies out.`" again where `<c>` is the colony from the input.

Sample Input:

```
0
1010
11011
01010
```

Sample Output:

```
0 dies out.
1010 has a period of 4 units.
11011 has a period of 1 units.
01010 dies out.
```