# NEW ZEALAND PROGRAMMING CONTEST 1997

## Problem A — Random Numbers — 5 points

IMPORTANT NOTE: An answer to this problem is needed for Problem I (15 points). While this problem has specific values for the parameters A, B, C and S, for the later problem you will have to vary these numbers so your solution should be written generally.

A common source of random numbers is a Congruential Random Number Generator. For this, three numbers A, B and C are chosen, and a seed number S < C. The first random number, S', is

$$S' = (A*S + B) \bmod C,$$

where "mod C" means the remainder after A*S + B is divided by C. The second random number, S", is found by using S' instead of S in the formula. Then a third random number is found by using S" in the formula, and so on. If A, B and C are chosen carefully then C numbers, all between 0 and C-1, will be produced, and there will be no apparent relationship between the numbers.

For example, if A = 5, B = 3 and C = 16, then picking any number less than 16 as the seed, say 4, we get the first random number 5*4 + 3 mod 16 = 7, and the numbers produced, in order, are:

7, 6, 1, 8, 11, 10, 5, 12, 15, 14, 9, 0, 3, 2, 13, 4

Once 4 is produced, the numbers will start to repeat.

There is no input for this problem.

Output, which must be written to standard output (the screen), must be the 1000th number produced when the Congruential Random Number Generator with A = 301, B = 57 and C = 1024 is started with a seed of 587.

Note that if you work out the number by hand and simply write a program which prints this number, you will receive zero points. In this competition, a program is not regarded as a solution unless it is a valid attempt to solve the problem as set (in the opinion of the judges).

EXAMPLE

**Input**

(There's no input)

**Output**

```
421
```

(Actually the number isn't 421 but this is to show that the output, on the screen, must be a single number starting at the left of the screen. Many of the answers to the problems are of this, or a similar, form. Do NOT try to "improve" this output by writing "1000th number = 421" or similar; this will be regarded as a wrong answer).

# Problem B Fees 5 points

A new tertiary institution, Tony's Tertiary Tutoring, has decided to get in on the game of getting paid by both the government and students for giving qualifications. In proper output-oriented managerial style, this institution guarantees that every fee-paying student will graduate with a degree, the level of the degree being, of course, dependent on the fee paid. In this competitive education market they have decided to advertise themselves as having the lowest starting fees, but in order to come out ahead they intend to charge higher fees for future years of the course, so that existing students will have to pay more to stay in the course. Your job is to assist the institution in working out its profit.

The institution plans to keep costs very low, but will have some costs, such as the cost of hiring and maintaining the buildings - this will come to $400,000 per year. To avoid the notoriously tricky problems of hiring and firing academic staff they have decided to not have any; students will be given a copy of the approved syllabus, access to the library and an Internet connection, and expected to teach and test themselves. So there will be costs for computer equipment and Internet user charges, expected to come to $150,000 per year. The only staff cost will be a secretary who will be paid $18,000 per year.

Input will be from the file PROBLEMB.DAT and will consist of various fee scenarios. Each fee scenario will be a single line, and will contain a set of three numbers (positive integers) separated by spaces. The first number will give the basic fee for starting the course, the next a percentage for the amount by which the fee will go up every year (eg if the first year fee is $365 and the increase is 10%, the second year fee will be $401, the third year fee $441 - the exact amount will be truncated to whole dollars and the next year's fees computed on the truncated amount), and the third will give the length of the course in years. The initial fee will never be more than $10,000, the percentage increase will be less than 100%, and the number of years will be no more than 6. Input will be terminated by a line containing three zeros.

Output, which must be written to standard output (the screen), will be a single number for each line of input, giving the profit expected if there are exactly 100 students in each year of the course ("steady-state" situation). This could be a negative number, in which case it must be preceded by a - sign.

EXAMPLE

**Input**

```
1000 5 6
700 10 4
0 0 0
```

**Output**

```
111700
-243200
```

# Problem C Report 5 points

Computers can produce and store vast quantities of numeric data easily, but this isn't much use for human beings. Often just the simple process of formatting data in a user-friendly way can make it more understandable for people. For this problem, input concerning sales in a firm is given, and you must format this so that it will be apparent, to anyone looking at the screen, what the information is.

The firm concerned has four sales representatives with code numbers as follows:
Jane, 2341
Chu-ling, 1419
Peter, 6132
Moana, 3498
It sells five different items with code numbers as follows:
Blankets, 451
Teaspoons, 363
Mercedes, 246
Sheep, 865
Cabbages, 449

Input will be from a file called PROBLEMC.DAT and will be lines each of contains three positive integers separated by one or more spaces. The first number is the code of a sales representative, the second number is the code of the item sold, and the third number is number of this item sold, which will always be less than 10,000. The input will be terminated by a line containing three 0s.

Output, which must be written to standard output (the screen), must be one line for each line in the input. This must give firstly the name of the sales representative, then the name of the item sold, then the number sold. The names must be left-justified, the sales representative name against the left margin and the item sold name on column 20 (ie there must be exactly 19 character positions before every item name). The number sold must be right-justified against column 40 (ie if 1 item is sold there must be 39 character positions before the 1, if 10 are sold there must be 38 character positions before the 1). If the sales representative or item number is not on the list given above you must use **???** as the name.

EXAMPLE

**Input**

```
6132  363    2
1419   451    1
3498  865     15
2341    449   241
1418   865    15
2341  452    1397
6132  246    16
0 0 0
```

**Output**

```
Peter                Teaspoons            2
Chu-ling             Blankets             1
Moana                Sheep               15
Jane                 Cabbages           241
**???**              Sheep               15
Jane                 **???**           1397
Peter                Mercedes            16
```

# Problem D **Balls** 5 points

Suppose you are working for a firm which manufactures balls of various kinds out of a type of rubber. These balls could be hollow or solid. Your job is to find out how many balls of a given type can be made from a certain volume of rubber. To work this out, you will need to know that the volume of a ball with diameter $D$ is $\frac{\pi}{6}D^3$. For the sake of consistency, take $\pi$ to be 3.14159 exactly.

Input will be from a file called PROBLEMD.DAT and will be a series of lines each containing three numbers (decimals). The first number gives the outer diameter of the ball to be manufactured in cm (never less than 1 nor more than 100), the second number gives the inner diameter of the ball (at least .2 cm less than the outer diameter), and the third number gives the volume of rubber available in cubic metres (one cubic metre is a million cubic centimetres). Solid balls have inner diameter equal to 0. There will never be more than 10 cubic metres of rubber. All numbers will be given to at most three decimal places. The input will be terminated by a line containing three 0.

Output, which must be written to standard output (the screen), must be a single integer for each line of input, giving the number of balls which can be made from the amount of rubber given. Any remaining rubber must be not enough to make one ball. Note that the amount of rubber needed for a hollow ball is the difference between the volume of the whole ball (using the outer diameter) and the volume of a ball with diameter equal to the inner diameter given.

EXAMPLE

**Input**

```
10 0 0.001
10 9.8 0.1
0 0 0
```

**Output**

```
1
3247
```

# Problem E **Change** 5 points

For this problem you must write a program which will calculate the change to be given, in normal New Zealand coins. Remember that there are $2, $1, 50c, 20c, 10c and 5c coins.

Input will be from a file PROBLEME.DAT, and will consist of lines giving firstly the price of an object, followed by the amount of money paid for the object. The second number will always be greater than or equal to the first number, but no more than 10, and the numbers will be separated by one or more space characters. The numbers will either be positive integers (whole dollars) or positive decimals with two places (dollars and cents). The number of cents will always be a multiple of 5. The input will be terminated by a line consisting of two zeros.

Output, which must be written to standard output (the screen), must be six numbers on a line (separated by single spaces) for each line of input. The first number of the six must be the number of $2 coins needed in the change, the second the number of $1 coins, and so on, so the sixth gives the number of 5 cent coins. Each number must be the maximum possible, ie X = (money paid) - $2*(number of $2 coins) must be less than $2, X - $1*(number of $1 coins) must be less than $1, and so on.

EXAMPLE

**Input**

```
0.05 10
5 5
1.85 2
0 0
```

**Output**

```
4 1 1 2 0 1
0 0 0 0 0 0
0 0 0 0 1 1
```

# Problem G **Shuffling** 15 points

For this problem you must write a program which will experiment with various different ways of shuffling cards. Each card is represented by two characters, the first being the suit, one of H, D, C, S, and the second being the rank, one of A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2. For each shuffling experiment you must assume that the 52 different cards in a pack are initially arranged first by suit order then by rank order (the order of the symbols is as given), that is, the pack initially looks like:

HA, HK, HQ, HJ, HT, H9, H8, H7, H6, H5, H4, H3, H2
DA, DK, DQ, DJ, DT, D9, D8, D7, D6, D5, D4, D3, D2
CA, CK, CQ, CJ, CT, C9, C8, C7, C6, C5, C4, C3, C2
SA, SK, SQ, SJ, ST, S9, S8, S7, S6, S5, S4, S3, S2

The shuffling methods which you are to experiment with are defined by a single number, N. The card in position 1 (the first card, HA) is to be moved after the card in position N, then the new first card (HK if N > 1) is to be moved after the card in position 2N, then the new first card to position 3N, and so on until the last card (S2) has reached the front position and has been moved from there. If a position greater than 52 is needed, assume the count starts again from 1, ie position 53 is the same as position 1. Some cards will be moved several times so the S2 card will be moved to a position much greater than 52N. You may assume the S2 card will always be moved.

For example, if N = 3, then the HA will be moved to after the HQ, then the HK to after the H9, then the HQ to after the H6, so the first line will look like:

HA, HJ, HT, H9, HK, H8, H7, H6, HQ, H5, H4, H3, H2, DA, DK, DQ

At this point the HA will be moved again, to after the H3., Eventually the H2 will be moved to after the S3 (position 51), then the DA to after the DK which will be at position "54" at that point. If a card needs to be moved after itself (eg to position "105"), it does not move at all, but will move at the next step (eg now to position "108").

Input will be from a file PROBLEMG.DAT, and will consist of lines giving the number N for a shuffling experiment. The input will be terminated by a line containing a single zero.

Output, which must be written to standard output (the screen), must be the set of shuffled cards, written in four lines as above, when shuffled using this value of N. The four lines must be followed by a blank line (ie an empty line).

EXAMPLE:

**Input**

```
3
52
0
```

**Output**

```
C4, H5, DJ, HT, C7, D7, D9, S9, C6, SJ, D2, C3, DK
H3, S5, CK, H2, S7, DQ, DT, H8, D6, S4, H9, C8, D5
SA, HQ, S2, CJ, S8, C5, H6, D3, DA, C2, SQ, HJ, S3
CA, D8, CQ, C9, HK, SK, HA, D4, H4, CT, S6, ST, H7

HA, HK, HQ, HJ, HT, H9, H8, H7, H6, H5, H4, H3, H2
DA, DK, DQ, DJ, DT, D9, D8, D7, D6, D5, D4, D3, D2
CA, CK, CQ, CJ, CT, C9, C8, C7, C6, C5, C4, C3, C2
SA, SK, SQ, SJ, ST, S9, S8, S7, S6, S5, S4, S3, S2
```

# Problem H **Enrolling** 15 points

Tony's Tertiary Tutoring wants you to write a program so students can enrol in their courses electronically, without having to fill in any forms. They have a database which gives the numbers of the courses they offer (3-digit numbers), together with the prerequisite courses (if any). A given course, say 345, has another course, say 245, as a prerequisite, if a student must pass 245 before they can enrol in course 345. They have another database which holds the ID numbers (a 5-digit number) of every student who has inquired about enrolling or who has passed one of their courses, with the courses passed (if any). A student will type in their ID number and the numbers of the courses they wish to take this semester, and your program must check that the ID number and the course numbers are valid, and that the student has passed the required prerequisites for the courses they have selected.

Input will be from a file PROBLEMH.DAT, and will consist of three parts. Each part consists of lines with varying numbers of numbers on them, which are separated by one or more space characters, and there is a 0 on the end of each line, which is **not** part of the data but merely a terminator. The first part is a database of courses offered, and consists of lines which start with a valid course number (a 3-digit number with no leading zeroes). After this, on the same line, will be the prerequisite course numbers (0 to 5 of them) for that course, in increasing order; note that some of them may not be listed as valid course numbers (they represent courses not taught this semester). The valid courses are listed in increasing order and the database is terminated by a line containing a single zero. The second part is the database of students and contains lines which start with an ID number (a 5-digit number with no leading zeroes), then the courses the student with that ID number has already passed (0 to 15 of them, in increasing order). These may or may not be currently valid course numbers. The ID numbers are listed in increasing order and the database is terminated by a line containing a single zero. Being realistic, the institution expects no more than 1000 students to ever enrol; ie no more than 1000 IDs will exist. The third part is the list of enrolment requests, in no particular order. Each request is a line which starts with an ID number (a 5-digit number, anyway) and is followed by the numbers of the courses this student requests for this semester (at least one and at most five course numbers requested; the order of the course numbers will be random). Each line will be properly terminated by a zero and all course numbers on the line will be three digits or fewer. The input is terminated by a line containing a single zero.

Output, which must be written to standard output (the screen), must be one line for each enrolment request. This must be one of the following:
"Invalid ID" if the ID is not in the database of students
"Invalid course" if one of the requested courses is not in the database of courses
"Not passed prerequisites" if the student has not passed a course listed as a prerequisite for one of the requested courses.
"Invalid enrolment" if the student has already passed one of the requested courses.
"Approved" if none of the above apply.
If more than one of these messages applies, the message printed must refer to the first thing which is wrong (ID number or course number) when the line is processed from left to right. Repeated course numbers on the same request line can be ignored.

EXAMPLE

**Input**

```
145 0
245 145 0
345 120 245 0
0
11111 110 120 145 245 0
22222 120 145 254 0
33333 0
0
11111 345 345 0
44444 345 0
22222 245 345 110 0
0
```

**Output**

```
Approved
Invalid ID
Not passed prerequisites
```

# Problem I **Good Randomness** 15 points

In Problem A a Congruential Random Number Generator was described, and a particular example given. Not all of the possibilities for the numbers A, B, C and S (the initial seed) lead to good random numbers. Badly chosen parameters can give very short cycles - for example, if A = 37, B = 54, C = 73, and S = 35 then the first number produced is also 35:

(37*35 + 54) mod 73 = 35

and so the "random" numbers produced are all the same! For this problem you have to test various Congruential Random Number Generators and find the how many different "random" numbers are produced, in the long term. Note that once a number is repeated, the random number generator must cycle through the same numbers. The repeated number could be the original seed, or it could be a later number. For example, if A = 9, B = 31, C = 72, S = 5, then the numbers produced are:

4, 67, 58, 49, 40, 31, 22, 13, 4

and then the numbers repeat (67, 58 etc), so the seed number, 5, is not repeated, but the first number produced, 4, is repeated. In this case there are 8 different random numbers produced in the long term, although initially 9 are produced.

Input will be from a file PROBLEMI.DAT, and will consist of lines containing four integers, separated by one or more spaces, which give the A, B, C and initial seed S for a Congruential Random Number Generator. None of the numbers will be greater than 1024, and A, B and S will all be less than C. The input will be terminated by a line containing four zeroes.

Output, which must be written to standard output (the screen), must be one line for each line of input, and must give the number of different numbers produced by the Congruential Random Number Generator in the repeating cycle of numbers which are produced after the first number is repeated.

EXAMPLE

**Input**

```
37 54 73 35
301 57 1024 587
9 31 72 5
0 0 0 0
```

**Output**

```
1
1024
8
```

# Problem J **Anagram Sort** 15 points

Two words are anagrams of each other if they contain the same letters, but in a different order, for example the words "pots", "stop", "opts" and "spot". One way of finding anagrams in a list of words is to sort the words in the list according to a special order. One word is said to be "anagrammatically less" than another word if, when the letters in the two words are counted, the first word has fewer a's, or the same number of a's but fewer b's, or the same number of a's and b's but fewer c's, and so on. For example, "airplane" is anagrammatically less than "aeroplane" because they both have 2 a's, 0 b's, c's and d's but "airplane" has only one e to "aeroplane"s two e's. Note that the word "z" is anagrammatically less than the word "a"! Two words are "anagrammatically equal" if neither is anagrammatically less than the other - in this case, they will be anagrams of each other. For this problem, you must sort a list of words so that it is in increasing order in the sense that if one word is anagrammatically less than another then it will appear earlier on the sorted list.

Input will be from a file PROBLEMJ.DAT, and will consist of lines each containing a single word, with no space characters. No word will be more than 20 characters and there will be no more than 1200 words The input will be terminated by a line consisting of a single #.

Output, which must be written to standard output (the screen), must be the words in the input arranged in increasing anagrammatic order, as described above, one word per line. Words which are anagrammatically equal must be arranged in increasing alphabetic order.

EXAMPLE

**Input**

```
cat
pots
a
z
stop
tack
act
opts
#
```

```
Output
z
opts
pots
stop
a
act
cat
tack
```

# Problem K Improving email 15 points

One difficulty with email communication is that some people SHOUT AT YOU BY WRITING EVERYTHING IN CAPITALS. others are very lazy and never use the shift key, so they don't use capital letters to start sentences and write i for I(as in "i think this is a good problem").

For this problem you must write a program which will turn bad email text into "normal" mixed-case text. All upper-case letters must be turned to lower case, except that:
(a) The very first letter must be uppercase.
(b) A ".", "!", or "?" followed by one or more spaces is assumed to represent the end of a sentence, so the first letter after the spaces (ignoring non-letters) must be uppercase.
(c) If the letter i appears with a space on each side, or with a space on one side and a non-letter on the other, then it must be uppercase ("I" in this case). A letter at the start of a line is treated as if it has a space before it.
All characters which are not letters should be left unchanged.

Input will be from a file called PROBLEMK.DAT and will be a series of lines containing text to be improved. No line will be more than 60 characters wide. You should assume that every line ends with a space character, even though some lines in the file may not actually end in spaces. The input will be terminated by a line consisting of a single #.

Output, which must be written to standard output (the screen), must be the improved text. Space characters at the end of lines will be ignored.

EXAMPLE

**Input**

```
One difficulty with email communication is that some
people SHOUT AT YOU BY WRITING EVERYTHING IN CAPITALS.
others are very lazy and never use the shift key, so they
don't use capital letters to start sentences and write i
for I(as in "i think this is a good problem"). #now@@junk
#
```

**Output**

```
One difficulty with email communication is that some
people shout at you by writing everything in capitals.
Others are very lazy and never use the shift key, so they
don't use capital letters to start sentences and write I
for I(as in "I think this is a good problem"). #Now@@junk
```

# Problem M Volleyball 50 points

The rules of Volleyball are quite simple. The game is played between two teams using a round ball on a flat area, mostly inside a rectangular region called the court. There is a net stretched across the court, and the teams play in the two halves of the court defined by the net. The ball is not supposed to touch the ground during the game. One team starts by "serving" - hitting the ball with a fist. The ball is then supposed to pass above the net inside the court into the other half, and the other team must try to get it back over the net again, by hitting it (catching is not allowed). Up to three hits are allowed before it must cross the net. Players are allowed to be inside or outside the court when they hit the ball. As long as the teams successively return it across the net inside the court, the ball is said to be "in play". It comes out of play when one of the following six events happens:
(a) More than 3 hits are made on the ball before it crosses the net (or more than one hit during a serve).
(b) The ball passes below the net (note that the ball is allowed to hit the net)
(c) The ball passes above the part of the net outside the court
(d) The ball touches the ground and it has not crossed the net since the last person hit it.
(e) The ball touches the ground outside the court and no-one has hit it since it crossed the net.
(f) The ball touches the ground inside the court and no-one has hit it since it crossed the net.
If any of (a) to (e) occurs, then the team which last hit the ball have "lost the play". If (f) occurs, then the team which last hit the ball have "won the play".

A team wins a point if they serve and win the play (ie the other team loses the play). Furthermore, they serve the ball for the next play. If they serve and lose the play, then no points are won or lost by either team, but the other team has the next serve. The winner of the game is the first team to win 15 points, as long as the other team has 13 or less. If, when one team wins 15 points, the other team has 14 points, then the game continues until one team wins two points more than the other team.

High-tech has now reached the game of Volleyball! By putting electronic devices in the ball, around the court and on the players, the following events can be automatically recorded:
H: the ball is hit by a person
G: the ball touches the ground
N: the ball passes over the net
U: the ball passes under the net
O: the ball passes over the court boundary from the inside to the outside
I: the ball passes over the court boundary from the outside to the inside
In addition, a human umpire signals S when a team serves - after the S, there will be an H as the server hits the ball. "Fouls" such as a serve from the wrong place, a player stepping on the wrong side of the court, or catching a ball instead of hitting it, are also detected by the umpire and recorded as Y if the fouling team is the serving team, and Z if the fouling team is the other team. A foul results in the ball coming out of play, and the fouling team have "lost the play".

For this question you will be given the names of two teams, then a sequence of letters indicating the events in a game of volleyball (H, G, N, U, O, I, S, Y, Z). You must work out the winning team, and the final score (you may assume each example game does finish). After the ball passes out of play, there may be some events such as ball hits, passing over or under the net, fouls, etc, as the ball is passed back to the team to serve again (or the umpire accidentally hits buttons) - you must ignore these events. There may also be input, includes serves, after a game is finished. The input will, in fact, closely resemble a random stream of letters (but with some letters occurring more frequently than others, and no invalid letters such as an S when the ball is still in play).

Input will be from the file PROBLEMM.DAT, and will consist of a series of games. Each game will be started by a line containing the names of the two teams, separated by a space. Each team name is more than one letter long and less than 20 letters, with no embedded spaces. The team named first is the team which starts serving. After this will be a set of lines which contain the letters identifying the events (H, G, O, U, B, S, Y, Z). The letters will be separated by spaces and each line will be less than 70 characters long. The sequence of letters will describe a complete game, together with extra letters after the game which must be ignored. The input will be terminated by a line consisting of a single #. No line will have leading space characters.

Output, which must be written to the screen, is one line for each game in the input, giving the name of the wining team, then a space, then the winning team's score, then a space, then the losing team's score.
EXAMPLE
See over page

EXAMPLE

**Input**

```
Blues Reds
S H N H H N O G H I N S H N H U G H U H S H N G O G H I N S H N H H G
H N S H N H H H N H H H N H H H H N S H N H O H H I N O H I N O H I G
H N S H Z U H N S H N H H N O G H I N S H N H U G H U H S H N G O G H
I N S H N H H G H N S H N H H H N H H H N H H H H N S H N H O H I H N
O H I N O H G H I N S H Y N S H H N S H N G H N H S H N O H I H N H O
H H I N H O H I H N H H N H G U H H N Y H N S H G S H N G H U S H N G
Blacks Whites
S H Y S H N H H N O G H I N S H N H U G H U H S H N G O G H I N S H N
H H G H N S H N H H H N H H H N H N H H H H N S H N H O H H I N O H I N O
H I G H N S H Z U H N S H N H H N O G H I N S H N H U G H U H S H N G
O G H I N S H N H H G H N S H N H H H N H H H N H H H H N S H N H O H
I H N O H I N O H G H I N S H Y N S H H N S H N G H N H S H N O H I H
N H O H H I N H O H I H N H H N H G U H H N Y H N S H G S H N G H U S
H N G
#
```

**Output**

```
Blues 15 0
Whites 15 0
```

# Problem N **Encryption** 50 points

A Playfair Cipher is a type of encryption which conceals messages by substituting pairs of letters in a message with pairs of letters obtained from a table. The table used is a 5x5 square which contains every English letter except Z. It is usually constructed by taking a codeword which contains no repeated letters, such as "ENCRYPT", and writing this in the first few squares, then filling in the table by writing in the remaining letters in the alphabet; in this case:

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

To encrypt a phrase (assumed to be entirely in uppercase with no punctuation and with single spaces separating the words), all repeated letters are replaced by single letters, all Z's changed to S, and all blanks replaced by J (except that if this would make two J's in a row, the blank is replaced by Q, or if this makes two Q's in a row then by X). Then, starting from the left, each pair of letters is located in the table. There are two cases:
(i) The pair of letters are on the corners of a rectangle (for example, AW or BL). The pair of letters on the other two corners of the rectangle is substituted for this pair, preserving the left-to-right or right-to-left order of the letters (so VB is substituted for AW and OT for BL).
(ii) The pair of letters are in the same column or same row (for example, CH, DY, FI or US). The matching pair of letters in the next column or next row is substituted for this pair, wrapping around to the first column or row from the fifth (so RI is substituted for CH, PE for DY, KO for FI and NE for US).
All pairs of letters are substituted - if there are an odd number of letters an extra J (or Q if this makes two J's) is tacked on the end of the modified message.

So the phrase "ZORRO SAYS KILL IRAQ JACK" is changed to "SOROJSAYSJKILJIRAQXJACKJ" then encrypted to "KWYQXFCDFXFOGQJYMDSFBRFQ". The encrypted phrase is called a "cipher". Getting the original (changed) phrase from a cipher is called "decrypting", and is easily done by reversing the specifications for encrypting (so VB decrypts to AW, PE to DY and NE to US).

Input will be from a file PROBLEMN.DAT, and will consist of sets of encryption/decryption problems. Each line in the input will be 70 characters or fewer in length and will consist entirely of uppercase letters or spaces. The first line in each set will begin with a C followed immediately (no space) by the codeword for the set. This will be 10 characters or less and will contain no repeated letters. After the codeword line will be a set of lines which either begin with an E (for encryption) or a D (for decryption). These initial letters will be followed immediately by the phrase to encrypt (for E) or the cipher to decrypt (for D). A phrase to encrypt will contain single spaces between words and no trailing spaces, but the words may have Z's and repeated letters. The cipher to decrypt will not contain any spaces, Z's or repeated letters. The input will be terminated by a line consisting of a single #.

Output, which must be written to the screen, is one line for each line in the input which starts with an E or D. Output for E lines must be the Playfair encryption (using the codeword given at the start of the set of E and D lines) of the phrase after the E, after double letters, Z's and spaces are eliminated. Output for D lines must be the decryption of the cipher after the D, ie the (modified) phrase encrypting to the cipher.

EXAMPLE
**Input**

```
CENCRYPT
EZORRO SAYS KILL IRAQ JACK
DKWYQXFCDFXFOGQJYMDSFBRFQ
CA
EEEEEEEEEE
#
```

**Output**

```
KWYQXFCDFXFOGQJYMDSFBRFQ
SOROJSAYSJKILJIRAQXJACKJ
AF
```

BLANK

# Problem Q — Form Letters — 50 points

For this problem you must write a program which will put names and addresses in a form letter.

Input will be from the file PROBLEMQ.DAT, and will contain a form letter of no more than 100 lines with no more than 60 characters on each line. There are no repeated spaces and no leading or trailing spaces on any line. In the letter, the given name of the person being written to is replaced by @, and the surname (family name) by &. Also, there will be a number in the form letter which is replaced by #. The end of the form letter is marked by a line starting with five * symbols (*****). After this come a set of lines each of which contains one given name, one surname, and one number. These are all separated by spaces. Every word in the form letter or name/number lines will be at most 30 characters long. The end of the name/number lines is marked by another line starting with *****.

Output, which must be written to the screen, is one copy of the form letter for each of the name/number lines in the input, followed by a line consisting of five *s (*****). You must replace all the @, # and & characters in the form letter by the appropriate word from each line, as shown below. If this replacement would make the line more than 70 characters long, you must take words off the end of the line and add them to the start of the next line, and so on down the letter if the next line becomes too long. Words should be removed one at a time, taking everything past the final space character in the line as the word to remove, until the line becomes less than 70 characters ie remove the fewest possible number of words). If the next line is entirely empty or is the terminating line starting with "******", then the overflowing words must form a new line. Make sure that the final letter contains no repeated spaces nor leading spaces on any line (trailing spaces in output lines are always ignored).

EXAMPLE

**Input**

```
Dear @ &,

@, have we got a deal for you! For only $#, @, you can be
the proud owner of a brand new whatsit! Yes, only $# just
for you, @!
*****
John Doe 10,000
Victoria University 1,000,000.00
*****
```

**Output**

```
Dear John Doe,

John, have we got a deal for you! For only $10,000, John, you can be
the proud owner of a brand new whatsit! Yes, only $10,000 just
for you, John!
*****
Dear Victoria University,

Victoria, have we got a deal for you! For only $1,000,000.00,
Victoria, you can be the proud owner of a brand new whatsit! Yes,
only $1,000,000.00 just for you, Victoria!
*****
```

# Problem R — Forest reflections — 50 points

Suppose a light source is inside a large forest of reflecting trees which are all the same size and placed at the points of a rectangular grid. Will any light emerge from the forest, or will it be all be reflected and absorbed inside the forest? This is an unsolved problem if the trees are cylindrical, but for perfectly reflecting regularly spaced rectangular trees it is easy to see that some light must emerge.

Consider a forest of square trees, all with side L (an integer), and a coordinate system based on a point in the centre of the forest. Suppose the trees are all aligned with their sides parallel to the coordinate axes, and all have their centres placed on points with coordinates (nD, mD) where D is an integer (D > L) and n, m = 0, ±1, ±2, ±3, ...±N for some maximum size N. Suppose the light source is placed at a point with integer coordinates (P, Q), not inside or on the boundary of a tree, and light rays from the source are perfectly reflected from every side of every tree, but if they exactly hit a corner of a tree or pass through the original light source point then they are absorbed. For this problem you will be given various forest scenarios (L, D, N, P, Q), then various light ray paths (integer coordinates of a point the light ray is beamed at). You must find out whether the light ray will escape from the forest or be absorbed, and you must give the number of reflections the ray undergoes before it is absorbed or reflected. Note that since everything is phrased in terms of integers, the light ray paths are known exactly and it makes sense to talk about a ray hitting a corner or passing through the original point exactly.

Input will be from a file PROBLEMR.DAT and will consist of lines describing forest scenarios each followed by lines giving light ray directions. The forest scenario lines will contain five positive integers giving L, D, N, P and Q. L and D will be less than 20, N will be less than 1000, at least one of P and Q will be bigger than L/2 and both will be less than D - L/2. These will be followed by lines containing two integers, defining points the light ray from the source at (P, Q) is aimed at. The points could be inside a tree or outside the forest and could be positive or negative. The list of points for each scenario will be terminated by a line containing two 0s. The input will be terminated by a line containing five 0s.

Output, which must be written to the screen, is one line for each point in the input, giving first the letter A or E describing whether the ray is absorbed or escapes from the forest, then a single space character, then the number of reflections the light ray undergoes before it is absorbed or escapes.

EXAMPLE

**Input**

```
2 5 1 2 2
5 5
7 2
7 0
12 -7
0 0
3 7 1000 1 4
0 5
0 0
0 0 0 0 0
```

**Output**

```
A 0
E 0
E 1
A 2
A 4
```

# Problem T — Getting there is half the fun — 150 points

In most countries nowadays, flying between main centres involves no more than being at the airport soon enough to check in and ensuring that there is a car waiting at the other end to get you to your hotel. However the situation is very different if you are flying between smaller cities - typically most of the legs are fairly short, there are several different airlines to choose from and not all of them fly all the legs you need to get from here to there. To further complicate the issue, most airlines offer discounts for continuous travel with them rather than switching between airlines at each stop (it is more convenient for you too, but that is a separate issue). Thus one can be faced with a bewildering array of choices, all involving different proportions of convenience and price.

In an attempt to help bewildered customers, ASS (Airline Support Systems) wants a system that will determine the cheapest ticket between given cities. If there are several cheapest tickets, then they wish to offer the most convenient. Write a program that will perform that service.

Input will be from a file called PROBLEMT.DAT and will consist of two parts. The first part will describe the airlines' pricing structure and the routes they fly, while the second will define a series of journeys for which you have to find tickets. The first part of the file will start with a series of five (5) lines, each line describing the pricing policy of one of the five airlines involved. Each line will consist of the 5 integer numbers giving the cumulative percentage discounts given for consecutive legs. Thus (assuming for now that all legs cost $100) an input of 2 3 5 7 10 means that one leg will cost $100, two legs with this airline will cost 2% less than the total cost, ie $196, three will cost 3% less than the total cost, ie $291, four will similarly cost $280, five will cost $465 and six will cost $540. In the unlikely event that one needs more than 6 legs to arrive at one's destination, the final discount will apply, thus seven legs would cost $630, eight legs would cost $720 and so on. The rest of the first part will consist of the routings available and will consist of a series of lines, each line describing a single possible leg, and consisting of the two city identifiers (always three uppercase letters - there will be at most 50 cities) separated by a -, and the prices charged by the five airlines. The prices will be integers representing the number of dollars charged by that airline for that leg, and is the same, regardless of direction flown. A 'price' of 0 indicates that the corresponding airline does not fly that route. These costs, and the discounts, are always arranged so that the cost of flying A to B to C is always greater than the cost of flying A to B, no matter how many legs are involved in the total trip. This part of the file will be terminated by the word END on a line by itself. The second part of the file will consist of a series of pairs of city identifiers separated by -, indicating desired journeys. Every city mentioned in this part of the file will have appeared in the first part of the file, however it may not be possible to fly between any given pair. The file will be terminated by a line consisting of a single #.

Output, which must be written to the screen, will consist of one line for each desired journey. Each line will specify the journey and the minimum possible price rounded to the nearest cent, as in the examples below. If it is not possible to fly between the towns requested, print "Impossible" instead of the price.

EXAMPLE

**Input**

```
2 3 5 7 10
1 2 3 4 5
1 3 4 5 8
1 2 3 5   10
2 3 5 7 11
AKL-WEL 100 99 101 0 0
WEL-DUD 150 0 145 144 146
AKL-SYD 300 250 310 302 295
WEL-HKG 682 0 671 666 671
END
AKL-DUD
SYD-HKG
#
```

**Output**

```
AKL-DUD $243.00
SYD-HKG $1011.51
```

# Problem U — Airport Construction — 150 points

In a certain region of the country the fields are laid out in a totally regular grid, each field being square and exactly one hundred acres in area (so each side is about 600m). Farms consist of a number of these square fields, each of which is connected to the other squares in the farm by at least one side. Thus in the diagram below, the left-hand picture shows a valid farm while the other pictures are not valid farms.

```
XXX      XX       X X
 XX      X XX     X X
XX         X      XX
```

There are also rivers, lakes and other non-farmable areas which do not obey the rules for valid farms - for example, the following could be a river:

```
 X   X
  XXX
```

The government has decided to build an airport in the region but is unsure how big to build it. One thing is certain, however; they want to buy land off as few farmers as possible, and they must leave only valid farms when the land is purchased. The land for the airport must be a rectangle, at least two fields in each dimension, and it must not include any rivers, lakes or other non-farmable areas. For this problem you must work out the "best" place to put an airport, given a map of a large piece of this region. "Best" is defined as the largest rectangle which lies in the map, which does not include any non-farmable areas, which includes parts of as few farms as possible and is such that all remaining fields form valid farms. The "largest rectangle" and "fewest farms" conditions are satisfied if the ratio (area of possible airport site)/(number of fields from different farms in airport site) is as large as possible. If we draw a map of part of the region, denoting fields in the same farm with the same letter and non-farmable regions with #, the best airport site in the following map is the region consisting of the 16 fields in **bold**:

```
ppppprrrr#
xpppprr#a
xppxpr#aa
xxxxx#aaa
```

Input will be from a file PROBLEMU.DAT and will consist of maps for several rectangular parts of this region, each side no more than 20 farms wide. There will be no more than 26 farms in each map, and the fields in each farm will all be denoted by a lower-case letter, a, b, x etc; different farms will have different letters. The non-farmable areas will be denoted by a #. There will always be at least one farm in each map. Each map will be followed by a single blank line (ie a line with no characters before the end-of-line character). The input will be terminated by a line consisting of a single #.

Output, which must be written to the screen, will consist of one line for each map in the input, giving the "best" airport site in that map. The site will be described by listing the letters corresponding to the farms from which fields are to be purchased, followed by the number of fields which must be purchased from that farm, with single space characters separating the letters and numbers. The letters for the farms must be listed in alphabetic order. If there is more than one "best" site, then the site involving the fewest farms, and the alphabetically least list of farm fields if there is still a choice, must be the one chosen (ie choose the site giving a 4 b 2 rather than b 4 c 2 or a 2 b 4 or any site involving 3 farms).

EXAMPLE

**Input**

```
ppppprrrr#
xpppprr#a
xppxpr#aa
xxxxx#aaa

fedcba
fedcba

#
```

**Output**

```
p 9 x 7
a 2 b 2
```

# Problem V Zero! 150 points

It is quite frequently necessary to find where a function becomes zero. This can be quite difficult in the case of functions which have a very complex description, but this problem is restricted to polynomials, ie functions of the form:

$$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_0$$

You will be given $n$ and the coefficients $a_n, a_{n-1}, a_{n-2}, \ldots, a_0$, and you must find the first positive value of $x$ at which the polynomial becomes zero, ie the first positive zero of the polynomial. Of course this can't be given absolutely accurately; for this problem, all you have to do is to give the number y, accurate to two decimal places, such that the first zero of the polynomial occurs for $x$ with $y < x < y + 0.01$. To avoid problems with rounding error, we will guarantee that the first zero is less than 1000, is not a double root (ie the derivative of the polynomial is not 0 at the first zero), and is never exactly equal to a number which is an integer multiple of 0.01.

Input will be from a file PROBLEMV.DAT and will consist of lines each giving a polynomial. The first number on the line will be $n$, the highest power of $x$, and will be followed by $n + 1$ numbers giving the coefficients in the polynomial, $a_n, a_{n-1}, a_{n-2}, \ldots, a_0$. All the numbers will be separated by spaces. The highest power $n$ will always be less than 8 and each coefficient will be between -1000 and 1000 and will have no more than two decimal places (ie each coefficient will be an integer multiple of 0.01). There will always be a positive zero less than 1000 for the polynomial and the first zero will not be a double root nor an integer multiple of 0.01. The input will be terminated by a line consisting of a single 0.

Output, which must be written to the screen, will consist of one line for each polynomial in the input, giving the positive number $y$ with exactly two decimal places (ie a positive integer multiple of 0.01) such that the first positive zero of the polynomial lies between $y$ and $y + 0.01$.

EXAMPLE

**Input**

```
3 600 -894 330.02 -36.03
3 600 -894 330.02 -36.01
0
```

**Output**

```
1.00
0.24
```

# Problem W **Year 2000 problem** 150 points

Imagine a conversation between a business-oriented programmer and a committed C hacker:

B: We are exposed to tremendous revenue and customer satisfaction danger by the upcoming millennium change! Many legacy systems have two-digit year fields which will revert to 00 on the millennial night, destroying time-order checking. We must upgrade our systems to the declared standard at this point in time!

C: No way, man! You're idiots to have used that *#$@ language COBOL! Convert to C is the way to go! Store your dates as bit fields, not decimals, and avoid overflow problems totally.

Actually the C hacker is wrong - storing dates as bit fields does not avoid overflow problems. For example, an old Burroughs operating system used a count of seconds from the start of the year 1900 as the system date, in a 32-bit variable. Sometime in the year 2036 this will overflow to 0 (Burroughs clearly believed, rightly, that they would be safe from prosecution by that year). In order to generate a number of new standards to which businesses must conform (ie to generate lots of jobs for the programming community), you are being asked to compute the time when various bit-count clocks will overflow. You will be given a base year, the number of bits in the bit-count clock, and the number of increments per second (eg 1000 = 1000 increments per second, ie 1 increment per millisecond). The count always starts from the exact beginning of the base year. You must find the last whole second given correctly by the clock before the count overflows.

Remember that there are 60 seconds in a minute, 60 minutes in an hour, and 24 hours in a day. There are 365 days in a year except for leap years, when there are 366. Leap years are all years divisible by 4 and not 100, except that those divisible by 400 are leap years - thus 1900 was not a leap year, 1904, 1908 ... 1996 were leap years, 2000 will be a leap year, 2100 will not be a leap year, etc. The number of days in each month in a normal year is 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31; in a leap year, the second month has 29 days.

Input will be from a file PROBLEMW.DAT and will consist of lines each giving three positive integers describing a bit-count clock. The first number on the line will be the base year (which will never be less than 1800), the next number will be the number of bits in the count (never more than 100) and the third number the number of increments per second (never more than 1,000,000). All the numbers will be separated by spaces. The input will be terminated by a line consisting of a three 0s.

Output, which must be written to the screen, will consist of one line for each line in the input, giving the last valid second before the clock overflows. This will be described first by a date, which must be of the form Year/Month/Day, where Year will be a number 1800 or greater, Month will be between 1 and 12 and Day will be between 1 and 31. After this will come a single space character then the time field, which must be in the form HH:MM:SS, where HH is a two-digit hour field using a 24-hour clock with value between 00 and 23, MM is a two-digit minute field with value between 00 and 59, and SS is a two-digit second field with value between 00 and 59.

EXAMPLE

**Input**

```
1900 32 1
1996 33 1000
2100 100 1000000
0 0 0
```

**Output**

```
2036/2/7 06:28:15
1996/4/9 10:05:34
40170248388637016/5/17 05:58:23
```