Problem S: Counting Out

Consider this simple form of shuffling a deck of cards. Hold the deck face down and count to a certain number (say k), on each count taking one card from the top of the deck and placing it at the bottom. When you have finished counting, turn the top card (the k+1'th card) over and place it face up on the table. Thus cards are continually moved from the top of the deck to the bottom, even when you have fewer cards than your chosen number. Continue in this way until you have no more cards left. Thus if k was 4, you would move 4 cards, one by one, from the top of the deck to the bottom and turn over the fifth. You would then move the sixth, seventh, eighth and ninth cards to the bottom and expose the tenth. Note that the number of the exposed card is always one more than k; if k was nine you would expose every tenth card, and so on.

Write a program that will read in the chosen number (k) and details of a deck of cards and simulate this process. Remember that a standard deck (or pack) of cards contains 52 cards. These are divided into 4 suits—Spades (♠), Hearts (♡), Diamonds (◇) and Clubs (♣). Within each suit there are 13 cards—Ace (A), 2–9, Ten (T), Jack (J), Queen (Q) and King (K).

Input will consist of a series of scenarios. Each scenario will start with a number k ($4 \leq k \leq 26$) on a line by itself. This will be followed by the description of a deck. A deck will be specified on four lines with 13 cards on each line and with cards separated by exactly one space (see the example below). The cards are shown as they would be seen if they were face up, thus the last card in the sequence would be the top of the deck as far as you are concerned. The file will be terminated by a line consisting of a zero (0).

Output will consist of a series of lines, one for each scenario in the input. Each line will consist of the last card played, in the format shown below.

Sample input

```
4
H2 H7 SA HJ D4 S4 HT DJ C9 HQ CT H8 SQ
DQ S6 D2 H5 CJ CK HA D5 D9 H9 S8 D3 C4
S2 C5 CQ D8 DT C6 DK H4 CA C8 C2 SK C7
S5 H6 H3 S9 S7 C3 SJ S3 ST HK DA D6 D7
0
```

Sample output

```
S8
```

Problem T: Tax Avoidance

Now that inflation is under control the tax base is fairly stable, and hence the Infernal Revenue Department is desperately seeking more ways of extracting more money from the citizenry. One proposal is to institute a capital gains tax—in particular a tax made on the profit realised by buying and selling shares on the Stock Exchange. This seems to be a safe bet, since most ordinary people are not involved and therefore it will only affect the "rich" who can afford to pay. However, the Society of Creative Acountants has heard of these proposals and realise that it could well affect them.

The problem arises from the fact that shares do not have a fixed (or even steadily moving) price, and thus one has to determine which of the entire portfolio were sold. IRD have identified two ways of determining this; called respectively First Bought, First Sold (FBFS) and Last Bought, First Sold (LBFS). These are most easily explained by an example. Assume that you bought 10000 shares at 100c per share, then bought another 10000 at 90c per share and then sold 15000 shares at 95c per share. The sale will realise $14,250. Under the FBFS method you will be deemed to have sold the oldest stocks first so they will have cost you $10,000 plus $4,500, i.e. $14,500. You have therefore suffered a loss of of $250 and would not be liable for tax. Under the LBFS scheme, you are deemed to have sold the youngest shares first and hence these would have cost $9000 plus $5000, a total of $14,000, meaning you would have to pay tax on $250.

Write a program that will read in a series of share transactions, and for each share determine which method is the optimum. The optimum is the one that minimises the profit (if both earn a profit) or maximises the loss (if at least one makes a loss). In case of a tie, choose LBFS.

Input will consist of a series of sets of share transactions. Each set will start with the name of the share (3 upper case characters) on a line by itself. This name will be unique in the data set. This will be followed by a series of buy and sell transactions. Each transaction will be on a line by itself and will start with one of the letters 'B' (Buy) or 'S' (Sell), followed by the number of shares involved (up to 100,000) and the price (in cents, up to 10,000) separated by one or more spaces. You will never sell more shares than you own. Each set of transactions will be terminated by a line starting with an 'E'. The file will be terminated by a line consisting of a single **#**.

Output will consist of a series of lines, one for each set of transactions. Each line will consist of the share name, a single space, the chosen method, a single space and the profit or loss on the overall transaction (in dollars). This should be in a field 9 characters wide with two digits after the decimal point.

Sample input

```
PCS
B 100 10000
B 100  9000
S 150  9500
E
CSC
B  100 10000
S   50 11000
E
#
```

Sample output

```
PCS FBFS   -250.00
CSC LBFS    500.00
```

Problem U: Calendar

Most of us have a calendar on which we scribble details of important events in our lives—visits to the dentist, the Regent 24 hour book sale, Programming Contests and so on. However there are also the fixed dates: partner's birthdays, wedding anniversaries and the like; and we also need to keep track of these. Typically we need to be reminded of when these important dates are approaching—the more important the event, the further in advance we wish to have our memories jogged.

Write a program that will provide such a service. The input will specify the year for which the calendar is relevant (in the range 1901 to 1999). Bear in mind that, within the range specified, all years that are divisible by 4 are leap years and hence have an extra day (February 29th) added. The output will specify "today's" date, a list of forthcoming events and an indication of their relative importance.

The first line of input will contain an integer representing the year (in the range 1901 to 1999). This will be followed by a series of lines representing anniversaries or days for which the service is requested. An anniversary line will consist of the letter 'A'; three integer numbers (D, M, P) representing the date, the month and the importance of the event; and a string describing the event, all separated by one or more spaces. P will be a number between 1 and 7 (both inclusive) and represents the number of days before the event that the reminder service should start. The string describing the event will always be present and will start at the first non-blank character after the priority. A date line will consist of the letter 'D' and the date and month as above. All anniversary lines will precede any date lines. No line will be longer than 255 characters in total. The file will be terminated by a line consisting of a single **#**.

Output will consist of a series of blocks of lines, one for each date line in the input. Each block will consist of the requested date followed by the list of events for that day and as many following days as necessary. The output should specify the date of the event (D and M), right justified in fields of width 3, and the relative importance of the event. Events that happen today should be flagged as shown below, events that happen tomorrow should have P stars, events that happen the day after tomorrow should have P-1 stars, and so on. If several events are scheduled for the same day, order them by relative importance (number of stars). If there is still a conflict, order them by their appearance in the input stream. Follow the format used in the example below. Leave 1 blank line between blocks.

Sample input

```
1993
A 23 12 5 Partner's birthday
A 25 12 7   Christmas
A 20 12 1 Unspecified Anniversary
D 20 12
#
```

Sample output

```
Today is: 20 12
20 12 *TODAY* Unspecified Anniversary
23 12 ***     Partner's birthday
25 12 ***     Christmas
```

Problem V: Beggar My Neighbour

"Beggar My Neighbour" (sometimes known as "Strip Jack Naked") is a traditional card game, designed to help teach beginners something about cards and their values. A standard deck is shuffled and dealt face down to the two players, the first card to the non-dealer, the second to the dealer, and so on until each player has 26 cards. The dealer receives the last card. The non-dealer starts the game by playing the top card of her deck (the second last card dealt) face up on the table. The dealer then covers it by playing her top card face up. Play continues in this fashion until a "face" card (Ace, King, Queen or Jack) is played. The next player must then "cover" that card, by playing one card for a Jack, two for a Queen, three for a King and four for an Ace. If a face card is played at any stage during this sequence, play switches and the other player must cover that card. When this sequence has ended (that is all covering cards have been played), the player who exposed the last face card takes the entire heap, placing it face down under her existing deck. She then starts the next round by playing one card face up as before, and play continues until one player cannot play when called upon to do so, because they have no more cards. If this occurs during a covering operation, the cards on the table remain there and are not picked up by the winner.

Write a program that will simulate playing this game. Remember that a standard deck (or pack) of cards contains 52 cards. These are divided into 4 suits—Spades (♠), Hearts (♡), Diamonds (♢) and Clubs (♣). Within each suit there are 13 cards—Ace (A), 2–9, Ten (T), Jack (J), Queen (Q) and King (K).

Input will consist of a series of decks of cards. Each deck will give the cards in order as they would be dealt (that is in the example deck below, the non-dealer would start the game by playing the H2). Decks will occupy 4 lines with 13 cards on each. The designation of each card will be the suit (S, H, D, C) followed the rank (A, 2–9, T, J, Q, K). There will be exactly one space between cards. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each deck in the input. Each line will consist of the number of the winning player (1 is the dealer, 2 is the first to play) and the number of cards in the winner's hand (ignoring any on the stack), right justified in a field of width 3.

Sample input

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

Sample output

```
1 44
```

Problem W: Rotations and Reflections

Many games, tricks and puzzles depend on determining whether two patterns on a rectangular grid are the "same" or not. For instance, the 96 different ways of arranging 8 queens safely on a chessboard can be shown to consist of rotations and/or reflections of only 12 basic patterns.

Write a program that will read in pairs of patterns and determine whether there is a simple transformation that will convert one into the other. Because symmetrical patterns bear many relationships to each other, the transformations must be checked in a specific order. The possible transformations (in order) are:

Preservation	The patterns are identical
90 degree rotation	The pattern was rotated clockwise by 90 degrees
180 degree rotation	The pattern was rotated clockwise by 180 degrees
270 degree rotation	The pattern was rotated clockwise by 270 degrees
Reflection	The pattern was reflected about the horizontal axis (effectively by a mirror held at the top of the pattern)
Combination	A reflection (as above), followed by one of the above rotations
Improper	The patterns do not match under any of the above transformations

Input will consist of a series of pairs of patterns. Each set will consist of a line containing a single integer N ($2 \leq N \leq 10$) giving the size of the patterns, followed by N lines. Each line will consist of N dots or 'x's (specifying a line of the original pattern), a space, and another set of N dots and 'x's (specifying a line of the transformed pattern). The file will be terminated by a line consisting of a single zero (0).

Output will consist of a series of lines, one for each pattern pair in the input. Each line will consist of one of the following: 'Preserved', 'Rotated through m degrees' (where m is one of 90, 180 or 270), 'Reflected', 'Reflected and rotated through m degrees', 'Improper'.

Sample input

```
5
x...x ....x
.x... ...x.
...x. .x...
..x.x ..x..
....x xx..x
2
x. xx
x. xx
4
..x. ...x
xx.. ....
.... xx..
...x ..x.
4
.x.. ..x.
.x.x x...
.... ..xx
..x. ....
0
```

Sample output

```
Rotated through 90 degrees
Improper
Reflected
Reflected and rotated through 270 degrees
```

Problem X: Word Crosses

A *word cross* is formed by printing a pair of words, the first horizontally and the second vertically, so that they share a common letter. A *leading word cross* is one where the common letter is as near as possible to the beginning of the horizontal word, and, for this letter, as close as possible to the beginning of the vertical word. Thus DEFER and PREFECT would cross on the first 'E' in each word, PREFECT and DEFER would cross on the 'R'. *Double leading word crosses* use two pairs of words arranged so that the two horizontal words are on the same line and each pair forms a leading word cross.

Write a program that will read in sets of four words and form them (if possible) into double leading word crosses.

Input will consist of a series of lines, each line containing four words (two pairs). A word consists of 1 to 10 upper case letters, and will be separated from its neighbours by at least one space. The file will be terminated by a line consisting of a single **#**.

Output will consist of a series of double leading word crosses as defined above. Leave exactly three spaces between the horizontal words. If it is not possible to form both crosses, write the message 'Unable to make two crosses'. Leave one blank line between output sets.

Sample input

```
MATCHES CHEESECAKE PICNIC EXCUSES
PEANUT BANANA VACUUM  GREEDY
A  VANISHING   LETTER TRICK
#
```

Sample output

```
  C
  H
  E
  E
  S
  E         E
  C         X
MATCHES   PICNIC
  K         U
  E         S
            E
            S


Unable to make two crosses

V
A    LETTER
N       R
I       I
S       C
H       K
I
N
G
```

Problem Y: Anniversaries

As far as our perceptions are concerned, a year runs from the midnight beginning the first of January to the midnight ending the 31st of December. Thus we can say that a year consists of:

12 months (of differing lengths)
52 (and a bit) weeks
365 days
8,760 hours
525,600 minutes
31,536,000 seconds.

Most of these numbers need to be adjusted in leap years.

A k'th anniversary of a date is usually defined as the same date (month and day) k years later, thus the 10th anniversary of 17th May 1983 is 17th May 1993. However we can also define other types of 'versaries'—the 1 week-versary of 28th February 1984 is 6th March 1984.

If we define a convention of naming dates (and times if necessary) by specifying successively smaller periods separated by colons (that is as yyyy:mm:dd:hh:mm:ss) we can label the units using the same convention, that is y for years, d for days, w for weeks, h for hours, m for minutes and s for seconds (we cannot use months, they are too variable). We can then summarise the above examples as:

1983:5:17 + 10y → 1993:5:17
1984:2:28 + 1w → 1984:3:6

Note that the specification of an event has to be as least as precise (that is employ units at least as small) as the specification of the desired versary.

Write a program that will read in details of events and desired versaries and calculate the required date (and time if necessary). Bear in mind that a year is a leap year if it is divisible by 4 unless it is divisible by 100 as well, in which case it is only a leap year if it is divisible by 400; thus 1984 and 2000 are leap years, but 1900 is not.

Input will consist of a series of lines, each line a specification of an event and a desired versary. The input will follow the format shown below, except that where one space is shown there may be more than one space. The event will be specified at least to the same precision as the desired versary. All events will start in this century, and no versary will be more than 100 years (or equivalent). Note that midnight starts a day rather than terminates it and thus is written as 0:0:0 and not as 24:0:0. The file will be terminated by a line consisting of a single 0.

Output will consist of a series of lines, one for each line of the input. Each line will consist of the specification of the desired versary, to the same precision as given in the input, in the format shown below.

Sample input

```
1983:5:17      10 y
1984:2:28      1   w
1993:7:25:13:23 5 h
0
```

Sample output

```
1983:5:17 + 10y -> 1993:5:17
1984:2:28 + 1w -> 1984:3:6
1993:7:25:13:23 + 5h -> 1993:7:25:18:23
```

Problem Z: Telephone Numbers

Many New Zealand telephones now have letters printed in association with the digits. This allows firms and organisations to "customise" their telephone numbers by incorporating their name or some other word. Thus one might be able to obtain information on training as a nurse by dialling 0–800–NURSING, or find out about courses at local universities by dialling 0–800–4–OTAGOU or 0–800–AUCKUNI.

There are two related problems associated with this approach—one is relatively easy to solve while the other is a little more difficult. If one can elicit the cooperation of one's local telephone company, then one can merely purchase a suitable telephone number that matches your word. However, if you already have a telephone number, then one needs to find the 'best' word that matches it.

Write a program that will do this. Input will be a list of words from a dictionary and a list of telephone numbers. Your program must determine suitable candidate words that fit all or part of the given telephone numbers. Since suitability is somewhat subjective the only criterion you should apply is length—only the longest matching words are considered candidates. Note also that matches can only apply at the end, sequences such as 5COSC23 are unacceptable.

For this problem assume the following allocation of letters to digits:

1	Q Z	2	A B C	3	D E F
4	G H I	5	J K L	6	M N O
7	P R S	8	T U V	9	W X Y

Input will consist of two parts. The first part will contain up to 2000 words ranging in length from 4 to 7 upper case letters, one word per line. These words form the dictionary, and will not necessarily be in English. The rest of the file will contain a series of telephone numbers, each containing 7 digits and punctuated conventionally. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each number in the input. Each line will consist of the original telephone number, a colon, a space and one or more words or number-word combinations. There must be a hyphen between numbers and letters. If there are several matching words, the words should appear in alphabetic sequence. If there are no matching words, then the message "No words" should appear. Follow the spacing and layout shown in the example.

Sample input

```
OTAGOU
UBINT
AUCKUNI
MUBGOT
468-2468
123-4567
282-5864
#
```

Sample output

```
468-2468: 4-MUBGOT 4-OTAGOU
123-4567: No words
282-5864: AUCKUNI
```