Spring 1993 Denison Computer Programming Contest
Problem 1
Identify Yourself

In this problem you are to write a program that finds all identifiers in a Pascal program and the number of times they are used. For this problem, an identifier is considered to be any alphabetic string (NOTE: not alphanumeric) which is

1. not a Pascal keyword (keywords are listed below)
2. not in a Pascal comment (comments will be delimited by the characters { and }).
3. not in a Pascal string constant (delimited by the `'` character).

The following table lists all names which are considered keywords for this problem:

`and`	`else`	`not`	`string`
`array`	`end`	`of`	`then`
`begin`	`for`	`or`	`to`
`case`	`function`	`procedure`	`type`
`const`	`if`	`program`	`until`
`div`	`in`	`record`	`var`
`do`	`mod`	`repeat`	`while`
`downto`	`nil`	`set`	`with`

Input will be a Pascal program with no alphabetic strings longer than 15 characters (which we will consider the maximum size of any identifier). Output will be a sorted list of the identifiers along with a count of the number of times they appear in the input. For each identifier, print the name first (using a field width of 16), a blank, and then the count (using a field width of 2). Print 4 identifiers per line. All identifiers should be printed using lower case.

The following information will be helpful in solving this problem:

- You may assume that input will be a valid Pascal program (i.e. one that compiles correctly).
- Pascal is not case sensitive (e.g. the names `temp` and `TEMP` denote the same identifier).
- Comments cannot be nested inside other comments.
- the characters string `''` is used to denote a single quote inside a string.

Example

On input:

```
{ Sample program }
program Sample;

  const MAX = 1000;
  var i : integer;

  procedure DigHole(var j : integer);
    var i : integer;
  begin
    for i := 1 to MAX do
      j := j - 1;           { subtract one from j }
  end;

  procedure FillHole(var k : integer);
    var i : integer;
  begin
    for i := 1 to MAX do
      k := k + 1;           { add one to k }
  end;

begin
  writeln('put your favorite comment here');
  i := 0;
  DigHole(i);               { can't think of any }
  FillHole(i);              { obvious comments to }
end.                        { put here }
```

the output should look like:

```
         dighole  2       fillhole  2              i  8         integer  5
               j  3              k  3            max  3          sample  1
         writeln  1
```

Spring 1993 Denison Computer Programming Contest
Problem 2
On The Level

In this problem you are to read in a description of a three legged table whose legs are of different lengths, along with a list of blocks which can be used to prop up each leg. The object of the program is to count the number of ways which the blocks can be used to level the table, and to determine the minimum and maximum heights of all possible leveled tables.

When counting the number of combinations, you should use the following rules:

1. the blocks are indistinguishable, so that substituting one block with another of the same size does NOT count as a new leveling combination.
2. the order of the blocks under a table leg does not matter

For example, if the table leg lengths are 3, 4 and 5 and you have 4 blocks which each have size 1, there is only one solution: two blocks under the 3 leg, one under the 4 leg. Similarly, if the table leg lengths are 3, 5, 6 and you have 3 blocks of sizes 1, 1 and 2, then there is only one solution: blocks of size 1 and 2 under the 3 leg (the order does not matter), and the remaining block under the 5 leg.

Input will consist of multiple problem instances. Each problem instance will consist of 3 or more lines. The first line will contain three unique integers corresponding to the lengths of the 3 table legs. The second line will contain one number n which represents the number of blocks. There will be no more than 8 blocks in any problem instance. The next n lines will contain the integer sizes of the n blocks, one per line. The sizes will be in no particular order and the size of any block will be ≤ 100. No block size will be less than 1. End-of-file will terminate input.

Output will consist of four lines for each problem instance. The first three lines will have the form

```
number of leveling combinations: xxx
maximum level table height: yyy
minimum level table height: zzz
```

where `xxx, yyy` and `zzz` are to be determined by your program. If no combination of blocks can be used to level the table, then you should substitute the word `none` for `xxx, yyy` and `zzz`. The third line should be blank, used to separate problem instances.

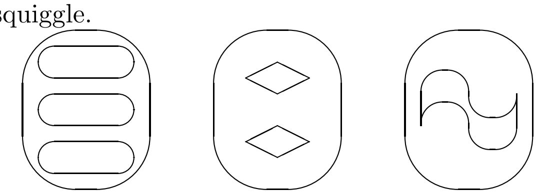
Example:

On input:	the output should look like

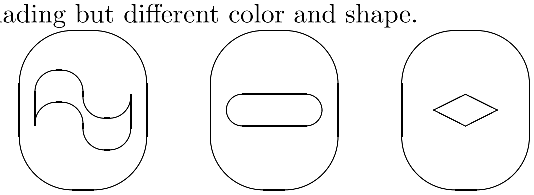
<pre>4 5 6
6
1
1
1
1
1
1
7 9 4
7
5
2
8
4
3
2
1</pre>
	<pre>number of leveling combinations: 2
	maximum level table height: 7
	minimum level table height: 6
	
	number of leveling combinations: 21
	maximum level table height: 15
	minimum level table height: 9</pre>

Spring 1993 Denison Computer Programming Contest
Problem 3
Game, Set, Match

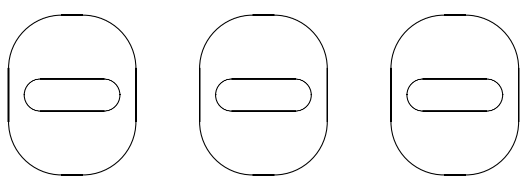
Set is a game sold by Set Enterprises that is played with a deck of 81 cards. Each card has one, two or three identical figures, in one of three colors (red, blue or green), in one of three shapes (diamond, oval or squiggle) and in one of three "shadings" (solid, cross-hatched or open). Three representative cards are given below. The first has three, red, open ovals; the second has two, green, solid diamond; the third has one, blue, cross-hatched squiggle.



The object of the game is to identify *sets*: A set is a collection of 3 cards where each of the four attributes is either the same on each card or all different. For instance, the above collection is a set since the three cards have different color, number, shape and shading. Below is another set since the three cards have the same number and shading but different color and shape.



The following is not a set since two cards are red and one is blue. (The other three attributes meet the set criteria, however.)



In this problem, you will be given a description of 12 cards and will print out all sets, if any. The description of the each card will be given as 4 characters left-justified on a line. The first card will be known as card 1, the next as card 2, etc. You are to print the card numbers in each set on a single line. Each card number should be in a field width of 3.

The attributes for each card are encoded as `WXYZ` where

- `W` is either `r`, `b` or `g` indicating color,
- `X` is either `1`, `2` or `3` indicating number,
- `Y` is either `o`, `d` or `s` indicating shape, and
- `Z` is either `s`, `c` or `o` indicating shading.

Thus, the first three cards given above would be encoded as

```
r3os
g2sc
b1do
```

Example

On input:

```
r2os
g3dc
r3os
b1oo
b1do
r1oc
r3sc
g2ss
b1so
b3ds
r1do
b3dc
```

the output should look like:

```
  1   2   9
  1   7  11
  4   5   9
  7   8   9
```

Spring 1993 Denison Computer Programming Contest
Problem 4
Mandatory Bussing

In this problem you are to simulate the activity of a bus during the course of its daily run. The bus services 8 stops on its routes labeled `A`–`H`. The schedule for the bus is as follows (where only minutes after the hour are shown).

BUS STOP							
A	B	C	D	E	F	G	H
:00	:07	:15	:22	:30	:37	:45	:52

For example, the bus is scheduled to arrive at stop `F` at 37 minutes past each hour during its scheduled hours.

Input for this problem will be several lines of text terminated by end-of-file. Each line represent passengers arriving at a stop and are listed in chronological order. Each line has the following form:

```
T S D
```

which specifies the time of arrival `T` (in the from hh:mm:ss), the starting bus stop `S` and the destination bus stop `D`. `D` will always be different from `S`. Arrival times will be listed using a 24 hour clock (i.e. 1:00:00 PM would be listed as 13:00:00, and 9:27:13 AM would be listed as 09:27:13).

The bus and passengers behave as follows:

- The bus runs its routes from 8:00 AM to 9:00 PM. The bus starts its day at stop `A`.
- The capacity of the bus is 30 people.
- The travel time between any two adjacent stops is 6 minutes.
- Picking up each passenger takes 10 secs, and discharging each passenger takes 5 secs.
- If after discharging and picking up passengers at a given stop the bus is ahead of schedule (i.e. the current time is ahead of the scheduled arrival time) it must wait at the stop until it's scheduled arrival time. While it is waiting there it may continue to pick up any passengers that arrive (including those that arrive exactly at the scheduled arrival time).
- If the bus cannot take all the passengers at a given stop, the excess passengers must wait for the next arrival (Bummer!).
- All passengers who miss the last bus (because they arrived too late or because the bus was full) should be ignored. Any passengers remaining on the bus at the end of the run at stop `A` should be discharged there.

Output will consist of a summary report for each stop. These reports will look like the following:

```
Bus Stop: A
Total # of passengers picked up: xxx
Average wait time for passengers: mm:ss
Total # of passengers discharged: xxx
Number of late bus arrivals: xxx
Bus Stop: B
Total # of passengers picked up: xxx
    (etc).
```

The wait time for a passenger is equal to the time between the passenger's arrival time at a stop and the bus's actual arrival time, or 0 if the bus is already at the stop when the passenger arrives. Display the average wait time as shown above, rounding to the nearest second.

Example:

On input:	the output should look like
`09:10:00 D E`	`Bus Stop: A`
`09:10:00 D E`	`Total # of passengers picked up:   0`
`09:10:00 D E`	`Average wait time for passengers: 00:00`
`09:11:00 D E`	`Total # of passengers discharged:   1`
`09:11:00 D F`	`Number of late arrivals:   0`
`09:11:00 D G`	`Bus Stop: B`
`09:12:00 D H`	`Total # of passengers picked up:   0`
`09:12:00 D A`	`Average wait time for passengers: 00:00`
`09:12:00 D B`	`Total # of passengers discharged:   2`
`09:15:00 C D`	`Number of late arrivals:   0`
`09:15:00 C D`	`Bus Stop: C`
`09:15:00 C D`	`Total # of passengers picked up:   6`
`09:15:00 C D`	`Average wait time for passengers: 00:00`
`09:15:00 C D`	`Total # of passengers discharged:   0`
`09:15:00 C D`	`Number of late arrivals:   0`
`09:23:30 D B`	`Bus Stop: D`
`20:30:00 A H`	`Total # of passengers picked up:  10`
	`Average wait time for passengers: 09:54`
	`Total # of passengers discharged:   6`
	`Number of late arrivals:   0`
	`Bus Stop: E`
	`Total # of passengers picked up:   0`
	`Average wait time for passengers: 00:00`
	`Total # of passengers discharged:   4`
	`Number of late arrivals:   1`
	`Bus Stop: F`
	`Total # of passengers picked up:   0`
	`Average wait time for passengers: 00:00`
	`Total # of passengers discharged:   1`
	`Number of late arrivals:   0`
	`Bus Stop: G`
	`Total # of passengers picked up:   0`
	`Average wait time for passengers: 00:00`
	`Total # of passengers discharged:   1`
	`Number of late arrivals:   0`
	`Bus Stop: H`
	`Total # of passengers picked up:   0`
	`Average wait time for passengers: 00:00`
	`Total # of passengers discharged:   1`
	`Number of late arrivals:   0`

Spring 1993 Denison Computer Programming Contest
Problem 5
Squaresville

An integer is called *square-free* if it has no factors of the form n^2 other than $n = 1$. So, 21 is square-free while 20 is not. Note that any four consecutive integers must contain at least one non-square-free number since 4 must be a factor of one. In other words, you can never have more than three consecutive square-free numbers.

Let's call three consecutive square-free integers a *square-free triple*. Thus 1, 2, 3 and 13, 14, 15 are square-free triples. In this program, you will be given two integers, n and m with $n \leq m \leq 500$, and are to print how many square-free triples are between n and m inclusive. Your program should process multiple input pairs with input terminating on end-of-file. Each input pair will occur on one line. Output for each input pair should simply be one integer on its own line.

Example

On input:

```
10 100
10 20
1 500
```

the output should look like:

```
12
1
65
```