Problem A - Sixth Grade
Standard Input

This summer, Mr. Fernandez will teach his sixth grade students a new game. With this new game, they will understand how the days of the week pass as the minutes advance.

Mr. Fernandez first explains that a day consists of 24 hours, that one hour consists of 60 minutes and that a new day initiates at 00:00 hours. This format is called **military format**.

Before beginning the game, Mr. Fernandez designates "**Sunday, 00:00**" like the day and hour initials (**DHi**). The game begins. Mr. Fernandez indicates a number of minutes (**NM**) and chooses one of his students. The chosen student must add **NM** to **DHi** and indicate the new **DHi**. This new DHi will be used by the following student. The game finishes when Mr. Fernandez indicates 0 (zero) as **NM.**

For example,

Sunday, 00:00	
83	Sunday, 01:23
935	Sunday, 16:58
663	Monday, 04:01
359	Monday, 10:00
1296	Tuesday, 07:36
978	Tuesday, 23:54
1120	Wednesday, 18:34
1	Wednesday, 18:35
540	Thursday, 03:35
828	Thursday, 17:23
1403	Friday, 16:46
172	Friday, 19:38
761	Saturday, 08:19
0 (End of Game)	

Your task is to help to Mr. Fernandez with a program that verifies the DHi's indicated by the students.

Input

The input is a series of integers (**NM**), one per line, ranging from 0 to 10,000. The number 0 (zero) will indicate the end of input.

Output

Your program should output a series of **DHi** (one per line), that shows the result of the sum of each **NM** and the **DHi** previously calculated. The first **DHi** is "Sunday, 00:00". The format of **DHi** output is **"**Name of Week Day**",** one comma**,** one space and the hour in military format**.**

Sample Input

```
83
935
663
359
1296
978
1120
1
540
828
1403
172
761
0
```

Output for the Sample Input

```
Sunday, 01:23
Sunday, 16:58
Monday, 04:01
Monday, 10:00
Tuesday, 07:36
Tuesday, 23:54
Wednesday, 18:34
Wednesday, 18:35
Thursday, 03:35
Thursday, 17:23
Friday, 16:46
Friday, 19:38
Saturday, 08:19
```

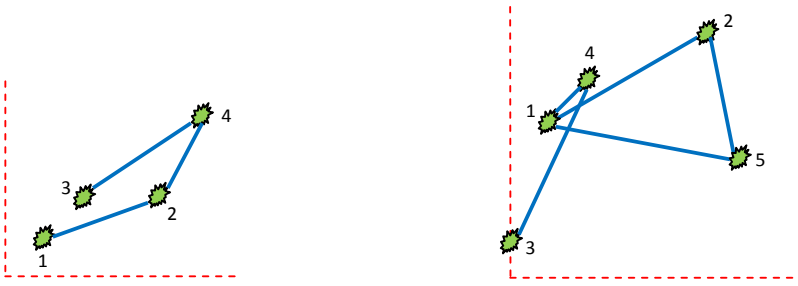
Problem B - Hammock in the Forest
Standard Input

After 5 hard-working days, the weekend is finally here; and a group of friends have decided to go eat at “La primavera” forest and spend the rest of the day resting, by simply lying on some hammocks. Therefore, they will have to look for an area that contains many trees, with the purpose of assembling as many hammocks as possible in a reduced space so that they are able to talk comfortably without having to yell or even getting off their hammock.

In order to assemble each hammock a tree is needed for each end, without having to worry about the distance between these two since hammocks can easily adjust, and in the group of friends there are short and tall ones. The only restriction is that, since all of the hammocks have to be hung at the same height, they cannot be crossing in the space that will be used. Many hammocks can however be hung in different directions, from the same tree.

The problem consists of determining if a given configuration of trees and hammocks may result in crossings and, if so, which pair of hammocks are involved in each crossing. Such a configuration includes: the coordinates (x, y) of each of the trees to be used and, for each hammock, the reference of the two trees that are used to hang it. For simplicity, the coordinates of the trees are specified with integer numbers.

The figure on the left side denotes the configuration of the first test case, in which four trees are contemplated with three hammocks without crossings. The figure on the right side is the configuration of the second test case. In this case, installing hammock (3,4) would cause a cross with hammock (1,2) already installed; similarly, hammock (1,5) would cross with hammock (3,4).



Input

The first line contains an integer number $N > 0$, that denotes the number of test cases. The following N lines contain the configuration of trees and hammocks. The configuration is given by two sequences separated by a semicolon. The first sequence consists of one or more space-separated pairs (x_k,y_k), where (x_k, y_k) is the coordinate of the tree with index $k \geq 1$ and is given by two integers, such that: $x_k, y_k \geq 0$. The second sequence consists of one or more space-separated pairs (k, k'), where k, k' denotes the indices of two trees connected by a hammock, such that: $k \neq k'$ and $k, k' \geq 1$. Consider that the input will not violate any of the restrictions that have already been mentioned.

Output

The output consists of N lines, where each $k \leq N$ contains the *Ok* word if in the current test case, no crossings of hammocks were found. However, if they were found it contains the phrase *Crossings found:* followed by the list of hammocks that crossed, separated by a semicolon. Each crossing is specified as follows: <space>(k,k')<space>(j,j'), meaning that hammock connecting trees k and k' have crossed with hammock connecting trees j and j'. Consider that both hammocks where specified in the input, and that hammock (j,j') was specified first.

Sample Input

2

(1,1) (4,2) (2,2) (5,4);(1,2) (3,4) (4,2)

(1,4) (5,6) (0,1) (2,5) (6,3);(1,2) (3,4) (1,4) (1,5) (2,5)

Output for the Sample Input

Ok

Crossings found: (3,4) (1,2); (1,5) (3,4)

Problem C - Sequence
Standard Input

Schaeffer's model for population growth of a bank of shrimp (Verhulst-Pearl but applying it to biomass instead of number of individuals and including fishing) is the population growth of shrimp.

Growth Rate = Biomass * Population Shrimp * Factor

The problem is to calculate the factor according to the following:

Observe the following sequence of natural numbers; number N indicates how many elements from the sequence will be generated.

N	Sequence
1	101
2	51,101
3	34,51,101
4	26,34,51,101
5	21,26,34,51,101
6	18,21,26,34,51,101
7	15,18,21,26,34,51,101
8	14,15,18,21,26,34,51,101

Input requirements

$0<N\leq100$

Sample Input

20

Output for the Sample Input

6,6,7,7,7,8,8,9,9,10,11,12,14,15,18,21,26,34,51,101

Problem D - Finding the closest triangle
Standard Input

Triangles are one of the most important types of geometrical shapes; they were widely studied by the ancient Greeks and other cultures, perhaps due to the fact that they are the polygon with the smallest number of sides. Several Greek mathematicians and geometricians came up with fundamental theorems that are widely used today in many fields, such as the Pythagoras theorem.

An important concept when studying triangles is the concept of similarity. Simply put, two triangles are similar if they are the same shape. So, a triangle can be scaled uniformly in both dimensions (height and width), it can be translated to a different position, it can be rotated and it can be flipped (or mirrored), and the resulting triangle will be similar to the original one. Two triangles, called ABC and DEF, are similar if any one of the following conditions is true:

- they have the same three inside angles. This means that one needs to prove they share only two angles, since the third will be the same if two angles are equal, so

$\angle BAC = \angle EDF$ and $\angle ABC = \angle DEF$. This implies that $\angle ACB = \angle DFE$,

or

- Corresponding sides have lengths with the same ratio,

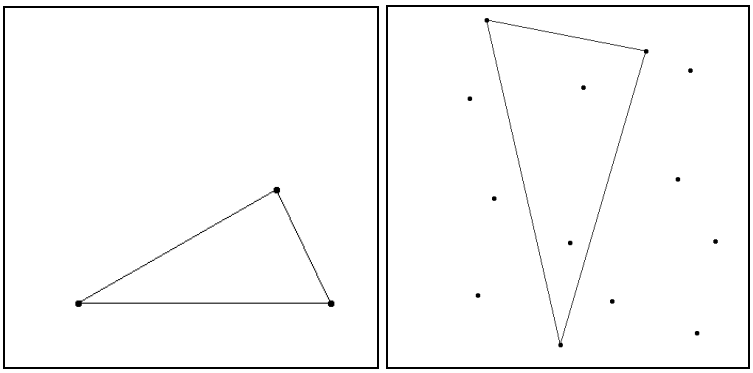
$$\frac{AB}{DE} = \frac{BC}{EF} = \frac{AC}{DF}$$

or

- the angle formed by two sides of one triangle is equal to an angle in the other triangle, and the ratios of the length of the sides forming the angle is equal in both triangles.

This problem is based on the concept of similarity, because it consists of finding the *most* similar triangle to a given triangle. You must write a program that, given a triangle, will determine which three of the ***n*** input points define the most similar triangle possible. The fact is that most likely no three points will produce a perfectly similar triangle to the original triangle; therefore, you need to devise a way to determine the most similar triangle possible.

The following figure shows how the most similar triangle to the sample triangle on the left is selected from the points in the space on the right.



Note that the second triangle may be scaled, rotated and flipped (mirrored) in relation to the original triangle; none of these transformations negate similarity.

Input

The input consists of:

- One line containing an integer, m, which indicates the number of test cases in the input file. For each one of the cases, there will be:

- Three lines containing the coordinates of three points, which define the original triangle. Each point is represented by two integers, one for the ***x*** coordinate (0<= ***x*** <=500) and the other for the ***y*** coordinate (0<= ***y*** <=500), separated by a coma and a space.

- One line containing one integer ***n***, where 4<= ***n*** <=100. This number represents the number of points in which to "fit" the original triangle.

- ***n*** lines containing ***n*** points. Each point is defined by two coordinates (***x*** and ***y***). Each pair of coordinates gives the point's position along the x and y axes, where positive numbers between 0 and 500 for each coordinate. Each pair of coordinates consists of its ***x*** coordinate followed by a coma (',') and a space (' '), and the ***y*** coordinate, on one line.

Output

The output will consist of three lines for each case in the input file, each one with one of the points of the triangle that is most similar to the original triangle. Each point will consist of its x (0<=x<=500) coordinate, followed by a coma (',') and a space (' '), and its y coordinate (0<=y<=500). The points for each triangle may be listed in any order. The triangle coordinates must coincide with the order in which the test cases appear in the input file.

Sample input	Output for the sample input
3 136, 355 141, 176 417, 30 6 113, 87 120, 240 182, 418 440, 350 402, 184 441, 90 175, 430 113, 199 419, 58 7 97, 233 212, 421 376, 389 423, 216 388, 73 33, 45 339, 464 240, 417 67, 390 398, 61 11 134, 425 19, 107 426, 70 222, 147 281, 259 161, 327 366, 470 471, 440 444, 277 230, 23 137, 22	113, 87 440, 350 402, 184 212, 421 376, 389 339, 464 134, 425 19, 107 137, 22

Problem E
Disassemble a machine language code for the processor POP11
Standard Input

Processor POP11

POP11 is a 2 address processor with its instruction set that conforms its machine language. These instructions are composed by bits sequences that include operation code, addressing mode of operands, and address fields with operands of instruction. This format is showed in Table 1

To make programming task easier, there is an assembly language that consist of mnemonic codes that are translated to machine language and they have one of the following formats:

Instruction source_and_dest_operand

Instruction source_operand,dest_operand

Operands Representation

The operands of the assembly language instructions for the POP11 processor may be located in: CPU Registers, memory (that could be accessed with direct addressing mode or using a register for memory addressing) or may be represented by constants (immediate addressing mode), by convention all numbers used in assembly codes are in hexadecimal and we use the following symbols to indicate the addressing mode of each operand. For example:

Operand	Meaning
0123	The constant number 0123 (hexadecimal), this is the immediate addressing mode.
[0123]	The data that is located in memory address 0123 (hexadecimal),this is the direct addressing mode.
[R1]	The data is in memory and the value of the register R1 is the memory address where the data is located, this is direct-register addressing mode.

The POP11 processor is Little Endian. That means that for all numbers that need more than 8 bits to be stored in memory, for example, a 16 bit number, the least significant 8 bits are stored in memory before the most significant 8 bits. For example:

Number (hexadecimal)	Bytes in memory (hexadecimal)
1234	34 12
78 AB	AB 78

Machine Language Instructions

The instructions of POP11 processor are formed according to the formats indicated on tables: Table 1, Table 2, Table 3 and Table 4. For example, the instruction: SUB R1, R2 when is assembled generates the bytes sequence 09 40 20 40.

<table>
<tr><td colspan="2">SUB</td><td colspan="2"></td><td colspan="2">R1</td><td colspan="2">R2</td></tr>
<tr><td colspan="2">Operation code of SUB instruction, look table 1</td><td colspan="2">Addressing mode, in this case both operands are register, so, this will be register to register

look table 2</td><td colspan="2">6 bits of adjustment, must be 0s</td><td colspan="2">Source operand 001 that indicates R1, a 5 bits adjustment in 0 a must be added to the right in order to complete 8 bits</td><td colspan="2">Destination operand 010 that indicates R2, a 5 bits adjustment in 0 a must be added to the right in order to complete 8 bits</td></tr>
<tr><td colspan="2">000010</td><td colspan="2">0101</td><td colspan="2">000000</td><td colspan="2">001 00000</td><td colspan="2">010 00000</td></tr>
<tr><td>0000</td><td>1001</td><td colspan="2">0100</td><td>0000</td><td>0010</td><td>0000</td><td>0100</td><td>0000</td></tr>
<tr><td colspan="2">09</td><td colspan="3">4 0</td><td colspan="2">20</td><td colspan="2">40</td></tr>
</table>

Some examples of assembly language instructions assembled to machine language for POP11 processor and its representation like a bytes sequence are shown on the next table.

Assembly language instruction	Machine language (bytes sequence)	Description
ADD R1,R2	05 40 20 40	Add R1 to R2, the result is stored in R2
NEG R2	29 00 40	Calculate the negative value of R2, result is stored in R2
MOV [0351],R3	16 40 51 03 60	Add the value stored in memory location 0351 to R3, the result is stored in R3
ADD R4,[0015]	05 80 40 15 00	Add the value stored in R4 to the value in memory location 15, the result is stored in memory location 15.
ADD 0156,R1	04 40 56 01 20	Add the constant value 156 (hexadecimal) to R1 register, result is stored on R1.

Your Challenge

Prog.in is a text file that contains a sequence of 8 bit numbers in hexadecimal where each number corresponds to a byte. The whole sequence of bytes represents a binary program for POP11 processor. This program consists of a bytes sequence that is machine language instructions. You have to create a disassembler tool that will read the bytes contented in the file and convert to its assembly language instructions sequence.

Instruction Format

POP11 instructions may have 1 or 2 operands and they may occupy 2, 3, 4 or 5 memory bytes, depends of addressing modes and operands used in instruction. If the operand is a memory address will have a 16 bits size.

OP Code (6 bits)	Addressing Mode (4 bits)	Adjustment 6 bits must be on 0s	Address field 1 (source operand 8 or 16 bits)	Address field 2. (destination operand 8 or 16 bits only for addressing modes with 2 operands)

Table 1

Operation Codes

6 bits, 64 operations maximum

ADD =	000001
SUB =	000010
MUL =	000011
DIV =	000100
MOV =	000101
AND	001110
OR	001111
XOR	001000
NOT	001001
NEG	001010
CMP	001011
JE[1]	010001
JNE[1]	010010
JA[1]	010011
JB[1]	010100
JMP[1]	010000

Table 2

[1] All control transfer instructions uses relative addressing mode

Valid addressing modes (4 bits)

0000	Immediate[2] (the source operand is a constant) , there´s no destination operand
0001	Immediate to register
0100	Register (For one operand instructions)
0101	Register to register
0110	Register to direct memory[3]
0111	Register to memory addressed with a register
1000	Direct memory (For one operand instructions)
1001	Direct memory to register
1100	Memory addressed with a register (For one operand instructions)
1101	Memory addressed with a register to register
1111	Relative, an amount of bytes that will be added to current memory address, used for control instructions like JMP, JE, JNE, etc.

Table 3

[2] All constants are 16 bits size and are stored in memory as little endian format

[3] All memory addresses are 16 bits and are stored with little endian format

In little endian: all numbers greater than 8 bits are stored like a bytes sequence where the first byte addressed its the most significant part of the number.

Registers

000	= R0
001	= R1
010	= R2
011	= R3
100	= R4
101	= R5
110	= R6
111	= R7

Table 4

Registers only use 3 bits of address field of instruction, but must have an adjustment of 5 bits with 0 at right.

Sample Input

16 40 00 04 60 05 80 40 20 00 04 40 00 40 20

Output for the Sample Input

MOV [0400],R3

ADD R4,[0020]

ADD 4000,R1

Problem F - Kids with obesity
Standard Input

Unfortunately, Mexico gained the top of world obesity charts, especially in kids. This has caused a lot of controversy and a lot of programs have been created to help prevent obesity in kids.

The school where your brother teaches has started to get information about the health of their students. Based on the BMI of every kid the examiner determines if the kid is under weight, normal, overweight or obese.

The BMI is calculated using this formula:

BMI = weight in kilograms / (height in meters)^2

The weight status in children depends on the sex as well. Your brother has shown you these charts:

Girls

Age	Low		Normal		Overweight		Obesity	
10	0	16.8	16.9	19.8	19.9	24	24.1	100
11	0	17.4	17.5	20.6	20.7	25.3	25.4	100
12	0	18	18.1	21.6	21.7	26.5	26.6	100
13	0	18.6	18.7	22.5	22.6	27.7	27.8	100
14	0	19.3	19.4	23.2	23.3	28.5	28.6	100

Boys

Age	Low		Normal		Overweight		Obesity	
10	0	16.5	16.6	19.7	19.8	23.9	24	100
11	0	17.1	17.2	20.5	20.6	25	25.1	100
12	0	17.7	17.8	21.1	21.2	25.9	26	100
13	0	18.4	18.5	21.8	21.9	26.7	26.8	100
14	0	19.1	19.2	22.5	22.6	27.5	27.6	100

Every teacher is responsible of keeping track of their student's BMI. The school has a nutritionist that will help balance of the kid's lunch of those who are overweight or obese. To make things easy for the nutritionist, each teacher filters the names of those who do not need any help.

Your job is to help your brother determine who needs the nutritionist is help, by calculating the BMI and just printing those who need help.

Input

The input consists of n lines with the kid's name, weight in kilograms, and height in meters, age and gender, F for female and M for male. The values are separated by coma.

Output

The output should be a list of the kids who are overweight or obese grouped by age.

Each line in the group should contain a line with the kid's name. In the next line the kid's BMI with 2 decimals, coma, space, and if the kid is overweight or obese. Each group must be separated by one blank line.

Sample Input

Elena Aquino Olivares,47,1.5,12,F
Mirna Juarez Ortega,49,1.62,12,F
Estefanie Martinez Olivares,74,1.6,12,F
Jennifer Gomez Cuadra,38,1.3,12,F
Guadalupe Barajas Garcia,78,1.62,12,F

Output for the Sample Input

12 years
Estefanie Martinez Olivares
28.91, obese
Jennifer Gomez Cuadra
22.49, overweight
Guadalupe Barajas Garcia
23.56, overweight