TCEA
HIGH SCHOOL
PROGRAMMING CONTEST

STATE PROBLEM SET
30 APRIL 2005

Texas Computer Education Association
2005 High School Programming Contest
State Problem Set

Problem 2.1 TCEA Scoring

General Statement: In the TCEA programming contest, there are problems worth 2, 5, and 9 points each. Each team's total score is based on the sum of the points of the problems that they solve. Thus, if a team solves four 2-point problems, three 5-point problems, and two 9-point problems, they have 41 (4*2 + 3*5 + 2*9) total points.

Your task is to write a program that computes the results for a team in the TCEA contest given the number of problems that the team solved.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains three integers representing the number of solved 2-point problems, solved 5-point problems, and solved 9-point problems for a single team, respectively.

Name of Data File: pr21.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain a single integer representing the total score of the problems in the corresponding input data collection.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The number of problems of a given type solved by a single team will be between 0 and 6, inclusive.

Sample Input:

```
3
3 0 4
1 6 1
0 1 6
```

Sample Output:

```
42
41
59
```

Problem 2.2 Look Before You Leap

General Statement: The guidelines for determining if a year is a leap year are more complicated than many people realize. The rule most people know is that if a year is evenly divisible by 4, then the year is a leap year. There are exceptions to that rule, however. If a year is evenly divisible by 100, then it is *not* a leap year. There is an exception to this rule as well because if a year is evenly divisible by 400, then it *is* a leap year. So, 1900 was not a leap year, but 2000 was.

Your task is to write a program that determines if an input year is a leap year.

Input: The first line of the input is an integer *n* that represents the number of data values that follow where each data value is on a single line. Each data value contains a year represented as a 4-digit number.

Name of Data File: pr22.dat

Output: Your program should produce *n* lines of output (one for each data value). Each output line should indicate if the corresponding input year is a leap year. If the year is a leap year, your program should print "`leap year`". Otherwise, your program should print "`not a leap year`".

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each year will be between 1583 and 9999, inclusive.

Sample Input:

```
4
1988
1998
2200
2000
```

Sample Output:

```
leap year
not a leap year
not a leap year
leap year
```

Problem 2.3 Electoral College

General Statement: The United States uses the electoral college system to determine the winner of the presidential election every four years. In this system, each state is assigned a certain number of electoral votes. The number of electoral votes a state has is based on the total number of congressional seats for that state. For this problem, a candidate receives all of the electoral votes in a state if he or she receives the majority of the popular vote of that state. The candidate that receives the most total number of electoral votes is the winner.

Your task is to determine the total number of electoral votes for each candidate in an election given the popular vote totals and the number of electoral votes for each state.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection represents the information for a single state in the election and is represented by three integers X, A, and B. For each state, the first integer, X, represents its total number of electoral votes, the second integer, A, represents the number of popular votes obtained by candidate A, and the third integer, B, represents the number of popular votes obtained by candidate B.

Name of Data File: pr23.dat

Output: Your program should produce exactly one line of output. The output should consist of two integers. The first integer should represent the number of electoral votes for the winning candidate, and the second integer should represent the number of electoral votes for the losing candidate. The two integers should be separated by exactly one space.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: There are only two candidates in the election.
Each state will have a popular vote winner (no ties).
One candidate will have more total electoral votes than the other.

Sample Input:

```
3
10 500 200
20 1000 1300
5 80 75
```

Sample Output:

```
20 15
```

Problem 2.4 Pizza Puzzler

General Statement: After suffering through their worst argument since they have known each other, a group of friends known as “The Circle” decide to take steps to ensure that they will never argue again. This argument occurred because when The Circle last ordered a pizza, some members had more slices than other members. They decided to avoid this argument by ensuring that they always ordered enough pizzas so that each member could have an equal number of slices.

The Circle has come to you for assistance to write a program they can use whenever they are ready to order pizza. Your task is to write a program that determines the *smallest* number of pizzas greater than 0 they have to order so that each member can have an equal number of whole slices. Your program must be able to compute this total given any number of Circle members present and number of slices per pizza.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow. Each data collection consists of two integers *x* and *y* separated by a single space. The first integer, *x*, represents the number of Circle members present. The second integer, *y*, represents the number of slices per pizza.

Name of Data File: pr24.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should contain a single integer greater than 0 that equals the minimum number of pizzas needed that allows the *x* members of the Circle to have an equal number of pizza slices.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each *x* and each *y* will be between 1 and 1000, inclusive.

Sample Input:

```
4
5 8
1 6
6 1
4 10
```

Sample Output:

```
5
1
6
2
```

Problem 2.5 Lucky Numbers

General Statement: Coaches of the teams competing in this year's baseball tournament are so superstitious that they all have unlucky numbers that they want to avoid. The first step in avoiding his or her unlucky number is to make sure no player on the team wears a uniform with that number. In addition, each coach also wants to ensure that no two players on the team have uniform numbers that add up to the unlucky number.

The coaches want a program that will input an unlucky number and a set of possible uniform numbers and will determine if the unlucky number can be formed as a sum of any two of the uniform numbers. Your task is to write that program.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of a set of integers with each pair of integers separated by a single space. The first integer in the set, *x*, represents an unlucky number. The remaining integers represent the possible uniform numbers.

Name of Data File: pr25.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should indicate whether there are two uniform numbers in the data set whose sum equals the unlucky number, *x*. If there are two such numbers, the output line for the data collection should be "`unlucky`". Otherwise, the output line for the data collection should be "`lucky`".

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each data collection will contain only positive integers.
There will be no more than 20 different uniform numbers in a single data collection, and there will always be at least 2 uniform numbers.

Sample Input:

```
3
12 8 3 1 10
15 9 4 2 8 11
21 1 2 3 4 5 6 7 8 9 10
```

Sample Output:

```
lucky
unlucky
lucky
```

Problem 2.6 Jedi Master

General Statement: After traveling all of the way across the galaxy to meet the Jedi Master and last hope for the rebellion, you discover that you cannot understand anything that he says. After listening for a while, you discover that whenever the Jedi Master says a sentence with a subject, verb, and direct object, he says the direct object before the subject and verb. You quickly realize that to speak to the Jedi Master, you need to speak with the same pattern. Thus, you decide a program can be written that will convert your sentences into the Jedi Master's form. Your task for this problem is to write that program.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection represents a sentence. The first word of the sentence is its subject, and the second word of the sentence is its verb. For this problem, treat the remaining portion of the sentence as the sentence's direct object. Thus, the direct object for each sentence may consist of one or more words where a word is any sequence of characters without spaces.

Name of Data File: pr26.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line be the Jedi Master's version of the input sentence which means that the subject and verb should be placed after the direct object.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each sentence will contain at least three words.
Each sentence will contain between 1 and 80 total characters, inclusive.
Each word in the sentence will consist of only lowercase letters.
There will be a single space between each pair of words in the input.

Sample Input:

```
3
you did find someone
they are strong in the force
fear leads to anger and suffering
```

Sample Output:

```
find someone you did
strong in the force they are
to anger and suffering fear leads
```

Problem 5.1 A Bit of Parity

General Statement: When a data packet is transmitted across a network, a parity bit may be included to allow the receiving computer to check if the packet has been corrupted. The parity bit is computed based on the sum of all of the bits of the original data packet mod 2. For example, if the packet 101011 should be transmitted, the bit 0 would be appended to the packet since (1 + 0 + 1 + 0 + 1 + 1) mod 2 equals 0. Thus, the packet that would be transmitted is 1010110. With this protocol, a computer receiving a packet can check if one of its bits was corrupted (changed) during transmission by examining if the last bit of the packet is equal to the sum of the other bits mod 2.

Your task is to write a program that determines if an input packet has been corrupted using the above approach.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection represents a packet and consists of a string of characters with each character representing a single bit. Thus, each character is either `0` or `1`.

Name of Data File: pr51.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should indicate if the corresponding input packet has been corrupted and should therefore be rejected. If the packet has been corrupted, your program should print "`reject`". Otherwise, your program should print "`accept`".

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each packet will contain between 2 and 80 characters, inclusive.

Sample Input:

```
4
1010
10101
00000000000000000000000000000011
11111111
```

Sample Output:

```
accept
reject
accept
accept
```

Problem 5.2 Finding Nemo

General Statement: Names can often be hidden within the words of a sentence. The words can be found by combining the letters of consecutive words in order. For example, the name "NEMO" can be found within the sentence "WE HAVE ONE MORE PROBLEM!" by combining the N and E from the word "ONE" with the M and O from the word that follows it, "MORE".

Your task is to write a program that determines if a name is hidden within a given sentence by joining the letters of one or more consecutive words in order. Note that a name is still hidden within a sentence even if there are punctuation symbols between the words that contain it.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow. Each data collection consists of two lines. The first line of the data collection contains a sentence, and the second line contains a name.

Name of Data File: pr52.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should indicate if the corresponding input name can be found in the associated sentence. If the name is found, your program should print "FOUND". Otherwise, your program should print "NOT FOUND".

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each sentence will contain between 1 and 80 characters, inclusive.
Each name will contain between 1 and 20 characters, inclusive.
Each name will consist of only letters (no spaces).
All of the letters in the input sentence and name will be in uppercase.

Sample Input:

```
3
SINCE HE MISSED A STEP, HE NEEDS TO RESTART.
STEPHEN
THE MAXIMUM SCORE FOR THIS CONTEST IS 96.
MAX
THE LINE FOR THE LEANING TOWER IS THE LONGEST.
HELEN
```

Sample Output:

```
FOUND
FOUND
NOT FOUND
```

Problem 5.3 Final Jeopardy (page 1 of 2)

General Statement: On the game show Jeopardy, three contestants compete by answering questions worth varying numbers of points. The contestant with the highest point total at the end of the game is declared to be the winner. The last question of the game is known as Final Jeopardy.

Final Jeopardy offers each contestant the opportunity to double his or her score. This is possible because each contestant can risk as many of his or her points as possible on the last question. If the question is answered correctly, the risked points are added to the contestant's score. Therefore, if contestant X wagers all of his or her points and answers the question correctly, X's score will double.

The usual strategy is that the leader at the start of Final Jeopardy wagers just enough points to ensure victory even if the other contestants double their respective scores. For example, if contestants A, B, and C have 1000, 800, and 400 points, respectively, contestant A will wager at least 601 points.

Your task is to write a program that determines the minimum wager the leader should make in Final Jeopardy to ensure victory even if the other contestants double their respective scores.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of three integers *x*, *y*, and *z* representing the scores of three contestants entering Final Jeopardy. There is a single space between each pair of scores.

Name of Data File: pr53.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should contain a single integer. That integer must be the minimum number of points the leader should wager to ensure victory even if the other contestants double their scores.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each score will be between 1 and 99999, inclusive.
No two scores within a single data collection will be the same.
In each data collection, the leader will not have more than twice the number of points of the contestant in 2nd place.

Problem 5.3 Final Jeopardy (page 2 of 2)

Sample Input:

```
3
2000 2500 800
99 100 101
5000 555 10000
```

Sample Output:

```
1501
100
1
```

Problem 5.4 Valid Passwords

General Statement: Passwords are valid if they have at least one lowercase letter, at least one uppercase letter, at least one digit, and at least one special character. If one or more of those components are missing, then the password is invalid. For this problem, a special character is one of the following characters `!`, `@`, `#`, `$`, `%`, `^`, `&`, `*`, `(`, or `)`. Your task is to write a program that determines if a given string of characters is a valid password.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a single string of characters without any spaces.

Name of Data File: pr54.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should indicate if the corresponding input string contains a valid or invalid password. If the string is a valid password, your program should print "`valid`". Otherwise, your program should print "`invalid`".

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each input string of characters will have between 4 and 20 characters, inclusive.
Each character in an input string will be an uppercase character, lowercase character, digit, or special character as defined in the General Statement.

Sample Input:

```
4
Contest##
abCDE012*
password4$
2^Ab
```

Sample Output:

```
invalid
valid
invalid
valid
```

Problem 5.5 Time Translations

General Statement: We know that there are 60 seconds in 1 minute, there are 60 minutes in 1 hour, there are 24 hours in 1 day, and there are 7 days in 1 week. Given this knowledge, we can convert amounts of time from one unit to another. For example, we know that we can convert 3 days into 72 hours, and we can convert 300 seconds into 5 minutes.

Your task is to write a program that reads an amount of time and converts it into another specified unit.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of a set of three components *k A B* where *k* is an integer and *A* and *B* are units of time. Each data collection represents a request to convert *k A* into *B*. Thus, given an amount of time expressed as *k* sets of *A* units, the task is to find an equal amount of time expressed using *B* units.

Name of Data File: pr55.dat

Output: Your program should produce *n* lines of output (one for each input data collection). Each output line should contain a single integer that is the number of *B* units that are equal to *k* sets of *A* units.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each unit will be one of the following: `seconds`, `minutes`, `hours`, `days`, or `weeks`.
The given number of units will convert to an integer value between 1 and 1,000,000, inclusive.

Sample Input:

```
4
28 days weeks
2 hours seconds
2 weeks minutes
300000 seconds minutes
```

Sample Output:

```
4
7200
20160
5000
```

Problem 5.6 Tic-Tac-Toe

General Statement: A game of Tic-Tac-Toe is played on a 3x3 grid by player X and player O. Player X wins the game if the player has three Xs in a straight line horizontally, vertically, or diagonally. Alternatively, player O wins the game if three Os are in a straight line horizontally, vertically, or diagonally. If neither player wins the game, then it is a draw.

Your task is to write a program that determines the winners of an input set of Tic-Tac-Toe games.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection consists of three lines, and each line contains three characters. Each data collection represents the board of a Tic-Tac-Toe game. Each character of a Tic-Tac-Toe game will be either an X or an O.

Name of Data File: pr56.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should indicate the winner of the corresponding input Tic-Tac-Toe game. If player X is the winner, your program should print "X WINS". If player O is the winner, your program should print "O WINS". If the game is a draw, your program should print "NO WINNER".

The output is to be formatted exactly like that for the sample output given below.

Assumptions: There will be at most one winner of each Tic-Tac-Toe game.

Sample Input:

```
3
XOO
XXO
OXX
OXO
XXO
XOO
OXO
XOX
XOX
```

Sample Output:

```
X WINS
O WINS
NO WINNER
```

Problem 9.1 Making Minesweeper (page 1 of 2)

General Statement: Minesweeper is a game that is commonly included with Microsoft Windows operating systems. The game is played on a two-dimensional board, and each position on the board either contains a mine, or it contains a number. If the board position contains a number, that number represents the total number of mines that are adjacent to the position in any of the eight directions, north, south, east, west, northeast, southeast, northwest, and southwest.

Your task is to write a program that computes the numbers for each space on a Minesweeper board given an input set of mine positions.

Input: The first line of the input contains three integers *r*, *c*, and *n*. The integers *r* and *c* represent the number of rows and the number of columns of the Minesweeper board, respectively. The third integer, *n*, represents the number of mines that are on the board.

The remainder of the input consists of *n* lines with each line representing an individual mine position. Each position consists of two numbers *x* and *y* meaning that the mine is located at row *x* and column *y*. The numbering of the board should be from top to bottom and left to right, with the upper left corner of the board being position (1, 1) and the lower right corner of the board being position (*r*, *c*).

Name of Data File: pr91.dat

Output: Your program should produce *r* lines of output with each line containing exactly *c* characters. If position (*r*, *c*) has a mine, then the character in row *r* and column *c* should be an asterisk, *. Otherwise, the character in row *r* and column *c* should be the number of mines adjacent to that position in the Minesweeper board. If no mines are adjacent to a position, the character for that position *must* be a 0.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *r* will be between 1 and 20, inclusive.
The value of *c* will be between 1 and 20, inclusive.
The value of *n* will be between 0 and 400, inclusive.
Each input mine position (*x*, *y*) will be a valid position on the board. Thus, each x will be between 1 and *r*, inclusive, and each *y* will be between 1 and *c*, inclusive.
A mine position will not be listed more than once.

Problem 9.1 Making Minesweeper (page 2 of 2)

Sample Input:

```
4 5 6
1 3
1 4
2 5
3 1
3 3
4 5
```

Sample Output:

```
01**2
1334*
*2*32
1212*
```

Problem 9.2 Spellchecker (page 1 of 2)

General Statement: An advanced feature of several programs such as word processors and e-mail clients is to correct the spelling of text composed by the user. This feature involves the program checking each word composed by the user against a dictionary of known words. If the word is not in the dictionary, then it is a misspelled word.

To suggest the correct spelling of a misspelled word, the program needs the ability to find the word in the dictionary that is the closest match for it. For this problem, the closest match is the dictionary word that has the most number of letters in common with the misspelled word. Two words have a letter in common if the letter appears in the same position in both words. For example, the words “contest” and “correct” have four letters in common. Those letters are the ones that appear in the first, second, fifth, and seventh positions in each word.

Your task is to write a program that finds the correct spelling for an input word. Thus, your program needs to be able to find the best matching word in a dictionary for the input word.

Input: The first line of the input contains two integers *m* and *n*. The next *m* lines of the input represent the dictionary of words known by the program. Each of those *m* lines contains one dictionary word apiece. The next *n* lines following the dictionary represent the program test cases with each test case being on a single line. Each test case consists of a single word that is not in the dictionary.

Name of Data File: pr92.dat

Output: Your program should produce *n* lines of output (one for each test case). Each output line should indicate the correct spelling of the word in the corresponding input test case. Thus, your program should output the dictionary word that is the closest match to the input test case word. Again, the closest match for an input test case word is the dictionary word that has the most letters in common with it.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The number of dictionary words, *m*, will be between 2 and 20, inclusive. Each input test case will have only one best match in the dictionary. All of the dictionary and test case words will have the same length. Each dictionary and test case word will contain only lower case letters.

Problem 9.2 Spellchecker

Sample Input:

```
5 3
contest
program
hundred
tuesday
compute
correct
bundled
contact
```

Sample Output:

```
contest
hundred
contest
```

Problem 9.3 Twenty-One (page 1 of 2)

General Statement: The game of twenty-one is played by multiple people where each player has a set of cards called his or her "hand", and each hand has a point value. The winner of the game is the person with the hand with the point value that is the closest to 21 without exceeding 21. Thus, a person with a hand whose total point value is 22 or higher cannot win.

The total point value of a hand is the sum of the point values of each individual card in the hand. The type of card represents its point value, and each card may be either A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, or K. If the card type is numeric, then the point value of the card is equal to its type. If the card is a T, J, Q, or K, then it is worth 10 points. Finally, if the card is an A, then its point value may be 1 or 11 depending on the total point value of the hand. It is worth 11 unless the total point value of the hand exceeds 21. In that situation, the A would be worth 1 point. Given a set of four such hands in a game of 21, your task is to write a program that determines the winning hand.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection represents a single game and consists of four strings where each string represents a player's hand. Each pair of strings is separated by a single space. Each character in a string is either A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, or K.

Name of Data File: pr93.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should indicate the winning hand in the corresponding input data collection. Specifically, your program should output 1, 2, 3, or 4 depending on if the first, second, third, or fourth hand wins the game, respectively. Thus, each line of output will consist of only a single number from 1 to 4.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: There will be exactly one winner for each hand (no ties).
Each hand will consist of between 2 and 10 characters, inclusive.

Sample Input:

```
3
T8 456 24Q JK2
362 J2 AAAA K3
A9AA A23456 JK QT
```

Problem 9.3 Twenty-One

Sample Output:

```
1
3
2
```

Problem 9.4 String Along with Subsequences

General Statement: Given a string *S1* and another string *S2*, *S2* is considered to be a subsequence of *S1* if it can be formed by selecting any set of characters in *S1* from left to right without rearranging their order. For example, the string "`care`" is a subsequence of "`character`" since it can be formed using its 1st, 3rd, 4th, and 8th letters. As an alternative, "`reach`" is not a subsequence of "`character`" since it cannot be formed from its letters without rearranging them.

Frequently, it is possible that one string is a subsequence of another in multiple ways, meaning that it can be formed using two or more different sets of letters from the larger string. For example, the string "`car`" can be formed from "`character`" using the 1st, 3rd, and 4th letters, the 1st, 3rd, and 9th letters, or the 1st, 5th, and 9th letters. Thus, "`car`" is a subsequence of "`character`" in 3 different ways. Your task is to find the total number of different ways a word is a subsequence of another.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of two strings *S1* and *S2*. Each string will be separated by a single space.

Name of Data File: pr94.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should contain a single integer representing the number of different ways *S2* can be formed as a subsequence of *S1*.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Both *S1* and *S2* will consist of only lowercase letters.
Both *S1* and *S2* will have between 1 and 20 characters, inclusive.
S1 will be longer than *S2*.

Sample Input:
```
3
character car
banana ana
finals fail
```

Sample Output:
```
3
4
0
```

Problem 9.5 Earth 2 (page 1 of 2)

General Statement: For a long time, scientists have known about the existence of Earth 2, a planet very similar to Earth. As the Earth orbits the sun, Earth 2 moves so that it is always located on the sun's opposite side. The result is that we can never see Earth 2, but we know it's there!

Earth 2 is just like Earth, except for one small difference. Each person on Earth 2 has an extra finger on his or her right hand. This results in them using a different system for representing numbers than us on Earth. While we represent our numbers using the symbol sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, people on Earth 2 represent their numbers using the symbol sequence 0, 1, 2, 3, 4, 5, #, 6, 7, 8, and 9.

The above difference means that numbers on the two planets are represented differently. The following table lists Earth's equivalent for the first 40 whole numbers in Earth 2's system

Earth 2	1	2	3	4	5	#	6	7	8	9	10	11	12	13	14	15	1#	16	17	18
Earth	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Earth 2	19	20	21	22	23	24	25	2#	26	27	28	29	30	31	32	33	34	35	3#	36
Earth	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

Your task is to write a program that translates an Earth 2 number into the equivalent number in our world.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection is a string of characters representing a whole number in Earth 2's system.

Name of Data File: pr95.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should contain a single integer representing the translated value of the corresponding input Earth 2 number.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each input string will contain between 1 and 8 characters.
Each input string will be a valid Earth 2 whole number.

Problem 9.5 Earth 2

Sample Input:

```
4
29
#00
5
#
```

Sample Output:

```
32
726
5
6
```

Problem 9.6 Final Examination (page 1 of 2)

General Statement: Normally, there are many components that make up a student's final grade in a class such as homework assignments, midterm exams, and, of course, a final exam. Each component is worth a varying percentage of the total grade. For example, the homework assignments may be worth 25% of the course grade, and the midterm exams may be worth 40%. So, the total course grade is computed as a weighted average of all of the grade components.

The final exam is normally the last component of the course grade. So, a student typically knows all of the scores he or she has made on the other course components before taking the final exam. This implies that the student should be able to calculate what score is needed on the final to obtain a desired grade.

Given a set of course components along with their respective weights, your task is to write a program that computes the minimum integer score that a student needs to obtain on the final exam in order to receive a 70% average for the total course grade.

Input: The first line of the input is an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection consists of four integer numbers *H h M m* with each pair of numbers separated by a single space. The first integer *H* represents the score obtained for the homework assignments. The second integer *h* represents the weight of the homework assignments. The third integer *M* represents the score obtained for the midterm exams. The fourth integer *m* represents the weight of the midterm exams. All four integers represent percentages.

Name of Data File: pr96.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should contain an integer which is the minimum final exam score needed by a student to obtain a total course grade of at least 70%.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: Each integer in the input data collections will be between 0 and 100, inclusive.
The weight of the final exam computed as 100 – (*h* + *m*) will be a positive integer.

Problem 9.6 Final Examination

Sample Input:

```
3
80 25 90 25
75 45 65 45
0 0 0 0
```

Sample Output:

```
55
70
70
```