University Interscholastic League

Computer Science Competition

2013 District 2 Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Names of Problems

Number	Name
Problem 1	Bus
Problem 2	Cheese Party
Problem 3	Concentric Squares
Problem 4	Eagle Nests
Problem 5	Escape!
Problem 6	EUREKA!
Problem 7	Old Folks
Problem 8	Password
Problem 9	Patterns
Problem 10	Possession
Problem 11	Projectiles
Problem 12	Spellcheck

1. Bus

Program Name: Bus.java **Input File: bus.dat**

The City of Silliness has purchased a fleet of new busses that are so skinny that people must stand in them single file and cannot walk around each other.

- The busses have two doors, one in the very front by the driver, and one in the very back.
- New passengers must enter through the front to pay their fare, but can exit the bus via the front or the back door.
- Passengers always enter the bus in the order listed. When passengers exit the bus to let other passengers exit, the re-entering passengers will always enter in the reverse order they exited so that their original order is maintained.
- Assume that everyone on the bus will want to inconvenience the fewest number of people when they exit.
  - A passenger can exit either the front door or the back door, whichever is closest to them.
  - If somebody needs to exit the bus that isn't the first or last person then people in front or behind that person will need to get off the bus to allow him or her to exit, then they must get back on to continue the trip to their stops. Re-entering passengers must maintain their original order.
  - If more than one passenger needs to exit at a stop, they can all exit the front door or the back door, or some can exit the front and others the back, whichever is most convenient for the other passengers. Then the other passengers will re-enter the bus, maintaining their original order.
- While this method is very tedious for people on the bus, this new model saves the city a lot of money!

The city is interested to know how many people walk in and out of the doors on a given bus route, whether they are getting on the bus for the first time, getting off the bus at their exit, or having to exit then re-enter because somebody needs to leave.

- A person getting on the bus counts as 1 enter/exit, and a person getting off at their stop also counts as 1.
- If somebody has to get off and then back in to let somebody out, both the exit and enter are counted, so the value is incremented by 2.

Input

The input file will contain an unknown number of lines:

- The file will not contain any blank lines.
- The file may contain multiple routes.
- Each line will be of the form:
  - `ON` followed by a space and a space-delimited list of upper case ASCII characters denoting the people getting on the bus at a particular point in the route.
  - `OFF` followed by a space and a space-delimited list of upper case ASCII characters denoting the people getting off the bus at a particular point in the route (these people are guaranteed to already be on the bus).
  - `END ROUTE` – This means the current route is ended, any remaining people on the bus must exit, and the data should be calculated and printed out.
- The last line is guaranteed to be `END ROUTE` so that no bus route is left in limbo.

Output

For each route you will print a single line containing a string in the form `People Entered/Exited: N`, where N is the number of exits/enters during the given route.

Example Input File

```
ON A B C
OFF C
ON D E F G
OFF E B
END ROUTE
```

Example Output to Screen

```
People Entered/Exited: 18
```

2. Cheese Party

Program Name: Cheese.java Input File: cheese.dat

Jonathan has organized a cheese tasting party and invited six of his cheese loving friends to bring two of their favorite cheeses to share. He has also developed a list of secret words having to do with cheese. Each time one of his guests says one of the secret words, conversation is stopped and the guest is allowed one roll of two standard 6 sided dice with numbers 1-6 on the sides of each die. The guest that rolls the dice will receive the total number points indicated on the dice. At the end of the night, the guest with the largest number of points will win the door prize – a chunk of Jonathan's favorite cheese. Jonathan wants you to write a simulation that will help him get a feel for the outcome of his little game.

For each simulation, you will need to construct one object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

For each simulation, you will need to generate three random numbers for each secret word uttered:

- The first random number `r1` will stand for a letter between `A` and `F`, inclusive, and will represent each of his six guests. This guest will roll the dice and receive points.
- The second and third random numbers `r2` and `r3` will each be a number between 1 and 6 inclusive and will determine the results of the roll of the two dice.

Input
The first line of input will contain a single integer `n` that indicates the number of simulations to follow. Each of the following `n` lines will contain a positive `long` that will be the seed for the random object in the simulation, followed by a space and an integer indicating the number of secret words that will be uttered.

Output
For each simulation, you will print each of the guests, `A` through `F`, and the number of points each received.

Example Input File

```
2
123543225422 20
546745453 25
```

Example Output to Screen (**Note:** Alternate sets of output are bolded for ease of reading.)

```
A 23
B 39
C 0
D 35
E 26
F 21
```
A 24
B 37
C 15
D 16
E 45
F 27

Random numbers generated:

```
123543225422
1 5 2 5 3 4 5 6 5 4 1 2 1 1 5 1 3 5 3 1 3 4 6 6 3 6 3 1 5 5 0 1 2 5 2 1 3 3 4
3 6 4 4 5 6 3 3 2 0 1 1 0 5 5 1 5 3 0 3 5
546745453
0 2 3 4 5 3 3 1 1 5 5 6 0 3 5 1 2 5 3 2 5 4 6 2 1 5 3 5 2 5 0 3 2 4 3 4 4 4 4
1 3 1 5 1 1 2 2 6 1 1 2 1 3 6 4 1 6 3 1 6 4 5 2 0 2 4 5 2 5 2 3 4 1 4 2
```

3. Concentric Squares

Program Name: CSquares.java **Input File: csquares.dat**

James is writing an application that involves a maze. He needs you to write a program that will draw as many concentric squares as possible around a given point, called the center, using the following method:

- The center of the maze will be a single cell and will contain a space.
- Around the center, there will be a square of cells that each contain an asterisk (`*`)
- Around the first square of asterisks, there will be a square of spaces.
- The space squares and the asterisk squares will continue to alternate as long as there is room to the left of the center of the maze.

James will then use your squares to create his maze.

Input
The first line of input will contain a single integer `n` that indicates the number of mazes you are to draw. Each of the next `n` lines will contain a positive even integer `r` ($2 \leq r \leq 20$), where `r` is the row and column that contains the center of the maze.

Note: The coordinates of the upper left corner of the maze are `1 1`.

Output
For each line of input, you will print the maze as described above. Print at least one blank line after each maze.

Example Input File
```
1
6
```

Example Output to Screen
```
**********
*        *
* ****** *
* *    * *
* * ** * *
* * * * *
* * ** * *
* *    * *
* ****** *
*        *
**********
```

Notes:

- The outside square will always begin in column one and will always contain asterisks.
- A blank line at the end of the output is optional.

4. Eagle Nests

Program Name: EagleNests.java **Input File: eaglenests.dat**

Hermit Joe lives in a desolate area of Alaska. The Alaskan Wildlife Management team has given him several maps of the area surrounding his cabin that show the location of the homes of different types of animals. The characters on the map represent different animals that live in the area. For example, the letter `E` represents an eagle's nest, the letter `B` represents a brown bear den, the letter `M` represents a moose habitat, etc. Joe especially loves to photograph the eagles in the area. You are to write a program that, given a 10x10 rectangular grid of the area, will output the coordinates of the eagles' nests in the area.

Input
The first line of input will contain a single integer `n` that indicates the number of 10x10 maps to be checked. For each map, there will be ten lines with ten characters and no spaces on each line. All characters will be either an uppercase letter of the alphabet that represents an animal's home or a period (.) that represents an area without a major animal home.

Output
For each map, you will print on a single line the coordinates `x y` of each eagle's nest on the grid. The locations will be listed in row-column order which means you will print all the nests in row one, in order from left to right, followed by the nests in row 2, etc. Print a blank line after the results of each map (blank line after the last map is optional).

Note: The top left cell is considered to be at `1 1` and the bottom right cell is considered to be at `10 10`.

Example Input File

```
1
E...E..B.M
...B....MM
EECCC.BBMM
..CCCFFBMM
C..F..B..M
B...B..FF.
....FFFFMM
CC..B..MMM
BCCFF...FB
..CC.FF.B.
```

Example Output to Screen

```
1 1
1 5
3 1
3 2
```

5. Escape!

Program Name: Escape.java **Input File: escape.dat**

Your paranoid friend is about to go camping and is worried about the various animals that inhabit the forest. He wants you to write a program that will help him figure out his survivability chances in various scary scenarios so that he will be prepared if the unthinkable happens. To survive a given scenario he has to be able to make it to an escape position before he is attacked by a vicious squirrel.

You have designed your program so the woods are represented by rectangular grid composed of the following characters:

- `S` – a vicious squirrel
- `@` – a tree (only squirrels can move onto these squares)
- `#` – obstruction that is impassable by all
- `.` – empty space (anybody can move here)
- `P` – your paranoid friend
- `E` – Escape position

Your program will work as follows:

- Your friend and the squirrel will alternate taking turns moving, with your friend moving first.
- Your friend's moves:
  - Your friend can move in any direction (horizontally, vertically or diagonally), but he cannot go back to a square he has been to before.
  - Your friend escapes when he gets to an escape space on any turn.
- The squirrel's moves:
  - The squirrel can move in any direction (horizontally, vertically and diagonally) and *can* go back to squares it has been to before
  - The squirrel will always move towards your friend.
  - If your friend's move is to the same row or column as the squirrel, the squirrel will move one square towards your friend, in that row or column.
  - If your friend's move is not to the same row or column as the squirrel, the squirrel will move diagonally toward your friend.
  - Squirrels have no problem moving into an escape space.
  - If the move the squirrel wants to make is into an obstacle, it will stay put and look frustrated.
  - If it is the squirrel's turn to move and the squirrel is adjacent to your friend, your friend gets attacked.
- If your friend cannot move on his turn because all adjacent spaces have been visited, are obstacles or contain the squirrel then it is considered an animal attack as the squirrel will eventually find your friend, lying on the ground in the fetal position.

Input
The first line will contain a single integer `n` that indicates the number of escape grids to follow. For each escape grid, the first line will contain 2 integers in the form `r c`, where `r` is the number of rows in the grid and `c` is the number of columns ($0 < r, c \leq 5$). The next `r` lines will contain `c` characters as described above with no spaces.

Note: Each grid is guaranteed to have at least one escape space, one version of your paranoid friend and one squirrel.

Output
For each grid you will output `YES` if there is any chance of escape or `NO` if there isn't.

Example Input File (cont. on next page)

5. Escape! (cont.)

Example Input File

```
3
6 5
@@...
..P..
.....
@@...
S....
@...E
4 4
E..P
.@@.
..S.
E...
4 4
E..P
.##.
..S.
E...
```

Example Output to Screen

```
YES
NO
YES
```

6. EUREKA!

Program Name: Eureka.java **Input File: eureka.dat**

You are to write a program that will generate the number of capital letters necessary to write a given word.

For each word you are to write, you will need to construct one object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

For example, if you are to write the word `EUREKA`, you will generate random letters `A-Z` until you have found the letters, in order, that spell the word `EUREKA`. Then, you will report the number of letters you had to create to spell the word.

Input
The first line of input will contain a single integer `n` that indicates the number of words to follow. Each of the following `n` lines will contain a `long` to be used as the seed for the random object followed by a space and a single word containing only uppercase letters of the alphabet.

Output
For each word, you will print the number of letters that you had to generate to complete the given word.

Example Input File

```
2
435673456 EUREKA
345664322234 FOOTBALL
```

Example Output to Screen

```
164
445
```

Random letters generated for each seed:

```
435673456
I G F J U Y M L P G X E F Z R J U A M X F H P X M A S A T I S H Y G N O B N F
B P M F E X B B T B V N H H R Q L P U K L P I R I M O I K H P V T C X M G S C
S L C P A M Z S S G Q C N Y O L O B R N Z B L D F G B S S C K Q Y Q I U N U D
D Y P F I T Y O E A D T H C V Q J X C K C P W N S Q C G M E J D P Y V P M K U
X W L C B H N A
345664322234
X O V A P C M Y B E Q O E P D M E V W Z V J I B P D P G C Z B S N O R D M R M
O W L T V C V X P M G H G E P B G E J W A Z O Z G H N L K M L F H R R J B W Z
U U G D A B N T S L F X F U K C Y R K T K A H H A I A H J L C I T R C Z C Z O
J Q I S T N Q Z K V L X Y H Y X D U D X J L G Y N H K R Y C S X V P H M N G L
W D P A S B O F Y Z C Y P B J E P E C M V U U S E X O E R E H Y N Y V D X Q X
E I N F O A M J O Q O B L U H G F S Z Z X N N N F U J W E B E A G Z S A G T E
T T V U E S B N Z U B V H G O P Y W V V J P K N H G D Z B S O H P E C C I W Z
P U O N M B Y L P W Y T J X K S Z V M F H N V J F N I O P T I O G H D I C L R
H K R T R T J F Y G R J V M E R T W V Z O Y Y N I I L L R Z D B Y L X T P M K
S V I J M S T I P C T L Y L P J T Y W D X C I M H F U W Y D V H M X Z C B V R
C X J J L Q T T A B C X B V X M K H Y T Q V V I N B E V B T E Q W A V Y H M O
V O B D Q G Z P C L E V P C K L
```

7. Old Folks

Program Name: OldFolks.java **Input File: oldfolks.dat**

According to one precocious four year old, people older than a certain age are considered to be "old folks". Being four years old, she has no real concept of age but thinks that anyone older than her grandparents are old but her grandparents, who were born in 1953, are not old.

You are to write a program that, given a list of people's names and their corresponding birth years, will print the names of the "old folks" born after 1953.

Input
The first line of input will contain a single integer `n` that indicates the number of people in the list. Each of the next `n` lines will contain a person's full name followed by a space and the four digit year in which they were born.

Output
In the order they appear in the original list, you will print the names of the "old" folks.

Example Input File

```
6
Joe E Lewis 1945
Mary Smith 1999
Anne Scott 1943
Larry E Jones 1964
Roger J Allen 2003
Virginia V Alvarez 1950
```

Example Output to Screen

```
Joe E Lewis
Anne Scott
Virginia V Alvarez
```

8. Password

Program Name: Password.java **Input File: password.dat**

Keith is sending passwords to sections of his website to various friends and does not want them to be intercepted by any hackers. Therefore, he has decided to send them in "code". He will send a string of letters containing the password that his friend will use to access his website as well as the length of the password. Unfortunately, the password is embedded in the string more than once. The desired password is the string of the given length that appears in the input string the greatest number of times. You are to write a program that will extract the password from the given string.

Note: The string will contain exactly one unique password.

Input
The first line of input will contain a single integer `n` that indicates the number of passwords to be sent. Each of the following `n` lines will contain a single string of lower case alphabetic characters followed by a space and an integer greater than 2 which denotes the length of the password to be found.

Output
For each string input, you will print the embedded password.

Example Input File
```
2
abccababccabc 3
treatreataaeesaaeesatreatreaterarederaeesarafea 5
```

Example Output to Screen
```
abc
treat
```

9. Patterns

Program Name: Patterns.java **Input File: patterns.dat**

Given two strings of characters, you are to write a program that determines if the patterns are the same. Two patterns are considered to be the same if each unique character in string one is replaced with a corresponding unique character in string two. For example, `abc44cd` and `defzzfx` have the same pattern since each `a` was replaced with a `d`, each `b` was replaced with an `e`, each `c` was replaced with an `f`, each `4` was replaced with a `z`, and each `d` was replaced with an `x`.

Input
The first line of input will contain a single integer `n` that indicates the number of pairs of strings to follow. Each of the following `n` lines will contain two strings of equal length that will be separated by a space. Each string will contain at least two characters.

Output
For each pair of strings input, you will print `YES` if the strings have the same pattern or `NO` if they do not.

Example Input File
```
4
abcdef w3455
rr555bbc ttfffccb
RRRsssTTss77 333YYYvvYYaa
gggg9999rrrr4444 aaaabbbbaaaabbbb
```

Example Output to Screen
```
NO
YES
YES
NO
```

10. Possession

Program Name: Possession.java **Input File: possession.dat**

The time that a football team possesses the ball is one of the most important statistics in football. The longer a team possesses the football, the more opportunities they have to score and the fewer opportunities their opponent has to score. The back-up quarterback is frequently assigned the job of keeping this important statistic.

There are four quarters in a football game and the time clock starts at 15:00 (15 minutes, 0 seconds) at the beginning of each quarter and counts down to 0:00 (0 minutes, 0 seconds). Every time his team gets the ball, the back-up quarterback records the minutes and seconds left in the quarter. Similarly, each time his team loses possession of the ball, he records the minutes and seconds left in the quarter.

You are to write a program that he can use to compute the time of possession for his team.

Input
The first line of input will contain a single integer `n` that indicates the number of games to follow. Each of the next `n` lines will have the name of the team, without spaces, followed by one or more data sets in the form `Qx mm:ss Qy mm:ss`. The quarter that his team began possession of the ball is denoted by `x` and is followed by the number of minutes and seconds left in that quarter when his team gained possession of the ball. The quarter his team lost possession of the ball is denoted by `y` and is followed by the number of minutes and seconds left in that quarter when his team lost possession of the ball.

Output
For each game, you will print his team's name and the number of minutes and seconds that his team possessed the ball during the ballgame, as shown below.

Example Input File
```
2
UT_Austin Q1 15:00 Q1 5:13 Q2 10:45 Q2 7:33 Q2 4:12 Q3 7:33 Q4 10:27 Q4 4:00
TX_Tech Q1 10:55 Q2 12:34 Q2 4:23 Q2 0:00 Q3 15:00 Q3 7:12 Q3 2:01 Q4 9:12 Q4
7:55 Q4 4:59
```

Note: The second input line in the printed version above stretches across two lines, but is actually only one line in the `possession.dat` input file.

Example Output to Screen
```
UT_Austin 31:05
TX_Tech 36:17
```

11. Projectiles

Program Name: Projectiles.java **Input File: projectiles.dat**

Mr. Appleworth is a Physics teacher and is teaching his students about projectiles and parabolas. His students will be constructing projectiles to shoot off the top of the football stands and into the woods. He wants them to create a simulation of the flight of their projectile prior to conducting the actual experiment.

The height of a given object projected into the air at a given time is given by the equation $H = V * T - 16 * T^2 + c$, where V is the initial velocity, T is the time in seconds since the projectile was fired, and c is the initial height of the object. He has asked you to write a program that will find highest integer height that the object will reach and the time in seconds that the object will first reach that height.

Input
The first line of input will contain a single integer `n` that indicates the number of simulations to follow. Each of the following `n` lines will contain two integers in the form `V c` representing the values of the Velocity and the initial height respectively of the equation that is in the form of the equation above. Each of items will be separated by a space.

Output
For each equation, you will print `HEIGHT h TIME t`, where *h* is the maximum integer height the object reached and *t* is the time in integer seconds that it first reached that height.

Example Input File
```
3
120 20
200 12
250 10
```

Example Output to Screen
```
HEIGHT 244 TIME 4
HEIGHT 636 TIME 6
HEIGHT 986 TIME 8
```

12. Spellcheck

Program Name: Spellcheck.java **Input File: spellcheck.dat**

Your boss has asked you to write a spellcheck algorithm for your company's text editor. Your program will contain a dictionary of words that will be used to check input words for typos and will output the dictionary word that matches based on the algorithm. An input word matches a dictionary word if, for each location in the input string, that character either matches exactly the character at the same location in the dictionary word or if the input character is adjacent to the dictionary character in the same string location on a standard US keyboard. Obviously, if a word matches a dictionary word exactly, then you shouldn't correct it to a similar dictionary word whose differing characters are adjacent (left, right, above or below) on the keyboard. For example, the word `SNELL` should be corrected to `SMELL` since `N` and `M` are adjacent on a keyboard, but `COG` shouldn't be corrected to `DOG` if both words are in the dictionary.

You will only care about uppercase characters `A` through `Z`. There will not be any spaces or non-alphabet characters.

Input
The first line of input will contain a single integer `n` that indicates how many dictionary words will follow, one per line. After the dictionary words there will be another line that contains a single integer `m` that indicates how many input words will follow, one per line. You can assume that input words will correct to match at most one dictionary word.

Output
For each input word you will output the dictionary word it corresponds to, or the word itself if no suitable word can be found in the dictionary.

Example Input File
```
4
CAR
SMELL
MONKEY
FEATHER
5
FAR
SPELL
SNELL
NONKRY
FEATHEG
```

Example Output to Screen
```
CAR
SPELL
SMELL
MONKEY
FEATHEG
```