5. Ren

Program Name: Ren.java **Input File: ren.dat**

Ren wants to help out the tabulation of many of the UIL event scores. For many events, the scoring is calculated where the competitor earns 6 points for each correct answer and is penalized 2 points for each incorrect answer. Ren needs to read in each student's first and last names as well as how many correct and incorrect responses each student earned. Ren needs to output the top 6 students and their scores.

Input:
The data file will be formatted as follows: an unknown number of lines, each containing a first name, a last name, an integer representing the number of correct responses, and an integer representing the number of incorrect responses, in that order.

Output:
The first line will consist of 30 dashes (-)
The title line will contain the word `Name` beginning in the 4th column of the line and will have the word "`Score`" right-aligned at the end of the 30th column.
The next 6 lines will consist of a place number followed by a period (`.`) and a space. Next comes the student's name (`Last, First`). Each line will conclude with the student's score, right-aligned ending in the 30th column.
The final line will consist of 30 dashes (-).

If there are ties, order the students with the same scores in alphabetical order by their last name. Additionally, in such a case, do not worry about changing the place numbering to reflect the numerical tie.

Sample input:
```
Aarav Patel 12 18
Sofia Rodriguez 5 25
Wei Chen 20 15
Amara Okafor 10 10
Liam O'Connor 8 22
Fatima Al-Fayed 15 20
Mateo Silva 30 5
Yuki Tanaka 18 12
Elena Ivanov 22 8
Kofi Mensah 14 16
```

Sample output:
```
------------------------------
   Name                 Score
1. Silva, Mateo           170
2. Ivanov, Elena          116
3. Chen, Wei               90
4. Tanaka, Yuki            84
5. Mensah, Kofi            52
6. Al-Fayed, Fatima        50
------------------------------
```

4. Dillen

Program Name: Dillen.java **Input File: dillen.dat**

Dillen enjoys riding his motorcycle far too fast, and the local police department has begun issuing tickets based on multiple factors. For each recorded incident, you will be given the posted speed limit, Dillen's actual speed, whether the incident occurred in a school zone, whether it occurred in a work zone, and whether workers were present in that work zone. Your program must determine either the final dollar amount of the ticket or if Dillen's license should be suspended.

If Dillen's speed is less than or equal to the posted speed limit, then he was not speeding and no ticket was issued.

If Dillen was speeding, let O be the number of miles per hour he was over the speed limit. A base fine is first assigned using the following rules: if O is between 1 and 9 inclusive, the base fine is $60; if O is between 10 and 19 inclusive, the base fine is $120; if O is between 20 and 29 inclusive, the base fine is $240; and if O is 30 or more, the base fine is $500.

If Dillen's actual speed is 100 miles per hour or greater, he is considered a super speeder and an additional $200 is added to the base fine before any other calculations are performed.

After the base fine and possible super speeder surcharge have been applied, a zone multiplier is applied. If the incident occurred in a school zone, the fine is multiplied by 2. If the incident occurred in a work zone without workers present, the fine is multiplied by 2. If the incident occurred in a work zone with workers present, the fine is multiplied by 3. If the incident occurred in both a school zone and a work zone, both multipliers apply and the fine is multiplied accordingly. If the work zone flag is listed as false, the workers-present value should always be treated as false regardless of the input.

After all multipliers have been applied, if the current fine is greater than $1000, an additional reckless driving penalty of $150 is added to the total.

In addition to the fine, demerit points must also be calculated. If O is between 1 and 9 inclusive, 1 point is assigned; if O is between 10 and 19 inclusive, 2 points are assigned; if O is between 20 and 29 inclusive, 3 points are assigned; and if O is 30 or more, 4 points are assigned. One additional demerit point is added if the incident occurred in a school zone, and one additional demerit point is added if the incident occurred in a work zone with workers present. If the total number of demerit points for an incident is 6 or greater, Dillen's license is suspended for that incident and no fine is to be displayed. For each incident, output No ticket if Dillen was not speeding. Otherwise, if the total demerit points are 6 or more, output License suspended. If neither of those cases applies, output the final ticket amount in the format $X, where X is the total fine in dollars.

Input:

The input will begin with a single integer n ($1 \leq n \leq 1000$), representing the number of recorded incidents. Each of the next n lines will contain five values separated by spaces. The first value is the posted speed limit L ($1 \leq L \leq 80$). The second value is Dillen's recorded speed S ($0 \leq S \leq 200$). The next three values are characters (Y or N) representing, in order, whether the incident occurred in a school zone, whether it occurred in a work zone, and whether workers were present.

Output:

For each incident, output exactly one line containing either No ticket, License suspended, or the dollar value of the ticket in the format $X.

Sample input:

```
6
35 35 N N N
30 38 Y N N
60 75 N Y Y
55 67 N Y N
40 72 Y Y Y
45 90 N N N
```

Sample output:

```
No ticket
$120
$360
$240
License suspended
$5
```

6. Kiran

Program Name: Kiran.java **Input File: kiran.dat**

Kiran is the stats-guru for the esports club. The club is hosting a tournament and Kiran needs to be able to quickly provide a full report for each team based on the raw data provided (kills and deaths) on each individual team member. The report will need to calculate the Kill/Death Ratio for each player and the total kills for the team. The Kill/Death ratio is calculated as kills / deaths. In the event of the "perfect game" (a player has 0 deaths), their Kill/Death ratio cannot be calculated normally (division by zero). In this case, their ratio is equal to their kills. If their ratio >= 2.0, print `(MVP)` next to their name.

Input:
The data file will be formatted as follows: a collection of lines starting with a team name (a single string). The next line is an integer (`n`) which is the number of members on that team, followed by `n` lines with a gamer tag, number of kills, and number of deaths

Output:
Each team report gets a full report as follows: The first line is "team report" in all capital letters followed immediately by a colon (`:`) and a space, and finally the name of the team. The next line will consist of 48 dashes (`-`). The next line will be a title line consisting of the string "`Player`", 9 spaces, the letter "`K`", 5 spaces, the letter "`D`", 5 spaces, and the string "`Ratio`". The next `n` lines will consist of the player name left-aligned in the first 15 columns, the number of kills left aligned in the next 6 columns, the number of deaths left aligned in the next 6 columns, and the calculated ratio in the next 8 columns formatted to 2 decimal places. The next line will consist of 48 dashes (`-`). The final line for the team report is the words "`Total Team Kills`" followed by a colon, a space, and the total number of kills from the entire team. One blank line will follow as a separator between each team's report.

Sample input:
```
CyberKnights
3
Blade 12 4
Shadow 5 5
Tank 2 8
PixelWarriors
2
Glitch 15 0
LagSpike 4 10
```

Sample output:
```
TEAM REPORT: CyberKnights
------------------------------------------------
Player          K       D       Ratio
Blade (MVP)     12      4       3.00
Shadow          5       5       1.00
Tank            2       8       0.25
------------------------------------------------
Total Team Kills: 19

TEAM REPORT: PixelWarriors
------------------------------------------------
Player          K       D       Ratio
Glitch (MVP)    15      0       15.00
LagSpike        4       10      0.40
------------------------------------------------
Total Team Kills: 19
```

7. Johnson

Program Name: Johnson.java **Input File: johnson.dat**

You have two new friends, both named Johnson. They own a pharmaceutical company, but you forget the name. They have a very interesting method of writing emails to their employees. They each will write exactly 29 characters, then the other will write 29 characters, and so on and so forth. The issue is, when doing this, they struggle to keep track of brackets, parentheses, and braces. They need you to write a program to check if their emails have valid structure when it comes to brackets, braces, and parentheses, i.e. for every opening symbol there is a matching closing symbol. Also, each open symbol must be followed next by either another opening symbol, or a closing symbol of the same type. There can be any other characters in between open and closing symbols, but any other opening and closing symbols will render the expression invalid.

Input:
The input will begin with one integer, `n` (`0 < n <= 1000`). Each test case will consist of one line, containing an expression made up of any alphabetic characters, spaces, punctuation, and braces, parentheses, and brackets.

Output:
If the given string is a valid format, output the string "`Johnson and Johnson`", otherwise output the string "`Invalid Johnsons`".

Sample input:
```
3
[{()}]
([{]})
Hello there (I like {this or [that]} one), but [someone] told me not to.
```

Sample output:
```
Johnson and Johnson
Invalid Johnsons
Johnson and Johnson
```

8. George

Program Name: George.java **Input File: george.dat**

George works as a truck driver for a regional delivery company. Each day he is given a list of roads that he is allowed to use while making deliveries. Unfortunately, his phone dies and his navigation system stops working. George still has the company's list of roads, but now he needs your help to determine the shortest route from his starting location to his destination.

Each road connects two locations, has a distance in miles, and has a maximum weight limit that it can safely support. Because George's truck is fully loaded, he may only travel on a road if the weight of his truck does not exceed the road's weight limit. All roads are bidirectional. If there is no valid route that George can take using only allowed roads, then no delivery is possible.

Input:
The first input line will contain a single value N ($1 \leq N \leq 50$) denoting the number of delivery scenarios.

Each scenario will begin with two integers R and W, where R ($1 \leq R \leq 500$) is the number of roads listed for that scenario, and W ($1 \leq W \leq 1000$) is the weight of George's loaded truck in tons.

The next line will contain two strings, start and end, representing the names of the starting location and the destination.

The following R lines will each contain four values A, B, D, and L, where A and B are the names of two locations connected by the road, D ($1 \leq D \leq 10{,}000$) is the distance in miles of the road, and L ($1 \leq L \leq 1000$) is the maximum truck weight that the road can support.

George may only use a road if $W \leq L$. You may assume that each scenario will contain at most 200 unique location names.

Output:
For each delivery scenario, output exactly one line.

If George can reach his destination using only valid roads, output: Shortest distance: X, where X is the minimum total distance in miles required to reach the destination. If George cannot reach the destination using only valid roads, output: No route possible

Sample input:

```
3
8 5
DEPOT YARD
DEPOT A 5 10
A B 4 8
B C 3 6
C YARD 5 12
A D 10 20
D YARD 4 20
B E 2 4
E YARD 7 4
8 9
DEPOT YARD
DEPOT A 5 10
A B 4 8
B C 3 6
C YARD 5 12
A D 10 20
D YARD 4 20
B E 2 4
E YARD 7 4
8 15
DEPOT YARD
DEPOT A 5 10
A B 4 8
B C 3 6
C YARD 5 12
A D 10 20
D YARD 4 20
B E 2 4
E YARD 7 4
```

Sample output:

```
Shortest distance: 17
Shortest distance: 19
No route possible
```

9. Golga

Program Name: Golga.java **Input File: golga.dat**

Golga is a mechanical engineer – a clearly less superior choice of profession compared to software engineering – and due to their poor choice of major, now needs your help in solving a problem of theirs. Namely, Golga has a set of n gears, each with a specific tooth size and number of teeth, and at most n axles that she can use and is interested in forming the largest gear ratio he possibly can. Given two gears, A and B, she can perform two actions:

1. If A and B have the same tooth size (i.e., $S_A = S_B$), then Golga can either place A and B on two separate axels, and enmesh them at their teeth, or can place A and B on the same axel.
2. If A and B do not have the same tooth size (i.e., $S_A \neq S_B$), then Golga can only place A and B on the same axel.

Golga – with their limited knowledge base – has informed you that the gear ratio between gears A and B is equal to

$$R = \frac{T_A}{T_B}$$

Where T denotes the number of teeth, A is the gear that is being driven (output), and B is the gear that is driving (input). Golga has also informed you that this only holds if $S_A = S_B$ and A and B are on different, but enmeshing axels. You may assume that all gears that are on the same axel will rotate at the same rate and that one axel of your choice will be spun at a constant rate of one revolution per second. Help Gogla determine what the largest gear ratio she can form, given this setup.

Input:
The first line of input will consist of a single integer m ($1 \leq m \leq 10^2$) denoting the number of test cases to follow. Each test case will begin with a single integer n ($2 \leq n \leq 5 \cdot 10^4$) denoting the number of gears and the maximum number of axels that Golga has at their disposal. The following n lines will each consist of two single-space-separated integers S_i and T_i denoting the size of the teeth and the number of teeth that the i^{th} gear has. It is also guaranteed that among all test cases, $1 \leq S_i \leq 10^5$, $3 \leq T_i \leq 10^5$ will hold.

Output:
For each of Golga's m requests, calculate the largest gear ratio possible. Since this number can be quite large, print the natural log of the gear ratio expressed to 8 decimal places of precision.

Sample input:

```
2
6
19 364
21 1023
19 66
19 242
21 807
19 675
4
33 10
33 27
44 10
44 27
```

Sample output:

```
2.97044519
1.98650355
```

Explanation:
The first test case, we have two sets of gears which can be enmeshed with one another. Those two sets are {(19,364), (19,66), (19,244), (19,675)} and {(21,1023), (21,807)}. By optimally enmeshing gears in the first set, we can get a gear ratio of 15.38317055, and doing the same for the second set, we get a gear ratio of 1.26765799. Then, by optimally choosing some axel to serve a gear from both sets, we get an optimal gear ratio of 19.50059906, which, when taken the natural log of, is 2.97044519.

10. Wyatt

Program Name: Wyatt.java **Input File: wyatt.dat**

Wyatt has recently completed his Bachelor's of Forestry degree, and he wants to test it out on some unkempt trails in a nearby national park. He needs you to write a program to determine if the latest map of the trail has a possible path through the forest. Each map will be made up of the following characters:

- `o` – Represents a large boulder on the map, cannot be traversed.
- `^` – Represents a tree on the map, cannot be traversed.
- `.` – Represents an open trail space on the map, can be traversed in one unit of time.
- `~` – Represents a space of river on the map, cannot be traversed.
- `T` – Represents the starting trailhead, where you begin.
- `P` – Represents the parking lot, where the trail ends.

Input:
The input will begin with one integer, `n` (`0 < n <= 1000`). Each test case will consist of one line, containing two space separated integers, `r` and `c` (`1 <= r, c <= 100`), denoting the number of rows and columns in the map given, respectively. A trailhead and parking lot are not guaranteed to appear on the map. You can only move around the map in the 4 cardinal directions, up, down, left, and right.

Output:
For each test case, if there is a possible path from `T` to `P`, output the string `"PHForestry"`, otherwise output `"No Can Do"`.

Sample input:
```
2
5 6
T..^^o
o^...^
~~~~..
o....^
P.^^^^
3 8
.T...oo.
~~~~~~..
^^P....o
```

Sample output:
```
PHForestry
No Can Do
```

11. François

Program Name: François.java **Input File: francois.dat**

François is an avid competitor and problem writer in programming contests and has recently thought of a new problem type that he thinks is going to be a real hit. François has identified that often problems require solvers to perform some algorithm on an input and produce an output, to show their understanding of a concept. However, this has its limitations in what kind of content and knowledge it can test on. Suppose a problem writer wanted to check if a solver understands how to program binary search: many languages have pre-defined functions that perform binary search, and so there is no way in knowing whether a participant has implemented their own version or simply proxied the existing built-in version. However, François has solved this issue through the use of object serialization!

Object serialization is the process converting an object's state into a format that can be stored or transmitted. JSON files are often used as an intermediate representation of different fields and values for serialized objects; however, this has its limitations as all data is transmitted in plain text. However, François has implemented a more primitive version of this which translates the object down to its raw bytes and then encodes it using a base 64 scheme.

To test whether François' new problem type is viable, he has asked you a simple request: given the serialized representation of a LinkedList of integers, print out the serialized representation of that LinkedList, but reversed. To help you with this, François has provided his implementation of his object serialization scheme for you to use.

Input:
The first line of input will consist of a single integer n ($1 \leq n \leq 10^3$) denoting the number of testcases to follow. The next n testcases will each consist of a single line that denotes the serialized LinkedList of integers. Each list L will have a length $1 \leq |L| \leq 8 \cdot 10^2$. It is guaranteed that all serialized object bytes are castable to the custom linked list class below.

Output:
For each of François' n requests, print out the serialized reversed-order LinkedList of integers.

Sample input: (*lines truncated at …*)
```
3
rO0ABXNyABBDdXN0b21MaW5rZWRMaXN0ULS/llaPaNACAAJMAARoZWFkdAAXTEN1c3RvbUxpbmtlZEx...
rO0ABXNyABBDdXN0b21MaW5rZWRMaXN0ULS/llaPaNACAAJMAARoZWFkdAAXTEN1c3RvbUxpbmtlZEx...
rO0ABXNyABBDdXN0b21MaW5rZWRMaXN0ULS/llaPaNACAAJMAARoZWFkdAAXTEN1c3RvbUxpbmtlZEx...
```

Sample output: (*lines truncated at …*)
```
rO0ABXNyABBDdXN0b21MaW5rZWRMaXN0ULS/llaPaNACAAJMAARoZWFkdAAXTEN1c3RvbUxpbmtlZEx...
rO0ABXNyABBDdXN0b21MaW5rZWRMaXN0ULS/llaPaNACAAJMAARoZWFkdAAXTEN1c3RvbUxpbmtlZEx...
rO0ABXNyABBDdXN0b21MaW5rZWRMaXN0ULS/llaPaNACAAJMAARoZWFkdAAXTEN1c3RvbUxpbmtlZEx...
```

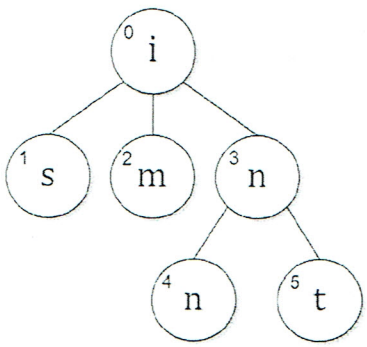
Explanation:
The first test case is the linked list $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow$ nil, so the reversed list is the linked list $5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow$ nil. The second case is the linked list $10 \rightarrow 3 \rightarrow 5 \rightarrow$ nil, so the reversed linked list is $5 \rightarrow 3 \rightarrow 10 \rightarrow$ nil. The last test case is the linked list $7 \rightarrow$ nil, so the reversed linked list is no different.

Important Notes:
There are many factors that affect what is output by the `marshall()` method that don't pertain to what the object logically represents. Because of this, you should avoid changing the names of existing methods/fields, creating additional `public` / `protected` methods (`private` is fine) and avoid creating any new fields as this affects what data is associated with the encoding of the object. Moreover, due to the way that Java handles dynamic memory allocation, all nodes created in your reversed linked list should be new objects (i.e., **NOT** references to `Node` objects from the original object). You may set the `toString()` method to simply return the empty string if you do not wish to type out the implementation.

12. Aline

Program Name: Aline.java **Input File: aline.dat**

Aline is secretly a bit of an obsessive counter and wants to make sure that she is perfectly aware of how many different words she knows. However, in her brilliant brain, she doesn't store these words as a list of words – that would be too space inefficient – instead, she stores them as a tree. Namely, given a tree T, consisting of a set of vertices V, where all $v \in V$ represents a single lowercase character, and a set of edges E, where all $e \in E$ takes the form (u, v), where $u, v \in V$, Aline knows that by selecting any two vertices in T, call them v_i, v_j, she can form a string $s = v_i v_l v_k \ldots v_j$. That is, string s is formed by concatenating all characters observed when traversing from v_i to v_j in tree T. Despite her brilliantly efficient storage technique, Aline has unfortunately forgotten how many unique strings taking the form of s exist in T and needs your help determining how many that is.



Input:
The first input will consist of a single integer n, $2 \leq n \leq 2 \cdot 10^3$, denoting the number vertices in V. The next line will consist of a string of n characters, the i^{th} of which denotes the character of vertex v_i in tree T. It is guaranteed that for all $v_i \in V$, $v_i \in$ {'a','b', ...,'z'} holds. The next $n - 1$ lines will consist of two single-space-separated integers, i and j, $i \neq j$, denoting that edges $e_1 = (v_i, v_j)$ and $e_2 = (v_j, v_i)$ exists in tree T.

Output:
Output a single integer: the total number of strings of the form s that can uniquely be formed in tree T.

Sample input:
```
6
ismnnt
0 1
0 2
3 0
3 4
5 3
```

Sample output:
```
34
```

Explanation:
Let S be the set of unique strings in T that can be constructed like string s. Then we have

$$S = \{\text{i, s, m, ... , t, is, im, in, nn, nt, si, mi, tn, ... , int, inn, sin, ... , sint, mint ...}\}$$

We can note that $|S| = 34$, and thus, that is our answer.