**Cypress Woods**

**Computer Science Competition 2016**

1. Problems may be solved in any order you choose. They do not have to be done in order from 1 to 18.
2. All problems are worth 40 points. Incorrect submissions will subtract 5 points from the points rewarded if the problem is submitted correctly. No points are subtracted if the problem is never submitted correctly.
3. There is no extraneous input. All input is exactly as specified in the problem.
4. Unless specified by the problem, integer inputs will not have leading zeroes. Your program should read to the end of file unless otherwise specified.
5. Your program should not print extraneous output. Follow the form exactly as given in the problem.
6. All programs must run under 2 minutes.

|  |  |  |
| --- | --- | --- |
| **Problem Number** | **Problem Name** | **Check Sheet** |
| 1 | 212 |  |
| 2 | Dance Party |  |
| 3 | Fight Club |  |
| 4 | Agent |  |
| 5 | Singing |  |
| 6 | Spencer |  |
| 7 | Agenda |  |
| 8 | Tetrimonium |  |
| 9 | Typing |  |
| 10 | Hacks |  |
| 11 | Letter |  |
| 12 | Dank Numbers |  |
| 13 | Homework |  |
| 14 | Knight |  |
| 15 | Will it Compile? |  |
| 16 | Space |  |
| 17 | Nick Needs A Ride |  |
| 18 | Grader |  |

**1. 212**

# Program Name: twotwelve.java Input File: None

Arami forgot what time it is…. again. So he has contracted you to create a program to remind him what time it is. Print out the elegant 212 shown below (Arami likes elegant things).

**Input**

None

**Example Output to Screen**

\_\_\_\_/\\\\\\\\\\_\_\_\_\_\_\_\_\_\_/\\\\_\_\_\_/\\\\\\\\\\_\_\_\_\_

\_\_/\\\///////\\\\_\_\_\_/\\\\\\\\_\_/\\\///////\\\\_\_\_

\_\///\_\_\_\_\_\_\//\\\\_\_\/////\\\\_\///\_\_\_\_\_\_\//\\\\_\_

\_\_\_\_\_\_\_\_\_\_\_/\\\/\_\_\_\_\_\_\_\/\\\\_\_\_\_\_\_\_\_\_\_\_/\\\/\_\_\_

\_\_\_\_\_\_\_\_/\\\//\_\_\_\_\_\_\_\_\_\/\\\\_\_\_\_\_\_\_\_/\\\//\_\_\_\_\_

\_\_\_\_\_/\\\//\_\_\_\_\_\_\_\_\_\_\_\_\/\\\\_\_\_\_\_/\\\//\_\_\_\_\_\_\_\_

\_\_\_/\\\/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\/\\\\_\_\_/\\\/\_\_\_\_\_\_\_\_\_\_\_

\_\_/\\\\\\\\\\\\\\\\_\_\_\_\_\/\\\\_\_/\\\\\\\\\\\\\\\\_

\_\///////////////\_\_\_\_\_\_\///\_\_\///////////////\_\_

**2. Dance Party**

# Program Name: Party.java Input File: party.dat

Krish is attending a dance party, Krish is a hardcore break dancer and has sick moves. However, some audiences are more difficult to please than others, so he dances EVEN HARDER. Try to see if Krish can please his audience and get a round of applause in the process.

**Input**

The first integer, n, will represent the number of test cases that follow, every test case has 2 decimals, the first decimal number, e, represents the expectations of the audience. The second decimal, s, represents Krish’s dance skill. If his dance skill is greater than or equal to the audience’s expectations, he impresses the audience.

**Output**

Print out Dance On! if Krish can impress his audience, or print out BOO! if Krish sucks at dancing and can’t impress the audience.

**Example Input File**

4

12.02 50.00

1.00 1.00

52.99 21.10

1.36 888.41

**Example Output to Screen**

Dance On!

Dance On!

BOO!

Dance On!

**3. Fight Club**

# Program Name: Club.java Input File: club.dat

Krish has recently been promoted to leader of a secret underground cult similar to the Illuminati called the Fight Club. As leader, he has tasked you, the newest recruit, to write a program that can print out as many flags as he wants in any size he wants.

**Input**

The first integer, N, will be the number of test cases. Each test case will contain a single integer J, and will always be an odd number.

**Output**

Print out the Fight Club flag, an inverted diamond with diagonal shapes of length J in a square of asterisks with sides of length J+2. Print a single blank line between each flag except after the last case.

**Example Input File**

3

1

3

9

**Example Output to Screen**

\*\*\*

\* \*

\*\*\*

\*\*\*\*\*

\*\* \*\*

\* \*

\*\* \*\*

\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*

\*\*\*\* \*\*\*\*

\*\*\* \*\*\*

\*\* \*\*

\* \*

\*\* \*\*

\*\*\* \*\*\*

\*\*\*\* \*\*\*\*

\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

**4. Agent**

# Program Name: Agent.java Input File: agent.dat

ASF Agent Spencer has intercepted the most secret message known to man from the Antarticans (not to be confused with the penguins). Spencer is under immense pressure to crack the code to protect the world from devastation. However, Spencer is on his lunch break. So now it’s up to you to save the world! Or just, you know, not save the world…

**Input**

The first line of input will consist of an integer, n, indicating the number of data sets to follow. Each data set will consist of a phrase that needs to be decrypted. To decrypt the phrase: reverse the phrase. For every third character in the reversed phrase, convert it to binary, count the number of zeroes and add 65, and convert that number to the letter that is associated with that ASCII value.

**Output**

Output the decrypted phrase.

**Example Input File**

4

.1oo taa I

dnair taem

sie o1dil kniN

!t?Hla nata. la sa t?hT

**Example Output to Screen**

I EatFooD.

meEt FriEnd

NiCk DidDo Dis

ThAt Es El CatEn ElHAt!

**5. Singing**

# Program Name: Singing.java Input File: singing.dat

Nick decides to prove his musical talent by taking part in a sing-off. In this contest, there are numerous rounds and each contestant will pick their own song for each round. Although Nick likes a few songs in particular that really put his vocal range to the test, the audience gets irritated when he sings the same song again. In order for a song to count as a duplicate song, it must have the same song name and the same artist.

**Input**

The first line of input will contain a single integer, n, which indicates the number of test cases that follow. Each n lines that follow will contain the name of the artist and the name of the song separated by a hyphen.

**Output**

Output the name of the songs once that are listed more than once in the following format: [Name of Song in all caps] AGAIN?! The songs should be printed in the order of their repeated appearances.

**Example Input File**

10

Smash Mouth - All Star

Europe – The Final Countdown

Smash Mouth – All Star

Smash Mouth – All Star

Rick Astley – Never Gonna Give You Up

Christina Aguilera – Candyman

Milky Chance – Stolen Dance

The Original Broadway cast of Hamilton – My Shot

Europe – The Final Countdown

Rick Astley – Never Gonna Give You Up

**Example Output to Screen**

ALL STAR AGAIN?!

ALL STAR AGAIN?!

THE FINAL COUNTDOWN AGAIN?!

NEVER GONNA GIVE YOU UP AGAIN?!

**6. Spencer**

# Program Name: Spencer.java Input File: spencer.dat

Spencer accidentally deleted his java file, but not before he was able to run it and get the output. He has recently been attacked by a pterodactyl and while on his death bed, wanted you to figure out what his original code was.

**Input**

The first line of input will contain only an integer, n, depicting how many datasets to follow. The next n lines will contain each a list of numbers depicting what Spencer’s output was. The numbers will not contain decimals, and can be stored in an integer.

**Output**

You will output the code that Spencer used to print out his output using the following guidelines:

* The loop Spencer used will always be a for loop.
* The increment variable in the for loop will be identified as x and will be an integer.
* The condition will always be x<= or >= the last number printed number.
* The only things you should change should be the starting value, the ending value, and the increment.
* The inside of the for loop will only contain one line of code, printing out the variable x and a space.
* If incrementing by one, use ++ or --, if not incrementing by one, use +=, -=, /=, or \*=
* Only one operation will happen in each for loop.
* The inside of the for loop will be tabbed over.
* The for loop will always be followed by System.out.println();
* There will always be three or more numbers per test case.
* You will never be incrementing by 0, or multiplying or dividing by 1 or a negative number.
* **The formatting should be exactly the same as the sample output.**

**Example Input File**

4

3 4 5 6 7 8 9

1 2 4 8 16 32

44 22 11 5 2 1

13 9 5

**Example Output to Screen**

for ( int x = 3 ; x <= 9 ; x++ )

{

System.out.print(x+" ");

}

System.out.println();

for ( int x = 1 ; x <= 32 ; x\*=2 )

{

System.out.print(x+" ");

}

System.out.println();

for ( int x = 44 ; x >= 1 ; x/=2 )

{

System.out.print(x+" ");

}

System.out.println();

for ( int x = 13 ; x >= 5 ; x-=4 )

{

System.out.print(x+" ");

}

System.out.println();

**7. Agenda**

# Program Name: Agenda.java Input File: agenda.dat

Huckleberry is now a college student and signed up for more classes than he can handle. He has embarrassed himself too many times by walking into the wrong class at the wrong time. He asks you to create a personal agenda where he can quickly see which classes he has on a given day. The following table describes the keys for each class:

|  |  |
| --- | --- |
| **Key** | **Description** |
| ODD DAYS | This applies if the day is an odd number.  Example: November 3, 2016 |
| EVEN DAYS | This applies if the day is an even number.  Example: November 4, 2016 |
| MONDAYS, THURSDAYS | This applies if the day of week is in the list separated by commas. In this case, a Monday or Thursday. Up to six days can be listed.  Example: July 24, 2017 |
| NOT SUNDAYS, SATURDAYS | This applies if the day of week is not in the list separated by commas. In this case, not a Sunday or a Saturday. Up to six days can be listed.  Example: July 24, 2017 |
| WEEK 3 | This applies if the day is in on the week of the month listed. In this case, the third week of the month.  Example: October 14, 2012 |

**Input**

The first line of input contains a single integer, n, which indicates the number of test cases that follow. For each n cases that follow will have one or more lines will follow containing a name followed by a hyphen and then a key from the table above. Class names may be more than one word and could also contain a hyphen. The one more line will follow that contains a date with a month and a day. If no year after the day is present, assume the year is 2016.

**Output**

Print out what classes Huckleberry has that day in alphabetical order separated by commas. If he has no classes that day, output NONE.

**Example Input File**

4

Class A - ODD DAYS

Class B - MONDAYS, FRIDAYS

Class C - MONDAYS, TUESDAYS

October 10

English - NOT WEDNESDAYS

Calculus - NOT THURSDAYS

Physics - WEEK 1

CompSci - WEEK 2

January 3, 2010

History - EVEN DAYS

December 21, 2012

Disco Class - ODD DAYS

(continued on next page...)

Choir - WEEK 20

December 31, 2016

**Example Output to Screen**

Class B, Class C

Calculus, CompSci, English

NONE

Disco Class

**8. Tetrimonium**

# Program Name: Tetrimonium.java Input File: tetrimonium.dat

Huckleberry, being too good for public education has spent all his time playing Tetris. However, he has been bored the classic 4-block long shapes, also known as polyminos, found in Tetris, having won the high score for the fifteenth time, so he wonders how many shapes exist if he could make them of a different length. To elaborate, polyminos are simply shapes built by tacking identical cubes together edge to edge. Furthermore, a free polymino is one such shape that has no other shape that is a rotation and/or reflection of it. But Huck knows that there could be many shapes if he had more blocks, so he wants them sorted too. Fear not, though, as he has provided a convenient method of sorting them:

* Sort them smallest-first by the areas of the smallest rectangle that could enclose the shape
* If the areas are same, the ones with the least rows should come first
* If the number of rows are still equal, put the ones that have the least columns first
* Otherwise, going from left to right and then from top to bottom, the shape to first have a block filling in a spot where the other doesn’t should come last.

BUT, he’s still not done as he also wants to know the degrees of polyminos which refer to what symmetries shapes have or lack. Shapes only have one degree which is the highest applicable one. The degrees are described as follows:

* Zeroth degree polyminos are completely asymmetrical and have 8 distinct orientations and reflections
* First degree polyminos have 4 distinct orientations and reflections
* Second degree polyminos have two lines of symmetry and have 2 distinct orientations or reflection
* Third degree polyminos have four lines of symmetry and have only 1 distinct orientation or reflection

**Input**

An indeterminate number of lines each containing a single integer representing the length of the polymino to make for that case.

**Output**

For each case, output the distinct orientations and reflections of each free polymino on a single “line” sorted with the above sorting criterion. The ordering of the lines is to be by the first polymino using the exact same sorting criterion. Each set of orientations is to be separated by a single blank line and each orientation by one space. At the end of all the shapes, output the total number of free polyminos and the number of zeroth, first, second, and third degree free polyminos for the given length. Extra spacing at the end of each line or one extra line at the end of the output may be added to the output if desired. Note that O is used to represent a “block” or a filled spot. Numbers are always left-justified.

\*Lookup tables and precomputation are not allowed and judges will verify that your code actually computes the results.

**Constraints**

0 < Number of Blocks < 11

0 < Number of Test Cases < 20

**Example Input File**

2

3

4

(continued on next page...)

**Example Output to Screen**

OO O

O

Total : 1

Degree 0 : 0

Degree 1 : 0

Degree 2 : 1

Degree 3 : 0

OOO O

O

O

O OO O OO

OO O OO O

Total : 2

Degree 0 : 0

Degree 1 : 1

Degree 2 : 1

Degree 3 : 0

OOOO O

O

O

O

OO

OO

O OOO O OOO O OO O OO

OOO O OOO O O O O O

OO O OO O

O OOO O O

OOO O OO OO

O O

OO OO O O

OO OO OO OO

O O

Total : 5

Degree 0 : 1

Degree 1 : 2

Degree 2 : 1

Degree 3 : 1

**9. Typing**

# Program Name: Typing.java Input File: typing.dat

Sammy doesn’t know how to type properly. He can only type predefined patterns and press the backspace key. He can hold the backspace key to delete multiple characters at a time only from the end. At contests his time is limited, so he can only type a few sequences per problem. Any sequence or pressing or holding the backspace key takes 1 second. He must type an entire sequence before moving to the next sequence.

**Input**

The first line contains an integer representing the number of test cases. The first line of each test case is amount of time he has. The second line is integer N, representing the number of sequences he knows. The next N lines are sequences that he knows how to type and the last line is what he wants to type.

**Output**

Output how many possible ways Sammy can type the target sequence within given time.

**Example Input File**

3

5

5

ST

STA

STT

AR

R

STAR

5

5

A

AA

AAA

AAAA

AAAAA

A

4

3

GAA

GG

AG

G

**Example Output to Screen**

258

3091

53

**10. Hacks**

# Program Name: Hacks.java Input File: hacks.dat

Steven is hacking companies and needs help finding which company is the best to hack. Companies often own subsets of other company’s data. You are to determine which companies own enough of another company’s data to essentially own that company.

Company B is considered to be owned by Company A if Company A owns 50% of Company B’s data or if Company A owns other companies that own a total of 50% of Company B’s data. However, the companies do not own themselves unless a cyclical ownership arises where a company owns some company that owns the original.

**Input**

The first line has the number of test cases. The first line of each data set contains integer N, the number of connections in each test case. The next N lines will be formatted A B X. Where A denotes the owning company, B denotes the company owned and X denotes the percentage of data owned. All company names will be named A-Z. Ownerships can be cyclical in nature, and companies can own themselves though companies can only “own” a company once.

**Output**

Print all companies that own other companies in alphabetic order and which companies they own also in alphabetical order.

**Example Input File**

3

6

A B 25

A C 60

A E 30

B E 20

C B 25

C D 50

11

A B 20

A C 50

A D 25

C B 10

C D 25

D B 20

E A 15

F A 10

F E 50

G A 25

G F 50

3

A B 50

A C 25

B C 25

**Example Output to Screen**

A: B C D E

C: D

A: B C D

F: E

G: A B C D E F

A: B C

**11. Letter**

# Program Name: Letter.java Input File: letter.dat

Ra-ul really likes pranking his friends with random letters. Only problem is that Ra-ul sucks at English, writing, creativity, and pranks… Anyway, he needs your help to revise his letters into a form that anyone can understand. This is actually a very simple process as all of Ra-ul’s letters make perfect sense, if and only if, you remove every letter except the third from each line. If the third character from a line does not exist, assume it is a space.

**Input**

The input will consist of a number of lines.

**Output**

Output the 3rd character of each line.

**Example Input File**

Dear Sammy,

You are the only thing that allows me to live. Without you I would be just a stupid little girl. (Will appear on one line)

You are the light at the end of my tunnel and you guide me through. I love you. (Will appear on one line)

From,

Meggie

**Example Output to Screen**

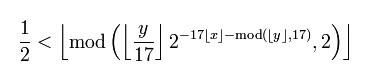
a uu og

**12. Dank Numbers**

# Program Name: dankNums.java Input File: dankNums.dat

Nothing makes Arami happier than those DANK MEMES. Unfortunately, his teammates found out about his plan to insert memes into the programming packet instead of creating an actual programming problem and promptly told him to “git gud” and “make a real problem”. Solving this problem may help cure his crippling depression, so you should probably do it. Or not; it’s up to you.

Given a number K make a 17 x 106 matrix. If the set of points (*x*, *y*) in 0 ≤ *x* < 106 and *k* ≤ *y* < *k* + 17 satisfies the inequality given below, place an asterisk at row Y-K (Y minus K) and column X. If it does not, put a space. In other words, X is the current column and Y is the sum of the current row and the number of the current test case.



Written Form: (same as above)

1/2 < floor(mod(floor(y/17)\*2^(-17\*floor(x)-mod(floor(y), 17)),2))

⌊ ⌋ denotes the floor function, and mod represents the modulo operation.

**Input**

The first number, N, will be the number of test cases. N lines will follow, each containing K, a number.

**Output**

Print out the resulting 17 x 106 matrix (including spaces). Also print a blank line after each test case.

**Example Input File**

3

960939379918958884971672962127852754715004339660129306651505519271702802395266424689642842174350718121267153782770623355993237280874144307891325963941337723487857735749823926629715517173716995165232890538221612403238855866184013235585136048828693337902491454229288667081096184496091705183454067827731551705405381627380967602565625016981482083418783163849115590225610003652351370343874461848378737238198224849863465033159410054974700593138339226497249461751545728366702369745461014655997933798537483143786841806593422227898388722980000748404719 (This will appear as one line.)

11446143048577322873420746886032253602081036176820637725351572728824205319356548595443573778191478330600315648025516347418384227839098139252614970555108049338384907856705947495396329029490965408180552069582726103040 (This will appear as one line.)

6064344935827571835614778444061589919313891311

**Example Output to Screen**

\*\*\* \* \*\*\* \*\*\* \* \*\*\*

\* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*\*\*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \*\*\*\* \*\*\*\* \* \*\* \* \* \* \*\*\* \*\* \* \* \* \* \* \*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \*\*\*\* \* \* \* \* \* \* \* \* \* \* \* \*\*\*

\* \* \* \* \* \*\*\* \*\*\* \* \*\*\* \*\*\* \* \* \* \* \*\*\* \*\* \* \*\* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*

\* \* \*\* \* \* \* \* \* \* \* \* \*\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*\*\* \*\*\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*\*\* \*\*\* \*\*\*\* \*\* \* \* \* \* \* \* \*\* \* \* \* \* \*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \*\* \* \* \* \* \* \* \* \*\* \* \* \*

\*\*\*\*\*\*

\*\*

\*\*

\*\*

\*\* \*\* \*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\* \*\*

\*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*\*

\*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*

\*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*\*\*\*\* \*\*

\*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*

\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*

\*\* \*\*

\*\* \*\*

\*\* \*\*

\*\*\*\*\*\*\*\*\*

\* \*

\* \*

\* \*

\* \* \* \*

\* \*

\* \* \*

\* \* \*

\* \* \* \*

\* \*\*\*\*\* \*

\* \*

\*\*\*\*\*\*\*\*\*

**13. Homework**

# Program Name: Homework.java Input File: homework.dat

Meggie is very bad at her homework and has so much that she needs to perform a triage situation on all of it. She needs you to build a sorting program to figure out in which order she needs to do it and which homework she can do with the amount of time and brainpower she has.

**Input**

The first line will contain an integer, n, or the number of datasets that will follow. The first line of each dataset will contain a list of classes Meggie has homework in and their respective averages. There will always be a class name with no spaces followed by a decimal number, which is the class average. There will be a space between the class names and the averages. The next line will have an integer, m, or the number of homework assignments to follow. Each line will contain a homework assignment. The first string is the name of the class that the homework assignment is for. The second string is the name of the assignment. Three integers: b, the amount of brainpower required to complete the assignment, t, the amount of time required to complete the assignment, and w, is the weight of the assignment. After m lines, there will be another line consisting of two integers, x and y, x being the amount of brainpower Meggie has and y, the amount of time Meggie has.

Meggie will sort her homework by the following:

* First by assignment weight, with the larger weight coming before the lower weight.
* If the weights are equal, sort by Meggie’s current average in the class the homework was assigned in, with the lower average prioritized over the higher average.
* If the averages are equal, sort by the amount of time required to finish the assignment, with the smaller time coming before the longer time.
* If the times are equal, sort by the amount of brainpower required. Assignments requiring less brainpower come before assignments requiring more brainpower.
* If the amount of brainpowers required are equal, sort the assignments alphabetically based off assignment names. You may assume that no two assignments will be equal after this condition.

**Output**

Output "DATASET: " followed by the dataset number, starting at 1. Print out all the homework assignments Meggie can finish with the brainpower and the time she has (ordered according to the rules above). If she has no homework assignments for that dataset, do not print anything. Print out a blank line after every dataset, even if there are no homework assignments.

**Sample data and output continued on next page.**

**Example Input File**

3  
Algebra 95.0 Physics 45.7 WorldHistory 85.8

3

Algebra Worksheet1 15 20 1

Physics Project 40 30 3

WorldHistory ReadingNotes 2 20 1

60 60

Spanish 34.5 English 98.0 Algebra 79.6 Geography 86.3

4

Spanish MiSemana 30 7 2

English Essay 20 15 3

Algebra Worksheet 8 5 1

Geography ReadingNotes 60 10 1

43 23

USHistory 24.8 Art 98.0 Physics 99.6 English 56.8 PreCalculus 87.6 Statistics 86.3 CompSci 91.2 (Will be on one line in input file)  
8

Art Portrait 10 10 3

USHistory Essay 20 15 2

PreCalculus Worksheet1 20 15 1

USHistory ReadingNotes 25 10 1

English Article 15 13 2

Physics Graph 10 5 1

Statistics Worksheet2 12 14 2

PreCalculus Worksheet3 13 12 1

55 65

**Example Output to Screen**

DATASET 1

Project

Worksheet1

ReadingNotes

DATASET 2

Essay

MiSemana

DATASET 3

Portrait

Worksheet2

Article

Essay

Graph

**14. Knight**

# Program Name: Knight.java Input File: knight.dat

Walter has recently hacked into the future and discovered Multi-Dimensional Chess™. Walter thoroughly enjoys normal modern chess and thinks he will enjoy this version. However it must first pass a test Walter has devised. If the knight can move throughout the board and land in every space once and only once, Walter will play it, otherwise he won’t.

Given this information it is your job to make a program to determine, with a rectangle of a given size, if it is possible for this process to be completed.

In Multi-Dimensional Chess™ the knight moves two squares in one direction then one in both perpendicular directions.

. . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . .  
. . . . . . . . . . . K . . . .  
. . . . . . . . . . . **.** . . . .  
. . K . . . . . . . **.** **.** . . . .  
. . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . .

In this example the Knight starts in position (4, 2, 0) and ends in position (2, 3, 1).

Given dimensions of a Multi-Dimensional Chess™ board determine whether or not it is possible for a knight starting in position (0, 0, 0) to move throughout the square land in every position once and only once.

**Input**The first line will contain a single integer N that determines the number of data sets that will follow. The next N lines will contain two integers: R, C, and H. R is the number of rows, C is the number of columns, and H is the height.

**Output**For each data set you will output whether or not it is possible for the Knight starting in position (0, 0, 0) to land in every square exactly once.

**Example Input File**31 1 13 4 32 2 2

**Example Output to Screen**

TrueFalseFalse

**15. Will It Compile?**

# Program Name: Compile.java Input File: compile.dat

The Dulles kids have just realized that they only have one computer running Java 8. As a result, the must figure which problems absolutely have to be graded on the one Java 8 computer so that the contest can proceed as quickly as possible. Make a program to determine if a submitted program has to be graded on the only Java 8 computer.

**Input**

A single integer, N, will be on the first line to indicate the number of java code sets to analyze. The first line of each test case will have, on one line, a single integer, L, for the number of lines in the source code. The next line will contain a valid file name (on windows) for the file of the source code with the extension. The next L lines thereafter will contain the source code that is contained in the file. There will **NOT** be a blank line between cases nor is there a new line at the end of the last line in the data file.

**Output**

Print Great for 8. if the code will compile ONLY in Java 8, output Even for 7. if it will compile in Java 7 and 8, or output Not Punny at All. if the code does not compile at all for Java 7 and 8.

**Constraints**

0 < N < 120

0 ≤ L ≤ 512

The total number of characters in the code (including newlines) will never be greater or equal to 65535 characters.

Code is considered to work for Java 8 if it compiles for “jdk1.8.0\_51” and for 7 if it compiles for “jdk1.7.0\_51”.

**\*Hint\*:** All judges will run windows and have “jdk1.8.0\_51” running and have the “javac” environment variable defined. The JDK will be available for the submissions to use.

**Example Input File**

5

5

A.java

//This is a comment

//And this is a blank class

public class A {

}

7

\_8Only.java

import java.util.function.\*;

public class \_8Only {

public static final void main(String[] args) {

Function<Integer, String> f = x -> "" + x;

System.out.println(f.apply(553));

}

}

7

DoesNotWorkForSoManyReasons.java

public class DoesNotWorkForThisReason {

public ArrayList<Integer> cannotImport = new ArrayList<>();

public void 1)0E51\107WORK ();

{

Function<String, String> ff = x -> x + x;

}

}

(continued on next page...)

8

Really7.java

//import java.util.Function.\*;

public class Really7 {

public static void main(String[] args)throws Exception {

String trick = " Function<Integer, Integer> f = x -> x + 4; \n";

trick = new String(new char[100]).replace("\0", trick);

System.out.println(trick);

}

}

1

A.java

import java.io.\*;import java.util.\*;public final class A{public static final void main(final String[]B)throws Exception{final Scanner C=new Scanner(new File("A.dat"));int D=C.nextInt();while(D-->0){int E=C.nextInt();C.nextLine();new File("Q").mkdir();String H="Q/"+C.nextLine();String G="";while(E-->0)G+=C.nextLine()+'\n';PrintStream F=new PrintStream(H);F.print(G);F.close();Process I=Runtime.getRuntime().exec("javac -source 1.7 "+H);I.waitFor();boolean J=I.exitValue()==0;I=Runtime.getRuntime().exec("javac -source 1.8 "+H);I.waitFor();boolean K=I.exitValue()==0;System.out.println(K?J?"L":"M":J?"N":"O");}}}

(this case is all on one line)

**Example Output to Screen**

Even for 7.

Great for 8.

Not Punny at All.

Even for 7.

Even for 7.

**16. Space**

# Program Name: Space.java Input File: space.dat

After testing his time machine Walter found himself trapped in N-Dimensional Space, and needs help getting out in the shortest amount of time. The space he is trapped in has equal edge lengths. Moving through the 1st dimension costs 1 unit of time. Moving through the 2nd dimension costs 2 units of time. Moving through the Nth dimension costs N units of time. The 1st dimension is defined as the x axis and the 2nd dimension is defined as the y dimension.

**Input**

The first number in the input is the number of test cases. The first and second lines of each test case are the number of dimensions N and the side length of the space S. An N dimensional maze of side length S is represented as S N-1 dimensional segments, and those also are represented as such. The next S ^ (N-1) lines will represent one dimensional segments of the maze. A # represents a wall, S represents the starting position, E represents the ending position and . represents a space that Walter can move to.

**Output**

Print the fastest amount of time that Walter can get to the end of the space

**Example Input File**

3

2

3

S##

...

##E

3

4

##S#

##.#

##.#

##.#

####

####

####

##.#

####

####

####

##.#

##E#

##.#

##.#

##.#

2

10

.#########

E.......##

..###.#.##

..###.#.##

..###.#.##

..###.#.##

......S.##

##########

##########

##########

(continued on next page...)

**Example Output to Screen**

6

21

16

**17. Nick Needs A Ride**

# Program Name: Nick.java Input File: nick.dat

Nick wants to attend computer science club after school, but he doesn’t own a car and his parents have more important things to do than pick him up. He systematically begs every person in computer science for a ride, but everyone is “busy” and unable to take him home. He is allergic to late buses so his only option is to suck it up and walk home. Unfortunately, Nick gets hungry very quickly during physical exertion. He starts his journey with a certain amount of hunger points. Every step he takes reduces his hunger points by a certain amount. Finding a trash can will replenish a specified amount of hunger points, because Nick can nourish himself off the scraps of food inside. If his hunger points ever reach zero, he collapses and doesn’t make it home (unless Nick arrives at home or a fresh trash can at 0 hunger points, then he will survive).

Write a program to figure out if Nick arrives home safely or has to spend the night on the side of the road.

**Input**

The first number, n, will be the number of test cases.

Each test case has 4 integers on the first line, followed by r number of lines.

The first integer of each test case, r, will be the number of rows of the map.

The second integer of each test case, d, will be the amount of hunger subtracted from each step Nick takes.

The third integer of each test case, t, will be the amount of hunger gained upon reaching a new trash can.

The fourth integer of each test case, s, will be the amount of hunger points Nick starts with.

‘S’ marks the school, or Nick’s starting point.

‘H’ marks his house, or Nick’s destination. If Nick arrives at home at 0 hunger points, he will survive.

‘T’s are trash cans, where Nick can replenish his hunger points by t. If Nick arrives at a trash can at 0 hunger points, he will survive. Nick can traverse through trash can tiles. Once a trash can is visited, it will not contain any food left (Nick cannot return to the same trash can and regain health points again).

‘@’s are obstacles that Nick cannot traverse through. The map will always be surrounded by walls.

‘.’s are empty spaces that Nick can travel through.

Nick can only travel north, east, south, or west.

**Output**

If Nick makes it to his house, print out STILL ALIVE. If Nick does not make it to his house, print out RIP IN PEPRI

**Example Input File**

3

4 2 10 30

@@@@@@@

@....S@

@H....@

@@@@@@@

10 5 50 60

@@@@@@@@@@

@@@@@@.....T...@

@...@..........@

@.S.@...........@@@@@@@@@@@

@.................T.@H@..T@

@.....................@...@

@........T...............@

@............@@@@@@@@@@@@

@...........@

@@@@@@@@@@@@

(continued on next page...)

4 2 10 30

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@................T....T......T......T....T....S@

@H....T........................................@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**Example Output to Screen**

STILL ALIVE

STILL ALIVE

RIP IN PEPRI

**18. Grader**

# Program Name: Grader.java Input File: Grader.dat

Mr. Armstrong, the coach of the amazing Cy-Woods Computer Science team, is assessing how well the members are preparing for written tests. However, manual grading is both tedious and has a high margin of error, so he has hired you to determine whether a particular student will live to see another day, or if Mr. Armstrong will have to bring out his trusted meter stick.

**Input**The first integer will indicate the number of data sets to follow. For each dataset, the first line will be the answer key, a string of letters denoting the right answer. The second line will contain 5 integers, n, r, w, s, and l. n represents the number of student answers to grade, and r, w, and s represent the number of points awarded for a right answer, a wrong answer, and a skip respectively. l represents the minimum score a student can get and pass. The next n lines will contain the student’s name (without spaces) followed by their answers, in the same format as the answer key. A space in the student’s answers denotes a skip. **Output**For each dataset, print out Test #m:, where m is the current dataset number. The next n lines should print out the students sorted by score, and in the case of a tie by name, in the format [Score] : [Name] [Condition], where [Score] is their calculated score, [Name] is their name, and [Condition] is either lives to see another day if they passed or gets the meter stick if they didn’t. A blank line should separate those who passed from those who didn’t and between each data set.

**Example Input File**

2

ACDBEEDACB

5 6 -2 0 30

Walter ACDBEEDACB

Huckleberry ACDBDEDACB

Meggie ACDB ED B

Steven BCDBEEDACB

Sammy FFFFFFFFFF

ABCDDE

2 1 0 0 3

Jimmy ECABDE

Bobathan A CDDE

**Example Output to Screen**

Test #1:

60 : Walter lives to see another day

52 : Huckleberry lives to see another day

52 : Steven lives to see another day

42 : Meggie lives to see another day

-20 : Sammy gets the meter stick

Test #2:

5 : Bobathan lives to see another day

2 : Jimmy gets the meter stick