Cypress Woods Computer Science Competition 2015

1. Do the problems in any order you like. 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 | 3N+1 |  |
| 2 | Liberties |  |
| 3 | Lucky 7 |  |
| 4 | Fancy Matrix |  |
| 5 | Bob’s Adder |  |
| 6 | Death Star Automation Software |  |
| 7 | Catch the Pig |  |
| 8 | Fun With Cyphers |  |
| 9 | Factor Fact |  |
| 10 | Pyramid |  |
| 11 | Yeezy |  |
| 12 | Quarterback Rating |  |
| 13 | Diamond |  |
| 14 | Zombies |  |
| 15 | Grader |  |
| 16 | Derivative |  |
| 17 | Are you ready? |  |
| 18 | Abandoned Mineshaft |  |

1. **3N+1**

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

The famous 3N+1 sequence is an infinite iteration of numbers that has gathered much attention. You are a student assistant helping your computer science teacher explain this concept. You need to provide the answer to several examples of data so that your teacher’s students can clearly understand how to get the answer.

**Input**

On the first line, the integer c indicates the number of line of data that follows. Every line after the first has x numbers (2 <= x <= 10).

**Output**

Based on the numbers in the line, find the biggest number. If the biggest number Ais even, divide it by two and print out the result. If A is odd, multiply it by three and add one and print out the result. If there are two or more max numbers that are equal, print out the required result once based on whether it is even or odd.

**Example Input File**

5

43 52

10 100 67

0 5 13

27 13 7 88 19

1 0

**Example Output to Screen**

26

50

40

44

4

1. **Liberties**

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

A Liberty is an open space directly next to a stone(not counting diagonals). If a stone is surrounded it will have no Liberties and is captured. Find and return the row and column (starting at 0) of any white stone that has at least one Liberty.

**Input**

W denotes a white stone. B denotes a black stone. **.** denotes anempty space First line will contain integer n, which will be the number of boards there will be. The next integers will be r and c, which will be rows and columns for the board. The board size will always be at least 2x2.

**Output**

Output the row and column of the white stones that have at least one Liberty. If there are no Liberties on the board, return NONE.

**Example Input File**

**Example Output to Screen**

1 2

2 1

2 8

0 6

3 3

6 2

1 6

NONE

NONE

5

5 10

..........

.BWB....B.

BWB....BWB

......B...

.....BWB..

9 7

WB...BW

B......

..BBB..

..BWB..

.......

.......

.BWB...

..B....

.......

4 7

WB....B

B....BW

..B....

.BWB...

3 3

.B.

BWB

.B.

3 3

...

...

...

**3. Lucky 7**

# Program Name: lucky7.java Input File: lucky7.dat

**Input**

The first line of input will contain a single integer n that indicates the number of data sets to follow. For each data set:

* The first line will contain one integer indicating the number of rows which meets the following criteria:
  + r > 2 is the number of rows in the grid.
  + The number of columns is 3

**Output**

For each test case, you will print “WINNER” if there are three 7s in a row, in the same row. If the test case does not have three 7s, print “LOSER.”

**Example Input File**

3

3

1 5 7

7 6 7

7 7 7

5

1 2 3

4 5 6

7 8 9

3 2 1

6 5 4

4

7 7 8

3 7 7

7 7 7

1 5 6

**Example Output to Screen**

WINNER

LOSER

WINNER

**4. Fancy Matrix**

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

Make a Fancy Matrix! It’s like a regular matrix but with a checkerboard design.

**Input**

The first line of input will contain a single integer n that indicates the number of sequences to follow. Each of the following n lines will contain a set of integers separated by a space that will be set to the dimensions of the matrix you will create.

**Output**

For each sequence input you will create a matrix with a checkerboard design consisting of “#” inside of the matrix with the borders filled in. Each sequence needs to be separated by an empty line.

**Example Input File**

3

5 5

10 10

4 4

**Example Output to Screen**

#####

## ##

# # #

## ##

#####

##########

## # # # #

# # # # ##

## # # # #

# # # # ##

## # # # #

# # # # ##

## # # # #

# # # # ##

##########

####

## #

# ##

####

**5. Bob’s Adder**

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

For Christmas, Bob’s parents got him a variety of number blocks, with each block having a different number on it. Bob wants to know if he adds some of the blocks, will they add up to his favorite number? You are to write a program that will determine from the blocks that Bob has if it is possible for them to sum up to his favorite number. If the sum of the numbers provided cannot add up to his favorite number, print out Not Possible.

**Input**

The first integer N (where N > 0) will represent the number of cases to follow. On the next line, there is an integer B (blocks) that will determine how many number blocks Bob has. The next line will show the numbers on the blocks Bob has, with the last integer on the line being an integer X, which is Bob’s favorite number.

**Output**

Output “Yes.” If the numbers Bob has can add up to his favorite number. Otherwise output “Not Possible.”

**Constraints**

1 <= B <= 20

0 <= value of each B <= 10000

0 <= X <= 10000

**Example Input File**

3

5

6 16 25 90 43 121

3

9 20 11 1000

1

5 17

**Example Output to Screen**

Yes.

Not Possible.

Not Possible.

**6. Death Star Automation Software**

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

Darth Vader is getting tired of manually deciding what planets to blow into smithereens. He knows that while the ability to destroy a planet is insignificant next to the power of the force, the power of coding is a close second. Lord Vader has tasked you with creating a program to automatically decide which planets to blow up. Darth Vader does not tolerate failure. Don’t disappoint him.

**Input**

The first line in the data file is an integer that represents the number of planets that follow. The following lines describe which faction the majority of the planet is sided with.

**Output**

If the planet is a part of the Rebellion then blow it up; sympathy towards the Rebel scum’s effort is not tolerated. However if the planet is aligned with the Empire, don’t destroy it; we need as many brothers in arms to fight the resistance as possible. If the planet needs to be destroyed, print “False” otherwise print “True”.

**Example Input File**

4

Empire

Rebellion

Rebellion

Empire

**Example Output to Screen**

False

True

True

False

**7. Catch the Pig**

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

Watch out! Hamlet, a slightly devious pig, has been terrorizing Cypress by breaking into houses and eating copious amounts of smoked ham. As the local deputy, you must find him within 1 week because after consuming so much of his own species, he becomes unstoppable in his rampage. Given a 2D character grid which is a map of a certain area in Cypress, the following symbols represent:

H – Houses

. – Sidewalk

D – Yourself

O – Oscar the pig

Travelling 1 tile in the grid requires one day, and you can only travel in the 4 cardinal directions. Your goal is to print out if you can successfully capture Oscar before he becomes invincible. There will always be a valid path. Note that you must actually move into Oscar’s grid coordinates to capture him.

**Input**

The first line of input will contain a single integer n that indicates the number of simulations to follow. The next N sets each start with two integers, R and C, which are the number of rows and columns in the grid. The next R lines contain C characters that create your map, with only the characters listed in the description. There will be an empty line between each of the matrices.

**Output**

For each line of input, you will print out BACON if you capture the pig in time or PIG ON THE RUN if you cannot.

**Example Input File**

3

5 5

D...H

HH..H

.HH.H

...O.

3 7

.......

.DHHH..

HHH.O..

4 4

.D.H

.H..

H...

..HO

**Example Output to Screen**

BACON

PIG ON THE RUN

BACON

1. **Fun With Cyphers**

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

Jake wants to send messages to his friends, but he doesn’t want anyone else to easily read the messages he and his friends send to each other. Because of this, Jake has decided to encrypt his messages with a Vigenere cypher, and give his friends the cypher to decrypt with a key he changes daily. Write a Java program to help Jake’s friends decrypt his messages so that they don’t have to spend so much time decrypting them in order to read them.

**Input**

The Vigenere cypher square will be provided in the first 27 lines (with 27 columns) with spaces between for use in decrypting the cyphers of n length such that 5<n<30 as well as keys with which to decrypt them. This is followed by a number indicating the number of cypher/key combinations and a space. Each cypher/key combination will have the cypher, followed by the key, followed by a space. To encrypt a Vigenere cypher, start at the row beginning with the letter of the key and move to the column starting with the letter of the plaintext (original message) to find the letter for the cypher. With a key of ‘HELLO’ and the plaintext ‘WHATSUP’ generating the cypher ‘DLLEGBT’, the letter ‘D’ is in the column of plaintext ‘W’ and the row containing ‘H’. In decrypting a cypher, the process is similar, where the plaintext can be found at the top row in the same column as the cypher letter in the row of the current key letter. All letters will be uppercase. For instances where the key is longer or shorter than the cypher, repeat or cut off the key where necessary. Ex: cypher - ‘SETWPICV’ key - ‘APPLE’ use ‘APPLEAPP’.

**Output** For each set of keys and cyphers, the plaintext must be printed in all uppercase with one line in between in the format cypher “decrypted with” key : plaintext. No spaces between words of plaintext are needed. There will be no special characters such as ‘,!? and all output will be comprised of words that exist in the English language.

**Continued on next page**

**Example Input File**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3

VVYTACK

ORANGES

SDXLRVBKEIWGXLNLC

KLEENEX

UZINIOCGHWOW

MOUSE

**Example Output to Screen**

VVYTACK decrypted with ORANGES : HEYGUYS

SDXLRVBKEIWGXLNLC decrypted with KLEENEX : ISTHEREATESTTODAY

UZINIOCGHWOW decrypted with MOUSE : ILOVECOMPSCI

**9. Factor Fact**

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

Yo dawg I heard you like factors, so we’re making a factor factory so you can factor your factors faster than factoring freehand.

**Input**

The first line of the input contains the number of cases that will follow. The following numbers will be on separate lines, within the range of 1 – 32000.

**Output**

For each number in the input file, you will find all of the positive whole number factors of the given number. For each of those factors, you must find the factors as well. When printing, print each line beginning with the factor of the input number, followed by the factors of that factor. The number being factored is separated by a space followed by a colon and another space, while all of the factors of the factor are separated only by a space.

**Example Input File**

4

8

32

81

100

**Example Output to Screen**

1 : 1

2 : 1 2

4 : 1 2 4

8 : 1 2 4 8

1 : 1

2 : 1 2

4 : 1 2 4

8 : 1 2 4 8

16 : 1 2 4 8 16

32 : 1 2 4 8 16 32

1 : 1

3 : 1 3

9 : 1 3 9

27 : 1 3 9 27

81 : 1 3 9 27 81

1 : 1

2 : 1 2

4 : 1 2 4

5 : 1 5

10 : 1 2 5 10

20 : 1 2 4 5 10 20

25 : 1 5 25

50 : 1 2 5 10 25 50

100 : 1 2 4 5 10 20 25 50 100

**10. Pyramid**

# Program Name: Pyramid.java Input File: pyramid.dat

Pharaoh Mathematica wants to build new pyramids for his kingdom. Each brick of each pyramid is labeled with letters for easier construction. It is your job to take the bricks and design the pyramids by sorting them in numerical order, top-to-bottom in decreasing order.

Ex: 55555

4444

333

22

1

**Input**

The first line of input will contain a single integer n that indicates the number of data sets to follow. For each data set:

* The first line will contain the starting integer of the pyramid thus defining how tall it is to be.

**Output**

For each test case, you will print the completed pyramid of numbers. If given a 0, then print NOT BUILDABLE.

**Example Input File**

4

5

8

0

3

**Example Output to Screen**55555

4444

333

22

1

88888888

7777777

666666

55555

4444

333

22

1

NOT BUILDABLE

333

22

1

**11. Yeezy**

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

Billy Bob needs a to buy a new pair of Yeezy’s that cost $2000, given the amount of money Billy Bob earns hourly at his local burger joint. If Billy Bob works an average 43 hours a week and puts 25% of his earning towards purchasing Yeezy’s, find out if he will have the shoes in 5 months so he can show off to his friends. Assume there are 4 weeks in a month.

**Input**

You will be given the hourly wage that Billy bob earns, calculate how many weeks it will take Billy Bob to buy the new Yeezy’s. If it takes him less than 5 months, return True. Otherwise return False.

**Output**

You should return if Billy will have the shoes in time.

**Example Input File**

4

7.25

10.99

13.70

8.50

**Example Output to Screen**

False

True

True

False

**12. Quarterback Rating**

# Program Name: Quarterback.java Input File: quarterback.dat

Your friend was hired a few months ago by the National Football League. However, he has not been performing well; he constantly procrastinates on creating weekly reports and commonly does a poor job. His boss decides to give him a different assignment and he now needs to calculate the passer rating for every quarterback that plays that week. But as expected, he waits on the last day and he won't be able to finish it in time. He asks you for help as he may be fired if he doesn't complete this assignment.

The quarterback rating is calculated by using the formulas below:

**Input**

The first line of input contains a single integer n which indicates the number of test cases that follow. Each n lines that follow will contain 5 integers, c (Completions), a (Attempts), i (Interceptions), y (Yards), t (Touchdowns).

**Output**

Print out the quarterback rating rounding to two decimal places.

**Constraints**

0 < n < 10  
0 ≤ c ≤ 2500

0 ≤ a ≤ 2500

0 ≤ i ≤ 2500

0 ≤ y ≤ 5000

0 ≤ t ≤ 2500

**Example Input File**

4

289 479 10 3238 21

163 272 4 2144 15

310 517 17 4051 21

20 28 0 218 1

**Example Output to Screen**

86.44

97.12

84.54

105.95

**13. Diamond**

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

Print a diamond made of the various powers of n increasing as you near the center and stopping once you reach the exponent of 0.

**Input**

The first integer n will be the number of test cases to follow and each number after will be the base to use in each diamond. Each test case will be between 1 and 6 inclusive.

**Output**

Print a diamond with aligned rows consisting of powers of n. The spacing between the numbers should be in such a way that the numbers are aligned in columns.

**Example Input File**

3

3

4

6

**Example Output to Screen**

1

1 3 1

1 3 9 3 1

1 3 1

1

1

1 4 1

1 4 16 4 1

1 4 16 64 16 4 1

1 4 16 4 1

1 4 1

1

1

1 6 1

1 6 36 6 1

1 6 36 216 36 6 1

1 6 36 216 1296 216 36 6 1

1 6 36 216 1296 7776 1296 216 36 6 1

1 6 36 216 1296 216 36 6 1

1 6 36 216 36 6 1

1 6 36 6 1

1 6 1

1

**14. Zombies**

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

You are trapped by zombies and must fight your way out. You will move one block along the path each day if there are no zombies at your position, otherwise you must spend your day killing zombies (one per day) until you are clear. Zombies will move toward you one block along the path each day. Zombies will NOT move if they are already in your area. Remember, you are faster than zombies, so you move first. If a zombie moves into a spot where there are already zombies, they will add together.

Day 0

|  |  |  |
| --- | --- | --- |
| @ | - | - |
| # | # | - |
| # | E | 5 |

Day 1

|  |  |  |
| --- | --- | --- |
| - | @ | - |
| # | # | 5 |
| # | E | - |

**Input**

The map of the area. @ = you, E = exit, #=not path.

**Output**

Every time you have a zombie in your area, output the day and block you killed the zombie also when you exit.

**Example Input File**

3

3 3

@--

##-

#E5

5 5

E----

####-

--@#-

-###-

-----

1 2

@E

**Example Output to Screen**

Day 3: Block 2

Day 4: Block 2

Day 5: Block 2

Day 6: Block 2

Day 7: Block 2

Day 10: Exit

Day 16: Exit

Day 1: Exit

**15. Grader**

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

Mr. Computica is a new teacher who is having issues understanding the Genovian grading system. He can’t seem to figure out that you get an A for an 89.5-100, a B for a 79.45-89.49, a C for a 69.45-79.44, and an F for a 69.44 or below. You have to help Mr. Computica grade his test.

**Input**

The first line in the data file indicates the number of students in Mr. Computica’s class.

**Output**

Return the letter grade of each student’s test followed by the student’s name.

**Constraints**

Input will always have a grade with a maximum of two numbers after the decimal.

**Example Input File**

5

Jimmy 99

Rebecca 82

Mason 37

Johnny 79

Logan 89

**Example Output to Screen**

A Jimmy

B Rebecca

F Mason

C Johnny

B Logan

**16. Derivative**

# Program Name: Derivative.java Input File: derivative.dat

Steven made a big mistake his senior year and took Calculus BC instead of Calculus AB. Because of his error, he now has to find the derivative of any equation given to him. Write a program to help Steven find the derivative of the equations given to him.

**Input**

The first line will contain a single integer n that determines the number of data sets to follow. Each line will contain a polynomial function of which you are to find the derivative. To find the derivative, simply multiply the exponent by the coefficient then decrement the exponent by one. For example: 2x^2 + 3x would be 4x^1 + 3 or 4x+3. Each line contains an integer d which denotes the number of terms in the equation following the rule 0<d<100.There will be spaces between each term and operator. The only operands to be used will be + and -. It can be assumed that the first term in every equation will be positive. Exponents will be designated by a ^ followed by the integer exponent value. Note: any term that does not contain an x has a derivative value of 0 and should not be included in the output equation.

**Output**

Each equation output should be in the form f’(x)=4x+3. There should be no spaces and if the exponent is equal to 1 it is left out.

**Example Input File**

4

2

2x^2 + 3x

3

5x^3 + 2x + 7

6

3x^7 + 6x^6 - 8x^5 + 7x^3 – 8x^2 + 7

1

3x^77

**Example Output to Screen**

f’(x)=4x+3

f’(x)=15x^2+2

f’(x)=21x^6+36x^5-40x^4+21x^2-16x

f’(x)=231x^76

**17. Are you ready?**

# Program Name: Ready.java Input File: ready.dat

Hello? Hello, Hello! Welcome to your new job as a night guard at Freddy Fazbear’s pizzeria! We are home to the world’s crowning achievement in fun and animatronic advances. Your job will be to guard the place, make sure no one breaks in or whatever. However, though we have made many advances, the animatronics may be a bit active at night. They each tend to wander around and will occasionally wander into your office… that would be bad. To prevent this, you should check the doors to your office to see if they are coming. We hope you have a great first night. Are you ready for Freddy?

**Input**

The first line of input will contain a single integer n denoting the number of cases to follow. In every data set there will be a single integer a denoting the number of animatronics followed by a line containing the names of the animatronics and the frequency f at which they attack represented by a decimal value. The final line of each case will contain a single decimal c which denotes the frequency at which you check the doors. For example, a typical case would be laid out like so:

1

Freddy 0.5

.25

Each decimal value is representative of a time. For instance, 0.5 means every half hour. The night always begins at 12:00 A.M. and continues on to 6:00 A.M. inclusive. It is your job to determine whether or not you survive. You will only survive if you check the doors at the exact same time as the animatronic is scheduled to strike. If you fail to check the door you did not survive.

**Output**

You are to read in each value appropriately and determine whether or not you survived. If you survived, then simply print out 6:00 A.M. Yay! If you did not survive, your output should contain the name of the animatronic that killed you and the time you died. If you are killed by multiple animatronics at one time, output the first one alphabetically that killed you.

**Example Input File**

4

1

Freddy 0.5

0.25

2

Bonnie 0.5 Foxy 1.0

1.0

1

Fredbear 0.75

0.5

4

Chica 0.5 Bonnie 1.167 Freddy 1.0 Foxy 1.5

.25

**Example Output to Screen**

6:00 A.M. Yay!

Bonnie 12:30 A.M.

Fredbear 12:45 A.M.

Bonnie 1:10 A.M.

**18. Abandoned Mineshaft**

# Program Name: Mineshaft.java Input File: mineshaft.dat

Oh no! You have fallen into an old abandoned mineshaft! In order to escape, you will have to dig your way through the collapsed walls to find the exit. Unfortunately, your trusty rusty shovel only has a limited number of uses (durability) and can only dig through certain ‘breakable’ walls.

The Following characters will be used to describe the mineshaft:

* . - Denotes a clear space
* # - Denotes an unbreakable wall
* % - Denotes a breakable wall
* S - Denotes the starting space
* E - Denotes the exit

Your goal is to find the quickest path to the exit where each movement in the four cardinal directions takes 1 second, each movement down a floor takes 2 seconds, each movement up a floor takes 3 seconds, and breaking any wall takes 3 seconds.

Remember that you can break walls that are directly below or above you.

**Input**

The first line of input is the number of test cases.

For each test case, the first line will consist of four integers in the form f r c a, where f is the number of floors (0 < f < 10), r is the number of rows (0 < r < 100), c is the number of columns (0 < c < 100), and a is the durability of the shovel (0 < a < 100). The next f floors will consist of r lines with c characters on each line. There will be no empty spaces between cases or between lines.

**Output**

The output should be one line per test case in the format # SECONDS where # indicates the number of seconds it took to escape the mineshaft. Output DEAD if there is no way to escape.

**Example Input File**

3

2 2 2 1

S%

#.

##

#E

2 1 1 50

E

S

1 4 4 4

S###  
####  
####  
###E

**Example Output to Screen**

7 SECONDS

3 SECONDS

DEAD