WALLER HIGH SCHOOL

BOO-VITATIONAL 2024

Student Programming Packet

Problem Number	Problem Name
1	Ghost
2	Party Prep
3	Costume Contest
4	Frankenstein
5	Conflicting Costumes
6	Candy Store
7	Trick AND Treat
8	Halloween Hayride
9	Spooky Numbers
10	Pumpkin Spice Treats
11	En-crypt-tion
12	Magic (File) System

Instructions:

1. Do the problems in any order you like. The problems do not have to be completed in order from 1 to 12.
2. For problems that read from a data file: When the program is judged, the data file will be in the same directory as your program. Do not include any extraneous path information in your program, just the name of the data file.
3. All problems are worth 60 points. Incorrect submissions receive a deduction of 5 points, but ONLY if the problem is ultimately judged correct. Problems may be reworked and resubmitted as many times as you like.
4. Your program shall not print extraneous output. Follow the format exactly as given in the problem statement and as shown in example output.
5. The time limit for problems is 120 seconds. If the program has not terminated after 120 seconds on the judge's computer, the program will be judged as an incorrect submission.

1. Ghost

Program Name: ghost.java Input File: ghost.dat

Billy has to draw a ghost for art class. Unfortunately, he was never really much of an artist. Instead of practicing, he decides it would be easier if you did it for him.

Input: No input available.

Output: Print out the ghost exactly as shown.

Note: Leave no trailing whitespace.

Example Output:

2. Party Prep

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

Amanda is throwing a Halloween party but is having trouble figuring out how many supplies to buy because her friends have only given her the odds that they'll show up instead of a yes or no. She's asked you for help to calculate her expected number of guests.

Input: Each line of input represents a guest. Each guest will have a value p (0<=p<=1) representing the probability the guest arrives and, separated by a space, a non-negative integer s representing the number of supplies that guest would use if they arrived.

Output: The expected number of supplies Amanda will need, rounded up to the nearest whole number.

Note: The expected value of a random variable is the sum of each value multiplied by the probability of that value

$$E(X) = \sum_{i} x_i * P(x_i)$$

Example Input:
```
.25 4
.9 1
.8 4
.56 2
0 20
1 1
```

Example Output:
```
8
```

3. Costume Contest

Program Name: costume.java Input File: costume.dat

Jaycee's neighborhood is having its local costume contests and Jaycee has been tasked with ranking the contestant's based on their scores. Unfortunately, he was never very proficient with numbers so he has asked for your help in determining each contestant's scores and ranking them.

Input: The first line contains an integer t, denoting the number of test cases. Each test case will begin with a number n denoting the number of contestants. Each contestant will consist of one line with their costume name and one line representing their 5 scores.

Output: For each test case, a numbered list of contestant names along with their scores in descending score order. All scores should be aligned with each other. Each test case should be followed by a line of dashes.

Note: No two contestants will have the same score. All total scores will be a non-negative integer less than 100.

Example Input:

```
2
5
Ghost
10 10 10 10 9
Rabbit
5 7 8 9 2
Batman
10 10 9 8 10
Spiderman
10 9 8 9 10
Dracula
1 2 3 2 1
2
Superman
4 5 7 2 3
Flash
10 10 10 10 10
```

Example Output:

```
1. (49) - Ghost
2. (47) - Batman
3. (46) - Spiderman
4. (31) - Rabbit
5. ( 9) - Dracula
----------------
1. (50) - Flash
2. (21) - Superman
----------------
```

4. Frankenstein

Program Name: frankenstein.java Input File: frankenstein.dat

Frankenstein's monster has once again escaped the lab. Frankenstein needs to find the fastest way to get to him; however, his tracking GPS is broken. Instead of fixing the GPS, he employs your help in finding the quickest path through town. However, Dr. Frankenstein is a man of (mad) science and won't travel through the dense forest in the area, strictly traveling only on the town roads.

Input: The first line contains an integer, t, denoting the number of test cases. Each test case will begin with an integer n (5<=n<=30) followed by n rows of n characters representing the search area.

"#" indicates dense forest.
"*" indicates a road.
"@" indicates the position of Dr. Frankenstein.
"!" indicates the position of Frankenstein's monster.

Output: For each test case, you are to output an n by n matrix with the fastest path between Frankenstein and his monster (including Frankenstein's start and end positions) indicated with "*" symbols. All other squares should be marked with a "#" symbol. Each test case should be followed by a blank line.

Note: Frankenstein can not move diagonally. There will be only one shortest solution.

Example Input:

```
2
10
##########
#@*******#
###*####*#
##**####*#
#**######*#
#********#
##*###*##
##*###*##
#**#####!#
##########
7
#@#####
#***##
#*#**##
#******
###*#*#
#******
#!#####
```

Example Output:

```
##########
#********#
########*#
########*#
########*#
########*#
########*#
########*#
########*#
##########

#*#####
#*#####
#*#####
#***###
###*###
#***###
#*#####
```

5. Conflicting Costumes

Program Name: confcostumes.java Input File: confcostumes.dat

A group of friends are picking what costumes they will wear to Jason’s Halloween costume party. The party has a strict rule that no two costumes should be the same. Unfortunately, some people want to wear the same costume. It is your job to determine whether each person can get one of their selected costumes while making sure it’s unique.

Input: The first line contains an integer t, denoting the number of test cases. Each test case will consist of a number n, denoting the number of people. Each person will provide the number z, indicating the number of costumes they are willing to wear, as well as the name of each.

Output: Print “Everyone's happy!” if all people are able to wear one of their selected costumes or “Someone’s out of luck!” if one or more people are unable to wear one of their selected costumes.

Note: The costume name will contain no spaces.

Example Input:

```
2
5
3
Spiderman Superman Batman
2
Robin Flash
4
Spiderman Hulk Robin Flash
5
Spiderman Superman Batman Robin Flash
2
Robin Flash
3
2
Invincible Omni-man
2
Omni-man Invincible
2
Invincible Omni-man
```

Example Output:

```
Everyone’s happy!
Someone’s out of luck!
```

6. Candy Store

Program Name: candystore.java Input File: candy.dat

Jimmy has been put in charge of buying candy for the halloween party. However, he has to be careful not to go over his budget. To maximize the amount of candy he is able to buy, he needs you to determine what candy he should get.

Input: The first line contains an integer t, denoting the number of test cases. Each test case will contain a number p dictating Jimmy's budget. Additionally, each test case will consist of a number n, denoting the number of candy options. The subsequent three lines will contain the names of every piece of candy, the amount available for purchase, and the prices of each piece, respectively, with each value separated by a space.

Output: The maximum amount of candy Jimmy can buy without going over budget.

Note: The candy names will contain only letters.

Example Input:

```
2
20
5
Reeses Snickers Hersheys Nerds Skittles
5 10 2 6 5
3 2 5 1 6
15
4
M&Ms KitKats PopRocks Toblerone
6 8 2 3
2 7 1 1
```

Example Output:

```
13
10
```

7. Trick AND Treat

Program Name: trick.java Input File: trick.dat

Jett is learning about logic and DeMorgan's law in his computer science class and has been wondering what the phrase "Trick OR treat" could mean. He's asked for your help to determine different versions of some common logical statements using DeMorgan's Law.

Input: Each line of input will contain a test case. Each test case contains a conditional statement in one of two forms: "p OR q" or "p AND q". p or q may contain the word "not" in which case they are considered to be the negation of a logical statement.

Output: The expression transformed by applying DeMorgan's Law.

Example input:
trick OR treat
not treat AND trick
good AND bad
not pizza hut OR not taco bell

Example Output:
not (not trick AND not treat)
not (treat OR not trick)
not (not good OR not bad)
not (pizza hut AND taco bell)

8. Halloween Hayride

Program Name: hayride.java Input File: hayride.dat

Skylar is planning the neighborhood's Halloween hayride and needs your help to find the best possible route. However, tractors move slowly and Halloween is a school night so Skylar needs to be sure the route takes the least amount of time possible so that the kids can get enough sleep!

Input: An integer t denoting the number of test cases. Each test case begins with an integer r denoting the number of rows of data that follow. Each line of data has two words a and b representing house names and an integer k representing the travel time in minutes between a and b.

Output: Output the shortest path from the "Start" house and "Stop" house.

Example Input:
```
2
7
Start Riley 15
Riley Kissinger 20
Riley Bokros 10
Kissinger Stop 2
Start Lomeli 12
Lomeli Hendricks 4
Bokros Stop 5
3
Start Mellon 10
Mellon Smith 12
Smith Stop 5
```

Example Output:
```
Start
Riley
Bokros
Stop

Start
Mellon
Smith
Stop
```

9. Spooky Numbers

Program Name: spooky.java Input File: spooky.dat

J'la is terribly superstitious, especially about "unlucky" numbers, and has recently found out a whole lot of numbers are considered unlucky by cultures around the world. She needs your help in determining if a number is unlucky. The rules for unlucky numbers are as follows:

- Any number that ends in a 4 or a 9
- Any number that, in hexadecimal, contains the digit sequence F13
- Any number that is divisible by 17, 39, or 43.
- Any number that contains the digit sequence 666

Input: Several base-10 integers N, all on one line, with one space between each.

Output: The integer N, followed by a space, followed by the phrase "AAAA!" if it is considered unlucky by two or more rules, "BOO!" if it is considered unlucky by exactly one rule, and "Candy" if it is not considered unlucky.

Example Input:
44 717109 22 585 666 27 34 4047153851

Example Output:
44 BOO!
717109 AAAA!
22 Candy
585 BOO!
666 BOO!
27 Candy
34 AAAA!
4047153851 AAAA!

10. Pumpkin Spice Treats

Program Name: spice.java Input File: spice.dat

Robert is obsessed with the phrase “Variety is the spice of life” and wants to mix up the spices in his Halloween recipes. However, Robert has had bad luck in the past with improper spice combinations so he needs your help to figure out how he should change his recipes.

Input: An integer k denoting the number of acceptable spice substitution lists. Each spice substitution list will be on one line beginning with a spice c and the possible substitutions for c in the recipe. Following the substitution lists, there will be an integer t denoting the number of recipes. Each recipe will begin with an integer N denoting the number of ingredients in the recipe. The next N lines will contain the ingredient list with each ingredient on a separate line.

Output: The recipe with all possible spices substituted in order of appearance in the corresponding substitution list.

Note: there will not be more occurrences of any substituted spice than corresponding acceptable substitutions.

Example Input:

```
2
thyme oregano basil parsley
cinnamon pumpkin allspice cloves nutmeg
2
8
1 cup milk
2 cups flour
1 tsp cinnamon
3 tbsp sugar
15 mL vanilla extract
1 egg
2 tsp cinnamon
1 tsp thyme
3
1 tsp thyme
2 tsp cinnamon
1 tsp thyme
```

Example Output:
1 cup milk
2 cups flour
1 tsp pumpkin
3 tbsp sugar
15 mL vanilla extract
1 egg
2 tsp allspice
1 tsp oregano

1 tsp oregano
2 tsp pumpkin
1 tsp basil

11. En-crypt-tion

Program Name: crypt.java Input File: crypt.dat

Francois has decided he wants to send spooky messages to all of his friends. However, he doesn't want them to get scared when reading the messages so he's decided to use a columnar transposition cipher to encode the words of his message. Francois has asked you to write a program that will encode his message for him. A columnar transposition represents the message, m, as an $|m|/|k| \times |k|$ matrix made up of the characters of the message in sequential row-column order, then shuffles up the columns based on a given keyword. For the nth column in the grid, its corresponding letter k is the nth character in the keyword. The nth column's new position is the position of k in the set of alphabetized letters of the keyword. The encoded message is then created by reading down each column, left to right.

Input: Test cases are separated by line. Each test case will consist of 1 word, the keyword, and the message to be encoded. The message and keyword will be separated by a space and the message will not contain any spaces. The message length will always be some multiple of the keyword length. The keyword will not contain any duplicate letters.

Output: The encoded message.

Example Input:
WOLF *SPOOKY_SCARY_SKELETONS*
BONES HE_A_OWANGE_WOUNDO_BOI^_^

Example Output:
O_RKT*PYASESSKC_LN*OSYEO
HOENOANO___AWO^EW_DI_GUB^

Visualization:

W	O	L	F
*	S	P	O
O	K	Y	_
S	C	A	R
Y	_	S	K
E	L	E	T
O	N	S	*

F	L	O	W
O	P	S	*
_	Y	K	O
R	A	C	S
K	S	_	Y
T	E	L	E
*	S	N	O

B	O	N	E	S
H	E	_	A	_
O	W	A	N	G
E	_	W	O	U
N	D	O	_	B
O	I	^	_	^

B	E	N	O	S
H	A	_	E	_
O	N	A	W	G
E	O	W	_	U
N	_	O	D	B
O	_	^	I	^

12. Magic (File) System

Program Name: magic.java Input File: magic.dat

Melinda has discovered that file systems use something called “magic numbers” to determine file types. Magic numbers are specific bytes (in this case represented as hex values) inserted at the beginning of the file data that corresponds to a known file type. Melinda also knows that some file types can be dangerous, so she’s asked for your help in determining which files are dangerous. The following file types (and their corresponding magic number) are potentially dangerous:

```
RPM   ED AB EE DB
EXE   4D 5A
PDF   25 50 44 46 2D
VBS   23 40 7E 5E
CLASS CA FE BA BE
ELF   7F 45 4C 46
BC    42 43
MSI   D0 CF 11 E0 A1 B1 1A E1
```

If a file is dangerous, Melinda needs a warning attached to the file name. If a file is safe, the file name should be left alone. Keep in mind that magic numbers are ONLY at the beginning of the file’s data and ONLY ONE magic number is inserted.

Input: Each line represents a file, which will contain a series of 2 digit hex values, separated by spaces.

Output: The name of the file. If the file is potentially dangerous, the phrase “BAD_” should be inserted before the file name and the corresponding file extension should be appended to the end (in lowercase).

Example Input:

```
48 41 50 50 59 20 48 41 4C 4C 4F 57 45 45 4E 2E 74 78 74
4D 5A 42 43 56 49 52 55 53
42 43 72 75 73 74 65 2D 6D 61 69 6C
6D 61 67 69 63 2E 6A 61 76 61
```

Example Output:

```
HAPPY HALLOWEEN.txt
BAD_BCVIRUS.exe
BAD_ruste-mail.bc
magic.java
```