A+ Computer Science
Computer Science Competition
Hands-On Programming Set

I. General Notes

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

2. All problems have a value of 60 points.

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

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

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	A Gobble Gobble
Problem 2	BaseCon
Problem 3	Chemistry
Problem 4	Disc Golf
Problem 5	Marble
Problem 6	Q That
Problem 7	RadDeg
Problem 8	SDR
Problem 9	Search
Problem 10	Twister
Problem 11	Vowels
Problem 12	Zoo

A+ Computer Science – 2223 Packet 02
1. A Gobble Gobble

Program Name: gobblegobble.java **Input File: none**

The Clements Computer Science club wants to host a Thanksgiving party which includes a roasted turkey. Unfortunately, due to dietary restrictions, the club was unable to serve the dish, so being the geeks they are, the club wants everyone to create the food items using ASCII art. You have been assigned to create the turkey. Help liven the Thanksgiving spirit at the Thanksgiving party by drawing a turkey using ASCII art!

Input
None

Output
Print the turkey exactly as shown in the sample input.

Output to Screen

```
          _____________
      _  /  "Gobble"  \
     / (    "Gobble"   )
    /_/  \_____________/
   ++          ____
 <(o )  ___(       \\
  U\  \/    \\       \)
     |     >   ||    //
      \______//__//
        | |
```

A+ Computer Science – 2223 Packet 02
2. BaseCon

Program Name: basecon.java **Input File: basecon.dat**

Base conversion in Java is easy, if you know the right commands. In this program, you are to input a value in a certain base, and output it in the four bases common to computer science: decimal, hexadecimal, octal, and binary.

Input
An initial positive integer n followed by n data pairs, each on its own line, each consisting of a single capital letter indicating the base - 'D', 'H', 'O', or 'B', followed by a value in that base, with one space of separation.

Output
All four base values in the order Decimal, Hex, Octal, Binary, as shown below.

Example Input File
```
4
D 34
H 8e
B 11
H f0f
```

Example Output to Screen
```
34 22 42 100010
142 8e 216 10001110
3 3 3 11
3855 f0f 7417 111100001111
```

A+ Computer Science – 2223 Packet 02
3. Chemistry

Program Name: chemistry.java **Input File: chemistry.dat**

Azura is analyzing some chemical reactions. Being an amateur chemist, she didn't keep accurate documentation of her experiment. An unusual reaction occurred, and she wants to be able to repeat the experiment, however, she only remembers the measurements of chemicals used, and alas, cannot recall the order that produced the unusual reaction! She has requested that you write a program to generate all the possible orders she could have mixed the chemicals so she can better attempt to replicate her discovery.

Input

The first line will contain a single integer **n** that indicates the number of data sets to follow. Each data set will consist of an integer **q** that indicates how many chemicals Azura used in her concoction, followed by **q** chemicals, each separated by one space

Output

For each data set, output all the possible ***unique*** orders that Azura could have mixed the chemicals, initially in alphabetical order, prefixed by the number, with a new line between data sets.

Example Input File

```
3
3
B Zn Fr
4
C H He O
3
A B B
```

Example Output to Screen

```
1. B Fr Zn
2. B Zn Fr
3. Fr B Zn
4. Fr Zn B
5. Zn B Fr
6. Zn Fr B

1. C H He O
2. C H O He
3. C He H O
4. C He O H
5. C O H He
6. C O He H
7. H C He O
8. H C O He
9. H He C O
10. H He O C
11. H O C He
12. H O He C
13. He C H O
14. He C O H
15. He H C O
16. He H O C
17. He O C H
18. He O H C
19. O C H He
20. O C He H
21. O H C He
22. O H He C
23. O He C H
24. O He H C

1. A B B
2. B A B
3. B B A
```

A+ Computer Science – 2223 Packet 02
4. Disc Golf

Program Name: disc.java **Input File: disc.dat**

Disc Golf is so much fun! Big J recently moved into an apartment in Round Rock and discovered an 18-hole disc golf course right behind his apartment complex, and he was hooked! This was the Brushy Creek Cat Hollow course, a really nice course! He also discovered a 9-hole practice course on the athletic fields of nearby Cedar Valley Middle School, excellent for practicing throws. After moving again farther down the road to the Bee Cave area, he found a world class course behind Austin Ridge Bible Church, designed by John Houck, nationally renowned disc golf course designer.

Big J's skills are still improving, and he tries to practice as much as he can. Besides, it's great exercise and great outdoor fun! Your job is to analyze his play by calculating and reporting various aspects of each practice round, including the name of the course he practiced, the overall par total of the course, his overall practice round total, the number of pars results, above par results, and below par results, as well as his overall result based on par for the course.

On each course, each hole is given an assigned par score, which is the number of throws a player is expected to make to complete the whole. The overall par total for the course is the sum of each of these hole-by-hole par scores. When Big J practices at a course, his score is calculated by the amount of throws he attempts on each hole. His overall practice round total is the sum of each of these attempts hole-by-hole. The number of pars that Big J receives is the number of holes in which his attempt score is equal to the par score. The number of above pars that Big J receives is the number of holes in which his attempt score is higher than his par score. The number of below pars that Big J receives is the number of holes in which his attempt score is lower than his par score. The overall score is "PAR" if his overall practice round total is equal to the overall par total for the course. The overall score is "BELOW PAR" if his overall practice round total is below the overall par total for the course. The overall score is "ABOVE PAR" if his overall practice round total is above the overall par total for the course.

Input

The three courses mentioned above, with hole-by-hole par listings for each course, then an integer **n** followed by **n** practice rounds, one on each line. Each practice round starts with the first letter corresponding to the course he practiced on, followed by his hole-by-hole attempts.

Output

The name of the course Big J practiced, the par total for the course, his score, the number of pars, above pars, and below pars, and finally his overall result. One blank line should follow each output sequence.

Sample Input Data

```
AUSTIN_RIDGE_BIBLE_CHURCH
3 4 4 4 3 3 3 4 3 4 3 4 4 4 3 4 3 3
BRUSHY_CREEK_CAT_HOLLOW
3 3 3 3 4 3 3 3 3 4 3 3 3 3 3 3 3 3
CEDAR_VALLEY_MIDDLE_SCHOOL
3 3 3 3 3 3 3 3 3
3
A 5 5 4 2 3 3 1 4 3 7 3 4 6 4 1 4 4 3
B 3 1 3 1 4 8 3 1 3 4 5 3 2 3 3 6 1 1
C 3 3 3 3 3 3 3 3 3 3
```

Sample Output to Screen

```
AUSTIN_RIDGE_BIBLE_CHURCH
COURSE TOTAL = 63
SCORE = 66
PARS = 10
ABOVE PARS = 5
BELOW PARS = 3
OVERALL RESULT: ABOVE PAR

BRUSHY_CREEK_CAT_HOLLOW
COURSE TOTAL = 56
```

```
SCORE = 55
PARS = 9
ABOVE PARS = 3
BELOW PARS = 6
OVERALL RESULT: BELOW PAR

CEDAR_VALLEY_MIDDLE_SCHOOL
COURSE TOTAL = 27
SCORE = 27
PARS = 9
ABOVE PARS = 0
BELOW PARS = 0
OVERALL RESULT: PAR
```

A+ Computer Science – 2223 Packet 02

5. Marble

Program Name: marble.java **Input File: marble.dat**

There are 25 bins, each with a maximum capacity of 10 grams, into which we will sort marbles of weight ranging from 1 to 9 grams each. The basic idea is to sort marbles into the minimum number of bins, starting with bin 1, then using bin 2, etc. There will be 5 different strategies to use.

First Fit: Pick up the next marble. Scan the bins from 1 to 25 and put the marble into the first bin encountered that it will fit into.
First Fit Increasing: Sort all the marbles. Pick up the marbles from lightest to heaviest and apply the **First Fit** algorithm.
First Fit Decreasing: Same as **First Fit Increasing**, but pick up the marbles from heaviest to lightest.
Best Fit: Pick up the next marble from the original unsorted list. Look at all empty or partially filled bins. Put the marble into the bin that will come closest to maximum capacity. If more than one bin comes equally close to max capacity, then put it into the lowest numbered of those bins.
Worst Fit: Pick up the next marble of the original unsorted list. Look at all empty or partially filled bins. Put the marble into the bin that is the lightest. If more than one bin is equally light, then put it into the lowest numbered of those bins.

Overall Rules:
For all five strategies, the following rules take priority:

1) Since each bin costs money to use, you start using a new bin ONLY when it is impossible to fit the marble into any bin that is already started. Unused bins cost nothing.
2) A bin can never have more than 10 grams worth of marbles.

Input

Several sets of marbles. Each set lists the weights of the marbles, indicated by single digit positive integers all on one line, each separated by a single space.

Output

For each method, determine the weight of each bin. All used bins will have a weight > 0 and <=10, with no more than 25 bins used. Output only non-empty bins, with bin 1 first, 2 next, and so on in order.

Example Input File

```
1 3 5 3 6 2 1 2 4 6 3 7
2 9 4 7 1 3 2 6 8 1 4 2 5 3 4
1 4 2 8 2 1 6 3 4 5 2 2 7 3 1 5 2
```

Example Output to Screen

```
FF: 10 9 8 9 7
FFI: 9 10 5 6 6 7
FFD: 10 10 10 10 3
BF: 10 9 8 9 7
WF: 9 9 9 9 7

FF: 10 10 9 10 10 8 4
FFI: 8 10 8 5 6 7 8 9
FFD: 10 10 10 10 10 9 2
BF: 9 10 10 10 10 8 4
WF: 10 9 9 7 8 10 8

FF: 10 10 10 9 9 10
FFI: 9 10 8 10 6 7 8
FFD: 10 10 10 10 10 8
BF: 10 10 10 9 9 10
WF: 9 9 9 9 8 7 7
```

A+ Computer Science – 2223 Packet 02
6. Q That

Program Name: q.java **Input File: q.dat**

The Clements library allows you to borrow three books at a time, one per day. When you borrow the first three books, each goes on the next empty shelf of three that you have, one book per shelf, starting with the uppermost empty shelf. For example, if you request books 5, 1 and 2, they go onto the top, middle and bottom shelves, and so the books on your shelves from top to bottom are 5 1 2. **Once a book is put on a shelf, it is kept there until you return it to the APLUS library.**

After borrowing three books, a request for each additional book requires the return of one of the books on the shelf, unless the requested book is already there and you just forgot about it in your absentmindedness. For example, if you request book 6, you must remove one of the books from the shelf and put book 6 in its place. If you request a book you already have, nothing changes, of course.

The question is which of the books should be returned? There are two methods for removing and returning books: **First In First Out (FIFO)** and **Least Used Returned (LRU)**. Your job is to simulate each of these methods, tracking a series of book requests and then reporting the remaining books on the shelf, in order from top shelf to bottom shelf.

Method 1 - FIFO - In this removal method, you find the book that has been **borrowed the longest** among the three books on the shelf, remove it from its shelf, and put the new book ***in its place***. For example, below is a simulation for the book request sequence 5 1 2 4 1 6 4 1 6 3 5 4 3 2 5. In the chart, a requested book is **bolded** if it is new on the shelf or already there, and also **underlined** if it is replacing a removed book.

Day	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***
Request	5	1	2	4	1	6	4	1	6	3	5	4	3	2	5
Top shelf	**5**	5	5	**4**	4	4	**4**	4	4	**3**	3	3	**3**	**2**	2
Middle shelf	-	**1**	1	1	**1**	**6**	6	6	**6**	6	**5**	5	5	5	**5**
Bottom shelf	-	-	**2**	2	2	2	2	**1**	1	1	1	**4**	4	4	4

The book order at the end of the sequence for FIFO is **2 5 4**, top shelf to bottom shelf.

Method 2 - LRU - In this method, remove the book that has been **used the least recently**, putting the new book ***in its place***.

Day	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***
Request	5	1	2	4	1	6	4	1	6	3	5	4	3	2	5
Top shelf	**5**	5	5	**4**	4	4	**4**	4	4	**3**	3	3	**3**	3	3
Middle shelf	-	**1**	1	1	**1**	1	1	**1**	1	1	**5**	5	5	**2**	2
Bottom shelf	-	-	**2**	2	2	**6**	6	6	**6**	6	6	**4**	4	4	**5**

Book order at the end of the sequence for LRU is **3 2 5**, top shelf to bottom shelf.

Input
An initial value **n** indicating **n** sequences of book requests, each on its own line, with books numbering from 1 to 9, separated by one space each.

Output - Report the remaining books as described above, formatted exactly as shown below.

Sample Input Data
```
2
5 1 2 4 1 6 4 1 6 3 5 4 3 2 5
1 2 3 4 3 5 1 2 3 5 3 2 2
```

Sample Output to Screen
```
FIFO:2 5 4
LRU:3 2 5

FIFO:2 3 5
LRU:5 3 2
```

A+ Computer Science – 2223 Packet 02

7. RadDeg

Program Name: raddeg.java **Input File: raddeg.dat**

Degrees and radians are the two most common measures of angles and rotation, and your job in this program is simply to convert from one to the other. In this case, any integer found will represent a degree measure, and a decimal value a radian measure.

Input

Input - A horizontal line of integer and decimal values representing degrees and radians, with single spaces of separation.

Output

The appropriate conversion based on the criteria listed above in the format as shown below. All degrees should be rounded to nearest integer, and all radians should be shown to 2 decimal points.

Example Input File

```
34 5.6 6.28 180
```

Example Output to Screen

```
34 degrees = 0.59 radians
5.60 radians = 321 degrees
6.28 radians = 360 degrees
180 degrees = 3.14 radians
```

8. SDR

Program Name: sdr.java **Input File: sdr.dat**

SDR stands for Split, Double, Reverse, which is what you are to do with each input string here. For each string, split it at the given integer, then output it in SDR format, as shown below. Study the output carefully, and you'll figure it out. Have fun!

Input

Several lines of data, each line containing a single capitalized word, followed by an integer guaranteed to be less than or equal to the length of the word.

Output

The SDR format of the given word and integer, as shown below.

Sample Input Data

```
HOLIDAY 3
SNOW 2
RACECAR 4
TATTARRATTAT 6
```

Sample Output to Screen

```
HOL-YADI-LOH-IDAY
SN-WO-NS-OW
RACE-RAC-ECAR-CAR
TATTAR-TATTAR-RATTAT-RATTAT
```

A+ Computer Science – 2223 Packet 02
9. Search

Program Name: search.java **Input File: search.dat**

A typical search process uses a sorted list and searches through the list for an item, or "key". The key is either in the list, or it is not. In this exercise, we will use an unsorted array of unique integers, within the range of -200 to 200. You are first to sort the list, then search for a given key to either find the position of the key, or report the position where it should be in the list.

For example, the list `22 33 25 81 -8 17 24 -27 95 100 -2` is sorted to become `-27 -8 -2 17 22 24 25 33 81 95 100`. To search for the key value 33, look through the list and find it, reporting its zero-based position, which is 7 in this list. If a key is not found in the list, like the value 2, report where it should have been found, position 3 in this list.

Input
An array of integers all on one line, followed by an integer n on the next line indicating n search keys to follow, one on each line below that.

Output
Output - The sorted array followed by a sentence for each search key reporting the result of the search, in the exact wording and format shown below.

Example Input File
```
22 33 25 81 -8 17 24 -27 95 100 -2
5
33
2
100
86
-27
```

Example Output to Screen
```
-27 -8 -2 17 22 24 25 33 81 95 100
33 was found in position 7
2 was not found and should be in position 3
100 was found in position 10
86 was not found and should be in position 9
-27 was found in position 0
```

A+ Computer Science – 2223 Packet 02
10. Twister

Program Name: twister.java **Input File: twister.dat**

Twister is that fun party game where you get on the floor, someone spins the spinner, gets either a RED, BLUE, GREEN, or YELLOW and you are to either step on that color or put your hand on that color. First contestant that falls is out! If you are limber, go for it, but if not, you might throw out your back, so be careful if you ever play this game!

In this problem, your job is to simulate the spinner and predict what color it will land on based on how fast it is spun, in which direction it is spun, and on what color it starts at the beginning of the spin. For simplicity's sake, the spinner will have only four sections for the colors, unlike the normal Twister spinner. The section (in clockwise order) from 0 degrees at the top of the spinner over but not including 90 degrees is GREEN, followed by RED from 90 down to but not including 180, then 180<=YELLOW<270 and 270<=BLUE<360 on around back to the top.

Furthermore, the spinner is automated and you get to place the spinner in the very middle of the color you want to start on (GREEN would start on 45, RED on 135, YELLOW on 225, and BLUE on 315) and push a button for one of five preset spin speeds. The speed choices are FAST, MEDIUM_FAST, MEDIUM, MEDIUM_SLOW, and SLOW. The engineer who designed the automated spinner selected appropriate decimal values for each speed, indicating the number of revolutions each spin would take, and we will assume for consistency's sake that there will be no variance on the spinner due to any factors such as friction or otherwise, with gravity the only consistent force to slow it down. A FAST spin will ***always*** revolve 13.4 times, a MEDIUM_FAST 9.7 times, a MEDIUM 5.3 times, MEDIUM_SLOW 2.125, and SLOW 1.5.

For example, a FAST spin starting on RED going clockwise will spin 13.4 times and land on BLUE. A MEDIUM speed spin going counter-clockwise from BLUE will spin 5.3 times and land on YELLOW. A clockwise MEDIUM_SLOW spin starting on RED will land on YELLOW, and a SLOW counter-clockwise spun from GREEN lands on RED.

Input
Initial value **n** with **n** sets of data to follow, each set consisting of the letter "CL" or "CO" indicating either clockwise or counter-clockwise initial direction, the selected rotation speed, and the starting color position.

Output
The color the spinner lands on.

Sample Input Data
```
4
CL FAST RED
CO MEDIUM BLUE
CL MEDIUM_SLOW RED
CO MEDIUM_FAST GREEN
```

Sample Output to Screen
```
BLUE
YELLOW
YELLOW
RED
```

A+ Computer Science – 2223 Packet 02

11. Vowels

Program Name: vowels.java **Input File: vowels.dat**

In the language of the land of APLUS, the vowels are the letters A, P, L, U and S. To form the plural of a word, use the following rules:

1) If the word ends in a single consonant, add "XY".
2) If the word ends in a single vowel, drop that vowel and add "X"
3) If the word ends in a double consonant or a double vowel, add a second final letter, and then add "Y". For example, PROBLEM becomes PROBLEMMY because it ended in a double consonant.
4) If the word ends in more than two consecutive vowels or consonants, drop the leftmost of the consecutive vowels or consonants, and then add "Y". For example, THEAPLUS becomes THEPLUSY because it ends with five consecutive vowels.

Input

The first line will contain a single integer **n** that indicates the number of data sets that follow. Each data set will be a single word that will contain only the capital letters A through Z.

Output

The plural of each word according to the rules listed above.

Example Input File

```
6
PROBLEM
THEAPLUS
CAST
ADA
POLL
MASS
```

Example Output to Screen

```
PROBLEMMY
THEPLUSY
CASTXY
ADX
POLLLY
MSSY
```

A+ Computer Science – 2223 Packet 02

12. Zoo

Program Name: zoo.java **Input File: zoo.dat**

Keeping zoo inventory is an important job, and two lists are required by your employer each time you perform this task. After taking stock of every different type of animal in the zoo, you are to output the list in alphabetical order, and also by the word length. If more than one animal has the same word length, output all of those in alphabetical order. One of those animals is the orangutan, but it is spelled "oranghutan" in this data, the true original spelling in the language **Bahasa Indonesia**, meaning "man (**orang**) of the jungle (**hutan**)", where "oranghutans" are prevalent in the jungles of Sumatra and Borneo.

Input

There will be several animals in this list, not necessarily in any order, each animal name on one line, with no multi-part names.

Output

Two lists, one item per line, with a blank line between lists. The first list will be the animal names in alpha order. The next listing will be the length of the animal's name, followed by all the animals with that name length, in alpha order if there are more than one, each separated by one space, as shown below.

Example Input File

```
zebra
elephant
lion
oranghutan
rhino
```

Example Output to Screen

```
elephant
lion
oranghutan
rhino
zebra

4 lion
5 rhino zebra
8 elephant
10 oranghutan
```