**Computer Science II**

**Program Packet**

**Instructions:**

1. There are 11 programs in this packet. **You are to select at least 5 programs to do.**
2. You may work in pairs if you would like.
3. In the computer science folder there is a folder called CS2DataFiles. Copy these files in your folder.
4. Each program will read the data from the appropriate file.
5. If you need help with the reading of the files let me know.
6. Each program description has an example of what the output should look like.

**GOOD LUCK!!!**

**1. Bridge Over the Cowhouse Creek**

**Program Name: Bridge.java          Input File: bridge.dat**

In a remote area of Fort Hood, there is a single lane bridge that spans Cowhouse Creek. Besides being single lane, there is a load limit of 42 tons for this bridge. Frequently, a convoy of military vehicles has to cross the bridge. The bridge gate keeper must divide the convoy into groups of one or more vehicles so that the load limit for the bridge is not exceeded and so the convoy can cross the bridge as quickly as possible. The number of minutes it will take a given group to cross the bridge is determined by the number of minutes of the slowest vehicle in the group. For example, if a group has three vehicles with times of 12, 30 and 4 minutes, it will take that group 30 minutes to cross the bridge.

You may assume the bridge is long enough to hold an entire group at one time and all of one group will completely cross the bridge before another group is allowed to start crossing the bridge. Each group of a convoy will start crossing the bridge immediately after the previous group has exited the other end of bridge. Groups will continue to cross the bridge until all groups in the convoy have completely crossed the bridge.

**Input**

The first line of input will contain a single integer n that indicates the number of convoys to cross the bridge. For each convoy, the first line will contain an integer m, 1≤ m ≤ 10, denoting the number of vehicles in the convoy. The next  m lines will each contain two integers separated by a space. The first integer w, 1 ≤ w ≤ 42, is the weight of a vehicle in tons and the second integer t, 1 ≤ t ≤ 30, is the time in minutes that it will take the vehicle to cross the bridge.

**Output**

For each convoy, you will print on a single line the minimum number of minutes that it will take for each convoy to cross the bridge.

**Example Input File**

2

8

10 10

5 25

40 5

35 15

12 23

30 20

42 25

8 30

10

42 10

23 30

40 5

2 10

1 20

4 30

6 28

28 3

17 8

35 10

**Example Output to Screen**

95  
66

**2. Dice**

**Program Name: Dice.java          Input File: dice.dat**

There are many games in which six-sided dice are used to determine a random outcome of the game being played. The dice numbered 1-6 can be represented with 7 characters across and 5 characters down and a lower case 'o' for each dot on a die:

-------   -------   -------   -------   -------   -------

|     |   | o   |   | o   |   | o o |   | o o |   | o o |

|  o  |   |     |   |  o  |   |     |   |  o  |   | o o |

|     |   |   o |   |   o |   | o o |   | o o |   | o o |

-------   -------   -------   -------   -------   -------

James is writing a game that uses dice like these in his program. You are to write a program that will print the dice for him.

**Input**

The first line of input will contain a single integer n that indicates the number of dice that you are to print. The next n lines will each contain a single integer m, 1 ≤ m ≤ 6, that indicates which die is to be printed.

**Output**

For each integer m you will print the corresponding die. Print exactly one blank line between each die printed.

**Example Input File**

2

3

1

**Example Output to Screen**

-------

| o   |

|  o  |

|   o |

-------

-------

|     |

|  o  |

|     |

-------

**3. Little Hands**

**Program Name: Little.java                            Input File: little.dat**

Two card poker is played using a standard 52 card deck. Each card in the deck has a rank and a suit. The possible ranks are Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, and Ace. The possible suits are Clubs, Diamonds, Hearts, and Spades. Each combination of suit and rank exists once in the deck leading to 52 cards. In two card poker a hand consist of only two cards.

The possible two card hands that can be formed are pairs, flushes, and high card.

Pair is the highest ranking hand in two card poker. To be a pair both cards in the hand must be of the same rank. So, if a hand consists of the Nine of Spades and the Nine of Hearts, the hand is a pair of Nines. Ties between pairs are broken by the rank of the cards in the pair. The list of ranks in the previous paragraph show the ordering of ranks from lowest, Two, to highest, Ace. Thus a pair of Aces is better than all others pairs except another pair of Aces. A pair of Kings is better than all other pairs except a pair of Aces and is equal to another pair of Kings, and so forth. If two pairs contain cards with the same rank, the tie is not broken.

The next highest ranking hand in two card poker is a flush. A flush is a pair of cards that have the same suit. So, for example, a hand consisting of the Nine of Spades and the Three of Spades is a flush. Ties between flushes are broken by the rank of the highest ranking card in each flush. For example a flush consisting of the Ace of Hearts and the Two of Hearts is better than a flush consisting of the King of Clubs and the Queen of Clubs because Ace is a higher rank than King. If there is a tie between the highest ranking card of flushes, the rank of the second card is used to break the tie. So, for example, a flush with the Ten of Hearts and the Eight of Hearts is better than a flush with the Ten of Clubs and the Four of Clubs because Eight is a higher rank than Four. If two flushes contain cards with the same rank, the tie is not broken.

The lowest ranking hand in two card poker is referred to as a high card hand.  This is a hand that is not a pair or a flush. Ties between high card hands are broken by the rank of the highest ranking card in the hand. Thus, a high card hand with the Jack of Hearts and the Ten of Clubs is better than a high card hand with the Ten of Hearts and the Eight of Spades. If there is a tie between the highest ranking card of the high card hands, the rank of the second card is used to break the tie. If two high card hands contain cards with the same rank the tie is not broken.

Write a program that determines the best two card poker hand in a group of hands based on the above rules.

**Input**

The first line of input will contain a single integer n that indicates the number of data sets to process. The remainder of the input consists of those n data sets.

 Each data set will consist of two parts. The first part of a data set is a single integer m that indicates how many hands of cards are in this data set. The next m lines of the data set are the hands. Both cards in the hand appear on the same line, separated by a comma and a single space. The cards in a hand are not in any particular order.

Each data set is assumed to be dealt from a standard deck so in a given data set no card (unique combination of suit and rank) will appear more than once. It is possible for the same card to appear in multiple data sets.

**Output**

For each data set, print one line. On that line print out what type of hand the highest ranking hand in the data set is using capital letters. The three possible types of hands are: PAIR, FLUSH, and HIGH CARD. Then print a single space and  the number of hands in the data set that qualify as the highest hand. This is not the same as the number of hands in the data set that are the same type as the highest hand. For example if a data set contains 3 pairs, but only two of those three are the highest hand print out 2, not 3.

**Example Input File**

5

1

Two of Clubs, Three of Clubs

3

Two of Clubs, Three of Clubs

Two of Hearts, Two of Spades

Ace of Hearts, King of Spades

4

Three of Spades, Ace of Spades

Ace of Hearts, Three of Hearts

Ace of Clubs, Three of Clubs

Ace of Diamonds, Two of Diamonds

4

Ace of Spades, Ace of Hearts

King of Diamonds, King of Hearts

Ace of Clubs, King of Clubs

Ace of Diamonds, King of Spades

5

Six of Diamonds, Seven of Clubs

Seven of Hearts, Five of Diamonds

Seven of Diamonds, Four of Clubs

Seven of Spades, Six of Clubs

Six of Hearts, Five of Spades

**Example Output to Screen**

FLUSH 1

PAIR 1

FLUSH 3

PAIR 1

HIGH CARD 2

**4. Mary Had a Little Lamb**

**Program Name: Mary.java          Input File: mary.dat**

Write a program that will print all of the data in the data file.

**Input**

One or more lines of data.

**Output**

Print the input file exactly as it appears in the input file.

**Example Input File**

MARY HAD A LITTLE LAMB.

ITS FLEECE WAS WHITE AS SNOW.

EVERYWHERE THAT MARY WENT,

THE LAMB WAS SURE TO GO.

**Example Output to Screen**

MARY HAD A LITTLE LAMB.

ITS FLEECE WAS WHITE AS SNOW.

EVERYWHERE THAT MARY WENT,

THE LAMB WAS SURE TO GO.

**5. Multiple**

**Program Name: Multiple.java          Input File: multiple.dat**

For a given positive integer a, you are to find the smallest multiple of a that does not contain any digits in a given set of digits.

**Input**

The first line of input will contain a single integer n that indicates the number of lines to follow. Each the following n lines will contain a positive integer j followed by a space and a string s of numeric characters (digits 0 through 9).

**Output**

For each integer j, print the smallest multiple of j that does not contain any of the digits in s.

**Note:** You may assume there will always be a multiple of j that fits the criteria listed.

**Example Input File**

3

22 017

15 3425

12 20465

**Example Output to Screen**

44

60

888

**6. Octo-man**

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

The famous crusader Octo-man keeps watch over illegal number patterns – especially numbers containing the digit eight. You are to write a program that will create this logo for his super-crusader shirt.

**Input**

No input.

**Output**

Print this logo exactly as it appears below.

**Example Input File**

There is no input for this problem.

**Example Output to Screen**

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

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

\*\*\*\*\*               \*\*\*\*\*

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

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

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

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

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

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

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

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

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

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

\*\*\*\*\*               \*\*\*\*\*

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

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

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

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

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

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

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

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

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

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

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

\*\*\*\*\*\*             \*\*\*\*\*\*

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

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

**7. Perfection**

**Program Name: Perfect.java          Input File: perfect.dat**

A perfect number is one that is equal to the sum of its factors excluding itself. The number 6 is a perfect number because the factors of 6, excluding 6, are 1, 2, and 3 and 1 + 2 + 3 = 6.

A number is considered to be deficient if the sum of its factors excluding itself is less than the number. The number 8 is deficient because its factors, excluding 8, are 1, 2, and 4 and 1 + 2 + 4 = 7 which is less than 8.

A number is considered to be abundant if the sum of its factors excluding itself is greater than the number. The number 12 is abundant because its factors, excluding 12, are 1, 2, 3, 4, and 6 and 1 + 2 + 3 + 4 + 6 = 16 which is more than 12.

**Input**

The first line of input will contain a single integer n that indicates the number of values to process.  Each of the next n lines will consist of one positive integer.

**Output**

On a single line for each input value print whether the input value is PERFECT, DEFICIENT, or ABUNDANT.

**Example Input File**

3

6

8

12

**Example Output to Screen**

PERFECT  
DEFICIENT  
ABUNDANT

**8. Prime Reduction**

**Program Name: Prime.java                            Input File: prime.dat**

Given an integer, find its largest prime factor.

**Input**

The first line will contain a single integer n that indicates the number of data sets in the input. Each data set will consist of a line containing a single integer i, 2 <= i <*=* 1000.

**Output**

For each data set in the input, output a single line with the largest prime factor for the given integer.

**Example Input File**

5

2

7

45

48

512

**Example Output to Screen**

2

7

5

3

2

**9. Status Arrays**

**Program Name: Status.java          Input File: status.dat**

The status of each element of an array of integers can be determined by its position in the array and the value of the other elements in the array. The status of an element e in an array of size n is determined by adding the position p,   1 <= p <= n, of the element in the array to the number of array elements in the array that are less than e.

For example, consider the array containing the integers:  6 9 3 8 2 3 1. The status of the element 8 is 9 because its position is 4 and there are 5 elements in the array that are less than 8.

**Input**

The first line of input will contain a single integer n that indicates the number of arrays to follow. Each of the following n lines will contain the elements of a single array. Each element will be an integer and the elements will be separated by a single space.

**Output**

You will print the elements of the original array from low to high status order. In the event there is a tie for status of two or more elements, you will output them in numerical order, if possible. For example, the array elements given above, 6 9 3 8 2 3 1, have statuses 5, 8, 5, 9, 6, 8, 7 respectively. Therefore the output will be 3 6 2 1 3 9 8.

Note: If two or more elements have the same status and the same value, either can be printed first.

**Example Input File**

3

6 9 3 8 2 3 1

14 -3 4 6 9 10 -2 4 0

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

**Example Output to Screen**

3 6 2 1 3 9 8

-3 4 -2 6 14 0 4 9 10

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

**10. Unite 4**

**Program Name: Unite4.java                            Input File: unite4.dat**

You are excited to learn your parents have returned from the toy store with a game for you.  You are not so excited to find it is called “Unite 4”, and is a generic knock-off of a popular game.  You are also not so thrilled to discover that there are many pieces missing, so much so that the game is unplayable.  Luckily, the rules are intact, so you decide to write a program that simulates the game.

“Unite 4” is similar to tic-tac-toe and is played on a vertical board consisting of 42 holes in 7 columns and 6 rows.  One player has red discs and the other has black discs.  Players drop their discs in the top of the board, which drop straight down, stacking on one another.  The object of the game is to get four of your colored discs in a row: either horizontally, vertically, or diagonally.

For example, consider a game where the following moves were made:

Red column 4, Black column 5, Red column 3, Black column 4, Red column 1, Black column 4, Red column 2

The board would look like the following after each of the moves (with 'R' representing red-colored discs, 'B' representing black-colored discs , 'O' (a capital letter 'o') representing an empty hole, and rows and columns numbered):

Red 4     Black 5   Red 3     Black 4   Red 1     Black 4   Red 2

 1234567   1234567   1234567   1234567   1234567   1234567   1234567

1OOOOOOO  1OOOOOOO  1OOOOOOO  1OOOOOOO  1OOOOOOO  1OOOOOOO  1OOOOOOO

2OOOOOOO  2OOOOOOO  2OOOOOOO  2OOOOOOO  2OOOOOOO  2OOOOOOO  2OOOOOOO

3OOOOOOO  3OOOOOOO  3OOOOOOO  3OOOOOOO  3OOOOOOO  3OOOOOOO  3OOOOOOO

4OOOOOOO  4OOOOOOO  4OOOOOOO  4OOOOOOO  4OOOOOOO  4OOOBOOO  4OOOBOOO

5OOOOOOO  5OOOOOOO  5OOOOOOO  5OOOBOOO  5OOOBOOO  5OOOBOOO  5OOOBOOO

6OOOROOO  6OOORBOO  6OORRBOO  6OORRBOO  6RORRBOO  6RORRBOO  6RRRRBOO

At this point the game would end as the player with the red discs has four in a row.

**Input**

* The first line will be a single integer n that indicates the number of data sets in the input.
* Each data set will consist of:
  + A line with a single integer m that indicates the total number of moves, 1 <= m <= 42.
  + The next line will contain a space-separated list of m integers, representing alternating moves by the players, with the player with the red discs always going first.  Each move will be an integer c, 1 <= c <= 7, that represents into which column the player drops their disc.  Note that all moves will be valid (a player will not drop their disc into a column that is completely full).

**Output**

For each data set in the input, output the final layout of the board after applying all the moves in the input, as shown in the above example.  On the line after the board layout, output “RED WINS” if the player with the red discs has won, “BLACK WINS” if the player with the black discs has won, or “NO WINNER YET”, if neither have won.  Note that if there is a winner, there will not be any moves after a player has won.

**Example Input File**

4

7

4 5 3 4 1 4 2

11

4 5 5 6 6 7 6 7 7 4 7

21

1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7

8

2 7 3 7 4 7 3 7

**Example Output to Screen**

 1234567

1OOOOOOO

2OOOOOOO

3OOOOOOO

4OOOBOOO

5OOOBOOO

6RRRRBOO

RED WINS

 1234567

1OOOOOOO

2OOOOOOO

3OOOOOOR

4OOOOORR

5OOOBRRB

6OOORBBB

RED WINS

 1234567

1OOOOOOO

2OOOOOOO

3OOOOOOO

4RBRBRBR

5BRBRBRB

6RBRBRBR

NO WINNER YET

 1234567

1OOOOOOO

2OOOOOOO

3OOOOOOB

4OOOOOOB

5OOROOOB

6ORRROOB

BLACK WINS

**11. Weak Passwords**

**Program Name: Weak.java          Input File: weak.dat**

The technology department requires that teacher passwords cannot be “weak” passwords. A password is considered to be “weak”, if one of the following is true about the password:

* Does not contain at least one digit.
* Does not contain at least one vowel.
* Does not contain at least one consonant.
* Contains three or more consecutive vowels.
* Does not contain at least 8 characters.

**Note:**

* Vowels are A, E, I, O, and U. All other letters are consonants. Case is ignored.
* Digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

It is your job to write a program to see if a given password is considered to be a “weak” password.

**Input**

The first line of input will contain a single integer n that indicates the number of passwords to follow. Each the following n lines will contain a single string of keyboard characters with no spaces.

**Output**

For each password, on a single line print “WEAK” if the password is considered “weak” by the criteria listed above or print “VALID” if it is not “weak”.

**Example Input File**

4  
MARY\_HADaLITTLElamb  
MARY\_HAD1LITTLElamb  
{RogerRabbit2}  
Raei0uST

**Example Output to Screen**

WEAK

VALID

VALID

WEAK