University Interscholastic League

Computer Science Competition

2002 State 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 10.

2. Problems have different point values, depending on their difficulty.

3. Several problems have a reference to columns of an input line, such as column 1 or columns 1-3. In these cases column is referring to the character position on the input line. Column 1 refers to the first character position on the line, while columns 1-3 refer to the first three positions on the line.

4. 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.

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

II. Point Values and Names of Problems

Number	**Name**	**Point Value**
Problem 1	Mine Hunt	6
Problem 2	Scoring Machine	5
Problem 3	Barney!	5
Problem 4	What Does It Look Like?	4
Problem 5	What's Next?	5
Problem 6	What Time Is It?	4
Problem 7	How Hard Can It Be?	5
Problem 8	Do You Mine?	4
Problem 9	A Little Poker Anyone?	6
Problem 10	I Want a Recount!	6
Total		**50**

Problem 1

Mine Hunt

6 Points

Program Name: minehunt.cpp   Input File: minehunt.dat

In the game Mine Hunt, you are trying to determine the position of some number of mines on a mine field (an NxN grid). Each position on the field can be represented in 1 of four ways:

Cell Type	Condition of Cell	Neighbors	Notation on Cell	System response
A	Unrevealed position – All positions start as unrevealed and are "revealed" when the player touches them.	Does not matter	x	N/A
B	Revealed position with a mine on it.	Does not matter	@	System displays the mine.
C	Revealed position with no mine on it.	1-8 neighbors have mines on them	The number (1-8) of neighboring positions that have mines.	System displays the number of neighbors with a mine.
D	Revealed position with no mine on it.	0 neighbors have mines on them.	.	System displays a . and processes a user touch on all unrevealed neighbors.

In this program, you will be given a definition of a mine field followed by a series of "touches". Your program is to display the mine field as it would appear after the touches. For example, given the mine field on the left, you can see that the series of touches on the initial screen of (9,4) – (2,5) – (9,7) would result in the screen on the right.

1	@									
2										
3			@					@		@
4								@		
5			@			@				
6										
7				@						
8			@	@						
9	@						@			
10		@			@					
	1	2	3	4	5	6	7	8	9	10

x	1	.	.	.	.	.	.	.	.
x	2	1	1	.	.	1	1	2	1
x	x	x	1	.	.	2	x	x	x
x	x	x	2	1	1	3	x	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	3	x	x	@	x	x	x
x	x	x	x	x	x	x	x	x	x
1	2	3	4	5	6	7	8	9	10

Input

Input to your program is a single mine field followed by a series of "touches". The first 20 lines contain the mine field each with a row of 20 mine field positions on it in columns 1-20. Each position contains either a "." period or an "@" with a period representing an empty position and the "@" representing a mine at that position. Following the mine field definition will be a series of "touches". Each touch (1≤row,column≤20) will be on a line by itself with the (row,column) combination representing the position found by counting from top to bottom starting with 1 for the row and from left to right starting with 1 for the column. The row value will start in column 1 of the input file and will be followed by exactly 1 blank space followed by the column.

Output

Using the mine field definition and the series of touches, your program should simulate the mine search starting from the initial state (all fields display “x” whether they contain a mine or not) by applying the touches 1 at a time per the rules in the above table. Your program should then print the completed table with the notations from the above table on a series of 20 lines of output with the columns of the mine field in columns 1-20.

Your program should print only the completed simulated mine field. Any other output will be considered extraneous output and will be judged incorrect.

Example: Input File

```
@.........@.........
....................
..@....@.@..@....@.@
.......@.........@..
..@..@......@..@....
....................
...@.........@......
..@@........@@......
@.....@...@.....@...
.@..@......@..@.....
@.........@.........
....................
..@....@.@..@....@.@
.......@.........@..
..@..@......@..@....
....................
...@.........@......
..@@........@@......
@.....@...@.....@...
.@..@......@..@.....
9 4
2 5
9 7
17 8
```

Output to screen

```
x1.......1xxxxxxxxxx
x211..1122xxxxxxxxxx
xxx1..2xxxxxxxxxxxxx
xxx2113xxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxx3xx@xxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxx3112xxxxxxxx
xxxxxx211..1xxxxxxxx
xxxx211....1xxxxxxxx
xxxx2......1xxxxxxxx
xxxx2111.112xxxxxxxx
xxxxxxx1.1xxxxxxxxxx
xxxxxxx1.1xxxxxxxxxx
```

Problem 2

Scoring Machine

5 Points

Program Name: scoring.cpp   Input File: scoring.dat

In a hypothetical programming contest all judging activity is logged, and those logs are then used to verify the outcome of the contest. Teams are ranked by a final score determined by the number of problems solved and the number of rejections. You are given the following assumptions for this problem:

1. There are 10 teams in the contest. The judges and scorers use only the team numbers (1-10) to for identification to preserve anonymity and guarantee the most accurate and fair outcome.
2. There are 6 problems in the problem set.
3. Each problem in the problem set has its own point value.
4. A rejection by a team on a problem costs a point against the problem's point value **only** if the problem is solved.
5. The minimum score awarded for a problem that is solved is 1 point even if the rejection penalties for the problem exceed the point total for the problem.
6. Teams are ranked by the final score (highest to lowest).
7. If two or more teams have the same final score, then the following tiebreakers are applied to determine rank:
   - The team with the fewest rejections (regardless of whether the problem was accepted) wins.
   - If the number of rejections is the same for two teams who have the same final score, then the team who had a problem accepted first wins.

Your program will read a contest definition along with the contest logs and determine the final rankings for all 10 teams in the contest.

Input

Input to your program consists of a contest definition followed by the contest logs. The first line of the input file contains the contest definition and consists of a series of integer scores ($1 \leq S_p \leq 6$) where the integers represent the point value of the problems in order (1-6). Each of the remaining input lines represent a judging action. Each judging action has the following format:

1. the result of the judging action ('A' for accept or 'R' for reject) and appears in column 1
2. a single space
3. the team number ($1 \leq T \leq 10$) for the judging action
4. a single space
5. the problem number being judged ($1 \leq P \leq 6$)

You can be assured that there will be no extraneous input including blank lines or trailing characters at the end of a line. You can also be assured that there will be a maximum of 1 acceptance judging line for a given team number/problem number combination. Your program should read to the end of file processing all judging actions before determining the final team rankings.

Output

The output from your program consists of a single line in which you will print the team numbers in rank order for the teams who solved at least one problem. Do not print anything for the teams that solved zero problems. The integer team numbers should be listed on a single line in order of rank starting with the winning team with exactly one blank space between each team number.

Example: Input File

```
4 8 2 4 10 1
A 1 3
A 4 1
R 10 5
A 2 3
A 8 1
A 7 4
R 5 6
R 7 5
R 2 6
R 7 5
A 10 4
R 4 2
R 1 4
A 5 3
R 7 2
R 5 6
A 2 4
R 10 2
A 1 4
R 4 4
A 7 5
R 5 6
A 4 3
R 4 2
R 8 3
A 4 4
R 2 6
R 10 2
A 7 1
A 5 6
R 10 2
R 4 2
A 7 2
R 4 2
A 8 3
A 10 2
```

Output to screen

```
7 10 4 2 1 8 5
```

Problem 3

Barney!

5 Points

Program Name: barney.cpp   Input File: barney.dat

Barney is a dinosaur from our imagination. But he is also a television show in which a big purple dinosaur sings children's songs. Many of Barney's songs are really just old songs with new lyrics. The purpose of this program is to use a set of new lyrics and find an old song with a tune that can be used for the new lyrics. Your program will do so by scanning a set of old songs and counting the number of syllables per line. Your program will try to find the old song whose syllables per line most closely match the corresponding syllables per line from a new lyric. To count the syllables on a line, your program should implement the following rules.

1. All words will have at least 1 vowel.
2. If a word has a set of one or more vowels {a, e, i, o, u, y} separated by one or more consonants {set of all letters minus the set of vowels) from another set of vowels, then each set of vowels in the word would constitute a syllable. The following table shows some examples that implement these rules.

Sample Word	# of Syllables	Explanation
chat	1	There is only 1 vowel set {a} and there are no other vowel sets that are separated from it by 1 or more consonants.
apple	2	There are 2 vowel sets, {a} and {e}, separated by a set of consonants {p, p, l}.
alphabetic	4	There are 4 vowel sets separated from one another
one	2	There are 2 vowel sets, {o} and {e}, separated by a consonant {n}. Now you can see the weakness in this method, but it is OK because we are only going to identify candidate songs.
bureau	2	There are 2 vowel sets, {u} and {eau}, separated by a consonant {r}.
bayou	1	Here is the weakness of including y as a vowel.
city	2	And here is an argument for including y as a vowel.

Each song that you examine (old song or new) will contain exactly 4 lines. For each of the four lines, you should compute the sum of the differences squared (SoDS) in the number of syllables for corresponding lines.

Old Songs						New Song	
"This Old Man"	**#**	**SoDS**	**"Twinkle Twinkle Little Star"**	**#**	**SoDS**	**"I Love You"**	**#**
this old man	3	1	twinkle twinkle little star	7	9	i love you	4
he played one	4	0	how i wonder what you are	8	16	you love me	4
he played knick knack	4	4	up above the world so high	8	4	we are a happy	6
on my thumb	3	0	like a diamond in the sky	8	25	family	3
Sum of the Differences Squared		**5**	**Sum of the Differences Squared**		**54**		

So the total of the sum of differences squared for "This Old Man" is 5 versus 54 for "Twinkle Twinkle Little Star". Therefore, we identify "This Old Man" as a better candidate for the new lyrics than "Twinkle Twinkle Little Star". In this program, you will be given a library of up to 20 old songs followed by a series of new songs. For each of the new songs, your program will identify the best candidate old song using the sum of the differences squared method explained above.

Input

Input to your program consists of two sections. The first section describes the library of old songs. Line 1 of the input file will contain a single integer (1≤L≤20) that describes the number of old songs in the library. Each 4 of the next L*4 lines of the input file will contain the lyrics of one old song. Each old song will be made up of 4 lines of 1 to 80 lower case alphabetic characters only. Words in the lyrics will be separated by a single space.

A series of "new songs" will start at line # 4L+2 each formatted with the same rules as those in the library. Your program should read each of the "new songs" and determine which "old song" is the best candidate based on the sum of differences squared methodology described above. If there is a tie for the best candidate, your program should select the song involved in the tie that appears first in the library.

Output

For each "new song" in the input file, your program should print exactly two lines of output. The first line of output should be the lyrics of the first line of the "old song" that is the best candidate. The best candidate line should be followed by exactly 1 blank line.

Example: Input File

```
3
twinkle twinkle little star
how i wonder what you are
up above the world so high
like a diamond in the sky
the itsy bitsy spider
went up the water spout
down came the rain and
washed the spider out
this old man
he played one
he played knick knack
on my thumb
i love you
you love me
we are a happy
family
a bee cee dee e ef gee
aich i jay kay ell em en oh pea
cue arr ess tea you vee
double u ex why and zee
```

Output to screen

```
this old man

twinkle twinkle little star
```

Problem 4

What Does It Look Like?

4 Points

Program Name: building.cpp Input File: building.dat

The planners for any new construction need a program that can help them envision what some potential new plans for buildings will look like in relationship to the buildings around them. You are going to write just such a program for this problem. The ground area under consideration is a 6x6-block area each with a building on it. Each building is uniquely numbered from the set {A-Z, 0-9}. The block area is arranged and numbered is shown below.

North

	A	B	C	D	E	F	
	G	H	I	J	K	L	
West	M	N	O	P	Q	R	**East**
	S	T	U	V	W	X	
	Y	Z	0	1	2	3	
	4	5	6	7	8	9	

South

You can see that if you view the block from the North that the building at block D will block some or all of the building at block J. In fact, you will only see that top part of J that towers above the top of D. The same can be said to be true for the building P when considering both building D and J. In the program, you will be given a height for each of the buildings in the block area and then print the view from one of the four directions. Although a person would actually see a perspective from a single point, we will assume that the view is from a plane such that you would not see building K if you were standing at a point such that you could really see it between L and R.

Your program is going to show the view for a block strategy that defines the height of each building and then uses the block ID itself to show the view from one of the four directions. If the building at position D were 3 units high and the building at position J were 7 units high, the Northern view on your program would show a column of three D's at the bottom with a column of four J's in the middle with a column of 3 space characters at the top. The sample below represents a block strategy and the view of the block strategy from the South in the same format that your program should print. (Note that all views will show the complete 10 units of height even if no building is that tall.)

```
5 3 10 4 6 8
2 8 3 8 4 1
4 7 3 9 9 2
1 5 5 2 3 6
3 4 4 5 2 4
2 4 4 1 7 3
```

Figure 1: Sample Building Strategy

```
  C
  CPQ
 HCPQF
 NCP8F
 NCP8X
ATU18X
M56183
Y56189
456189
456789
```

Figure 2: Southern View of Sample Building Strategy

Input

Input to your program consists of one building strategy. The strategy consists of 6 lines of input and each line of input contains 6 integers (1≤H≤10) separated by integers. Each integer represents the height of the building at the block location corresponding with the picture above such that the first integer in the strategy represents the building at block A and the last represents the building at block 9. Your program should read the block strategy and print the view from the South for the strategy.

Output

Output for your program should be exactly 10 lines of output. The lines represent the view from the south of the block set described in the input file. For each position in the 10 rows by 6 columns of output, you will print the appropriate character if a building is seen at that position. If no building occupies that position, you should print a space for that position on the southern view.

Example: Input File

```
5 3 6 4 6 8
3 8 10 7 4 1
6 7 5 10 9 2
6 5 3 2 3 6
3 6 3 5 2 4
2 3 2 1 7 7
```

Output to screen

```
  IP
  IPQ
 HIPQF
 NIP89
SZIP89
SZO189
SZO189
Y50189
456189
456789
```

Problem 5

What's Next?

5 Points

Program Name: next.cpp   Input File: next.dat

There are several applications in which the order of character sequences in the English language is important. For example, when designing the keyboard of a PDA, typewriter, or other text entry machine, the ergonomic movement of fingers can be optimized by properly arranging the keys. In this problem, you will build a program that analyzes some plain text to determine the frequency of combinations of characters in English words.

Your program will read a file of sample English text. For every alphabetic character in the file, your program will determine what alphabetic character (if any) follows. Your program will then count the number of times that each alphabetic character follows all others. The text will contain non-alphabetic characters and all of these including the beginning and end-of-line should be treated as word breaks. You program will not consider combinations that include or span word breaks. After analyzing the entire input file, your program should determine the 5 most frequently occurring alphabetic character paired sequences and print them in order starting with the most frequent. In case of a tie, the 2-character sequence should be listed in alphabetical order.

For example, consider the following line.

> This is the sample text for your program. Notice the many "th" sequences.

would yield the following list of 5-most frequently occurring alphabetic paired sequences. Notice that your program must convert all alphabetic characters to lower case to produce the correct answer.

```
1: t h
2: a m
3: c e
4: h e
5: i s
```

Input

Input to your program is a free-form text file that can contain any printable character in the ASCII range 0-127. (Hint: Do not be tempted to make this harder than it is.) The only restriction on the input file is that no single line in the input file will contain more than 80 printable characters.

Output

After reading the input file, your program should print the 5 most frequently occurring pairs of alphabetic characters each on a line by itself. Each line of output should contain: the rank (1-5) in column 1, a colon in column 2, a space in columns 3 and 5, the first character of the sequence at that rank in column 4, and the second character of the sequence at that rank in column 6.

Example: Input File

```
There are several applications in which the order of character sequences in
the English language is important.  For example, when designing the keyboard
of a PDA, typewriter, or other text entry machine, the ergonomic movement of
fingers can be optimized by properly arranging the keys.  In this problem,
you will build a program that analyzes some plain text to determine the
frequency of combinations of characters in English words.
```

Output to screen

```
1: e r
2: i n
3: t h
4: h e
5: e n
```

Problem 6

What Time Is It?

4 Points

Program Name: logtime.cpp   Input File: logtime.dat

Compute the number of seconds since January 1, 2002 at midnight for any point in time for the year. You are examining some log data from some high-performance runs as part of your job and need to do some data modeling. Unfortunately, the log contains dates and times that you need to convert into the number of seconds elapsed since the beginning of the year (2002).

The date/time combination in the log is in the fixed format: “mm/dd/ccyy hh:mm:ss”. For example, the date/time (military time) from the input log might look like “04/08/2002 19:27:43” which is April 8, 2002 at 7:27:43 p.m. Your program would then compute the elapsed time (not including the current second) since 01/01/2002 00:00:00. In the case of our example, you would compute the elapsed time as 8450863 seconds.

Input

Input to your program will consist of a series of date/time test points. Each one will be formatted as

```
mm/dd/ccyy hh:mm:ss
```

with exactly one space between the date and the time. There will be no extraneous data on any line and all dates and times will be valid. You may also assume that the year will always be 2002.

Output

For each line of input, you should print the following output line.

```
mm/dd/ccyy hh:mm:ss => X
```

where X is the number of seconds elapsed. Print the output line exactly in the format described above (paying attention to the “ => “ because it makes it easier and obvious to judge).

Example: Input File

```
07/01/2002 18:54:19
04/08/2002 19:27:43
12/31/2002 23:59:59
```

Output to screen

```
07/01/2002 18:54:19 => 15706459
04/08/2002 19:27:43 => 8450863
12/31/2002 23:59:59 => 31535999
```

Problem 7

How Hard Can It Be?

5 Points

Program Name: baker.cpp **Input File: baker.dat**

There is a card game that resembles Klondike solitaire called “Baker’s Game” or “FreeCell”. The object of the game is not important in this problem because the first step is to deal cards and then determine just how difficult the dealt hand is to play. The 52 cards are dealt into 8 piles. The 4 leftmost piles each have 7 cards and the 4 rightmost piles each have 6 cards. A hand is scored for difficulty based on the depth and position of the key cards (aces, 2’s, 3’s and 4’s). Difficulty points are added as described in the table below.

Card/Rank	Points Added	Multiplier
Aces	5	Number of non-key cards on top of the key card. The top card is considered to be a depth of zero. So, if a ‘key card’ has two key cards on it and 3 non-key cards, the multiplier is 3. The total difficulty points for an individual card is its difficulty points * the multiplier for the position.
2’s	4	
3’s	2	
4’s	1	

Points are then subtracted while multiple key cards appear on the same pile. Specifically, for each pile, if the number (K) of key cards in the pile exceeds 1, K-1 points are subtracted from the difficulty total.

Consider the sample hand below. Note that Aces are denoted as a ‘1’; Tens are denoted as a ‘T’; Jacks are denoted as a ‘J’; Queens are denoted as a ‘Q’; and Kings are denoted as a ‘K’.

Pile Number	1	2	3	4	5	6	7	8
Piles	2	9	T	7	K	4	2	2
	J	6	K	5	9	8	K	9
	T	1	6	J	8	T	1	1
	T	3	9	Q	K	3	6	J
	J	5	4	3	4	7	5	8
	5	Q	4	1	Q	8	Q	7
	2	7	3	6				
Difficulty Points	4*5	5*3 + 2*3	0	2*1 + 5*1	1*1	1*4 + 2*2	4*4 + 5*3	4*4 + 5*3
Deductions	1	1	2	1	0	1	1	1
Pile Total	19	20	-2	6	1	7	30	30
Grand Total								**111**

You can see from the work in the above table that this hand is rated with 111 difficulty points.

Input

Input to your program consists of a series of dealt hands. Each hand consists of 6 lines of 8 ranks and 1 line of 4 ranks. These 7 lines form 4 piles of 7 cards and 4 piles of six cards where the first line of the hand being at the bottom of the piles. Each line will contain only valid ranks and there will be no spaces or other characters in the input file. Your program should read to end of file.

Output

Output from you program will consist of one line per dealt hand. Your program should print the difficulty score for each dealt hand starting in column 1 on consecutive lines. Your program should not print any other extraneous output or blank lines.

Example: Input File

```
29T7K422
J6K598K9
T16J8T11
T39QK36J
J5434758
5Q41Q8Q7
2736
94J4TT13
4K63T3Q9
7KQ25156
52J791Q8
K4T7Q263
9J178KJ8
6528
Q9T2J53K
T2J7182K
67446398
18755393
1921Q45T
48K66KJQ
7QTJ
```

Output to screen

```
111
89
116
```

Problem 8	Do You Mine?	4 Points

Program Name: mine.cpp   Input File: mine.dat

Your company mines for several minerals. Before they mine an area, they take a core sample and analyze it. That core sample is broken down into smaller uniform “samplets” and run through a scanning machine that identifies the minerals in it. Your program will analyze the report from the scanning machine and then determine if further mining is warranted based on the following cost/benefit parameters.

Cost/Product	Samplet Code	Revenue per Unit
Nickel	N	0.35
Platinum	P	4.93
Uranium	U	2.32
Copper	C	0.63
Iron	I	0.86
Sulfur	S	1.15
Dirt	D	0.00

In addition, there is a cost of mining of 0.17 for every unit mined no matter what is produced from the unit. This includes all costs for equipment, labor, transportation, processing, packaging, taxes, insurance, environmental restoration, and research.

Your program will be given the results from the scanning as a block of 50 samplets and determine the cost/benefit total for this sample. You will then determine which of the three strategies should be followed.

Strategy	Criteria	Description
Do Not Mine	Total < -1.00	Sample indicates mining will not be profitable.
Further Sampling	-1.00<=Total<=5.00	Sample is inconclusive and further samples should be taken.
Begin Mining	Total > 5.00	Sample indicates mining will be profitable.

For example, if your program is given the following sample report from the scanning machine:

```
DDDDCDDDDUUUCCCCDDDDDDDDIDDDDDSDDDDSSSDDDDDDDSDDDN
```

You would determine that the total cost is $8.50 and that the revenue would be $17.07. Therefore, the cost/benefit total for this sample would be $8.57 and the strategy would be to begin mining.

Input

Input to your program consists of a series of samples. Each sample is a string of 50 characters from the list of Samplet Codes above on a line by itself.

Output

For each sample, your program should print one of the strategies on a line by itself based on the evaluation of the samplets. You should print the strategy exactly as it appears in the Strategy column above starting in column 1.

Example: Input File

```
DDDDCDDDDUUUCCCCDDDDDDDDIDDDDDSDDDDSSSDDDDDDDSDDDN
DDDDDDCCCDDDDDDUDDDDDDDDDDSSDDDSSSDDDDDDNNNNNDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
```

Output to screen

```
Begin Mining
Further Sampling
Do Not Mine
```

Problem 9

A Little Poker Anyone?

6 Points

Program Name: poker.cpp **Input File: poker.dat**

In the card game "Poker" it is very important to be able to quickly identify what hand you have. Cards are described by rank and suit. Ranks are 2-9, T for 10, J for Jack, Q for Queen, K for King and A for Ace. There are four suits: S for Spades, H for Hearts, D for Diamonds, and C for Clubs. Poker is only played with a complete deck (exactly 1 card for each of the suit/rank combinations).

There are 9 types of hands that can be evaluated as described in the table below. They are in descending order of rank with Straight Flush being the highest hand.

Hand	Explanation	Example
Straight Flush	Five cards in uninterrupted rank sequence all of the same suit.	`4H 5H 6H 7H 8H`
Four of a Kind	Four cards of the same rank.	`7S 7H 7D 7C 3D`
Full House	Three cards of the same rank and two cards matching in another rank.	`KD KH KS 9D 9C`
Flush	Five cards not in rank sequence but all of the same suit.	`3H 6H 7H TH AH`
Straight	Five cards in uninterrupted rank sequence of more than one suit. Note that ace is considered the highest rank and 2 is the lowest rank.	`8S 9C TD JH QS`
Three of a Kind	Three cards of the same rank.	`2H 6D 6C 6H JS`
Two Pair	Two cards of one rank and two cards matching in another rank.	`4S 4C 7H TD TC`
One Pair	Two cards of one rank	`3D 4H 8D 8C JH`
High Card	None of the above hands applies and the player should evaluate based on the highest card.	`2C 5S 6H TS KD`

Input

Input to your program consists of a series of five-card dealt hands. Each card is represented by 2 characters: the first is the rank and the second is the suit. There will be exactly one space separating the cards from one other.

Output

For each dealt hand in the input file, your program will produce exactly one line of output. Your program should print the dealt hand exactly as it appears in the input file followed by a space in column 15. Your program should then print the highest hand that the dealt hand qualifies for starting in column 16.

Example: Input File

```
7D 8D 4D 5D 6D
2D 2H 3S 2S 2C
8S 5D 5H 8C 8D
2C 7C TC JC 9C
TD KC QD JD AD
QS 7D 6D QH QC
8D 5H 5C 6H 8S
QD 4H 5S 6C 4D
2C JC QC KC AS
```

Output to screen

```
7D 8D 4D 5D 6D Straight Flush
2D 2H 3S 2S 2C Four of a Kind
8S 5D 5H 8C 8D Full House
2C 7C TC JC 9C Flush
TD KC QD JD AD Straight
QS 7D 6D QH QC Three of a Kind
8D 5H 5C 6H 8S Two Pair
QD 4H 5S 6C 4D One Pair
2C JC QC KC AS High Card
```

Problem 10

I Want a Recount!

6 Points

Program Name: recount.cpp   Input File: recount.dat

In this problem set, you have seen some card games that require a complete set of cards. It always seems that there are 3 or 4 decks that have been put together to make a complete one. Your job is to write a program that reads a list of cards and determines which are missing and which are duplicates.

Cards are described by rank and suit. Ranks are 2-9, T for 10, J for Jack, Q for Queen, K for King and A for Ace. There are four suits: S for Spades, H for Hearts, D for Diamonds, and C for Clubs. A complete deck should have exactly one card for each rank/suit combination for a total of 52 cards. If more than one card of a rank/suit combination is found, it is said to be a “duplicate”. If any rank/suit combination is not contained in the deck, it is considered to be “missing”.

Input

The input file contains descriptions for series of decks. A card deck description starts with a single integer (1<=C<=100) on a line by itself that indicated how many cards are in the deck. The next line contains C 2-character card descriptions with the rank as the first character and the suit as the second character. A single blank character will separate the C card descriptions from each other.

Output

For each deck in the input file, your program should print a list of missing and duplicate cards.

The missing cards list starts with the label “`Missing: `”. Note that there is a single space after the colon. The missing cards then start in column 10. Each missing card consists of 2 characters: a single character rank followed by a single character suit. There should be a single space between characters.

The second line of output is the duplicate cards list which starts with label “`Duplicates: `“. The list of duplicated card rank/suit combinations is then printed in the same format described above for missing cards. The rank/suit combination is listed only once regardless of the number of times that the card is duplicated.

In the case of no cards qualifying as missing or duplicate, your program should print the word “None”. There should be a blank line printed after the duplicated cards list.

There should be no extraneous output or blank lines other than those explicitly described here.

Example: Input File (backslash means line is continued on next line)

```
52
3S KC AH 2D 9D 3D TD 4C 9C 5S AS 2C 9S 7D 3H KD QD 6D AC 2S 5D KH 8C QH JH JS 7C \
5C 7H 6C QC 7S TS 5H 2H 4H 6S 6H 8S 8D 4S KS 8H 4D 9H TC 3C TH AD QS JC JD
51
TC 9D KH 4H 4C 9C 9H KS 3H 4S 8S 5C AC JD 4D 8H TS JS KD TH 9S 7D 6D 8D AC QD 2D \
3S 2C TD AH 2S 2H JH KC AD 3H QS 5H 7C JC 6S 5S 8H 7H 7S 5D AS 6C 3C 8H
53
KD TD 7C 2H 3D 9S 7H TS KC JS 2D AH 7S 8D 3C 6H JC 2S 5C 2C QH QD QC 3S QS 7D AS \
JH KS 4C 4D 8S 6C 9D AC 6D 4S 3H 9H 6S 9C 8H JD 8C 5S TC 5H 4H TH KH 5D AD 7S
```

Output to screen

```
Missing: None
Duplicates: None

Missing: 6H QH 3D 8C QC
Duplicates: 3H 8H AC

Missing: None
Duplicates: 7S
```