University Interscholastic League

Computer Science Competition

2003 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. All problems have a value of 6 points.

3. Some 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	Space Camp	6
Problem 2	Cashing Out	6
Problem 3	Changing Tense	6
Problem 4	Virtual Computing	6
Problem 5	Pretty Poor Encryption	6
Problem 6	The Fraction Factor	6
Problem 7	Par for the Course?	6
Problem 8	Kennel Kritters	6
Problem 9	Just Picture It	6
Problem 10	You May Already Be A Winner!	6
Total		**60**

Problem 1

Space Camp

6 Points

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

You have made it through three grueling months at Space Camp, only to be confronted with the harsh final exam to graduate. As you don your space suit, Drill Instructor Grimley barks, "The rules are simple, maggots. You start in room 0 (west of force field 1), with force fields 0 and 1 closed. There is a single button in each room that will affect each force field in some manner. Your job is to navigate through the rooms until you reach room 4, but be sure not to open all the force fields behind you or you will be sucked out into space. Oh, and one more thing, you have one minute to complete the task."

```
              0___1___2___3___4___   <-- Force field numbers
[Outer Space] |_0_|_1_|_2_|_3_|_4_   <-- Room numbers
 Start in room 0 with Force fields 0 and 1 closed. Head east to room 4! -->
```

One minute! With your weak space legs, you realize your only hope is to rush to the next closed force field and push the button in that room, hoping it will open some force fields to allow you to proceed. You have no time to backtrack. It's not the best strategy, but maybe fate will be on your side.

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 3 components:

Start line - A single line, "START *V W X Y Z*", with each letter corresponding to the initial status of force fields 0-4, respectively. Each letter will either be "C", signifying the force field is initially closed, or "O", signifying the force field is initially open. Note that *V* and *W* (force fields 0 and 1) will always be "C".

Force Field Buttons - Each of the next 4 lines will contain 5 action codes showing how pressing the button in that room will affect each of the force fields. The first line will correspond to room 0, the next line for room 1, the next line for room 2, and the next line for room 3. The action codes will be in the format "*D E F G H*", corresponding to the action performed on force fields 0-4, respectively. Each action code will be one of the following:

"O" -- Open the force field. If it is already open, it remains open.
"C" -- Close the force field. If it is already closed, it remains closed.
"T" -- Toggle the force field. If it is open, close it. If it is closed, open it.
"N" -- Do nothing to the force field. If it is open, it remains open. If it is closed, it remains closed.

End line - A single line, "END".

Output Description

For each data set, there will be exactly one line of output. The line of output will be determined based on the result of your final exam. Remember your strategy:

1. Proceed to the easternmost room possible (until you reach a closed force field, which initially, will always be force field 1). If you are able to reach room 4, you have passed! Consider yourself, and your line of output, "SPACE CADET".

2. Push the button located in the room you are in, if you have not already done so. If you *have* already pushed this button, then you realize that you are not going to be able to make it, and that you are, and your line of output is, "SPACE MONKEY". If pushing the button causes all of the force fields behind (west of) you to be open, you will be sucked into space and become, as your line of output is, "SPACE GHOST". Consider force fields opening and closing to be instantaneous.

Just repeat your 2-step strategy until one of the three end conditions is met!

Sample Input

```
START C C C C C
O O O O O
O O O O O
O O O O O
O O O O O
END
START C C O O O
N N O O O
C O O O O
C O O O O
C O O O O
END
START C C O C O
N T T T T
T T T T T
T T T T T
T T T T T
END
```

Sample Output

```
SPACE GHOST
SPACE MONKEY
SPACE CADET
```

Problem 2

Cashing Out

6 Points

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

"Hurry up, Jorge, we're going to be late for the Seigfreid and Roy show, " George complained, folding his arms.
"I can't help it if I'm a better gambler than you, " Jorge smiled, playing with the casino chips in his hands. He waited patiently in the casino cashier's line for his turn to "cash out".
"Be sure to get enough bills, " George warned, "You still have to buy your ticket to the show and it costs $40, but you have to have exact change."
Jorge glanced down at the $45 worth of chips in his hands and smiled. "Well, it doesn't matter what bills the cashier gives me, I'll have $40 in exact change."
George looked at him questionably, thought for a second, and with a raising of his eyebrows in realization, admitted, "You're right. That's interesting. I wish there were a way to figure out, given two monetary amounts, if you are able to have change for the first amount, but not for the second amount."
"Hmmm," Jorge shrugged, "sounds like a good programming problem."

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 1 component:

Start line - A single line, "*A B*", where:
A : $(1 \leq A \leq 200)$ is an integer amount of dollars that you are caching out.
B : $(1 \leq B \leq A)$ is an integer amount of dollars that your ticket will cost.

Output Description

For each data set, there will be exactly one line of output. If it is possible to have a set of bills that add up to the first dollar amount such that no subset of those bills add up to the second amount, the output will be a single line with the statement "I MIGHT NEED CHANGE". Otherwise, the output will be a single line with the statement "I'VE GOT CHANGE". The possible denominations (values for a single bill) for this problem are $1, $5, $10, and $20.

Sample Input

```
45 40
40 10
```

Sample Output

```
I'VE GOT CHANGE
I MIGHT NEED CHANGE
```

Problem 3

Changing Tense

6 Points

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

"We have entertained audiences for many years. We are entertaining you tonight. We will entertain audiences for many more years, " Seigfreid stated, standing stoically on the stage.
"We entertained an audience last night. We will have entertained you at the end of the show. Now we entertain, " Roy continued.
"These guys sure do change tense a lot, " George leaned over in his front-row seat and whispered to Jorge.
"Yeah, " Jorge whispered back, "I wish there were a way to keep track of all these verb tenses."
"Hmmm, " George shrugged, "sounds like a good programming problem."

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 1 component:

Paragraph - A single line (up to 200 characters) containing three sentences. Sentences are defined as a list of words (words are consecutive groups of letters delimited by a single space). Sentences are delimited by a single period. Each sentence will contain exactly one verb phrase, where the verb phrase is one of the following:

"will have <word ending in the letters 'ed'>" (Example: “will have entertained”)
"have <word ending in the letters 'ed'>" not immediately preceded with the word "will" (Example: “have entertained”)
"<word ending in the letters 'ed'>" not immediately preceded with the word "have" or the words "will have" (Example: “entertained”)
"<word> <word ending in the letters 'ing'>" (Example: “are entertaining”)
"will <word other than 'have'>" (Example: “will entertain”)

Notes:

- If none of the phrases above are found in the sentence, the verb phrase will be the last word in the sentence.
- Consider the verb phrase rules listed above to be **case-insensitive** (i.e., "Have entertained" and "have entertained" are both verb phrases).

Output Description

For each data set, there will be exactly one line of output. The line will be a list containing the verb phrase for each of the three sentences, delimited by a single comma. When listing the verb phrases, use the case as was given in the sentence.

Sample Input

```
We have entertained audiences for many years.We are entertaining you
tonight.We will entertain audiences for many more years.
We entertained an audience last night.We will have entertained you at the end
of the show.Now we entertain.
I have a car and I once loved to drive it.Ed has a car.He does not like to
drive it.
Sometimes guessing where verbs are can lead to problems.We need a better
algorithm.But you must have the will to try this one first.
```

Sample Output

```
have entertained,are entertaining,will entertain
entertained,will have entertained,entertain
loved,Ed,it
Sometimes guessing,need,will to
```

Problem 4

Virtual Computing

6 Points

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

Write a virtual computer that runs programs written in a simple programming language.

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has up to 1000 lines, each of the format "<*Line Number*> <*Instruction*>" where:

Line Number : (1 ≤ Line Number ≤ 1000) is an integer indicating the line number for the following *Instruction*. Within a given data set, *Line Number*s are unique, and *Line Number*s always increase from one line to the next.

Instruction : One of the following:

"LOAD *X value*" – Set variable *X*'s value to *value*

"ADD *X value*" – Increase variable *X*'s value by *value*

"PRINT *X*" – Display the value of variable *X* to standard output followed by a newline.

"IF *X* == *value* GOTO *line*" – If the value of variable *X* equals *value* continue execution at the instruction with *Line Number* equal to *line*, otherwise execution moves to the next instruction as usual.

"END" – The last instruction in every data set (which will only appear once).

Note:

- *X* is a non-empty character string of up to 10 characters representing a variable name
- *value* is an integer (0 ≤ *value* ≤ 1000)
- *line* is a valid *Line Number* from the current data set
- Assume that every variable has an initial value of 0 that is reset at the beginning of each data set.

Output Description

For each data set, there will be at least one line of output. The first line of output for each data set will read, "START *N*" where *N* is an integer identifying which data set is being processed. *N* will be 1 for the first data set and increment by one for each additional data set. Also output will be the results from all of the PRINT *X* statements executed by the virtual computer.

Sample Input

```
1 LOAD var 1
10 PRINT var
11 END
5 END
19 PRINT noinit
20 LOAD noinit 10
21 PRINT noinit
30 LOAD x 5
35 LOAD y 1
40 ADD x 5
45 ADD y 1
50 IF y == 5 GOTO 70
60 IF z == 0 GOTO 40
70 PRINT x
80 END
```

Sample Output

```
START 1
1
START 2
START 3
0
10
25
```

Problem 5

Pretty Poor Encryption

6 Points

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

You are the top software engineer for a software company named Delusional Software that is releasing a new encryption package that will allow messages, such as email and chat messages, to be encrypted for transmission and then decrypted by the receiver. Your job is to develop the decrypting side of the package. You do have concerns however about the encryption algorithm being used in this package since all it involves is converting each character in the message to a binary representation of that character's ASCII value. The designers believe that this encryption algorithm will be uncrackable (you're starting to realize how this company got its name), and you are more than happy to develop it and see them proven wrong!

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 2 components:

Start line – A single line, "MESSAGE *X*", where *X* is the number of bytes that are to be decrypted, and ($1 \leq X \leq 80$).

Message line – This line will consist of *X* binary byte strings. Every line will consist of between 1 and 8 bytes of data, where each byte (series of eight 0's and 1's) represents the binary equivalent of the ASCII value of each ASCII-printable character found in the message (including spaces, but not newlines or tabs). Note that the binary byte strings on each line will not be delimited by any spaces.

Output Description

For each data set, there will be two lines of output. The first line will be a replication of the Start Line from the input file, and the following line will consist of the actual text message that was decrypted from the series of binary byte strings from the input.

Sample Input

```
MESSAGE 29
01010100011010000110010100100000010101010100100101001100001000000
01001010011101010110010001100111011001010111001100100000011000001
01110010011001010010000001100011011011110111011110110110000100000
01100111011101010111100101110011001000001
MESSAGE 42
01000101011100110111000001100101011000110111010010110000101101100
01101100011110010010000001110100011010000110010100100000011000001
01110101011101000110100001101111011100100010000001101111011001100
00100000011101000110100001101001011100110010000001110000011100010
01101111011000100110110001100101011011010010000100100000000100000
0011101100101001
```

Sample Output

```
MESSAGE 29
The UIL Judges are cool guys!
MESSAGE 42
Especially the author of this problem!  ;)
```

Problem 6

The Fraction Factor

6 Points

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

Reduce a given fraction to its simplest form. In this form, the numerator and the denominator are both integers and are relatively prime.

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set consists of a single line, "*N D*", where:

N : ($1 \le N \le 10000$) is an integer numerator.
D : ($1 \le D \le 10000$) is an integer denominator.

Output Description

For each data set, there will be exactly one line of output. The output line will express the reduced version of the input fraction using precisely the same formatting as the input, "*n d*".

Sample Input

```
100 1
1 100
100 100
75 20
```

Sample Output

```
100 1
1 100
1 1
15 4
```

Problem 7

Par for the Course?

6 Points

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

What would happen if a pro golfer enlisted the help of a talented programmer to improve his game? Probably not much, but we're going to fake it anyway. For this problem, the goal is to determine the minimum number of strokes needed to get the golf ball from its starting position to the hole. [FYI, a stroke equates to a single hit of the ball.]

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 2 components:

Size line - A single line, "*X Y Z*", where:

X : ($1 \le X \le 20$) is an integer number of columns in the *Hole Map*.
Y : ($1 \le Y \le 20$) is an integer number of rows in the *Hole Map*.
Z : ($3 \le Z \le 5$) is an integer number representing *Par* for this *Hole Map*.

Hole Map - A series of *Y* lines, each of length *X*. Each character in the *Hole Map* will represent one of the following:

'*' (asterisk) – represents the starting position of the golf ball. There will be exactly one golf ball.
'O' (capital letter) – represents the hole. There will be exactly one hole.
'T' – represents trees.
'.' (period) – represents fairway.

Note the following important facts:

- The golf ball may only be hit in one of the four cardinal directions; there can be no diagonal hits.
- The golfer only has four clubs, each ALWAYS hits a precise distance (a 'unit' is one row or column):
  - Putter – hits 1 unit
  - Wedge – hits 3 units
  - Iron – hits 5 units
  - Driver – hits 10 units
- When a ball is hit, it travels over any trees.
- A ball may never be hit outside the *Hole Map* or onto a tree.
- The number of strokes needed to finish a hole will never be less than (*Par*-2)
- The number of strokes needed to finish a hole will never be more than (*Par*+2)
- All holes will be completable.

Output Description

For each data set, there will be exactly one line of output indicating the best (lowest) possible score for the hole using the English text representation. The relationship between the number of strokes and *Par* gives the numerical score for the hole, which relates to the English text version of the score according to the following table:

Numerical Score (# Strokes minus Par)	**English Score**
2	"Double Bogey"
1	"Bogey"
0	"Par"
-1	"Birdie"
-2	"Eagle"

Do not print the quotes!

Sample Input

```
3 3 3
TTT
TOT
T*T
5 5 3
TTTTT
TTT.O
TTTTT
TTTTT
*TT.T
11 11 4
*TTTTTTTTT
TTTTTOTTTTT
TTTTT.TTTTT
TTTTT.TTTTT
TTTTT.TTTTT
TTTTT.TTTTT
TTTTTTTTTTT
TTTTTTTTTTT
TTTTTTTTTTT
TTTTTTTTTTT
...........
```

Sample Output

```
Eagle
Par
Bogey
```

Problem 8

Kennel Kritters

6 Points

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

Grok caveman. Grok have kennel. Grok have kritters. Grok notice some kritters eat other kritters. Grok notice some kritters fight other kritters. Kritters eat other kritters, kritters fight other kritters, bad for business. Help Grok put kritters in cages, so no kritter eat other kritter, no kritter fight other kritter. Grok draw picture of kennel:

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

Each square cage. Nine squares, nine cages. Grok good artist, no?

Grok smart. Grok learn:

1. Cage either have no kritter or one kritter. Grok want kritters uncrowded, happy.
2. Dog in cage next to 2 or more dogs in cages, dogs fight. Fish in cage next to 3 or more fish in cages, fish fight.
3. Dog in cage next to cat in cage, dog eat cat. Cat in cage next to mouse in cage, cat eat mouse. Cat in cage next to bird in cage, cat eat bird. Cat in cage next to fish in cage, cat eat fish. Bird in cage next to fish in cage, bird eat fish.
4. Cage "next to" cage if it shares side with cage (cage 1 next to cages 2 and 4, cage 2 next to cages 1, 3, and 5, ...cage 5 next to cages 2, 4, 6, and 8, etc.).

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 1 component:

Cage Configuration - Three lines representing a configuration of animals placed in cages. The arrangement of the cages will be as indicated in Grok's drawing above. Each cage will be represented by a single letter, where:

"B" indicates a bird is in this cage,
"C" indicates a cat is in this cage,
"D" indicates a dog is in this cage,
"F" indicates a fish is in this cage,
"M" indicates a mouse is in this cage,
"N" indicates there is no animal in this cage

Note that there will be no spaces separating the letters in the input data sets.

Output Description

For each data set, there will be exactly one line of output. If the input configuration is one in which no kritter will eat another kritter and no kritter will fight with another kritter, the output will be a single line with the statement "GROK HAPPY". Otherwise, the output will be a single line with the statement "GROK SAD".

Sample Input

```
BDD
BMF
DFF
CMM
NMM
MMM
```

Sample Output

```
GROK HAPPY
GROK SAD
```

Problem 9

Just Picture It

6 Points

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

The Graphics Image Format (GIF) was developed by CompuServe back in the dark ages as a way to compress images for transfer between customers. The compression was achieved by the combination of an 8-bit Color Look Up Table (CLUT) and Run Length Encoding (RLE).

RLE is a relatively simple and loss-less compression scheme, in contrast to JPEG compression which is lossy and much, much more complicated. The most basic form of RLE entails replacing a run of same value pixels with a single (pixel, run length) pair. So, if there were 100 black pixels, instead of encoding 100 individual black pixel values in a row, the compressed version would have a single black pixel value with a run length of 100.

In this problem, you will be presented with an image compressed using this basic form of RLE, and your program will have to output the uncompressed image.

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 2 components:

Start line - A single line, "START X Y", where X ($1 \leq X \leq 20$) is the number of columns in the uncompressed picture, and Y ($1 \leq Y \leq 20$) is the number of rows in the uncompressed picture.

Compressed Picture – There will be a series of lines "V R", where V ($0 \leq V \leq 9$) is the next pixel value, and R is the run length for that pixel. There will be precisely enough lines in each *Compressed Picture* to represent $X \times Y$ pixels (the entire picture).

Output Description

For each data set there will be an $X \times Y$ rectangular output showing the uncompressed picture (no spaces). There will be **no blank lines** between output sets.

Sample Input

```
START 6 4
0 5
4 1
0 4
4 1
0 4
4 1
0 4
4 1
0 3
START 8 8
0 8
1 4
2 4
3 4
4 8
5 8
6 4
7 4
8 4
7 4
8 4
7 4
8 4
```

Sample Output

```
000004
000040
000400
004000
00000000
11112222
33334444
44445555
55556666
77778888
77778888
77778888
```

Problem 10

You May Already Be A Winner!

6 Points

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

Your high school is in desperate need of new computer equipment, and the Computer Club is hosting a raffle to raise money for this worthy cause. In an effort to maximize participation in the raffle, the Computer Club has come up with a clever way to give out the prizes. There are a set number of prizes to be given away. When a person buys a raffle ticket, they will submit a prioritized list of the items they want to win, starting with the prize they most want and ending with the prize they want the least. After the raffle is over, names will be drawn from a hat. The first person drawn will win the first prize on his/her list. The next person drawn will win the first prize on his/her list that has not already been won. This will continue until all of the prizes are handed out or all contestants have been drawn. This gives people a greater probability of winning something they want, thus increasing the number of raffle tickets sold. Your job as president of the Computer Club is to write a program that will determine, given the order in which the names are drawn, who wins each prize.

Input Description

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 5 components:

Start line - A single line, "START *C P*", where *C* is the number of contestants ($1 \leq C \leq 10$) and *P* is the number of prizes in the raffle ($1 \leq P \leq 10$).

Prize List – The next *P* lines will consist of the prizes available to be won by the contestants.

Contestant Info – There will be *C* of these, each Contestant Info entry will consist of the contestant's name, followed by that contestant's prize priority list (each prize on a separate line). A contestant's prize priority list will list each prize from the *Prize List* exactly once.

Drawing Order – The next *C* lines will consist of all of the contestants' names in the order in which they were drawn.

End line - A single line, "END"

Note:

- Contestant and prize names may consist of multiple tokens (e.g. 'John Doe' and 'Sports Car').

Output Description

Each dataset's output will start with the following string:

```
Raffle #<NUMBER>:
```

Where <NUMBER> begins as 1 for the first data set and increments by 1 for each data set thereafter. The remaining output for each dataset will consist of the following string for each person who has won a prize from the prize list:

```
<CONTESTANT> Wins <PRIZE>!!
```

Note:

- The order in which the names were drawn is the order in which the winners should be listed.

Sample Input

```
START 3 5
Video Game
CD
Mouse Pad
Pocket Protector
Backpack
Jimmy
Backpack
Video Game
CD
Mouse Pad
Pocket Protector
Betty
CD
Video Game
Pocket Protector
Backpack
Mouse Pad
James
Video Game
CD
Mouse Pad
Pocket Protector
Backpack
James
Betty
Jimmy
END
START 1 1
Prize 1
Contestant 1
Prize 1
Contestant 1
END
```

Sample Output

```
Raffle #1:
James Wins Video Game!!
Betty Wins CD!!
Jimmy Wins Backpack!!
Raffle #2:
Contestant 1 Wins Prize 1!!
```