University Interscholastic League

Computer Science Competition

1999 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 8.

2. Many of the example input data sets and example outputs contain comments set off by "/*" and "*/". These are just to help you understand the examples. The comments are not included in the real input or output.

3. Problems have different point values, depending of their difficulty.

II. Point Values and Names of Problems

Number	Name	Point Value
Problem 1	Stone Wheels	3
Problem 2	Ant Trails	4
Problem 3	Blowing the Curve	5
Problem 4	Spell Checker	6
Problem 5	Binary Transmissions	4
Problem 6	Big Time Math	8
Problem 7	Fidel's Bingo	9
Problem 8	Drainage	10
Total		**49**

Problem 1	**Stone Wheels**	2 Points

Program Name: PROG1.CPP **Input Data File: DATA1.IN**

The year is about 27000 B.C. and your tribe has just discovered that a round rock can be mounted to a stick to make an axle and placed on a flat piece of wood to make a crude wagon. A particularly wise person in your group (the world's first mechanical engineer) has figured out that a small wheel means that the axle will wear out faster to travel a given distance. He has also determined that a large wheel will wear out the poor guys who have to pull the wagon because stone wheels are very heavy.

In order to help him with his evaluations, he wants you to determine the number of times that a wheel will turn to travel a particular distance when given a radius to that wheel. You have determined that the circumference around a wheel is equal to its twice its radius times a value that you have called cake. You have computed cake to be 3.14. (Later in history, some Greek guys will refine your number for cake, call it Pi, and take credit for it!)

Your mechanical engineer friend has given you a list of distance/radius pairs. For each pair, he wants you to tell him how many times a wheel with the given radius will have to turn to travel the given distance to an accuracy of 1 decimal place. Fortunately, you are very inventive and have invented the computer (they won't catch on for almost 29000 years) and can write a program to compute the distances automatically.

Input
Your program is to read in one or more data sets from the file DATA1.IN with each data set using one line of input. The data sets are terminated by the end-of-file. There will be 2 real (float) numbers per line of the file. The first number (0.1 <= radius <= 10.0) is the radius of the wheel under consideration. The second number (0.1 <= distance <= 1000.0) is the distance to travel under consideration.

Output
For each data set that is input, print the number of rotations required for a wheel of the given radius to travel the given distance. For each input set, your program should print the following message to the screen.

```
Radius: x.x, Distance: x.x, Rotations: x.x
```

In this message, your program is to print the radius, distance, and rotations to an accuracy of 1 decimal place. Your program should use 1 leading zero only for values < 1.0.

Example: Input File
```
0.7 10.6
5.3 426.6
4.3 426.6
7.5 1000.0
10.0 50.0
1.2 753.6
```
Output to screen
```
Radius: 0.7, Distance: 10.6, Rotations: 2.4
Radius: 5.3, Distance: 426.6, Rotations: 12.8
Radius: 4.3, Distance: 426,6, Rotations: 15.8
Radius: 7.5, Distance: 1000.0, Rotations: 21.2
Radius: 10.0, Distance: 50.0, Rotations: 0.8
Radius: 1.2, Distance: 753.6, Rotations: 100.0
```

Problem 2	Ant Trails	3 Points

Program Name: PROG2.CPP **Input Data File: DATA2.IN**

A queen ant is interested in determining just how far her workers will have to travel to collect some food. Unfortunately, her workers are unable to express a distance to her since they get mixed up trying to count the steps (having six feet and all). They can help her by giving a series of points which represent a very close approximation to the path traveled. The queen needs you to write her a program which tells her the actual distance to a food source and the approximate distance the workers will have to travel using the trail defined by the series of points.

Your program will read the series of points as coordinate pairs (-100.00<=X,Y<=100.00) which represents a trail, add up the distances of the trail/line segments, and print a message telling the queen the distance from the ant hill opening (at the origin of the graph) to the food and the actual distance the workers are traveling to the food. See the picture below for an illustration.

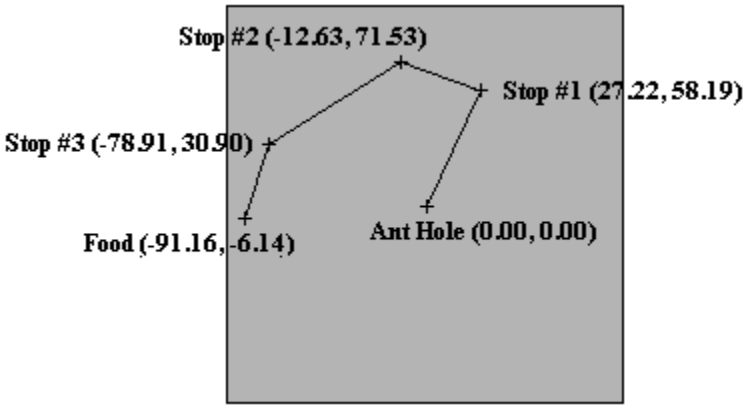

Figure 1: Ant Trail Example

For the above figure, your program will compute the actual distance to the food to be 91.37 and the distance traveled by the workers to be 223.02. (The lengths of the line segments along the path are approximately 64.24, 42.02, 77.74, and 39.01 traveling from the hole to the food). Your program will allow rounding as per the abilities of your computer and will print to the precision of 2 decimal digits. In this case, you will notice that the sum of the line segments actually sums to 223.02, but your program should not round (beyond the limits of your compiler and hardware) until it is time to print results. Do not worry, the judges will allow a little leeway to allow for rounding errors by your hardware and compiler. (Remember that .10 is an irrational number when expressed as binary form bbb.bbb…)

Input
Your program is to read in one or more data sets from the file DATA2.IN with each data set using one line of input. The data sets are terminated by the end-of-file. Each line of input will first contain an integer (1<=N<=5) followed by N coordinate pairs (-100.00<=X,Y<=100.00) on the same line. Coordinate values will be separated by exactly 1 blank and there will be no trailing blanks on the line.

Output
For each data set that is input, print the total distance being traveled by the workers along the path of line segments along with the distance from the whole to the food (at the last point). Output to the screen should use the following message.

```
Source #F, the travel distance is ddd.dd and the actual distance is aaa.aa
```

In this message, your program should number food sources starting with 1 and the travel and actual distances should be printed to an accuracy of 2 decimal digits.

Example: Input File

```
1 58.94 19.26
4 27.22 58.19 –12.63 71.53 –78.91 30.90 –91.16 –6.14
2 27.01 17.85 –35.50 –20.16
```

Output to screen

```
Source #1, the travel distance is 62.01 and the actual distance is 62.01
Source #2, the travel distance is 223.02 and the actual distance is 91.37
Source #3, the travel distance is 105.53 and the actual distance is 40.82
```

Problem 3	**Blowing the Curve**	5 Points

Program Name: PROG3.CPP **Input Data File: DATA3.IN**

Maybe you have not noticed, but sometimes a teacher will have one or two students who will score so high on an exam as to "blow the curve". In this context, "the curve" is an adjustment that the teacher makes to the exam scores based on the difficulty of the exam and the resulting raw scores on the exam. Among the ways to curve the scores are the "no curve", "straight add", and "squares times 10".

No Curve
The "no curve" method is the simplest. In this method, each student receives the Raw Score as his Final Score.

Straight Add
In the "straight add" method, each student's final score is computed by adding the difference of 100 and the highest score. For example, in the following sorted list of grades, you will see that the highest score is 93. With a straight add of 7 points (100-93) to each score, you then see the final grades.

Raw Score	93	91	87	82	81	79	76	73	73	71	70	69	65	64	62	58	47	46	31	19
Curved Score	100	98	94	89	88	86	83	80	80	78	77	76	72	71	69	65	54	53	38	26

Squares Times 10
The "squares times 10" method gives recognition that just a little more knowledge gains you more points on an exam if your score was lower than if your score was near the top. In this method, each student's final score is computed by multiplying the square root of the raw score by 10 with the final score being rounded to the nearest integer. This method also allows a curve to occur even if one or more raw scores of 100 are scored on the exam because the student's final score would still be 100.

Raw Score	93	91	87	82	81	79	76	73	73	71	70	69	65	64	62	58	47	46	31	19
Curved Score	96	95	93	91	90	89	87	85	85	84	84	83	81	80	79	76	69	68	56	44

The Problem
Your teacher wants you to write a program which will determine which of the curve methods she should used based on the least squares value (LSV) of each method. For each test set (every set is exactly 20 scores), the teacher has a target of 4 A's, 6 B's, 6 C's, 3 D's, and 1 F. A grade of A is given for final scores of 90 and higher; a B is given for final scores of 80 to 89; a C is given for final scores of 70 to 79; a D is given for final scores of 60 to 69; and an F is given for final scores below 60. The LSV value is computed as follows.

LSV = (# A's – 4) ^ 2 + (# B's – 6) ^ 2 + (# C's – 6) ^ 2 + (# D's – 3) ^ 2 + (# F's – 1) ^ 2
Where ^ 2 indicates arithmetic squares (multiplying the number by itself).

So, LSV's for the three methods for the raw scores given above would be computed as:

LSV (no curve) = (2-4)^2 + (3-6)^2 + (6-6)^2 + (4-3)^2 + (5-1)^2 = 4 + 9 + 0 + 1 + 16 = 30

LSV (straight add) = (3-4)^2 + (6-6)^2 + (5-6)^2 + (2-3)^2 + (4-1)^2 = 1 + 0 + 1 + 1 + 9 = 12

LSV (squares times 10) = (5-4)^2 + (9-6)^2 + (2-6)^2 + (2-3)^2 + (2-1)^2 = 1 + 9 + 16 + 1 + 1 = 28

Given the above computed LSV's, your program should choose the "straight add" method because it is the least of the 3 values. In case of a tie, your program should give precedence to the "squares times 10" method followed by the "straight add" method and then "no curve" with the lowest precedence.

Input
Your program is to read in one or more data sets from the file DATA3.IN with each data set using one line of input. The data sets are terminated by the end-of-file. Each line of input will contain exactly 20 integer scores between 0 and 100 inclusive separated from one another by exactly 1 space.

Output
For each data set that is input, print one of the following three messages.

```
For Exam #E, use no curve
For Exam #E, use straight add
For Exam #E, use squares times 10
```

In these messages, your program should number exams starting with 1.

Example: Input file
```
87 73 58 91 70 64 93 46 69 19 31 47 62 82 71 65 76 82 73 79
81 72 82 93 83 73 63 74 94 66 96 96 87 77 67 58 88 79 79 80
92 92 88 86 77 76 69 67 66 64 62 62 59 53 55 52 45 44 40 34
```

Output to screen
```
For Exam #1, use straight add
For Exam #2, use no curve
For Exam #3, use squares times 10
```

Problem 4	Spell Checker	6 Points

Program Name: PROG4.CPP **Input Data File: DATA4.IN**

The year is 1980, and the new IBM PC has just been invented. An upstart software company is writing a word processor and they need you to write a crude spell checker. Your spell checker shall check the spelling of words in a document against those in a dictionary and suggest a word from the dictionary if the word is not found exactly in the dictionary as it appears in the document.

Your program will determine a "similarity rating" for each word in the dictionary versus a candidate word (from the document). Similarity ratings are computed by counting the number of letters of the alphabet that appear at least once in both words and subtracting the number of letters of the alphabet that appear at least once in only one of the words. Multiple appearances of a letter appearing in a word does not change the similarity rating of the word and ratings are to be entirely case insensitive. (Moreover, all input to your program will be in lower case.) For example, given the candidate word "alfalfa" from a document and dictionary entries of "alfie" and "falala", your program would compute a similarity rating of 1 for "alfie" and 3 for "falala".

Your program will read in a dictionary and then read a document and verify the spelling for words in the document one at a time.

Input
Input to your program will consist of a single file (DATA4.IN) which will contain a dictionary followed by a document. The first line of the input file will be a single integer (1<N<=200) which indicates the number of words in the dictionary. Lines 2 through N+1 each contain exactly one dictionary word from 1 to 20 lower case alphabetic characters in length. All subsequent lines each contain exactly one word from the document with between 1 and 20 alphabetic characters in length with the end-of-file of DATA4.IN indicating the end of the document to be spell-checked. There are no blanks embedded anywhere in the input file.

Output
For each word from the document, your program shall check the word to see if it appears exactly in the dictionary (case insensitive). Each time your program finds a word in the document that does not appear in the dictionary, it shall find the word in the dictionary which most closely matches the document word based on the similarity rating. Any ties in the similarity rating between two or more dictionary words shall be resolved by using the word that appears first in the dictionary. For each word which does not appear in the dictionary, your program shall print the word followed by "->" (dash greater than) and the best similarity rating word from the dictionary (exactly as it appears in the dictionary). If the best similarity rating is less than zero, your program shall print "NO SUGGESTION" instead of the dictionary word. Misspelled words and the suggestions shall be printed only one to a line. See the Sample Output for further formatting help.

Example: Input File

```
14
buckwheat
pancakes
is
a
word
but
not
waffles
another
dictionary
word
shovel
seed
dues
buckwheat
loves
darla
but
porkwheat
does
not
```

Output to screen

```
loves->shovel
darla->NO SUGGESTION
porkwheat->another
does->seed
```

Problem 5

Binary Transmissions

7 Points

Program Name: PROG5.CPP **Input Data File: DATA5.IN**

Serial transmissions are really just a sequence of bits (0's and 1's) which form integers. When transmitting a byte (typically 8 bits or binary digits), the most significant bit (MSB) is transmitted first and the least significant bit (LSB) is transmitted last. These received bits are received into a queue, and when all 8 bits are received, they can be "viewed" as decimal values among other things. For example, the bit string '10010110' is equal to 150 in decimal. The table below should remind you of the values for bits (binary digits) for each position in an 8-bit byte.

	MSB							**LSB**	
	2^7 = 128	**2^6 = 64**	**2^5 = 32**	**2^4 = 16**	**2^3 = 8**	**2^2 = 4**	**2^1 = 2**	**2^0 = 1**	**Total**
Bit String	1	0	0	1	0	1	1	0	
Decimal Values	+(128*1)	+(64*0)	+(32*0)	+(16*1)	+(8*0)	+(4*1)	+(2*1)	+(1*0)	= 150

Your program is to read in a series of 8-bit bytes (expressed to you 1 bit at a time). In any input line, the leftmost bit on a line is the most significant (a value of 128 or 0 in decimal) and the rightmost bit is the least significant (a value of 1 or 0 in decimal). With a little work, you can prove to yourself that the range of decimal values for an 8-bit byte is 0 to 255. For each bit string, your program is to compute and print the decimal equivalent of the byte.

Input
Your program is to read in one or more data sets from the file DATA5.IN with each data set using one line of input. The data sets are terminated by the end-of-file. Each line of input will contain exactly 8 bits each of which can only be a 0 or a 1.

Output
For each data set that is input, print the bit string exactly as it appears on the input line followed by a " = " (space-equals-space) and then the decimal value of the bit string. The decimal value should be left justified and should not have leading zeroes.

Example: Input File
10010110
00001000
10000001
10100101
00011000
Output to screen
10010110 = 150
00001000 = 8
10000001 = 129
10100101 = 165
00011000 = 24

Problem 6

Big Time Math

8 Points

Program Name: PROG6.CPP **Input Data File: DATA6.IN**

Maybe you have noticed that computers can only add and subtract numbers up to 2^32 (2 raised to the 32nd power) without some sort of firmware or software intervention. This is because the registers and the ALU (arithmetic logic unit) are typically only 32 bits in size. Unfortunately, your company needs to add and subtract integer numbers up to 60 digits in length (plus a single character for the sign). They have turned to you to write a sample program demonstrating how this can be done.

Your program is to read in two signed integers each up to 60 digits in length and add them. (Remember that adding a negative number is the same as subtracting). The integers to be added do not contain leading zeroes. Your program then must print out the result of the addition.

Input
Your program is to read in one or more data sets from the file DATA6.IN with each data set using two lines of input. The data sets are terminated by the end-of-file. Each data set is an integer pair. An integer pair consists of two long integers which your program will add together. Each integer in a pair consists of a sign character ('-' for negative or '+' for positive) followed by 1 to 60 decimal digits representing the integer. Each integer is on a line by itself and there are no embedded blanks or empty lines anywhere in the input file. Integers are paired together with integers on odd-numbered lines paired to (or added with) the integers on the even-numbered line below it. Lines in the input file are numbered starting with 1.

Output
For each input integer pair, you are to print four lines of output. The first two lines of output should be an exact echo of the two lines of input. The third line of output is the result of the addition with the first column being the sign of the result ('-' for negative or '+' for positive) followed by 1 to 61 digits. The fourth line is empty. There must be nothing in your output except digits, sign characters and end-of-line ('\n' for you 'C' programmers) characters. See the sample output for illustration of the required format.

Example: Input File

```
+1234
+5678
-55555555555555555555555555555555555555555555555555555555555555
-55555555555555555555555555555555555555555555555555555555555555
+99999999999999999999999999999999999999999999999999999999999999
+2
+4
-0
```

Output to screen

```
+1234
+5678
+6912

-55555555555555555555555555555555555555555555555555555555555555
-55555555555555555555555555555555555555555555555555555555555555
-11111111111111111111111111111111111111111111111111111111111110

+99999999999999999999999999999999999999999999999999999999999999
+2
+10000000000000000000000000000000000000000000000000000000000001

+4
-0
+4
```

Problem 7

Bingo Anyone?

8 Points

Program Name: PROG7.CPP **Input Data File: DATA7.IN**

Having a passion for bingo, you have found that it is difficult to keep track of multiple cards at once. So, you have decided to write a program that will help score your bingo game. Bingo cards are arranged as 5x5 segments where each segment contains exactly 1 number. Segments are noted by a (row, column) combination each numbered 1 to 5. Each contains a number. Segments in column 1 contain numbers between 1 and 15; column 2 contains numbers between 16 and 30; column 3 contains numbers between 31 and 45; column 4 contains numbers between 46 and 60; and column 5 contains numbers between 61 and 75. The only exception is segment (3, 3) which is considered a "free" segment and is always considered "occupied". Each card is unique in that numbers are assigned to segments at random.

Bingo is played by calling out a series of numbers between 1 and 75 (with no repeats). With each called number, players mark "occupied" any segments containing the called number. A player calls "Bingo" declaring himself the winner when all segments of any one row, any one column, or either of the diagonals are all occupied. The following three cards show an example of three winning cards for each of the three possible ways to win. Note that the diagonal and the "row 3" win use the "free" segment at location (3, 3).

5	21	44	51	61		11	23	45	60	**75**		1	**19**	34	55	70
12	17	41	53	69		7	16	38	**59**	71		8	**22**	43	52	72
3	**29**	**X**	**47**	**73**		3	30	**x**	57	74		6	**24**	X	48	66
2	16	38	60	66		12	**22**	33	58	69		9	**17**	31	56	74
10	18	42	49	63		**10**	17	42	45	70		4	**28**	37	60	67

Further, after "calling" the following numbers, the cards from above would look as follows with the left card winning with a row 3 bingo.

Called numbers: 47 32 61 73 22 19 71 26 64 42 45 32 62 65 01 03 59 57 29

5	21	44	51	**61**		11	23	**45**	60	75		**1**	**19**	34	55	70
12	17	41	53	69		7	16	38	**59**	**71**		8	**22**	43	52	72
3	**29**	**X**	**47**	**73**		3	30	**x**	**57**	74		6	24	**X**	48	66
2	16	38	60	66		12	**22**	33	58	69		9	17	31	56	74
10	18	**42**	49	63		10	17	**42**	45	70		4	28	37	60	67

Input

Your program is to read in one or more data sets from the file DATA7.IN with each data set using one line of input. The data sets are terminated by the end-of-file. Each line of input will contain exactly 25 numbers between 1 and 75 (with no repeats). For each input set, your program is to read the 25 "called" numbers and determine which, if any, of the three cards (from above) will achieve a bingo within those 25 numbers.

Output

For each input set, your program is to determine which card achieves a bingo using the 25 called numbers in order. Ties are broken by determining which card achieves bingo first. If a tie still exists, ties are broken by choosing the cards from left to right in highest to lowest preference. If no bingo is achieved within the 25 numbers, your program will have to recognize and declare that condition. Your program will print one of the four messages below with the game number starting from 1.

```
Game #x: Left Bingo
Game #x: Middle Bingo
Game #x: Right Bingo
Game #x: No Bingo
```

Example: Input File

```
47 32 61 73 22 19 71 26 64 42 45 32 62 65 01 03 59 57 29 14 41 21 74 02 55
05 19 55 38 10 13 47 01 22 38 51 56 14 67 30 18 20 71 06 11 75 20 02 58 29
57 21 19 44 20 32 61 35 55 36 13 29 14 37 60 39 11 75 15 40 58 26 25 31 27
```

Output to screen

```
Game #1: Left Bingo
Game #2: Right Bingo
Game #3: No Bingo
```

Problem 8	**Drainage**	10

Program Name: PROG8.CPP **Input Data File: DATA8.IN**

You have been looking at land to purchase but a lot of what you have seen tends to not drain very well. Although there are drainage ditches all the way around the land you have seen, water often puddles all over the land every time it rains. The land areas you are considering are so large that it is difficult to see from direct observation where the water will drain. So, you can take data from satellite images to help determine where the water will drain.

The satellite image you are given comes to you in the form of a 20x10 grid of single digit numeric characters ('1' – '9') representing the altitude of the land relative to the surrounding ditches. (The drainage ditches are at altitude 0.) The purpose of your program is to determine whether puddles will form on your potential land purchases.

Drainage rules…

1. Any water that drains off your input grid is said to drain into one of the ditches (remembering that the input grid is bounded on all sides by ditches which are at altitude 0.
2. If for any position, any of the 8 neighboring positions is lower in altitude, all of the water from the position will drain from the position.
3. Some percentage of water at a given grid position will drain to each of the up to 8 neighboring grid positions which are lower in altitude.
4. If for any position, any of the 8 neighboring positions will drain and are at the same or lower altitude, the position will drain.
5. If no neighbors of a position are lower and no neighbor positions equal in altitude will drain, water will form a puddle at the position. (Remember that a group of positions may also form a large puddle if all are at the same altitude and the group is completely bounded by positions of higher altitudes.

Your program does not have to be concerned with how much water ends up in one of the drainage ditches or in one or more puddles. Instead, your program only needs to determine if the water will all drain into a ditch or if any of it ends up in a puddle. For each water test, your program is to determine if the test results in "perfect drainage" or if it results in "puddled drainage".

The following example shows a sample which would result in "Puddled Drainage". Some (but not all) of the areas subject to forming puddles are shown in bold.

```
98876654444555566655
87654333333556644455
87765333566356666654
87765366655355662221
88577665433333332777
88876664467872667789
88777744556654211177
98876544455666611677
99876666666777777888
99988777766667778899
```

Input

Input to your program will consist of a series of land samples from the file DATA8.IN. Each land sample is given by 11 lines. The first line of a land sample is a string up to 40 alphanumeric characters in length which is the address of the land in consideration with no leading or trailing blanks. The next 10 lines of the land sample each contain exactly 20 numeric digits (between 1 and 9 inclusive) which represents the altitudes from the satellite image. Your program should continue reading land samples to the end of the input file.

Output

For each land sample, your program is to determine the type of drainage for the sample. Your program must print, on a line by itself, the address of the land sample followed either by

“ has perfect drainage.” if the land sample has no puddles, or “ has puddled drainage.” if the land sample has 1 or more puddles.

Your program must print these messages exactly as described here. See the sample output for further illustration of the expected output format from your program.

Example: Input File

```
Number 2 Jellystone Pond
98876654444555666655
87654333333556644455
87765333566356666654
877653366655355662221
88577665433333332777
88876664467872667789
88777744556654211177
98876544455666611677
99876666666777777888
99988777766667778899
1600 Pennsylvania Avenue
666666662126666666666
667666662126666666666
667666662126666666666
667666662126666666666
667666662126666666666
667666666666666666666
667655555555555555556
667644444444333333333
667665555555555555556
667666666666666666666
1992 Enchanted Forest Drive
888888888888888888888
881888888888888888888
888888888777777888888
888777777775555778888
888755555555557888888
888755555555557888888
888777777776667788888
888788788885588888888
888877778888558888888
888888888888888888888
```

Output to screen

```
Number 2 Jellystone Pond has puddled drainage.
1600 Pennsylvania Avenue has puddled drainage.
1992 Enchanted Forest Drive has perfect drainage.
```