Problem #1

Code Wars II

Danger Level: Green (Bunny Slope)

"Eschew Obfuscation"

Point Value: 2

Problem Statement

Some programmers absolutely refuse to write code that others can easily understand, or to write documentation for their programs. Sometimes these people are called *unemployed*. However, there is a safe outlet for those who wish to express themselves by writing really awful code. It's called the *International Obfuscated C Code Contest*, or IOCCC (see http://www.ioccc.org). The goals of the contest are:

- To write the most Obscure/Obfuscated C program under the contest rules.
- To show the importance of programming style, in an ironic way.
- To stress C compilers with unusual code.
- To illustrate some of the subtleties of the C language.
- To provide a safe forum for poor C code. :-)

Program Input

You should compile the following program. This was the 1990 winner of the *Best Small Program* award. When you run the program it will wait for a single integer input from the keyboard before execution. Try using numbers between 4 and 8.

```
int v,i,j,k,l,s,a[99]; main() {
 for(scanf("%d",&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<s&&(!k&&!!
   printf(2+"\n\n%c"-(!l<<!j)," #Q"[l^v?(l^j)&1:2])&&++l||a[i]<s&&v&&
   v-i+j&&v+i-j))&&!(l%=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i])
  ;
}
```

Program Output

The program displays a solution for the N-queens problem, where N is the integer taken from the input line. But the real "output" from this exercise is, as one of the contest goals states, to show the importance of programming style, in an ironic way.

Problem #2

Code Wars II

Danger Level: Green

"Spellbinder"

Point Value: 2

Problem Statement

Letterman's arch nemesis Spellbinder has escaped Semitree Seed Asylum and is now terrorizing the city with his devastating cruelty. Write a program to determine Letterman's arsenal to fight Spellbinder.

Program Input

Each line in the input file PROB02.IN contains two words. The first is the "original" word, and the second is the word as modified by Spellbinder:

```
fountain mountain
pet pen
check chuck
Mike bike
```

Program Output

The program should write the name of the required letter to PROB02.OUT in traditional campy Letterman style:

```
Ripping the lower-case "f" from his shirt, Letterman
changes "mountain" back to "fountain".
Ripping the lower-case "t" from his shirt, Letterman
changes "pen" back to "pet".
Ripping the lower-case "e" from his shirt, Letterman
changes "chuck" back to "check".
Ripping the capital "M" from his shirt, Letterman
changes "bike" back to "Mike".
```

Problem #3

Code Wars II

Danger Level: Green

"Palindrome"

Point Value: 2

Problem Statement

Determine if an input line is a palindrome. A palindrome is a word, verse, sentence, or number that reads the same backward or forward.

Program Input

A series of input lines from the file PROB03.IN:

```
1881
Madam, I'm Adam.
Lisa Bonet ate no basil.
Taste penne pasta.
```

Program Output

Echo the input line to the output file PROB03.OUT, then indicate whether it was a palindrome.

```
1881
   is a palindrome.
Madam, I'm Adam.
   is a palindrome.
Lisa Bonet ate no basil.
   is a palindrome.
Taste penne pasta.
   is *not* a palindrome.
```

Problem #4

Code Wars II

Danger Level: Green

"Nth Prime Number"

Point Value: 3

Problem Statement

Prime numbers are used in many program algorithms, including hash tables and encryption. Write a program to display the Nth prime number for a given value of N.

Program Input

The program should read a single integer from the input line (keyboard). This is the value of N to be used later. After displaying the output, the program should continue reading values for N until the user presses the '.' key.

```
1
4
10
.
```

Program Output

The program should display the value of N and the Nth prime number to the screen. The program may assume that N will be in the range 1 through 9999, inclusive. Here are the values for the above input:

```
The 1st prime number is 2.
The 4th prime number is 7.
The 10th prime number is 29.
Done.
```

Code Wars II

Danger Level: Green

"Random Number Generator Analysis"

Point Value: 3

Problem Statement

Verify the randomness of the random number generator.

For each 'n' from the input file, analyze the effectiveness of the random number generator by creating 20,000*(n+1) random numbers and printing out the count for each number. There should not be an obvious difference in randomness. Also verify that the sum of the counts is equal to 20,000*(n+1).

Program Input

From the input file '`PROB05.IN`', each line will contain a number 'n' to be used for the random number generator analysis. For each 'n', create random numbers from 0-n inclusive.

```
4
7
14
```

Program Output

Output the counts for all numbers that are selected by the random number generator to the output file '`PROB05.OUT`'. The sample output below represents n=4 only.

```
Random number range is 0-4 (n=4)
0 = 20031
1 = 19982
2 = 20028
3 = 19973
4 = 19986
Total = 100000
```

Problem # 6

Code Wars II

Danger Level: Green

"Success comes in cans, not can'ts"

Point Value: 3

Problem Statement

Determine the optimum radius and height of a can such that the minimum material is needed for a defined volume of product.

R = Radius in inches
H = Height in inches
$\pi = 3.14159$
Area of can: $A = 2\pi R * (R + H)$
Volume of can: $V = H\pi R^2$

Program Input

The input will be from file '`PROB06.IN`' listing the desired volumes (in cubic inches).

```
60.5
125.3
```

Program Output

Output the can Volumes, Areas, and optimum Heights and Radii to the output file '`PROB06.OUT`'.

```
Volume=60.5 in^3, Area=85.3 in^2, Height=4.2 in, Radius=2.1 in
Volume=125.3 in^3, Area=138.6 in^2, Height=5.4 in, Radius=2.7 in
```

Problem #7

Code Wars II

Danger Level: Blue

"Account Balancing"

Point Value: 5

Problem Statement

The First Fiscal Bank of Financial Fidelity needs a program to keep account balances up to date. The old program runs on a large, expensive mainframe, and you've been hired to rewrite it for Windows NT. In this problem you will write a simple prototype for a full account balancing package.

Program Input

The input file PROB07.IN will be a series of deposits and withdraws, each associated with a person's name. The program should create a new bank account when it encounters a name with the key word BALANCE. It should then add and subtract the given values as appropriate per the DEPOSIT and WITHDRAW transactions. Here's some sample input lines:

```
DIANE BALANCE 450.00
LARRY BALANCE 721.50
DIANE WITHDRAW 63.50
BEATRICE BALANCE 550.00
LARRY BALANCE 1344.71
STEVE WITHDRAW 19.62
LARRY WITHDRAW 246.19
DIANE DEPOSIT 925.22
LARRY WITHDRAW 515.00
BEATRICE DEPOSIT 750.00
STEVE BALANCE 200.00
DIANE WITHDRAW 37.50
BEATRICE DEPOSIT 200.00
STEVE WITHDRAW 19.54
LARRY WITHDRAW 5.15
XAVIER BALANCE 23400.00
DIANE DEPOSIT 925.23
```

Program Output

The program should print error conditions and final account balances to PROB07.OUT. It should print a line for each occurance of any of these three error conditions: (1) any DEPOSIT or WITHDRAW transaction for an account that does not exist, (2) assigning a BALANCE to an account already open, and (3) attempting to overdraw the account. The program should print the line number with the error message. After reading the entire file, the program should print all the account names, in alphabetical order, along with the current balances. All dollar values in the output file should be prefixed with a $. Dollar amounts should be delimited by commas every three significant digits, as shown in the sample output. Here's the example output from the above input data:

```
ERROR LINE 5: LARRY CANNOT REASSIGN ACCOUNT BALANCE
ERROR LINE 6: STEVE DOES NOT HAVE AN ACCOUNT
ERROR LINE 9: LARRY CANNOT WITHDRAW $515.00 - BALANCE IS $475.31
--- FINAL BALANCES ---
BEATRICE $1,500.00
DIANE $2,199.45
LARRY $470.16
STEVE $180.46
XAVIER $23,400.00
```

Problem # 8

Code Wars II

Danger Level: Blue

"Traveling Salesperson"

Point Value: 6

Problem Statement

Determine the shortest route to follow while making sales visits.

A salesperson must visit a list of clients and return back home, and travel the shortest total distance possible. Each client will be located on an x-y grid where home base is at 0,0. The salesperson is only allowed to travel on straight lines between any two points.

Program Input

The input will be from file '`PROB08.IN`' with a list of coordinates representing the clients that must be visited. Each line of input will contain the x-y coordinates of only one client, and the end of a defined list of clients will be the home base coordinates of 0,0. The first set of coordinates represents client #1, the second set of coordinates represents client #2, and so on.

```
123, 14
356, 228
39, 40
48, 223
266, 366
329, 10
0, 0
```

Program Output

Create a list of clients in an order that will allow the salesperson to travel the minimum total distance. Output these results to the output file '`PROB08.OUT`'.

```
Client 1 = 123, 14
Client 6 = 329, 10
Client 2 = 356, 228
Client 5 = 266, 366
Client 4 = 48, 223
Client 3 = 39, 40
```

Problem #9

Code Wars II

Danger Level: Blue Cruiser

"Reversi"

Point Value: 6

Problem Statement

Because the U.S. has imposed export restrictions on encryption software, some smaller nations have no choice but to use simplistic encoding for secret government communications. Write a decoder for reverse coded messages.

Program Input

The input file PROB09.IN will contain a series of words, and occasionally the key word REVERSI. Example:

```
jumped
REVERSI
over
the
REVERSI
fox
brown
quick
REVERSI
lazy
dog
REVERSI
the
REVERSI
```

Program Output

Store the words in order as they appear in the input file and write them to the screen. Each time the word REVERSI appears, reverse the order of the words, then add new words to the end of the new list. Here's the result from the above input file:

the quick brown fox jumped over the lazy dog

Problem #10

Code Wars II

Danger Level: Blue Cruiser

"Word Sorter"

Point Value: 6

Problem Statement

A classic word sort problem with a special bonus!

Program Input

The input file PROB10.IN will contain several lines of text. For example:

```
Santa, you just better watch your step!
How much is that doggie in the window?
The quick brown fox jumped over the lazy dog.
Have you read Knuth's programming series?
It just doesn't get any better than this!
```

Program Output

The program should sort the words on each line and print them to the output file, PROB10.OUT, like this:

```
better just Santa step watch you your
doggie How in is much that the window
brown dog fox jumped lazy over The the quick
Have Knuth's programming read series you
any better doesn't get It just than this
```

1 POINT BONUS: Count the number of occurrences of each word in the entire file and print the list at the end of the output file, sorted in alphabetical order. (Sample output is in columns, yours shouldn't).

```
any 1        get 1      just 2           read 1     this 1
better 2     Have 1     Knuth's 1        Santa 1    window 1
```

```
brown 1      How 1      lazy 1           series 1   watch 1
dog 1        in 1       much 1           step 1     you 2
doggie 1     is 1       over 1           than 1     your 1
doesn't 1    It 1       programming 1    that 1
fox 1        jumped 1   quick 1          the 3
```

Problem # 11

Code Wars II

Danger Level: Blue

"Mission Possible"

Point Value: 6

Problem Statement

Soldiers get in line for special assignments and the officers (you) must keep track of them.

In a selection room, there can be a maximum of 10 soldiers, while the others wait outside. As a soldier is selected for an assignment, they then leave the room for further instructions. For most basic assignments, the first one in the line leaves; however, for special assignments, any of the soldiers in the selection room may be chosen to leave. At some point during the selection process, the soldiers currently in the selection room will be given the most challenging assignment, compete in the annual Code Wars competition.

Program Input

A list of soldiers' names will be input from a file named '`PROB11.IN`'. Within this list there will be two special cases that will cause soldiers to leave the selection room. If <Assignment> is encountered, then the first soldier in line will exit. If an identified soldier's name enclosed by <> is encountered, then that soldier will leave the room.

```
Mark Sanders, Jane West, Charly Charleston, <Assignment>, Bob Brainard,
Samantha Sigert, <Charly Charleston>, Zeke Trainer, Sue Truit, Baxter
Billingsly, Ian Fleming, Paul Winston, <Assignment>, <Assignment>,
Edward Overton, Jose Jimenez, Huong Chu, Betty Evers, Larry Manchester,
Dan Blocker
```

Program Output

After all data from the input file has been entered, list the remaining soldiers in the selection room that will have to compete in the Code Wars competition. Output these results to the output file '`PROB11.OUT`'.

```
Samantha Sigert, Zeke Trainer, Sue Truit, Baxter Billingsly, Ian
Fleming, Paul Winston, Edward Overton, Jose Jimenez, Huong Chu, Betty
Evers
```

Problem #12

Code Wars II

Danger Level: Aggressive Blue

"Magic Square"

Point Value: 7

Problem Statement

A classic math puzzle. Given a semi empty 4x4 matrix, determine which values to place in each element to match the row, column, and diagonal sums.

```
 _____________
|   |   |13|   |  67
|_3_|___|___|___|  37
|   |   |   |11|  34
|   |15|   |   |  69
|___|___|___|___|

 52 48 31 76   38
```

Program Input

Each line in the file PROB12.IN gives the values for each row in the magic square. Blank values that need to be filled in are represented by XX, like this:

```
XX XX 13 XX 67
 3 XX XX XX 37
XX XX XX 11 34
XX 15 XX XX 69
52 48 31 76 38
```

Program Output

Determine the missing values and print the entire magic square to the output file, PROB12.OUT. You may assume all values in the square, including the sums, are within the range 0 to 99, inclusive.

```
 9 27 13 18 67
 3  1 12 21 37
16  5  2 11 34
24 15  4 26 69
52 48 31 76 38
```

Problem # 13

Code Wars II

Danger Level: Black

"128 Bit Math"

Point Value: 10

Problem Statement

Develop a program that will add or subtract up to 128 bit hex values.

Program Input

The input will be from file '`PROB13.IN`' as follows:

```
0x00001234 + 0xefC0111
0xFC5308Ca389BBE07-0x383F073D905ba3BB
```

Program Output

Output the input expression with the result to the output file '`PROB13.OUT`'.

```
0x1234 + 0xEFC0111 = 0xEFC1345
0xFC5308CA389BBE07 - 0x383F073D905BA3BB = 0xC414018CA8401A4C
```

Problem #14

Code Wars II

Danger Level: Black

"Arithmetic Expression Evaluation"

Point Value: 11

Problem Statement

Write a program to evaluate numerical expressions that include addition, subtraction, multiplication, and division. You will need to use proper operator precedence and parenthetical grouping.

Program Input

Each line of the input file PROB14.IN is an expression to be evaluated. The expression syntax is identical to C, except all numbers are real (float) by default. The program should be able to handle up to 16 nested parenthetical groups. Here are some example lines from an input file:

```
3.2 + 7.091 * (-19.4 - 6.3)
(1.3+0.1*(4.3+7.4*19.3-72/(3.14+2.4*9.2)+96))/(64.3-13*(9-4/-3))
8.4 * 19.6)
5.6 + ((32 / (71.3 - 9)
```

Program Output

The program should numerically evaluate the expression and print the result to PROB14.OUT, the output file. If the program detects a group of incorrect parentheses then it must display the proper error message (only the first error if there are more than one). Here are the output lines for the above input:

```
-179.0387
-0.3616351112589
error: unexpected )
```

```
error: unmatched (
```

Problem #15

Code Wars II

Danger Level: Black

"Matrix Amazement"

Point Value: 13

Problem Statement

Political prisoner Bobby "Byte" Bouchet has been captured in a hostile military zone and confined to a cell four feet on each side. You work for a foreign intelligence service assigned to free Bouchet. The operative needs to override the access rights of the security doors, and your task is to write the decoder.

Each door has a local security controller with a 4x4 matrix of byte control codes that are changed on a daily basis. The matrix override sequence is a "path" of values through the matrix which conforms to the following rules:

1. the sequence begins at position 0,0
2. the sequence ends at position 3,3
3. as the sequence progresses, each number in the sequence is directly adjacent (up, left, down, or right) to the previous number (no diagonal connections)
4. the path cannot contain loops, and
5. the value at position 3,3 is the sum of all the previous numbers in the sequence.

Program Input

Each security door is described in the input file PROB15.IN by a series of five lines. The first line in each descriptor is an identifier, and the next four lines are each a sequence of four numbers, with each line representing a matrix row and each number in column sequence. There may be several doors in the file. For example:

```
DOOR NUMBER 1
 12  26   7  31
101   8  61  44
 18  82  13 119
 83   3  47 251
```

Program Output

For each security door the program should print, on one line, the identifier and the command override sequence. For example:

```
DOOR NUMBER 1: 12 26 7 61 13 82 3 47 251
```

Problem #16

Code Wars II

Danger Level: Black

"McFood Lines"

Point Value: 14

Problem Statement

Did somebody say McFood? This problem is all about *your* McFood!

It seems the bigwigs at McFood want to study how people respond to long lines in their restaurants, and you're going to help them by writing a program to simulate people getting in line and ordering McFood. No cutting in line, now...

Program Input

There are two input files associated with this problem. The first file, MCFOOD.DAT, is a data file listing all the McFood items available to order. Each item is listed by name along with the average amount of time required to retrieve that McFood item in seconds.. Here are some example lines from that file:

```
ITEM=Double Pounder;TIME=45
ITEM=Soda;TIME=20
ITEM=McChitterlings;TIME=71
ITEM=McPasta;TIME=53
ITEM=Big Mic;TIME=35
ITEM=French Fries;TIME=28
ITEM=Salad;TIME=17
ITEM=Nuglets;TIME=24
ITEM=Tea;TIME=20
ITEM=McPeanut Butter Sandwich;TIME=39
```

The second input file specifies (1) the number of lines to simulate, (2) the people who enter the restaurant (*patrons*), (3) the time at which each patron enters, and (4) what McFood each patron orders.

The exact format is as follows: The first line indicates the number of lines to simulate in the restaurant. This number may vary between 1 and 10, inclusive. The remaining file is separated into time segments divided by lines with the text TIME=N. After an individual TIME=N line, each patron is listed who is to enter the restaurant at that time. One or more people may be specified for this time. Thereafter the file may contain another TIME=N line, followed by more patrons, indefinitely. the value of N in the line TIME=N is measured in tens of seconds; that is, TIME=2 means "at the time 20 seconds after the beginning of the simulation." The simulation begins at TIME=0 (zero).

Each patron record is started with a NAME=qqq line, where qqq is the patron's name. This is followed by on or more lines of ITEM=xxx;COUNT=c, indicating an item the patron wishes to order, followed by the number of those items. Here's an example input file:

```
LINES: 3
TIME=0
NAME=Bill
ITEM=Double Pounder;COUNT=1
ITEM=Big Mic;COUNT=2
ITEM=French Fries;COUNT=3
ITEM=Soda;COUNT=2
NAME=Vaneesha
ITEM=Salad;COUNT=1
ITEM=Soda;COUNT=1
TIME=2
NAME=Kathleen
ITEM=Double Pounder;COUNT=1
ITEM=French Fries;COUNT=1
ITEM=Nuglets;COUNT=1
ITEM=French Fries;COUNT=1
ITEM=Soda;COUNT=2
TIME=3
NAME=Piotr
ITEM=McChow;COUNT=3
ITEM=Tea;COUNT=2
ITEM=Soda;COUNT=1
TIME=7
NAME=Hyung
ITEM=McChitterlings;COUNT=1
ITEM=Soda;COUNT=1
```

Program Output

The program should print the name of each patron to separate line in the output file PROB16.OUT. Each output line should also include which McFood line served the patron, and two times:

1. the patron's wait time, in minutes & seconds, from restaurant entry to transaction

completion, and
2. the simulation's clock time, in seconds, at transaction completion.

The output lines should appear in the output file sorted by simulation clock time.

Because items can be retrieved in parallel, you should subtract 10 seconds from the total time required for each item above one that an individual patron orders. (i.e., 2 items means subtract 10 seconds, 3 items means 20 seconds, etc.) If the patron orders a McFood item that the restaurant doesn't serve, then ignore the item, but add fifteen seconds for the delay from the confusion. To account for the time required to place the order, add an additional 7 seconds for each ITEM=xxx line for that patron (regardless of the item count).
Finally, add another 17 seconds to pay for the McFood.

If a line becomes empty while another line has more than one patron, the patron at the end of the longest line moves to the empty line. In the case of two lines tied for longest, select the patron from the highest numbered line, i.e., line 5 over line 2. Upon entry into the restaurant, patrons should be assigned to a line according to these rules:

1. the patron first chooses the shortest line.
2. if two or more lines are equal in length, the patron chooses the line with the lowest number. i.e., she chooses line 2 over line 5.

Here is the output for the above sample input file:

```
Vaneesha was served in line 2; wait time = 0:58; sim time = 58s
Hyung was served in line 2; wait time = 1:52; sim time = 182s
Kathleen was served in line 3; wait time = 2:47; sim time = 187s
Bill was served in line 1; wait time = 3:34; sim time = 214s
Piotr was served in line 2; wait time = 1:46; sim time = 288s
```

Problem #17

Code Wars II

Danger Level: Double Black ◆◆

"Airport Geography"

Point Value: 15

Problem Statement

The FAA regulates and restricts air flight in certain regions, especially near airports. Each airport has an air traffic control tower that controls the airspace within a designated region. A pilot who enters a controlled airspace must usually communicate with the tower, possibly changing frequencies while flying through that airspace. In this problem we'll write a program to determine which air spaces are entered by a specific flight plan.

Reality: In reality, air traffic control spaces have irregular 3D shapes. Reality is also quite complex in that our planet is apparently spherical. I haven't actually observed this myself, but it is a well accepted postulate. It certainly makes the math for this problem quite a bit harder than I want it to be.

Simplicity: For simplicity, we'll model this complex reality by oversimplifying. *Radically* oversimplifying. First, we'll assume that we can map longitude and latitude to miles in a linear fashion. This is patently untrue, and will make our results wholly inaccurate. But this is only a contest. To make our results a little more meaningful, we will restrict our map of sites to Texas, because Texas is fairly flat, heh, heh. Based on our first assumption we'll also assume that flight paths are straight lines in a rectangular (longitude, latitude) coordinate system. Next we'll define the controlled airspace as a vertical cylinder with the control tower at the center. Finally, based on our first wildly inaccurate assumption, we'll convert degrees to miles using the following formulae:

MILES = 50.23 * DEGREES LONGITUDE
MILES = 60.07 * DEGREES LATITUDE

The program should read a data file describing all these controller air spaces. The program will then read an aircraft departure and destination from a separate file. Finally, the program will write the names of all of the air spaces through which the aircraft must fly in order of their

occurrence.

Program Input

The file AIRSPACE.DAT contains six lines of text for each air traffic control tower. Each airport data block contains the name of the airport, the tower's longitude and latitude in degrees, and the radius of that tower's controlled air space in miles. All airports are located in Texas (mostly!). The names and locations of the airports are accurate, but the tower control radius figures are fictional for this problem. Here are some sample lines for a single airport:

```
Airport Name: GEORGE BUSH INTCNTL HOUSTON
IATA: IAH
Location: HOUSTON, TX, UNITED STATES
Latitude: 29° 58' N
Longitude: 95° 20' W
Radius: 25 MILES
```

After reading the airspace data file, the program should read a series of flight paths from the input file PROB17.IN. Each line of the input file contains (1) the IATA codes for the departure and arrival locations, in that order, (2) the departure time in 24 hr format, and (3) the average aircraft speed in miles/hour. For example:

```
SAT IAH 08:14 120MPH
DFW SAT 08:14 120MPH
ELP IAH 17:53 185MPH
```

Program Output

The program should write the results to the output file PROB17.OUT. For each flight plan, the program should write the IATA codes for the departure and arrival airports on a single line. Then it must print the names of the airports through whose airspace the aircraft will fly. The airports should be listed in the order they will be crossed in flight. The actual departure and arrival should be considered separate "intersections" as shown in the example output. For each airport print (1) the airport name, (2) the name of the city where the airport is located, (3) the longitude and latitude of the intersection between the flight path and the airspace. Note that some air spaces overlap the towers of other airports. All airspace's which include the flight path must be reported. For example, the output for the first input line above is:

```
From: SAT   To: IAH
DEPART: SAN ANTONIO INTL(SAT)
        SAN ANTONIO, TX, UNITED STATES
        29° 32' N by 98° 28' W
CROSS:  NEW BRAUNFELS MUNICIPAL(3R5)
        NEW BRAUNFELS, TX, UNITED STATES
        29° 34' N by 98° 9' W
CROSS:  GERONIMO FIELD(T90)
        SEGUIN, TX, UNITED STATES
```

```
        29° 35' N by 98° 4' W
CROSS:  SAN ANTONIO INTL(SAT)
        SAN ANTONIO, TX, UNITED STATES
        29° 35' N by 98° 4' W
CROSS:  GERONIMO FIELD(T90)
        SEGUIN, TX, UNITED STATES
        29° 36' N by 97° 53' W
CROSS:  NEW BRAUNFELS MUNICIPAL(3R5)
        NEW BRAUNFELS, TX, UNITED STATES
        29° 37' N by 97° 51' W
CROSS:  GEORGE BUSH INTCNTL HOUSTON(IAH)
        HOUSTON, TX, UNITED STATES
        29° 53' N by 95° 55' W
ARRIVE: GEORGE BUSH INTCNTL HOUSTON(IAH)
        HOUSTON, TX, UNITED STATES
        29° 58' N by 95° 20' W
```

2 POINT BONUS: In addition to the requirements above, the program should also display (1) the average speed on the first line, (2) the prompt ENTER or EXIT instead of CROSS, and (3) the estimated time at which the aircraft enters and exits each controlled airspace (assuming the aircraft maintains the average speed throughout the entire flight). Times should be displayed in the same 24 hr format as the input. The same example input flight plan would yield this result:

```
From: SAT   To: IAH   AvgSpeed:   120MPH
DEPART: SAN ANTONIO INTL(SAT)
        SAN ANTONIO, TX, UNITED STATES
        08:14   29° 32' N by 98° 28' W
ENTER:  NEW BRAUNFELS MUNICIPAL(3R5)
        NEW BRAUNFELS, TX, UNITED STATES
        08:21   29° 34' N by 98° 9' W
ENTER:  GERONIMO FIELD(T90)
        SEGUIN, TX, UNITED STATES
        08:23   29° 35' N by 98° 4' W
EXIT:   SAN ANTONIO INTL(SAT)
        SAN ANTONIO, TX, UNITED STATES
        08:23   29° 35' N by 98° 4' W
EXIT:   GERONIMO FIELD(T90)
        SEGUIN, TX, UNITED STATES
        08:28   29° 36' N by 97° 53' W
EXIT:   NEW BRAUNFELS MUNICIPAL(3R5)
        NEW BRAUNFELS, TX, UNITED STATES
        08:29   29° 37' N by 97° 51' W
ENTER:  GEORGE BUSH INTCNTL HOUSTON(IAH)
        HOUSTON, TX, UNITED STATES
        09:18   29° 53' N by 95° 55' W
ARRIVE: GEORGE BUSH INTCNTL HOUSTON(IAH)
        HOUSTON, TX, UNITED STATES
        09:33   29° 58' N by 95° 20' W
```

Problem #18

Code Wars II

Danger Level: Double Black ◆◆

"Musical Transposition"

Point Value: 16

Problem Statement

Some musical instruments, such as the flute and the clarinet, require written music to be transposed for that instrument's tuning. For instance, music for a B^b clarinet is written 2 half steps above the actual notes. Write a program to transpose music according to the rules in the appendix.

Program Input

The input file PROB18.IN begins with a line indicating the tuning of the target instrument. For this problem you may assume the only instrument tunings will be B^b, D and E^b. The next line specifies the key signature of the source music. The notes from the source music begin on the next line separated by white space. The program may assume that no line will be longer than 80 characters, and that the source music is written for an instrument tuned to C.

```
TARGET: Bb CLARINET
KEY SIGNATURE: F
A5 B5 C5 D5 E5 A6 G#5 E5 B5 D5
C#5 A5 G5 F#5 E5 F#5 D5
F5 E5 D5 C5 Bb5 A5 G4 F4 E4
F4 A6 E4 G5 D4 G5 F5 E5 D5 C5 Bb5 A5
```

Program Output

The program should write the transposed music to the output file PROB18.OUT. The first line should be identical to the first line of the input file, except the program should change the key word TARGET to INSTRUMENT. The second line should indicate the transposed key signature, and the notes should follow thereafter. The program should maintain the same line breaks between notes as in the input file.

```
INSTRUMENT: Bb CLARINET
KEY SIGNATURE: G
```
B5 C#5 D5 E5 F#5 B6 A#6 F#5 C#5 E5
D#5 B5 A6 G#5 F#5 G#5 E5
```
G5 F#5 E5 D5 C5 B5 A5 G4 F#4
```
G4 B6 F#4 A6 E4 A6 G5 F#5 E5 D5 C5 B5

Appendix A: Basic Music Theory

Tranditional western music, like math, has a sophisticated written symbolic language and vocabulary. We will present a small subset here for use in solving this problem. The solution to this problem is quite simple once you understand what is being asked.

There are seven basic named notes: A, B, C, D, E, F, and G, which occur in a rotating sequence called an ***octave***. In written music, the octave of a particular note is indicated by its position on the staff. In our shorthand notation we'll write a number after each note to indicate its octave, such as E4 or C5. The octave number is incremented each time the cycle restarts at A, so the sequence looks like this: A1, B1, C1, D1, E1, F1, G1, A2, B2, C2, D2, etc.

There are 12 ***half-step*** intervals in each octave. The half-steps are shown in figure 1. A ***whole-step*** interval is two half-steps, i.e., A to B or D to E. The notes in between the basic named notes are called ***sharp*** ($^{\#}$) or ***flat*** (^{b}). The general term for sharps and flats is ***accidental***. When one of the basic notes appears without an accidental, it is called a ***natural***. (The symbol for a natural isn't included in most standard fonts, so we won't show it here. This symbol is rarely drawn, so for this problem we'll assume any note without a $^{\#}$ or ^{b} is a natural.) Notice that there is no half-step inteval between B and C, nor between E and F. In our notation we will write the accidental before the octave number, i.e., G$^{\#}$6 means "the G sharp in octave six".

1/2 steps	sharps	flats
0	A	A
1	A$^{\#}$	B^{b}
2	B	B
3	C	C
4	C$^{\#}$	D^{b}
5	D	D
6	D$^{\#}$	E^{b}
7	E	E
8	F	F
9	F$^{\#}$	G^{b}
10	G	G

11	$G^{\#}$	A^{b}
12	A	A

figure 1

Key	Notes in the key (scale)						
B^{b}	B^{b}	C	D	E^{b}	F	G	A
C	C	D	E	F	G	A	B
D	D	E	$F^{\#}$	G	A	B	$C^{\#}$
E^{b}	E^{b}	F	G	A^{b}	B^{b}	C	D
F	F	G	A	B^{b}	C	D	E
G	G	A	B	C	D	E	$F^{\#}$
A	A	B	$C^{\#}$	D	E	$F^{\#}$	$G^{\#}$

figure 2

Traditional western music is written around a certain ***key***, which is a grouping of seven of the twelve notes. The ordered sequence of these notes is called a ***scale***. The keys which we will use for this problem are shown in figure 2. Sometimes the intermediate notes (between the basic named notes) are sometimes referenced by the $^{\#}$, sometimes by the ^{b}, depending on the key signature. This means that although $G^{\#}$ may be the same frequency (or ***pitch***) as A^{b}, one notation will be preferred over the other in certain circumstances. Using these notes by way of example, if the key signature is A, the note should be written as $G^{\#}$, but if the key signature is E^{b}, that same note should be written as A^{b}.

Appendix B: Transposition

trans•pose - to alter a sequence of musical notes by a fixed interval.

As indicated in the problem statement, some musical instruments require written music to be transposed for that instrument's tuning. The instruments are often named by their tuning. For instance, a B^{b} clarinet produces a B^{b} pitch when fingered for a C. An E^{b} clarinet, as you might guess, produces an E^{b} pitch when fingered for a C. The music for these instruments must therefore be transposed appropriately. In the case of the B^{b} clarinet, each note must be raised by two half steps, B^{b} to C, D to E, etc. When music is transposed this way the key signature is also typically transposed. For the B^{b} clarinet the key of F would transpose up a whole step to the key of G. Music for the D clarinet must be transposed down a whole step, i.e., from the key of A to the key of G. An E^{b} clarinet requires music to be transposed down three half-steps, such as from the key of F to the key of D.

The transposed notes should be written in such a way as to conform to the transposed key signature. For all the notes that are in the key this is a simple mapping, but for notes with accidentals outside the key it becomes trickier. The rule is to transpose a note from its position (or interval) in the original key to the corresponding interval in the transposed key. Let's stick with the Bb clarinet and examine some of the possibilities:

oringinal key	note	transposed key	note	comments
B^{b}	G^{b}	C	A^{b}	The key of C has no accidentals, so the transposed note retains the accidental of the original note.
G	$D^{\#}$	A	$E^{\#}$	The keys of G and A both have sharps, so $D^{\#}$ (the 5th note in the scale) is transposed to $E^{\#}$ (also the 5th in the scale). F is not used even though is is the same pitch as $E^{\#}$ and may appear simpler.
E^{b}	B	F	$C^{\#}$	Since a B natural is a *half step above* the 5th note in the scale, we transpose to 1/2 step above the 5th note in F (the transposed key), to a $C^{\#}$.
F	E^{b}	G	F	The E^{b} is a half step below the 7th note in the scale, so we transpose to 1/2 step below the 7th note in G (the transposed key), to an F.

The program may assume that the note data will not contain double-sharps or double-flats, either in the original or transposed data. These rare cases do sometimes occur in written music.