A General Summary of COMCON
Programming Contest

(Some sample contests follows this description: newest March 2010)

Contests are held about five times per school year, usually about once a month. The **contests are free; there is no fee** of any kind. While coming to the contest site is encouraged, teams **may participate over the internet: there is no travel required.**

The contest consists of four programs. As many of these as possible are to be solved by a team (up to four people) using one computer within a two hour limit. **Schools can compete alone or against other schools;** to compete against other schools successful program solutions must be emailed (see address below).

Programs are worth 100 points each. There is a time penalty of one point for every five minutes of elapsed time before a program is solved. A "bad run", that is a program that does not give the correct output (or that crashes or fails to run properly) is an additional ten point penalty. See the "additional scoring" notes below.

A submission is made with the team's flash drive or server, indicating the team name, program number, and the language used. (If a school has more than one team, the designation "A team", "B team", etc. is also included). The judges will fill in the time the program was submitted.

The program submission is named PROGRAM1, PROGRAM2, etc. and has an extension indicating the language in which it is written: .CPP, .perl, .java, etc. The programs may be submitted in any order. Because COMCONs are fundamentally contests of speed, a good strategy is to submit the easiest program first.

The programs are tested by running the program, which opens the test data file on the C: drive or some other location (C:\PROGRAM1.DAT, C:\PROGRAM2.DAT, etc.). Programs must complete execution and terminate within 60 seconds. Judges will run the program using test data, determine successful runs by visual inspection of the output, and will report the results as soon as possible.

If a program does not work with the test data, a bad run will be noted, and a comment will be given to the team indicating the general nature of the problem.

A scoreboard will indicate what teams have solved which programs, and the relative standings of the meet.

The team with the highest score wins, with the exception that a team that solves more total programs than another wins, regardless of time.

The decisions of the judges are final.

Questions? Please email Rich Lamb: Rlamb@cranbrook.edu

Additional Scoring Notes

The contest runs for two hours. There are four problems in each contest. A team may have one or more students to a maximum of four. Regardless of the number of team numbers, a team may only use one computer. Programs may be submitted in any order. Because the contest is largely one of speed, it is a good strategy to submit the easiest program first.

Here is an example of scoring: If I begin a contest at 4:00 and turn in my first problem say at 4:17, that means I have used 3 five-minute time blocks (4:00 to 4:05, 4:05 to 4:10, and the next two minutes, but I haven't used the

full block from 4:15 to 4:20). Thus, I have three points of time against me, so I score a 97 (100 - 3) for the first problem.

Now let's say that my second correct problem comes in at 4:31. The clock has been ticking since 4:00, so I have 7 points against me, and I score a 93 (1- 7) for this second problem.

Each time I have a "bad run" (the program crashes, takes more than one minute to execute, or gives the wrong output), I receive a ten point penalty **whether or not I subsequently successfully complete that problem.** (Penalties hurt!)

So, for example, let's say I submitted program 3 at 4:17, then program 2 right after, but it did not work, and then I submitted program 1 at 4:31. My score would be 97 + 93 - 10 = 180. Then at 5:01 I correctly submit program 2, and make no more submissions. My final score is 97 + 93 + 89 - 10 = 269.

"Spokes II"
COMCON
17 March 2010
Program 1

From the Internet…

A rectangle of up to 5 X 5 cells, each containing a number, is described by a data set of R rows each of C numbers per line, separated by spaces. The numbers, from 0 to 8 inclusive, indicate to how many adjacent cells that particular cell is connected, either vertically, horizontally, or diagonally. A cell cannot be connected to any but its immediately adjacent neighbors. The first line of the data set contains R and C, the number of rows and columns, respectively, in the data. If a cell number is 0, it will obviously be connected to no other cells. If we then number each cell from 1 to (R X C), we can then write out a description of which cells are connected to what others. For example the data set below describes the picture:

SAMPLE INPUT (C:\PROGRAM1.DAT)


```
4 5
1 0
2 3 6 7 8
3 2 4 8 9
4 3 9
5 0
6 2 7 11 12
7 2 6
8 2 3 9 12 13 14
9 3 4 8 14 15
10 15
11 6 12 17
12 6 8 11 18
13 8 14 18
14 8 9 13 15 18 19
15 9 10 14
16 0
17 11 18
18 12 13 14 17 19
19 14 18
20 0
1 4
2 18
6 10
3 4
12 13
```
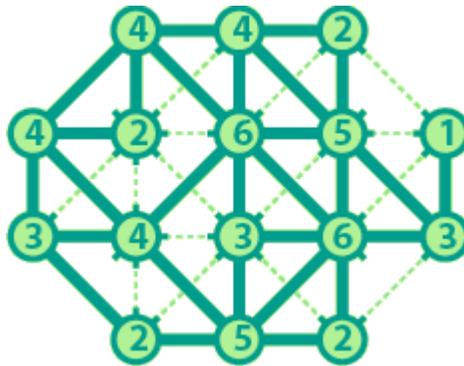
Wait a minute! What are those five data lines after the "20 0"? Well, that's where the fun starts. These five pairs represent beginning and ending cells where you need to give the shortest path length from one cell to the other. It may be that the two cell numbers are the same, then the path is zero. If the path is impossible, the output is "YCGTFH", you can't get there from here.

SAMPLE OUTPUT   Explanation (*NOT* part of the output)

YCGTFH   No path exists from cell one
4   Shortest path length is 2 4 13 18, a total of 4
6   6 2 8 9 15 10 or 6 2 3 9 15 10, for example, both are 6
2   From 3 to 4 is two cells
3   12 18 13 or 12 8 13, both are 3

"Mega Word"
COMCON
17 March 2010
Program 2

In the years before the advent of computers, a magazine ran a contest asking readers to find an English word whose letters, when each given a numeric value and multiplied together, came closest to one million. The value of each letter was given by its position in a string of all letters of the alphabet. The first letter has the value one, the second two, and so on with the last letter having the value twenty-six.

The data, all in uppercase letters, consists of the twenty-six letters in a random order, followed by five words. The output is the words and their values according to the scheme above, sorted in decreasing order.

For very large word values, output in scientific notation is permitted.

SAMPLE DATA: (C:\PROGRAM2.DAT)

QAZWSXEDCRFVTGBYHNUJMIKOLP
ORANGE,GREEN,WHITE,PINK,BLACK

SAMPLE OUTPUT:

ORANGE	846720
PINK	236808
BLACK	155250
GREEN	123480
WHITE	136136

This program uses 52 ordered integers given in data, each integer representing one of the 52 cards of a regular deck of playing cards, to display a bridge hand according to the instructions below. You do not have to know how to play bridge to write this program.

The actual integers in the DATA will be the counting numbers, 1 to 52, in some scrambled order. Each integer stands for one of the 52 cards in a deck of playing cards. The following table shows these values:

```
   A  2  3  4  5  6  7  8  9  T  J  Q  K
  ----------------------------------------------------
C | 1  2  3  4  5  6  7  8  9  10 11 12 13
D | 14 15 16 17 18 19 20 21 22 23 24 25 26
H | 27 28 29 30 31 32 33 34 35 36 37 38 39
S | 40 41 42 43 44 45 46 47 48 49 50 51 52
```

The suits are Clubs, Diamonds, Hearts, and Spades, denoted above by C, D, H, and S at the left of the table. Individual card names are across the top of the table (A = ace, T = 10, J = jack, Q = queen, K = king). Examples: The ace of spades is represented by the integer 40. The integer 23 represents the ten of diamonds.

The following is an example of how the DATA might look:

SAMPLE DATA:

42,30,11,48,17,19,14,31,45,4,5,6,46
2,36,3,18,16,24,40,47,29,51,32,20,28
35,8,37,10,23,12,50,21,49,15,52,9,34
33,25,39,44,22,27,38,43,1,7,26,41,13

The ordered data should be treated as a shuffled deck of cards. When a bridge hand is dealt, the 52 cards are dealt one at a time into four groups, or hands, denoted (in order) as WEST, NORTH, EAST, and SOUTH.

Thus, from the sample data above, 42 is WEST's card, 30 is NORTH's card, 11 is EAST's card, 48 is SOUTH's card, 17 is WEST's card, and so on.

When each player's hand is displayed, it must be sorted into suits and each suit must be arranged from highest card to lowest card. The ace of a suit is regarded as highest, followed by king, queen, jack, ten, 9, ... 3, 2. The suits are displayed in the order: Spades, Hearts, Diamonds, and Clubs, each labeled with S:,H:,D:,and C: as shown below.

After each label, each card is displayed using the single characters shown atop the table above. Each is separated by a single space. If any suit in a hand is void (meaning there are no cards in that suit) the suit label should be followed by a space and three consecutive dashes (---).

Continued...

When the above data is separated in hands, the following numbers would form the NORTH hand:

30,19,4,2,16,29,28,10,21,9,39,38,26

When displayed on the screen, this NORTH hand would be sorted and displayed as follows:

S: ---
H: K Q 4 3 2
D: K 8 6 3
C: T 9 4 2

The final output will be all four hands in the format just shown, in positions on the screen relative to the compass directions after which the hands are named. That is, the NORTH hand is the hand displayed roughly centered towards the top, the WEST hand is at the left, the EAST hand is at the right, and the SOUTH hand is towards the lower part of the screen.
If your program used the sample data above, the output would appear as shown below. Note that each hand is single spaced. The NORTH spade suit is on the top line of the screen. There are two blank lines below the club suit displays before the next spade suits are displayed. All output is upper case. Finally, note the logo in the center.

SAMPLE OUTPUT:

```
                                    S: ---
                                    H: K Q 4 3 2
                                    D: K 8 6 3
                                    C: T 9 4 2


S: K J 8 7 6 3                                              S: Q T 5 4 2
H: A J                      N                               H: T 9 8
D: Q 7 5 4                  W+E                             D: A J T
C: 7                        S                               C: J 5


                                    S: A 9
                                    H: 7 6 5
                                    D: 9 2
                                     C: A K Q 8 6 3
```

"Mirror Matrix"
COMCON
17 March 2010
Program 4

The first line of the data file defines the *row, column* dimensions of the matrix (no more than 10 x 10), the starting cell, and the direction (1 to 4, inclusive). The remaining data lines are *row* lines of *column* numbers each, to populate the matrix. All the data are integers.

The sample data describes a 4x5 array with the starting cell at (3,1) and a direction of 1. The upper-left cell is (1,1) and the directions are as follows: 1, up and to the right; 2, down and to the right; 3, down and to the left; 4, up and to the left.

SAMPLE INPUT (C:\PROGRAM4.DAT)

4,5,3,1
1,0,2,4,2
3,1,7,10,6
5,2,1,1,8
6,3,2,2,4

The program will travel through the array diagonally summing the values of the array cells through which it passes. The outer edges of the array are to be treated as "mirrors" during the journey. If such a cell is encountered, the path "bounces" and continues in a new diagonal direction, at a right angle relative to the old direction. The journey continues until the path comes to a corner of the matrix. The data will be such that the path is guaranteed to terminate in a corner.

During the journey, the value of each cell through which the journey passes are summed. However, each cell is summed only once, regardless of the number of times a path may go through a cell. If the path goes through a cell already visited, it should pass through it and continue in the same direction.

At the end of the journey the program will output the sums of both the visited cells and the sum of the cells not visited.

In the sample data, the value of sum of the visited cells (in the order in which they were visited) would be:
5 + 1 + 2 + 10 + 8 +2 + 1 + 1 (each cell is counted only once).

SAMPLE OUTPUT

THE SUM OF VISITED CELLS IS 30
THE SUM OF THE CELLS NOT VISITED IS 40

TEST DATA – program 1 SPOKES II

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

TEST DATA – program 2 megaword

"QWERTYUIOPASDFGHJKLZXCVBNM"
"SQUIRREL","DISHES","WOMBAT","LIGHT","FOOBAR"
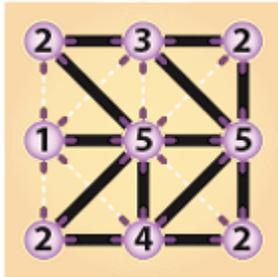

TEST DATA – program 3 BRIDGE

22,10,32,16,4,25,6,33,51,34,5,35,15
48,27,52,20,28,49,21,19,2,46,43,9,41
3,44,1,24,39,29,12,11,13,38,40,17,50
8,47,36,37,30,14,31,42,23,45,26,7,18

TEST DATA – program 4 Mirror Matrix

6,8,3,4,1
5,3,4,0,8,10,15,16
17,8,9,6,12,32,8,27
4,12,7,1,2,4,6,70
8,14,32,50,75,3,18,85
3,1,15,16,61,42,69,91
11,22,17,31,42,18,37,12

TEST OUTPUT – program 1 SPOKES II

4
3
0
3
3

TEST OUTPUT – program 2 megaword

FOOBAR   1197504
DISHES   718848
LIGHT   182400
WOMBAT   617760
SQUIRREL   612864

TEST OUTPUT – program 3 BRIDGE


```
                          S: 9 2
                          H: T 8 5 2
                          D: K Q J 4
                          C: J T 2



S: A Q 8 6                                              S: J T 7 3
H: ---                          N                       H: A K J 6
D: A 9 7 6 2                   W+E                       D: ---
C: A Q 9 4                      S                        C: K 7 6 5 3



                          S: K 5 4
                          H: Q 9 7 4 3
                          D: T 8 5 3
                          C: 8
```

following for test only – not part of output

```
WEST:   9D  4C  QS  2D  7D  6D  9C  AC  QC  AS  8S  AD  6S
NORTH:  TC  QD  8H  9S  2H  2C  2S  JD  JC  4D  TH  5H  KD
EAST:   6H  6C  5C  AH  TS  7S  3C  KH  KC  JS  JH  3S  7C
SOUTH:  3D  7H  9H  KS  8D  4S  5S  3H  QH  8C  4H  TD  5D
```
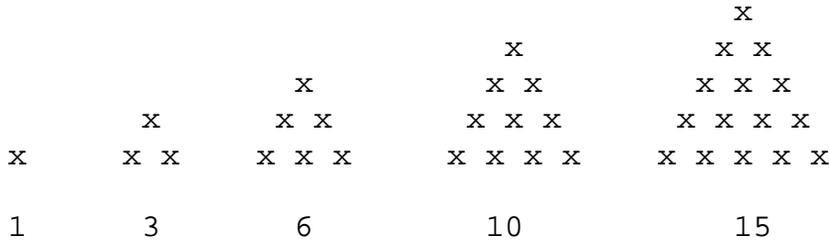
TEST OUTPUT – program 4 Mirror Matrix

THE SUM OF VISITED CELLS IS 541

THE SUM OF THE CELLS NOT VISITED IS 518

"Triangle Numbers"
COMCON
9 November 2005
Program 1

Triangle numbers are those from which a triangular shaped pile may be made from that number of objects:

```
                                        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

1      3      6        10            15
```

The sequence is 1, 3, 6, 10, 15, 21… and is infinite. In the 5 groupings above, the *number* of the groups are 1, 2, 3, 4, and 5 (that is, the first, second, third, fourth and fifth triangle groups) and the *values* of the groups are 1, 3, 6, 10, and 15, respectively.

The input file consists of numbers, which either is preceded by an "N" for number, or a "V" for value. For the "N" number, you are to print out the corresponding value; for the "V" number, you are to print the corresponding number. It may be that some "V" numbers are not correct, that is, they are not triangle numbers. In this case, you are to print "incorrect number". The flag of "END" will terminate the data. The data file will not exceed 1,000,000 for a value.

```
SAMPLE INPUT (C:\PROGRAM1.DAT)

N3
V6
V200028
N1983
N51
V1000
END
```

```
SAMPLE OUTPUT

6
3
632
1967136
1326
INCORRECT NUMBER
```

This is an adaptation of an old puzzle. A jailer is in charge of N cells, $1 \leq N \leq 1000$. One day to relieve boredom, the jailer decides to go to every Xth cell, with wrap-around if necessary, and to visit T cells in total. When arriving at a cell, the lock is toggled, that is, if it is locked, it is now unlocked, and vice-versa. The cells are all initially locked. The program will then print a range of cells given by the data and printed the number, in order, of each unlocked cell.

The data has values of N, X, and T in that order, followed by a pair of numbers that represent the range of cells to print. The first of the pair will always be less than or equal to the second. X will be less than N, and T will be no greater than one million (the jailer is *very* bored).

```
SAMPLE INPUT (C:\PROGRAM2.DAT)



10
4
7
3
8
```

Here the jailer has 10 cells, and will visit every 4th cell for seven times. Hence, the cells visited are 1, 5, 9, 3, 7, 1, and 5, in that order.

SAMPLE OUTPUT

```
CELL 3 IS UNLOCKED
CELL 7 IS UNLOCKED
```

"Squares & Cubes"
COMCON
9 November 2005
PROGRAM 3

This is strange and yet weird problem, as it has no input file, only output.

There is one three-digit number and one four-digit number that are both perfect squares and perfect cubes. Find and print these numbers.

Then, to help alleviate your boredom, print the next highest number that has this property.

Thus, the program will output three numbers. A reminder your program must execute in one minute or less.

```
SAMPLE INPUT  (none)



SAMPLE OUTPUT (but not correct! ☺)



111
2222
10000
```

"Some Assembly Required"
COMCON
9 November 2005
Program 4

The COMCON assembler consists of an accumulator, a temporary storage area, and 10 registers, R0 through R9. The machine has the following instructions, where the operand is either an identifier or a storage location.

L	load the operand into the accumulator
A	add the operand to the contents of the accumulator
S	subtract the operand from the contents of the accumulator
M	multiply the contents of the accumulator by the operand
D	divide the contents of the accumulator by the operand
N	negate the contents of the accumulator
T	store the contents of the accumulator in the operand location

An arithmetic operation replaces the contents of the register with the expression result. Temporary storage locations are allocated by the assembler for an operand of the form "Rn" where n is a single digit.

The input file consists of a (syntactically correct) postfix expression. Expression operands are single letters and operators are the normal arithmetic operators (+, -, *, /). Output must be assembly language code that meets the following requirements:

1. One instruction per line with the instruction mnemonic separated from the operand (if any) by one blank.
2. One blank line must separate the assembly code for successive expressions.
3. The original order of the operands must be preserved in the assembly code.
4. Assembly code must be generated for each operator as soon as it is encountered.
5. As few registers as possible should be used (given the above restrictions).
6. For each operator in the expression, the minimum number of instructions must be generated (given the above restrictions).

Sample input
(C:\PROGRAM4.DAT

AB+CD+-

Sample output

```
L A
A B
T R0
L C
A D
N
A R0
```

(note only one register is used)

TEST DATA – November 2005

Triangle numbers   PROGRAM1.DAT

V1234567
N1571
V7112106
N700
N2145
N4471
V10000000
N17906
END

Bored Jailer   PROGRAM2.DAT

170
4
253
8
15

Squares & Cubes   PROGRAM 3

<no data file>

Some Assembly Required PROGRAM4.DAT

AB+CD+EF++GH+++

TEST OUTPUT

Triangle Numbers - Program 1

INCORRECT NUMBER
1234806
3771
245350
2301585
9997156
INCORRECT NUMBER
160321371

Bored Jailer - Program 2

cell 9 is unlocked
cell 11 is unlocked
cell 13 is unlocked
cell 15 is unlocked

Squares and Cubes - Program 3

729
4096
15625

Some Assembly Required - Program 4
Note, that judges should examine output if it does not exactly match what is given here. There are a number of possibilities that can be correct.

L A
A B
T R0
L C
A D
T R1
L E
A F
A R1
T R1
L G
A H
A R1
A R0