1987 ACM Scholastic Programming Contest XI

Anagram Dictionary (*A*)

Source File:	`ANAGRAM.PAS`, *if Pascal*
	`ANAGRAM.FOR`, *if FORTRAN*
Data File:	`ANAGRAM.DAT`
Output File:	`ANAGRAM.OUT`

An anagram of a word is a different word formed using the same characters that comprise the original word. For example, the word BRAINY is an anagram of the word BINARY.

An anagram dictionary showing all words that can be constructed using a given set of letters is useful for certain kinds of word puzzles and games. Each entry in such a dictionary contains a key giving the set of letters in alphabetic order, and one or more words that can be formed using the letters found in the key. The anagram dictionary entry for the key ABINRY might then appear like this:

```
ABINRY       : BRAINY       BINARY
```

For this problem you are to produce an anagram dictionary containing a given set of words. The words will appear one per line in the input data, starting in column one. There will be no duplicate words in the input, and each word will contain 12 or fewer uppercase alphabetic characters. There will be no more than 1000 words in the input, and no more than 20 anagrams will be formed from any key.

Your program must produce the anagram dictionary so the keys themselves are in alphabetic order. Each line of each entry in the dictionary will contain five anagrams, except for the last line of each entry, which may contain fewer than five anagrams. The anagrams in each entry need not be in any particular order. The first line of each entry will begin with the key, left-justified in a 12 character field, followed by a colon. Each additional line of each entry will begin with 13 blanks. The five or fewer anagrams on each line are displayed left-justified in 12 character fields starting in column 14, each field preceded by a blank.

For example, if the input data contained the words:

```
CRAPES
INTEGRAL
RELATING
ESCARP
PACERS
ALTERING
SYZYGY
ALGORITHM
CAPERS
TRIANGLE
MARTLESSNESS
SCRAPE
LOGARITHM
BINARY
SPACER
TRAMLESSNESS
ALERTING
STY
BRAINY
RECAPS
```

(over)

then the corresponding anagram dictionary your program might produce would be:

```
ABINRY      : BRAINY      BINARY
ACEPRS      : RECAPS      SPACER      SCRAPE      CAPERS      PACERS
              ESCARP      CRAPES
AEELMNRSSSST: TRAMLESSNESS MARTLESSNESS
AEGILNRT    : ALERTING    TRIANGLE    ALTERING    RELATING    INTEGRAL
AGHILMORT   : LOGARITHM   ALGORITHM
GSYYYZ      : SYZYGY
STY         : STY
```

1987 ACM Scholastic Programming Contest XI

Chess (*B*)

```
 Source File:  CHESS.PAS, if Pascal
               CHESS.FOR, if FORTRAN
   Data File:  CHESS.DAT
 Output File:  CHESS.OUT
```

You are to write a program which will determine if a set of chessmen are attacking a given position on a chess board. (You do *not* need to know how to play chess to solve this problem, the limited amount of information about chess that you will require is provided below.) The program is to read in the positions of three chess pieces: a Knight, a Bishop, and a King; determine which pieces, if any, are attacking a set of board positions; print out the results of this determination; and then loop back to repeat the process. The procedure is to terminate when a particular board configuration is read in.

The following information about the game of Chess is sufficient to solve this problem:

Chess is played on a square board consisting of eight rows and eight columns (i.e. an 8 by 8 grid). Board positions are described with a set of ordered pairs (r,c) where 1 <= r <= 8 and 1 <= c <= 8 are the row and column numbers of a position.

Chessmen, or pieces, may move from a given position to a new position based on their type and the following rules: (i) No piece is allowed to move off the 8 x 8 board, and (ii) a piece is not allowed to move to a square occupied by another piece (you Chess players do not need to concern yourselves with the exception to this rule that applies when capturing an opponent's piece).

The Knight can move from a given (i,j) board position to any one of eight other positions (subject to the restrictions listed in the prior paragraph). The possible new positions are two rows up or down and one column left or right from the initial position (four combinations), or two columns left or right and one row up or down from the initial position (four more combinations). Thus, if the Knight is initially at position (4,4), then it can move to positions (2,3), (2,5), (6,3), (6,5), (3,2), (5,2), (3,6), and (5,6), assuming that none of these eight positions is already occupied.

The Bishop can move from an (i,j) board position to any of the diagonal positions given by (i±k,j±k) subject to the additional restriction that it can not "jump over" any other pieces that might be located along one of the diagonals. Thus, if the Bishop is initially at position (3,2), then it can move to positions (2,1), (4,3), (5,4), (6,5), (7,6), (8,7), (2,3), (1,4), or (4,1), assuming there are no other pieces along these diagonals. If there were some other piece at position (6,5), then the moves to (6,5), (7,6), and (8,7) would not be allowed.

The King can move from a given (i,j) board position to any one of the eight immediately adjacent positions which are not already occupied. Thus, if the King is initially at position (4,4), then it can move to positions (3,3), (3,4), (3,5), (4,3), (4,5), (5,3), (5,4), or (5,5).

A piece at position (i,j) is "attacking" a different position (r,c) if the piece can move from (i,j) to (r,c). (Multiple pieces can simultaneously be attacking a given position.) A "configuration" is a given set of chess pieces and their board positions.

(over)

Your program should read in a configuration, together with a set of board positions to be checked, and print out a message indicating which pieces of the configuration, if any, are attacking each of the board positions to be checked. The process is repeated for multiple sets of configurations.

A configuration is represented by a set of six integers in the range of zero through eight which are on a single input line and separated by one or more blanks (there will be no blank lines in the data). The first pair of integers represent the row and column position of a Knight, the second pair of integers represent the row and column position of a Bishop, and the third pair of integers represent the row and column position of a King. The zero values will be used only for a "dummy" sentinel configuration that indicates that there is no more input data (i.e. your program should terminate when it reads in a configuration consisting of six zero values).

Each position that is to be checked for a given configuration is represented by a set of two integers in the range of zero through eight. Each position will be entered on a separate line with one or more spaces between the integers on a line (there will be no blank lines in the data). Each pair of integers represent the row and column number of the position to be checked. The zero values will be used only for a "dummy" sentinel position that indicates that there are no more positions to be checked for the current configuration (i.e. when a zero zero position is read in then a new configuration will follow).

For each position that is to be checked, your program should print out a separate line in the following format:

```
N:a,b   B:c,d   K:e,f - g,h UNDER ATTACK BY xyz
```

where "`a`" and "`b`" are the row and column numbers of the Knight, "`c`" and "`d`" are the row and column numbers of the Bishop, "`e`" and "`f`" are the row and column numbers of the King; "`g`" and "`h`" are the row and column numbers of the position being checked (row and column numbers are single digits); and "`xyz`" is one of the eight possible values "`NOTHING`", "`N`", "`B`", "`K`", "`NB`", "`NK`", "`BK`", or "`NBK`" corresponding to the (g,h) position being under attack by none of the pieces, by the Knight (`N`) alone, the Bishop (`B`) alone, the King (`K`) alone, the Knight and Bishop (`NB`) together, the Knight and King (`NK`) together, the Bishop and King (`BK`) together, or the Knight and Bishop and King (`NBK`) together. This information should be output in the same order that it is input, that is, information about the first position for the first configuration, information about the second position for the first configuration, ..., information about the final position for the final configuration. No information should be output for dummy sentinel positions or configuration. As an example, assume the following data is input:

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

then the resulting output should be:

```
N:1,1   B:2,2   K:3,3 - 3,2 UNDER ATTACK BY NK
N:1,1   B:2,2   K:3,3 - 7,7 UNDER ATTACK BY NOTHING
N:5,6   B:4,4   K:2,5 - 3,5 UNDER ATTACK BY NBK
```

1987 ACM Scholastic Programming Contest XI

Splay Crossings (*C*)

Source File: `CROSS.PAS`, *if Pascal*
`CROSS.FOR`, *if FORTRAN*
Data File: `CROSS.DAT`
Output File: `CROSS.OUT`

A splay is defined as a set of line segments in the cartesian plane. You are to write a program that reads in a splay and counts the number of pairs of intersecting segments.

There are four special cases to consider in finding intersections:

1. A splay may contain identical segments, in which case only one of the identical segments should be considered when finding intersections with other segments. Two segments are identical if they have the same endpoints, although they need not be listed in the same order in the input data set.

2. Two parallel segments are considered not to intersect each other, even if they have an endpoint or a shorter segment in common.

3. A segment of length zero is considered parallel to all other segments, and thus cannot intersect any other segment.

4. Two nonparallel segments intersect if and only if they have exactly one point in common (which may be an endpoint).

The data set for this problem will consist of several lines of data, each containing the coordinates of the two endpoints of one line segment, in the order x1, y1, x2, y2. Each coordinate will be an integer in the range 0 to 50 (inclusive). Consecutive values are separated by at least one space. There will be at most 50 segments in the splay.

The output from your program should contain two values, each appropriately labeled. The first is the number of distinct (non-identical) segments in the splay, and the second is the number of pairs of intersecting segments in the splay.

1987 ACM Scholastic Programming Contest XI

Quick and Dirty Message Encryption (*D*)

Source File: `ENCRYPT.PAS`, *if Pascal*
`ENCRYPT.FOR`, *if FORTRAN*
Data File: `ENCRYPT.DAT`
Output File: `ENCRYPT.OUT`

While on an unscheduled and unsanctioned mission to deliver "goods" in a none-too-friendly country, it becomes necessary to communicate with encrypted messages. The method of encryption has to be simple and fast to implement (no more than 4 programmers, 1 PC, and less than 6 hours to complete the program).

The lines of the message to be encrypted will be contained in a file. Each line will be no more than 80 characters in length, and there will be an indeterminate number of lines to be encrypted. The end of the message will be designated by end-of-file.

The method of encryption will be as follows:

- Place the first line of the message row-by-row into the smallest square matrix that will hold it. (For our purposes, we will consider the line ended at the rightmost nonblank character, even if additional blanks follow. The effective message length is therefore between 0 and 80 characters, inclusive.) Pad out the matrix at the end with successive characters starting with "A". (Thus, if 4 blanks are unfilled by the line at the end of the matrix, fill them with "A", "B", "C", and "D".)
- Shift each row of the matrix one column to the right. The contents of the rightmost column will wrap around to fill the leftmost column.
- Invert each column of the matrix.
- Flip the matrix along its upper-left-to-lower-right diagonal.
- Remove the encrypted line of the message row-by-row from the matrix and write it on its own line in the output file.
- Repeat the process for additional lines of input until no more lines are encountered.

The sample input message

```
This is
a message
```

should generate the following output:

```
BiissTA h
esmaeags
```

1987 ACM Scholastic Programming Contest XI

Illegal Transmissions (*E*)

Source File:	`XMIT.PAS`, *if in Pascal*
	`XMIT.FOR`, *if in FORTRAN*
Map Data File:	`XMIT.MAP`
Transmission Data File:	`XMIT.DAT`
Output File:	`XMIT.OUT`

You are a member of a highly secretive radio telemetry team assigned to investigate suspicious radio signals emanating from a cluster of islands in your region. It is your task to ferret out the sources of these illegal transmissions.

A map which represents the islands in your region as convex polygons in an integer-based rectangular coordinate system is stored in the map data file. A top secret transmission tracer produces the transmission data file which contains the origins of the illegal transmissions.

Write a program that, for each transmission in the transmission data file, identifies the name of the island from which the transmission originated or, in the event that the transmission did not originate from an island, reports that it *originated at sea*. Your output should look like this.

```
Transmission #1 at site (20,17) is from the island "Pangea".

Transmission #2 at site (35,30) is from the island "Oil Platform".

Transmission #3 at site (0,0) originated at sea.
```

Input Data Description:

The *map data file* consists of a list of multiple-line data sets each representing part of the territory of an island. The first line of such a data set is the name of the island in which the territory resides, left-justified in the first 20 columns. The second line contains the number of vertices of the convex polygon that defines the territory's borders, right-justified in the first 5 columns. Each of the remaining lines contain the rectangular coordinates of one vertex of the polygon with the x- and y-coordinates right-justified in columns 1-5 and 7-11, respectively. It is assumed that consecutive vertices are connected by a side of the polygon and that the first and last vertices are connected by a side as well. An island is comprised of the interiors and borders of the territories in the map data file having the island's name. No two islands touch.

The *transmission data file* is a sequence of lines each containing the coordinates of the origin of a single transmission in the rectangular coordinate system of the map with the x- and y-coordinates right-justified in columns 1-5 and columns 7-11, respectively.

Special Judging Criteria:

All coordinates in the test data sets have integer values from 0 to 1000.

1987 ACM Scholastic Programming Contest XI

Integer Decompositions (*F*)

Source File: `DECOMP.PAS`, *if Pascal*
`DECOMP.FOR`, *if FORTRAN*
Data File: `DECOMP.DAT`
Output File: `DECOMP.OUT`

It is known that every integer *n* greater than 17 may be written as the sum of three distinct integers, each greater than one, that are pairwise relatively prime; that is, no pair of the three have a common prime factor. To illustrate this fact, write a program that reads values of *n* from successive input lines and, for each *n*, show a decomposition of *n* into the sum of three such integers *a*, *b*, and *c* by writing a line of the form

$$n = a + b + c.$$

For example, if *n* = 18, the output below would be acceptable.

```
18 = 4 + 9 + 5
```

Your program should process as many values as there are in the input file. Assume that every line contains exactly one input value in the range 18 to 32,767 right-justified in columns 1-5.

Your program may output any correct decomposition. Furthermore, the order of the addends is immaterial. The following lines are among those that are acceptable for *n* = 18.

```
18 = 9 + 5 + 4
```

or
```
18 = 2 + 3 + 13
```

1987 ACM Scholastic Programming Contest XI

Word Ladders (*G*)

Source File: `LADDER.PAS`, *if in Pascal*
`LADDER.FOR`, *if in FORTRAN*
Data File: `LADDER.DAT`
Output File: `LADDER.OUT`

Given two words with the same number of letters, a *word ladder* is a sequence of words that changes the first word into the second by changing a single letter each time. Thus to change COLD to WARM, we might construct the word ladder:

```
C O L D
C O R D
C A R D
W A R D
W A R M
```

For this problem you must write a program that can read in a "dictionary" of four letter words and then construct word ladders (or report that none exists) for specific pairs of words.

The first part of the input data will contain an arbitrary number of lines (maximum 100), each containing a four letter word, all uppercase letters, in the first four positions on the line. Next there will be a dictionary terminator line, containing a dollar sign ($) in the first position.

The second part of the input data will consist of several lines, each containing two four letter words. The first word is in the first four character positions on the line, then there is exactly one blank space, and then the second four letter word in the next four character positions. This part of the input is terminated by end-of-file.

Your program should first print the two words. If a ladder exists, using only the words in the dictionary, print that ladder in the vertical format shown above. Otherwise, print the message NO LADDER EXISTS.

For some pairs of words, there may be more than one ladder possible. Any correct ladder is acceptable.

1987 ACM Scholastic Programming Contest XI

Refinery Pipes (*H*)

Source File: `PIPES.PAS`, *if in Pascal*
`PIPES.FOR`, *if in FORTRAN*
Data File: `PIPES.DAT`
Output File: `PIPES.OUT`

A refinery has three large pipes running through the space between two buildings. The pipes and the building walls are all parallel. The drawing below shows the walls and pipes from the end, thus the pipes appear as circles.

The refinery wants to add a fourth pipe in the space between the other pipes, as shown by the broken line circle in the drawing. Your program must determine the radius and location of the largest pipe that can be placed there, touching all three of the existing pipes running parallel there. Neither walls, nor eaves, nor ground, nor dark of night shall constrain the solution of this problem.

The data for your program will describe the current arrangement of pipes. For each pipe, one line of data will give the x and y coordinates of the center of the pipe and the radius of that pipe as three real numbers (decimal points present, digits fore and aft) separated by at least one space. Thus there will be three lines of data. All values are in centimeters. The origin of the coordinate system is as shown in the diagram: at the ground (x-axis) and at the left wall (y-axis).

Your answers should also be in centimeters and should show the radius of the new pipe and the coordinates of its center, appropriately labeled. Answers must be accurate to one millimeter.