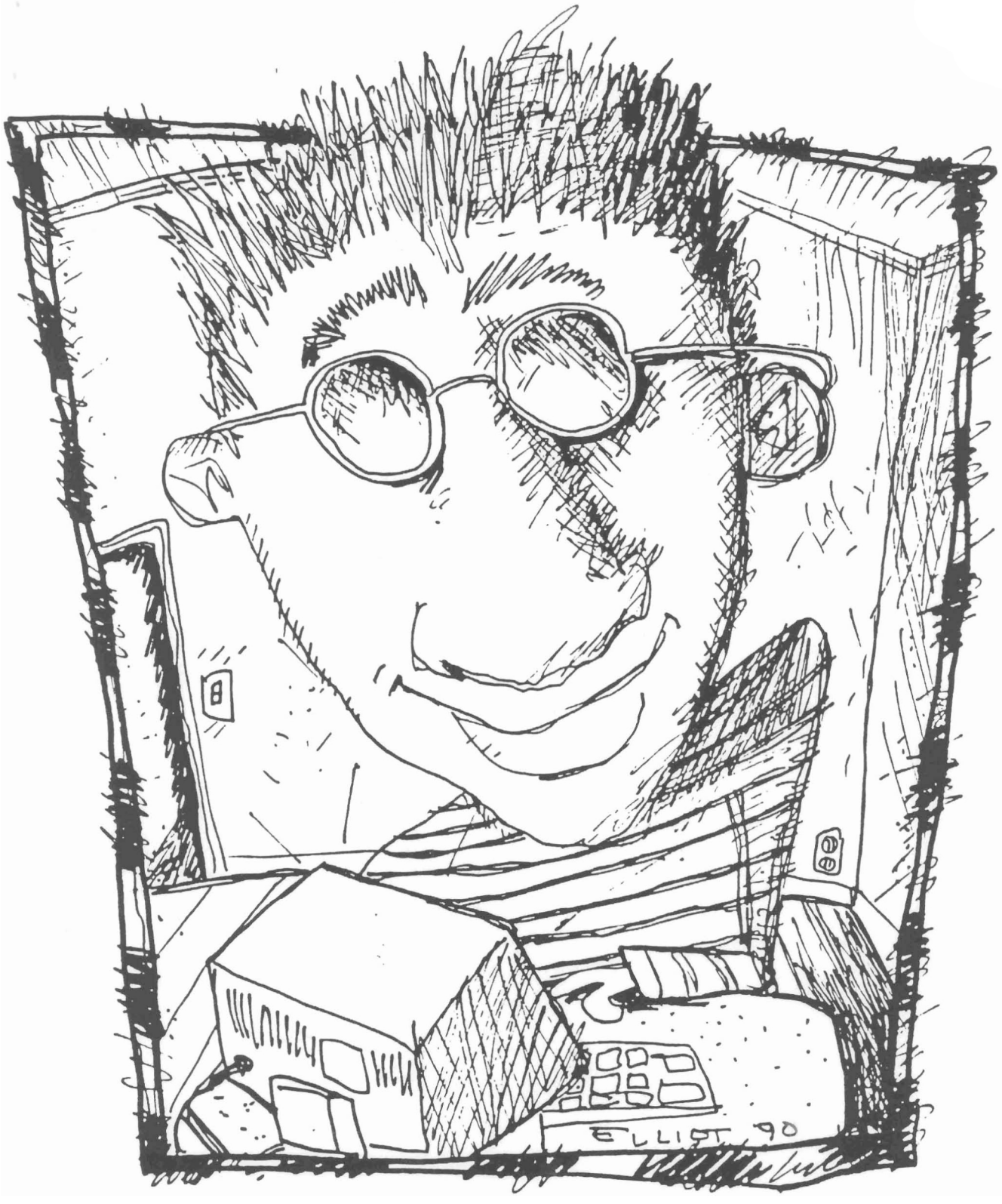


4TH ANNUAL
UNIVERSITY OF CENTRAL FLORIDA
HIGH SCHOOL PROGRAMMING TOURNAMENT

PROBLEMS

Fourth Annual
University of Central Florida

High School Programming Tournament

Problems

Problem Name	Filename
Damaged Digit Display	DIGIT
On the Wild Side	WILDCARD
Legal Words	WORDS
Chess Adjournment	ADJOURN
Yo!	TOYS
Got it Covered	COVERED
Bingo!	BINGO
Keyed Up	MOUSE

Call your program file: *Filename*.PAS
Call your input file: *Filename*.IN

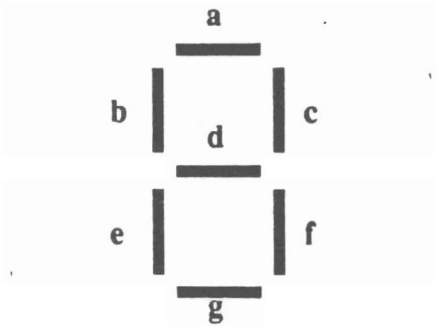
For instance, if you are solving Keyed Up:

Call your program file: MOUSE.PAS
Call your input file: MOUSE.IN

Damaged Digit Display

Filename: DIGIT

On most digital clocks, each digit is displayed as a pattern of seven on or off light-emitting diodes (LED's) which resemble that number. Here is an example of such a display, with the LED's labelled with the letters a through g:



The digits zero through nine are represented by the following on/off configurations:

Digit	0	1	2	3	4	5	6	7	8	9
"on"	abcefg	cf	acdeg	acdfg	bcdf	abdfg	abdefg	acf	abcdefg	abcdfg
"off"	d	abdeg	bf	be	aeg	ce	c	bdeg	(none)	e

Suppose that some of the LED's that make up a certain digit on a digital display are obscured or broken; that is, it is not possible to determine whether they are off or on. This may cause observers to interpret the LED pattern incorrectly (as the wrong digit) or to be unable to determine which digit it represents.

The Problem:

Write a program that will list all possible digits that a broken display might be representing, when given the letters of the broken LED's and the letters of the "on" LED's (all other LED's are to be assumed "off").

The Input:

There will be several broken displays to analyze. The information for each display will use two lines of the input file--the first line will list letters of broken LED's and the second will list letters of "on" LED's. The letters will be lower case letters, corresponding to the positions of the LED's of the previous diagram.

The Output:

For each display configuration in the input file, print out all of the digits that it could possibly represent. The digits should be all on the same line, in five-character fields. They may be in any order on that line.

Sample Input:

```
ec
dagfb
ec
dagf
abd
cf
```

Sample Output:

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

On the Wild Side

Filename: WILDCARD

A *Wildcard* is a symbol which can be used to match zero or more other symbols. An asterisk (*) is often used in this capacity. For instance, **a*m** can match **am, arm** or **antidisestablishmentarianism.**

The Problem:

Given a string with a single wildcard in it and a list of other strings, determine which of the other strings match the string with the wildcard.

The Input:

There will be several sets of data. Each set will begin with the wildcard string on the first line. The wildcard string is guaranteed to be at least one and at most 50 characters in length, and is guaranteed to have **exactly** one wildcard, represented by an asterisk (*). The other characters (if any) are guaranteed to be lower case letters. On the next line will be an integer *n* indicating the number of other words in the list. On the next *n* lines will be the other words, one per line. The other words are guaranteed to be at least one and at most 50 characters in length, all of which will be lower case letters.

The Output:

For each of the other words, print one of the following messages:

WildcardString matches *String*

or

WildcardString does not match *String*

whichever is appropriate. Separate the output for different data sets by a blank line.

Sample Input:

```
a*m
6
am
arm
antidisestablishmentarianism
manna
army
fudge
co*er
4
computer
courtney
casper
courier
```

Sample Output:

```
a*m matches am
a*m matches arm
a*m matches antidisestablishmentarianism
a*m does not match manna
a*m does not match army
a*m does not match fudge

co*er matches computer
co*er does not match courtney
co*er does not match casper
co*er matches courier
```

Legal Words

Filename: WORDS

The grammar police have enlisted you to act as a word checker. Sentences with illegal words have been trying to creep into stories and reports all over. Of course, your keen eye can spot these words in an instant. They contain no vowels! For this problem, *Y* is considered a legal vowel.

The Problem:

Your job is to read a list of words and determine which of them is illegal.

The Input:

The data set will consist of a list of words, one per line. The words will consist of at least one and at most 20 letters. The letters may be capital or lower case.

The Output:

For each word, print:

```
word is legal
```

or

```
word is not legal
```

whichever is appropriate. Print one message per line.

Sample Input:

```
This
is
a
test
Mr
Jones
Eh
```

Sample Output:

```
This is legal
is is legal
test is legal
Mr is not legal
Jones is legal
Eh is legal
```

Chess Adjournment

Filename: ADJOURN

Chess tournaments are very rigorously officiated. Every game has an *Arbiter*, an official who oversees the game. A game may be *adjourned* and continued at a later date, but the arbiter must fill out a form indicating the current state of the game. You are to write a program which will, given the state of the game, print out an adjournment form.

The Problem:

Given a chess board, the names of the players, and who moves next, print an adjournment form according to the specifications below.

The Input:

There will be several forms to print. For each form there will be ten lines of information. The first line will have the names of the players, black player first, separated by a comma and a space. The next eight lines will each have eight characters, indicating the contents of the board according to the following code:

White		Black
P	Pawn	p
N	Knight	n
B	Bishop	b
R	Rook	r
Q	Queen	q
K	King	k
.	Empty	#

The black player's side of the board will be on line 2, and the white player's on line 9. The tenth line will have either a capital W or a capital B, indicating which player should move next. Thus, if Fred were playing Sam, the board at the start of the game would be represented like this:

```
Fred, Sam
rnbqkbnr
pppppppp
.#.#.#.#
#.#.#.#.
.#.#.#.#
#.#.#.#.
PPPPPPPP
RNBQKBNR
W
```

The Output:

Print an adjournment form which looks like this (this one is for the example at the end of the last section):

```
Black Player: Fred

     King: e8
  Pawn(s): a7 b7 c7 d7 e7 f7 g7 h7
Knight(s): b8 g8
Bishop(s): c8 f8
  Rook(s): a8 h8
 Queen(s): d8

   a  b  c  d  e  f  g  h
8 BR BN BB BQ BK BB BN BR 8
7 BP BP BP BP BP BP BP BP 7
6 .. ## .. ## .. ## .. ## 6
5 ## .. ## .. ## .. ## .. 5
4 .. ## .. ## .. ## .. ## 4
3 ## .. ## .. ## .. ## .. 3
2 WP WP WP WP WP WP WP WP 2
1 WR WN WB WQ WK WB WN WR 1
   a  b  c  d  e  f  g  h

White Player: Sam

     King: e1
  Pawn(s): a2 b2 c2 d2 e2 f2 g2 h2
Knight(s): b1 g1
Bishop(s): c1 f1
  Rook(s): a1 h1
 Queen(s): d1

White to move.
```

Each piece is represented by two capital letters: the first, a **W** or **B** indicating the color of the piece, and the second, a **K, Q, R, B, N** or **P** indicating the kind of piece, as in the input. The rows of the board are numbered 1-8 and the columns are lettered a-h, with a1 being the lower left corner and h8 being the upper right corner. Keep in mind that each player must have exactly one king and may have up to eight pawns, and due to pawn promotion may have up to ten rooks, ten bishops, ten knights, nine queens.

You may print the piece positions (a5, f3, etc.) in any order. Aside from that, you must print the form **exactly** as it appears here--pay particular attention to spacing, punctuation and capitalization! Print a blank line between forms.

Sample Input:

```
Fischer, Spassky
rnbqkbnr
pppppppp
.#.#.#.#
#.#.#.#.
.#.#.#.#
#.#.#.#.
PPPPPPPP
RNBQKBNR
W
Van Horn, Von Trapp
kr.#.Q.#
ppN.#.#.
.#.#.#.#
#.#.#.#.
Q#P#P#.#
#.#P#P#.
.#.#.#PP
R.#.#RK.
B
```

Sample Output:

```
Black Player: Fischer

     King: e8
  Pawn(s): a7 b7 c7 d7 e7 f7 g7 h7
Knight(s): b8 g8
Bishop(s): c8 f8
  Rook(s): a8 h8
 Queen(s): d8

   a  b  c  d  e  f  g  h
8 BR BN BB BQ BK BB BN BR 8
7 BP BP BP BP BP BP BP BP 7
6 .. ## .. ## .. ## .. ## 6
5 ## .. ## .. ## .. ## .. 5
4 .. ## .. ## .. ## .. ## 4
3 ## .. ## .. ## .. ## .. 3
2 WP WP WP WP WP WP WP WP 2
1 WR WN WB WQ WK WB WN WR 1
   a  b  c  d  e  f  g  h

White Player: Spassky

     King: e1
  Pawn(s): a2 b2 c2 d2 e2 f2 g2 h2
Knight(s): b1 g1
Bishop(s): c1 f1
  Rook(s): a1 h1
 Queen(s): d1

White to move.
```

Sample Output (continued):

```
Black Player: Van Horn

     King: a8
  Pawn(s): a7 b7
Knight(s):
Bishop(s):
  Rook(s):
 Queen(s):

   a  b  c  d  e  f  g  h
8 BK BR .. ## .. WQ .. ## 8
7 BP BP WN .. ## .. ## .. 7
6 .. ## .. ## .. ## .. ## 6
5 ## .. ## .. ## .. ## .. 5
4 WQ ## WP ## WP ## .. ## 4
3 ## .. ## WP ## WP ## .. 3
2 .. ## .. ## .. ## WP WP 2
1 WR .. ## .. ## WR WK .. 1
   a  b  c  d  e  f  g  h

White Player: Von Trapp

     King: g1
  Pawn(s): c4 e4 d3 f3 g2 h2
Knight(s):
Bishop(s):
  Rook(s): a1 f1
 Queen(s): a4 f8

Black to move.
```

Yo!

Filename: TOYS

A small child has ruined his toy yo-yo. The string has become knotted at the spindle, thus creating a very boring new toy called a yo. The child's older brother, who is a high school student, decides to use this toy as an example of simple geometry.

The Problem:

If the student knows the diameter of the spindle, the diameter of the yo, and the length of the string, then how many revolutions will the yo make before jerking to a halt at the bottom of the string? Assume that windings of the string do not increase the diameter of the spindle. Also, a length of string equal to one radius of the yo is needed to extend out of the yo to be able to hold the end before throwing.

Note: $\pi \approx 3.14159$

The Input:

There will be several data sets. Each data set wil consist of three lines. The first will contain the diameter of the yo, the second will contain the diameter of the spindle, and the third line will contain the length of the string (measured from holding end to knot at spindle).

The Output:

The number of revolutions should be listed accurate to two decimal places and formatted as follows:

```
Yo will make nnn.nn revolutions before jerking to a halt!
```

Sample Input :

```
5
16
110.1
1
45
300
```

Sample Output:

```
Yo will make   6.50 revolutions before jerking to a halt!
Yo will make  88.33 revolutions before jerking to a halt!
```

Got It Covered

Filename: COVERED

The Problem:

Given two (possibly intersecting) rectangles in the Cartesian plane, determine the total area that is covered by the two of them together.

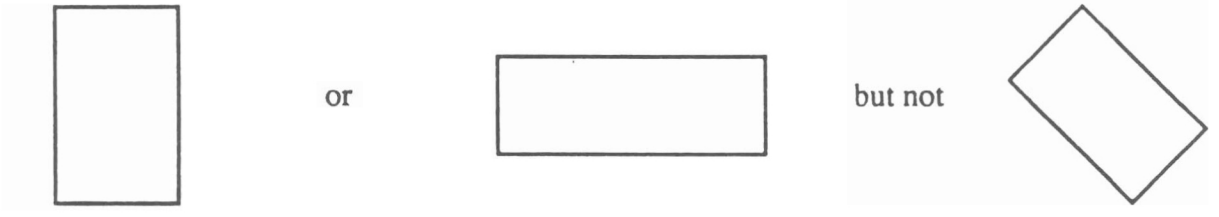
The Input:

There will be several data sets. Each data set will contain two pairs of coordinates, one pair for each rectangle. The coordinates will be specified in the following order:

```
x1  y1  x2  y2  x3  y3  x4  y4
```

where ($x1$, $y1$) and ($x2$, $y2$) are ordered pairs designating the coordinates of opposite corners of the first rectangle while ($x3$, $y3$) and ($x4$, $y4$) denote opposite corners of the second rectangle. The coordinates will be expressed in real numbers. All points for one data set will be on the same input line.

Rectangles are guaranteed to have non-zero areas. All rectangles will be vertical or horizontal in orientation. For example:



The Output:

For each data set, print the message:

```
Covered Area = area
```

where *area* is the total area covered by the pair of rectangles, accurate to three decimal places.

Sample Input:

```
2.0 4.0 4.0 2.0 6.0 4.0 8.0 2.0
0.0 0.0 1.0 1.0 10.5 10.5 9.0 9.0
```

Sample Output:

```
Covered Area = 8.000
Covered Area = 3.250
```

Bingo!

Filename: BINGO

A bingo card is a 5 by 5 matrix of positive integers with the center entry absent. The first column contains entries in the range 1 to 15 inclusive, the second column's entries are in the range 16 to 30, and so on, using numbers up to 75. The center space, or *Free Space*, is considered to be *covered* at the beginning of the game, and other positions are covered when their respective numbers are *called* by whoever is running the game. Bingo is won when five positions in the same row, column, or diagonal on the card are covered.

The Problem:

Given a bingo card and a list of numbers called, determine whether or not the card is a winner, and if so, say why.

The Input:

There will be several bingo cards, each followed by a list of called numbers. Each bingo card will be represented by five lines of input: each line will contain five positive integers--except the third, which will only have four due to the free space in the center. Each list of called numbers will immediately follow the card, consist of one number per line, and will be terminated by a 0.

The Output:

For each card, if the called sequence does not produce a bingo, print the message "`No Bingo.`" If the card is a winner, print the message "`BINGO! after` *n* `calls`" where *n* is how many numbers were needed from the list to win, along with a listing of the spaces covered on the board that produced the bingo, including "`Free Space`" if appropriate. If more than one bingo is created at the same time, print any one of them. Print a blank line between output sets.

Keyed Up

Filename: MOUSE

A local "hands-on" science museum for children has contacted you to help them implement part of a program that will be used in an upcoming display. The exhibit will allow a child to use a mouse as an input device to a computer. The computer screen will contain regions called *keys* each of which has a character on it. The child will select a key, and the computer will tell the child which key was selected.

A key is selected by using the mouse to point to it and then *clicking* the mouse button. Any click which does not correspond to a key is considered invalid.

When a child selects a key, the computer responds with a "Beep" and a message indicating the character on the key that was selected. However, when an invalid selection is made, the computer responds with a "BOING."

The Problem:

Your task is to write the program which will convert the mouse click coordinates into the proper responses from the computer.

The screen is 2500 x 2500 pixels in size, and each key is 250 pixels square. Screen coordinate (0, 0) is in the upper-left corner of the screen, while coordinate (2499, 2499) is in the lower-right corner of the screen. No key will extend beyond the boundaries of the screen, and no two keys will overlap. Note: if a border pixel of a key is clicked, the key is considered to have been pressed.

The Input:

Your program will process several data sets. Each set will begin with a line which contains two integers *m* and *n*: the number of keys on the screen and the number of mouse clicks in the data set, respectively. The following *m* lines will contain information about the keys: the character on the key and the coordinates of the upper-left pixel of each key. The remaining *n* lines of each data set contain coordinates corresponding to the locations of the mouse clicks.

The Output:

The output of your program will consist of one of the following messages for each mouse click:

```
Beep.  The key selected at coordinate (x, y) had "character" on it.
BOING!  The coordinate (x, y) does not correspond to a key.
```

where `(x, y)` is the screen coordinate of the mouse click and `character` is the character on the selected key.

Sample Input:

```
1 2
A 0 0
1000 1000
100 100
2 1
A 0 0
Q 500 0
500 100
```

Sample Output:

```
BOING!  The coordinate (1000, 1000) does not correspond to a key.
Beep.  The key selected at coordinate (100, 100) had "A" on it.

Beep.  The key selected at coordinate (500, 100) had "Q" on it.
```