leuven ACM88ek

European Regional Finals of the 1988–89 ACM Scholastic Programming Contest

Held at
Eindhoven University of Technology
The Netherlands

November 12, 1988

[7 problems A–G on 13 numbered pages]

Acm 88/89k

Problem A: From Prefix to Postfix

1988–89 ACM Scholastic Programming Contest
European Regional Finals

November 12, 1988

Source File:	`PREPOST.PAS`
Input File:	`PREPOST.INP`
Output File:	`PREPOST.OUT`

WARNING The characters in this problem are purely fictional. Any resemblance to real-life persons must be attributed to your own imagination. Good luck!

The Contest Director

Three well-known notations for expressions are prefix, infix, and postfix. Parentheses are needed in infix notation to eliminate ambiguities. For instance, the infix expression $a \star b \diamond c$ either means $(a \star b) \diamond c$ or $a \star (b \diamond c)$. Often priority rules are introduced to save on parentheses again. In prefix and postfix notation, however, there is no need for parentheses or priority rules. The above expressions in prefix notation are $\diamond \star abc$ and $\star a \diamond bc$, respectively. Here, each operator *precedes* its arguments, which themselves can be expressions. In postfix notation the operators are placed *after* their arguments: $ab \star c \diamond$ and $abc \diamond \star$, respectively.

The *arity* of an operator is the number of arguments it operates on. An operator with arity zero can be regarded as a constant. Several operators with their arity are given in Table 1 below. The arity of operator ? depends on the expression. Within an expression each occurrence of ? has the same arity, but for two different expressions the arity of ? may be different.

Operators	Arity
`0 1 2 3 4 5 6 7 8 9`	0
`- $ !`	1
`* +`	2
`f g h`	3
`P Q`	4
`?`	variable

Table 1: Operators and their arity

Not just any string of operators is a valid expression in prefix notation. For instance, the four strings

```
7
++345
-?12345609
?2?34
```

are valid prefix expressions. (Notice that 345 is not a single number but three separate constants. Also notice that the operator ? has arity 8 and 2 respectively in the last two expressions.) But the four strings

```
34
+34+56
1$
??1234
```

are not valid prefix expressions. In the last string, no proper arity for ? can be found.

The input for your program is a textfile. Each line consists of at least 1 and at most 80 operators from Table 1. The input file ends with the standard end-of-file marker. Your program determines for each input line whether it is a valid expression in prefix notation and if so, it computes the postfix notation for this expression.

The output is also a textfile. Each line contains the answer for the corresponding input line, that is, the expression in postfix notation if the input is a valid expression in prefix notation and the message 'Error' otherwise.

An example input file and corresponding output file are:

PREPOST.INP	PREPOST.OUT
7	7
++345	34+5+
34	Error
-?12345609	12345609?-
+34+56	Error
1$	Error
??1234	Error
?2?34	234??

END OF PROBLEM A

ACM 88ek

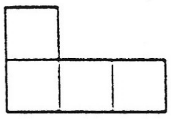
Problem B: The Snake in the Grass

1988–89 ACM Scholastic Programming Contest
European Regional Finals

November 12, 1988

Source File:	`SNAKE.PAS`
Input File:	`SNAKE.INP`
Output File:	`SNAKE.OUT`

Pieter's Garden is a rectangular piece of ground with integer dimensions $M \times N$ square units (M and N at least 2 and at most 30). Each square is identified by a coordinate pair (x, y), $0 \leq x < M$ and $0 \leq y < N$. His Garden is a big mess, it is infested with huge 1×1 holes and rocks. The worst thing, however, is the presence of a *monotonic snake*, yukkee[1]. When Yukkee slithers around she is always within the boundaries of a 2×3 (or 3×2) L-shaped area of 4 square units. This is called her (momentary) *configuration*. There are eight configuration classes, one of them is



The question is whether Yukkee can move from one corner of the Garden to the other. The holes and rocks are inaccessible. Moving, in this case, means changing the coordinate pair of one square of Yukkee's configuration under invariance of the L-shape. You may choose any initial configuration for Yukkee such that it covers square $(0, 0)$. Yukkee has reached the other corner when one of her coordinate pairs is $(M-1, N-1)$.

Oh, by the way, you know, of course, that the species of monotonic snakes has a limited ability to maneuver. Let $\check{x}$ be the minimum of the x-coordinates of the squares in the current configuration, and similarly $\check{y}$ the minimum of the y-coordinates. (N.B. $(\check{x}, \check{y})$ need not be in the configuration.) The new coordinates x' and y' of the changed square (i.e. *after* the move) must still satisfy $x' \geq \check{x}$ and $y' \geq \check{y}$. For instance, from the configuration $\{(10,10),(10,11),(10,12),(11,10)\}$ Yukkee can move to $\{(10,10),(10,11),(12,10),(11,10)\}$, but not to $\{(10,10),(10,11),(10,12),(9,12)\}$ (even though the latter configuration is correctly shaped).

Write a program to determine (the existence of) a path for Yukkee in a number of Gardens.

The input file is a textfile that presents a sequence of Gardens. Each Garden is described as follows. The first line gives M and N. The next line gives k, the number of inaccessible squares

[1] This is not the name of the snake, but an expression of disgust. I just don't like snakes. But, since you insist, let's call her Yukkee anyway.

in the Garden. The following k lines each contain the coordinate pair of one inaccessible square. Gardens are not separated in the input file.

The output file is also a textfile. For each input Garden you put Yukkee in some initial configuration and move her to the other corner. If this cannot be done, then your program outputs a line with the message 'NO'. Otherwise, it outputs a line with the message 'YES', followed by the description of a path. A path is specified by a sequence of coordinate pairs (each pair on a line by itself, coordinates separated by at least one blank). The first four lines must represent the initial configuration, each following line gives the new coordinates of the square that changed. The last line should encode $(M-1, N-1)$. Solutions are not separated in the output file.

Here follows an example of input with acceptable output.

SNAKE.INP

```
4 4
3
0 2
1 0
3 1
```

SNAKE.OUT

```
YES
  0  1
  1  1
  0  0
  2  1
  2  2
  2  3
  1  3
  3  3
```

END OF PROBLEM B

ACM 88ek

Problem C: The Jolly Jumper

1988–89 ACM Scholastic Programming Contest
European Regional Finals

November 12, 1988

Source File:	`JOLLY.PAS`
Input File:	`JOLLY.IN~~T~~P`
Output File:	`JOLLY.OUT`

A sequence of n, $n > 0$, integers is called a *jolly jumper* if and only if the absolute values of the difference of successive elements take on all of the values 1 through $n-1$. For instance, the sequence

```
1 4 2 3
```

is a jolly jumper, because the absolute differences are 3, 2, and 1, respectively. Notice that by definition a sequence consisting of a single integer is a jolly jumper. Besides itself, the above sequence contains 6 other contiguous subsequences that are also jolly jumpers.

For a number of non-empty sequences of non-negative integers, your program determines the length of a longest jolly jumper *in* that sequence. "In" here means "occurring as contiguous subsequence in".

The sequences are given in a `file of integer`, separated by `-1` and terminated by `-2`. The output of your program is also a `file of integer`, containing for each input sequence the desired answer (in the same order as the input, of course).

You may assume that the integers are at most 3000. For example, the following input

```
JOLLY.INT
1 1 1 -1 3 0 1 4 2 3 4 2 6 3 -2
```

should produce as output

```
JOLLY.OUT
1 5
```

END OF PROBLEM C

ACM 88EK

Problem D: El Puzzlo

1988–89 ACM Scholastic Programming Contest
European Regional Finals

November 12, 1988

Source File:	`LPUZZLE.PAS`
Input File:	`LPUZZLE.INP`
Output File:	`LPUZZLE.OUT`

Given is a square board of $n \times n$ fields, of which one field has been marked. Fig. 1 shows a 4×4 board where the field in the upper left-hand corner is marked. An L-piece has the shape of the letter 'L' and covers three neighboring fields (see Fig. 1). The objective of the L-puzzle is to cover the unmarked fields of the board with L-pieces, each field being covered by exactly one L-piece. For your convenience, Fig. 1 also shows a solution.

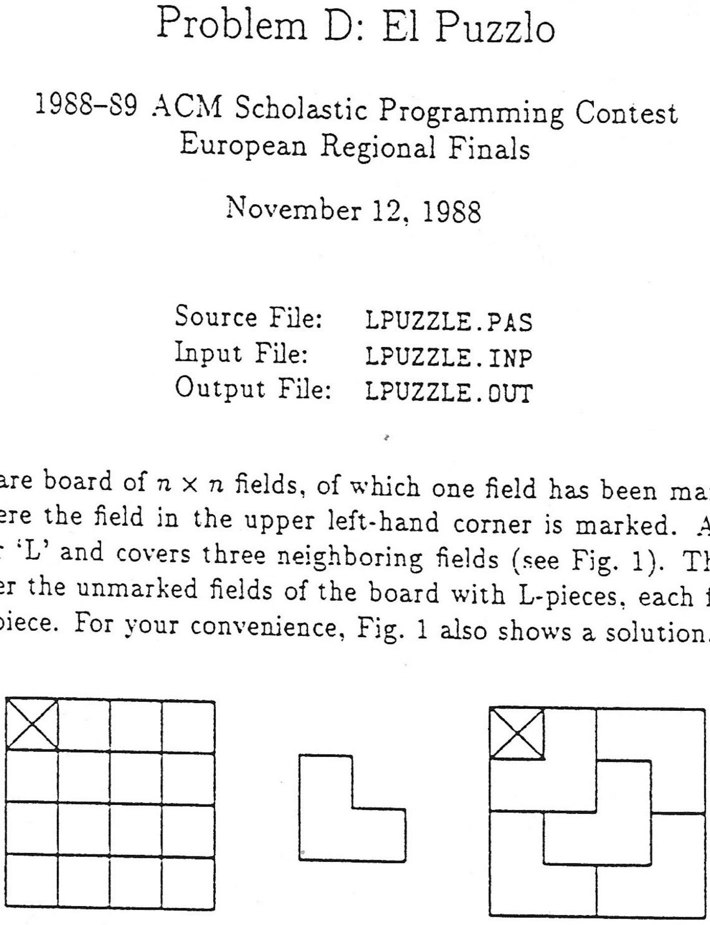


Figure 1: A board, an L-piece, and a solution

Your program solves a sequence of L-puzzles, where n is restricted to powers of 2. (Prove that in this case there is always a solution!) Furthermore, n is at most 32.

The input for your program resides in a textfile. Each line specifies an instance of the L-puzzle by giving integers n, r, and c (in that order, separated by at least one space): the board is $n \times n$ and the marked field is in row r/column c, where rows are numbered from the top, columns from the left, and numbering starts at 0. The case with $n = 0$ terminates the input, and otherwise $n > 0$, $0 \leq r < n$, and $0 \leq c < n$.

The output of your program is a textfile. It contains the solutions separated (but not terminated) by a single blank line. Your solution of an $n \times n$ L-puzzle is encoded in 1 line of $2n$ characters followed by n lines of $2n + 1$ characters. The characters that may occur are: space (' '), underscore ('_'), and vertical bar ('|'). If we number the characters on a line from 0 upwards, then 'vertical' information is encoded on the even positions and 'horizontal' information on the odd positions. Vertical information can be either a vertical bar or a space.

horizontal information can be either an underscore or a space. Rather than formally defining the encoding scheme we give an example. The solution of Fig. 1 (which could have been triggered by `4 0 0` in the input file) is encoded as follows:

```
 _ _ _ _
|_| |_  |
|_ _| |_|
| |_ _| |
|_ _|_ _|
```

Notice that the marked field is not encoded in a special way. The first line begins with a space, but ends with an underscore, and contains no vertical bars (that is why it very much looks like an empty line). The last line ends with a vertical bar, of course.

For example, the following output file is acceptable for the given input file.

LPUZZLE.INP

```
4 2 1
2 0 0
0 0 0
```

LPUZZLE.OUT

```
 _ _ _ _
|  _|_  |
|_|_  |_|
| |_|_| |
|_ _|_ _|


 _ _
|_| |
|_ _|
```

End of Problem D

ACM 88ek

Problem E: The Easy Part

1988–89 ACM Scholastic Programming Contest
European Regional Finals

November 12, 1988

Source File:	`MASTER.PAS`
Input File:	`MASTER.COD`
Output File:	`MASTER.HIN`

MasterMind is a game for two players. One of them, *Designer*, selects a secret code. The other, *Breaker*, tries to break it. A *code* is no more than a row of colored dots. At the beginning of a game, the players agree upon the length n that a code must have and upon the colors that may occur in a code.

In order to break the code, Breaker makes a number of guesses, each guess itself being a code. After each guess Designer gives a hint, stating to what extent the guess matches his secret code. The game is over when Breaker's guess equals the secret code.

The topic of this programming problem is not to guess the secret code. We stick to the easy part of the game: supplying the hints.

Given a secret code $s_1, \ldots, s_n$ and a guess $g_1, \ldots, g_n$ the hint consists of a pair of numbers determined as follows.

A *match* is a pair (i, j), $1 \leq i \leq n$ and $1 \leq j \leq n$, such that $s_i = g_j$. Match (i, j) is called *strong* when $i = j$, and it is called *weak* otherwise. Two matches (i, j) and (p, q) are called *independent* when $(i = p) \Leftrightarrow (j = q)$. A set of matches is called independent when all its members are pairwise independent.

Designer chooses an independent set M of matches for which the total number of matches and the number of strong matches are both maximal. The hint then consists of the number of strong followed by the number of weak matches in M. Note that these numbers are uniquely determined by the secret code and the guess. If the hint turns out to be $(n, 0)$, then the guess is identical to the secret code, and the game is over.

The input of your program is in a textfile. On this file a number of games are represented.

- Colors are represented as integers from 1 through k, where k is the size of the set of colors that is agreed upon.
- A code is represented by the blank-separated list of the representations of the colors of its successive dots.

- A game is represented on a number of consecutive lines. The first line contains the two parameters n and k of the game, in this order, separated by blanks. The second line contains the representation of the secret code. The remaining lines contain the representations of the successive guesses, one per line.

- Two successive games are separated by a line with the character 'Y' in the first position. The last game is followed by a line with some character different from 'Y' in the first position.

You may assume that for each game $0 \overset{<}{\leq} n \leq 1000$ and $0 < k \leq 10000$.

The output of your program has to appear in a textfile. For each guess in the input file the output file contains (in the same order) the pair of integers representing the hint. These integers are preceded by blanks and/or end-of-line markers.

For example, we have the following input file with acceptable output file.

```
MASTER.COD
4 6
1 3 5 5
1 1 2 3
4 3 3 5
6 5 5 1
6 1 3 5
1 3 5 5
N
```

```
MASTER.HIN


1 1
2 0
1 2
1 2
4 0
```

END OF PROBLEM E

Acm88EK

Problem F: Series Parallel Graphs

1988–89 ACM Scholastic Programming Contest
European Regional Finals

November 12, 1988

Source File:	`SP.PAS`
Input File:	`SP.INP`
Output File:	`SP.OUT`

The class of *series parallel* (SP) graphs is defined recursively as follows.

(i) A directed graph consisting of two (hence, distinct) vertices a and b connected by a single arc is a SP graph:

$$a \longrightarrow b$$

(ii) If G is a SP graph, then so is the graph obtained from G by

(a) doubling an arc of G (creating arcs in parallel):

$a \longrightarrow b$ is replaced by $$\begin{tikzcd} a \arrow[r, bend left] \arrow[r, bend right] & b \end{tikzcd}$$

(b) chopping an arc of G (creating arcs in series), thereby introducing a fresh vertex x:

$a \longrightarrow b$ is replaced by $a \longrightarrow x \longrightarrow b$

Thus, SP graphs are directed graphs, possibly with multiple arcs between vertices. Furthermore, SP graphs have a unique vertex without incoming arcs, called the *source*, and a unique vertex without outgoing arcs, called the *sink*.

An example of a SP graph is given in Fig. 1.

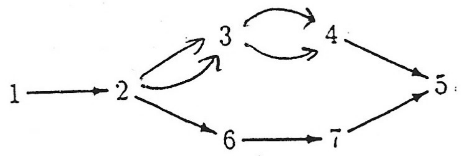


Figure 1: Example SP graph

Your program determines for a sequence of directed graphs whether they are SP or not. For each SP graph, the program also determines the source and the sink.

The input file is a textfile and contains the description of a number of directed graphs. The vertices of each graph are numbered from 1 upwards. A graph is represented by listing for each vertex the numbers of its successors. More precisely, a graph with N vertices is represented on $N + 1$ successive lines:

- line 0 contains N,
- line i contains the successors of vertex i ($1 \leq i \leq N$).

The representations of successive graphs are *not* separated. The input is terminated by the standard end-of-file. You may assume that the number of vertices of each input graph is at most 100.

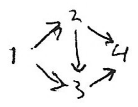
The output file is also a textfile. For each graph the output file contains 'YES' or 'NO' (on a line by itself), indicating whether the corresponding input graph is SP or not, respectively. If a graph is SP, then two more lines follow, containing the number of the source and sink, respectively, in strict format.

Sample input with corresponding output:

```
SP.INP
7
2
3 6 3
4 4
5

7
5
4
2 3
4 3
4
```

```
SP.OUT
YES
1
5
NO
```



END OF PROBLEM F

Acm88ek

Problem G: Contest Aid

1988–89 ACM Scholastic Programming Contest
European Regional Finals

November 12, 1988

Source File:	`LCHECK.PAS`
Input Files:	`LCHECK.INP`, `LCHECK.SOL`
Output File:	`LCHECK.OUT`

Imagine you are organizing a programming contest, just like this one, where programs are to be judged by checking their output for certain test runs. When the input uniquely determines the output, the output can be checked by a simple comparison with the known correct output. When you are less fortunate, you need the aid of a special program that examines the output and checks whether it is correct for the given input (of course, you cannot rely on 'visual' inspection by a human using some text editor).

This problem requires you to write a program that does the output checking for problem D (El Puzzlo), so you'd better read that one too.

The input for your program resides in two textfiles. The format of `LCHECK.INP` is exactly the same as that of the input file of Problem D. The values of n are at most 1024, and not necessarily powers of 2.

In general, `LCHECK.SOL` could be just any horrible textfile of purported L-puzzle solutions. To simplify matters, you may assume, however, that it consists of the right number (as determined by the other input file) of non-empty blocks of non-empty lines, where blocks are separated (but not terminated) by a single empty line.

The output is a textfile. For each input case your program produces one output line with the test result. This can be either of the following three messages: '`Wrong Format`', '`Wrong Answer`', or '`Ok`'. 'Wrong Format' indicates that the format is wrong, i.e. one (or more) of the following holds: wrong number of lines; wrong number of characters in some line; some character is not a space, underscore, or vertical bar: an underscore occurs in a vertical information position, or a vertical bar occurs in a horizontal information position. 'Wrong Answer' applies if the format is correct, but it does not encode the solution for the desired L-puzzle. 'Ok' is what you are aiming for in problem D.

Example:

LCHECK.INP
```
4 2 1
2 0 0
5 0 4
1000 3 12
0 0 0
```

LCHECK.SOL
```
 _ _ _ _
|  _|_  |
|_|_| |_|
|  _|_ _|
|_|_ _ _|


 _ _
|_| |
|_-_|


 _ _ _ _ _
|  _|_  |_|
|_|_  |_| |
|_  |_|_ _|
| |_|  _| |
|_ _|_|_ _|

 _ _ _ _ @#>!<=?
I give up!
How about you, Sally?    _
May be this fools them: |_|
```

LCHECK.OUT
```
Wrong Answer
Wrong Format
Ok
Wrong Format
```

END OF PROBLEM G