Concours Informatique Luxembourgeois 2002
EPREUVE DEMI-FINALE

Problemstellung I - Rösselsprung

Der Rösselsprung ist ein Strategiespiel für eine Person. Das Spielfeld ist wie ein Schachbrett. In unserer Aufgabe hat das Brett n Zeilen und k Spalten.

Der Spieler hat nur eine einzige Figur: ein Springer der sich so bewegen kann wie der Springer des Schachspiels (siehe Regeln des Schachspiels). Zu Spielbeginn befindet sich der Springer auf dem Feld A1 (oben links).

Der Spieler muss jetzt den Springer ziehen bis der Springer alle Felder ein einziges Mal besucht hat und dann so stehen, dass er mit dem nächsten Zug zurück auf A1 bewegt werden kann. Wenn dies nicht gelingt hat der Spieler verloren.

Die folgenden Abbildungen zeigen 2 mögliche Lösungen für n=8 und k=8. Die Zahlen in den Feldern ergeben die Reihenfolge der Spielzüge an. Der Springer beginnt auf 1, zieht dann auf 2 und so weiter bis auf 64 und von dort könnte er wieder auf Feld 1 springen.

1	62	43	58	3	54	45	52
42	59	2	63	44	51	4	55
61	64	39	48	57	6	53	46
38	41	60	7	50	47	56	5
25	8	49	40	23	16	29	14
34	37	24	11	28	13	20	17
9	26	35	32	19	22	15	30
36	33	10	27	12	31	18	21

1	4	61	12	59	56	47	14
62	11	64	3	16	13	58	55
5	2	9	60	57	48	15	46
10	63	6	17	8	27	54	49
31	18	41	28	37	50	45	26
40	21	30	7	42	53	36	51
19	32	23	38	29	34	25	44
22	39	20	33	24	43	52	35

Es ist von einem entscheidenden Vorteil, wenn euer Programm schnell ist, das heißt die Lösung schnell berechnet (unter einer Minute für die Werte n=10 und k=10 auf einem Intel P4 Prozessor)!!!

Eingabe und Ausgabe

⇨ Das Programm liest 2 Integerwerte von der Standardeingabe: in der 1. Zeile n (Zahl der Zeilen des Bretts) und in der 2. Zeile k (Zahl der Spalten) wobei gilt $2 \le n \le 10$ und $2 \le k \le 10$.

⇦ Das Programm schreibt in die Standardausgabe „kein Weg gefunden“ wenn es keine Lösung gibt (z.B. wenn n und k beide ungerade sind). Falls das Programm eine Lösung findet schreibt es diese in die Standardausgabe in der Form der beiden Beispiele. Es genügt eine einzige Lösung.

Das Programm soll die Lösung neu berechnen und nicht etwa aus einer vorbereiteten Tabelle lesen!

Das Programm soll unter dem Namen **SPRUNG.xxx** abgespeichert werden (wobei xxx=PAS, xxx=C oder xxx=CPP). Zuzüglich wird ein ausführbares Programm **SPRUNG.EXE**.

Zusätzlich dazu wird eine Texdatei **SPRUNG.TXT** verlangt. Diese Datei soll (in der Form des Beispiels) je eine fertige Lösung für folgende Werte enthalten:

a) n=4; k=6 **b)** n=4; k=8 **c)** n=6; k=6 **d)** n=10; k=10

Problemstellung II - Die hungrige Schlange

Schreibe ein Programm welches eine Variante des Spiels "Snake" spielt, welches auf vielen Mobiltelefonen installiert ist.

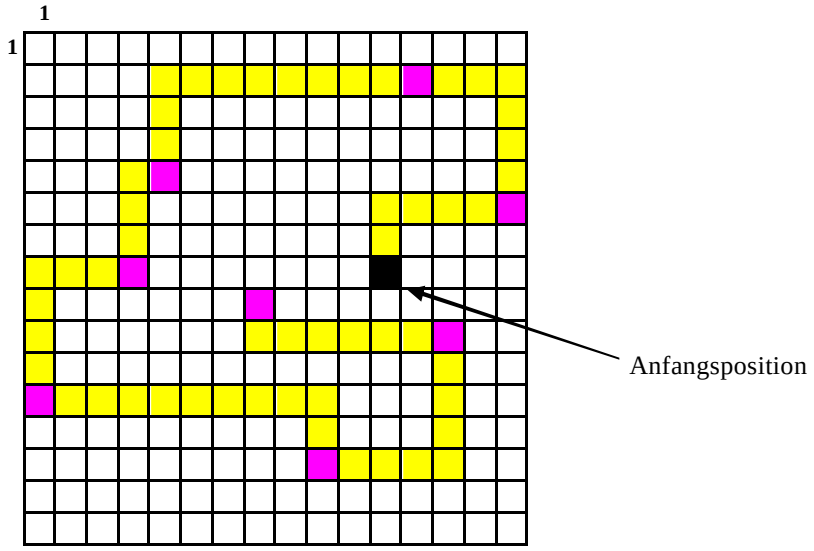
Spielerklärungen

Eine hungrige Schlange auf Nahrungssuche schleicht in einem Planquadrat mit den variablen Dimensionen N×N umher. Die Koordinaten (1; 1) befinden sich oben links. Das Schicksal will, dass der Schwanz der Schlange immer an der Anfangsposition befestigt bleibt, die Länge der Schlange also bei jeder Fortbewegung vergrößert wird. Der Kopf der Schlange kann sich innerhalb des Planquadrats in die 4 Richtungen fortbewegen. Eine diagonale Fortbewegung ist ausgeschlossen.

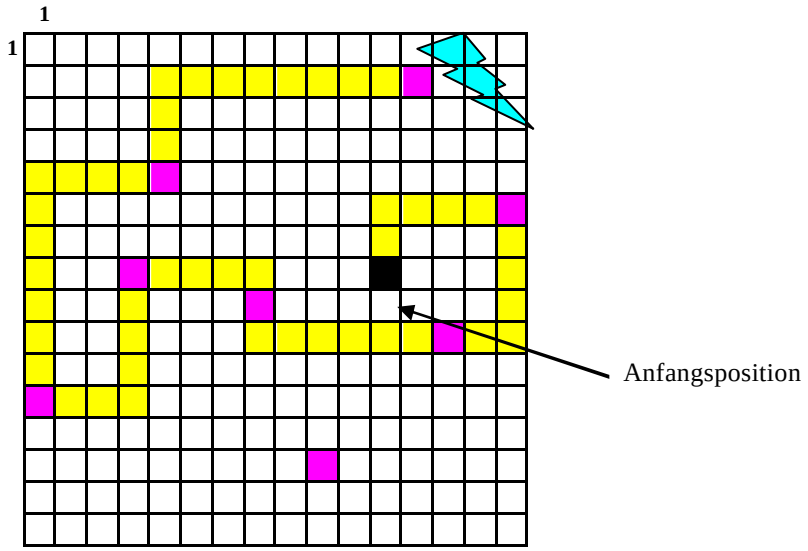
Im Planquadrat sind genau INT(N/2) Kaninchen in verschiedenen Positionen verteilt. Das Ziel des Spiels ist es einen Weg zu finden, auf welchem die Schlange alle Kaninchen fressen kann, ohne dass sie das Planquadrat verlässt und ohne dass sie sich selbst den Weg kreuzt oder zweimal den selben Weg nimmt.

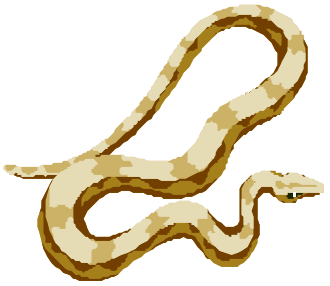
Die Anfangsposition der Schlange sowie die Positionen der Kaninchen sind von einem Spiel zum andern variabel.

Beispiel einer möglichen Fortbewegung in einem Planquadrat 16´16



Beispiel einer Fortbewegung mit Sperre (das Kreuzen ist nicht erlaubt)





Ein- und Ausgabe des Programms

⇨ Das Programm liest die Eingabedatei **SNAKE.IN** welches die Anfangsposition der Schlange sowie die verschiedenen Positionen der Kaninchen angibt. Die Textdatei hat folgendes Format (Beispiel für N=16):

```
. . . . . . . . . . . . . . K . .
. . . . . . . . . . . . . . . . .
. . . . K . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . A . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
K . . . . . . K . . . . . . . . K
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . K . . . . .
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . . K . . . . . . . . . . . K
```

. = A für die Anfangsposition
. = K für die Position eines Kaninchens

⇦ Das Programm erstellt die Ausgabedatei **SNAKE.OUT** welches den Weg der Schlange darstellt. Die Textdatei hat folgendes Format (Beispiel für N=16):

```
. . . . . . . . . . . . . . K x x
. . . . . . . . . . . . . . x . x
. . . . K x x x x x x x x x x . x
. . . . . . . . . . . . . . . . x
. . . . . . . . . . . . . . . . x
. . . . . . . . . . . . . . . . x
x x x A . . . . . . . . . . . . x
x . . . . . . . . . . . . . . . x
K x x x x x x K . . . x x x x K
. . . . x x x x . . . x . . . .
. . . . x . . . . . . x . . . .
. . . . x . . . . . . K x x x x
. . . . x . . . . . . . . . . x
. . . . x . . . . . . . . . . x
. . . . x . . . . . . . . . . x
. . . . K x x x x x x x x x x K
```

. = A für die Anfangsposition
. = K für die Position eines Kaninchens
. = x für den Weg

Bemerkungen

- Das Programm muss nicht den kürzesten Weg im Planquadrat finden.
- $4 \leq N \leq 100$

Das Programm soll unter dem Namen **SNAKE.xxx** abgespeichert werden (wobei xxx=PAS, xxx=C oder xxx=CPP). Zuzüglich wird ein ausführbares Programm **SNAKE.EXE** verlangt. Speichert ebenfalls eine Einabedatei eurer Wahl ab um das Programm zu testen.