Canadian
Mathematics
Competition

An activity of The Centre for Education
in Mathematics and Computing,
University of Waterloo, Waterloo, Ontario

Canadian Computing Competition

for the SYBASE® **Awards**

Tuesday, March 2, 1999

Problem 1
Card Game

Input file : card.in Output file : card.out

Write a program that will keep score for a simple two-player game, played with a deck of cards. There are 52 cards in the deck; four of each of 13 possible names: *two*, *three*, *four*, *five*, *six*, *seven*, *eight*, *nine*, *ten*, *jack*, *queen*, *king*, *ace*. The cards labelled *jack*, *queen*, *king*, *ace* are collectively known *as high cards*.

The deck is shuffled and placed face-down on the table. Player A turns over the top card and places it on a pile; then player B turns over the top card and places it on the pile. A and B alternate until the deck is exhausted. The game is scored as follows:

- if a player turns over an ace, with at least 4 cards remain to be turned over, and none of the next 4 cards is a high card, that player scores 4 points
- if a player turns over a king, with at least 3 cards remain to be turned over, and none of the next 3 cards is a high card, that player scores 3 points
- if a player turns over a queen, with at least 2 cards remain to be turned over, and none of the next 2 cards is a high card, that player scores 2 points
- if a player turns over a jack, with at least 1 card remains to be turned over, and the next card is not a high card, that player scores 1 point

The input file will contain 52 lines. Each line will contain the name of a card in the deck, in lower case letters. The first line denotes the first card to be turned over; the next line the next card; and so on. Whenever a player scores, print the line

```
Player X scores n point(s).
```

where *X* is the name of the player (A or B) and *n* is the number of points scored (1, 2, 3, or 4). At the end of the game, print the total score for each player, on two lines:

```
Player A: n point(s).
Player B: m point(s).
```

Sample Input

```
three
seven
queen
eight
five
ten
king
eight
jack
queen
six
queen
jack
eight
seven
three
```

```
ten
four
king
nine
six
seven
ace
four
jack
ace
ten
nine
ten
queen
ace
king
seven
two
five
two
five
nine
three
king
six
eight
jack
six
five
four
two
ace
four
three
two
nine
```

Output for Sample Input

```
Player A scores 2 point(s).
Player A scores 1 point(s).
Player A scores 3 point(s).
Player B scores 3 point(s).
Player A scores 1 point(s).
Player B scores 4 point(s).
Player A: 7 point(s).
Player B: 7 point(s).
```

Problem 2
Year 2000

Input file : y2k.in Output file : y2k.out

OK, you knew there had to be a Y2K problem, so here it is.

You are given a document containing text and numerical data, which may include dates. Your task is to identify (two-digit) years and to reprint the document with these two-digit years replaced by four-digit years. You may assume that any year numbered 24 or less is in the 2000's, while any year numbered 25 or more is in the 1900's (e.g. 16 represents the year 2016 and 26 represents the year 1926*). Yes, we know this rule may imply that your grandmother hasn't been born yet.*

Your program is to recognize dates in any of three formats:

```
dd/mm/yy
yy.mm.dd
Month,dd,yy
```

where *dd* is a two-digit day between 01 and 31, *mm* is a two-digit month between 01 and 12, *yy* is a two-digit year between 00 and 99, and *Month* is one of: January, February, March, April, May, June, July, August, September, October, November, December. The first two formats contain no spaces, and the third format contains a single space after *Month* and a single space after the comma.

Dates should appear on a single line. Dates traversing two lines or dates immediately adjacent to a letter of the alphabet or a digit should not be recognized. Non-existent data such as February 30, 99 do not need to be checked for.

The first line of input to your program will contain a positive integer *n*, indicating the number of lines of text to follow. In each of the lines of text you are to find all dates that occur in any of the formats above, and replace the two-digit year by the appropriate four-digit year as described above.

Sample Input

```
4
Before 02/03/04, but not after December 19, 99,
there was a rehash of the 55.34.02 meeting.  A date, like November 15,
95 cannot traverse two lines, nor can it be surrounded by alphabetics
or numerics like this:  78November 01, 88, or 6801/12/03, or 02/03/04x.
```

Output for Sample Input

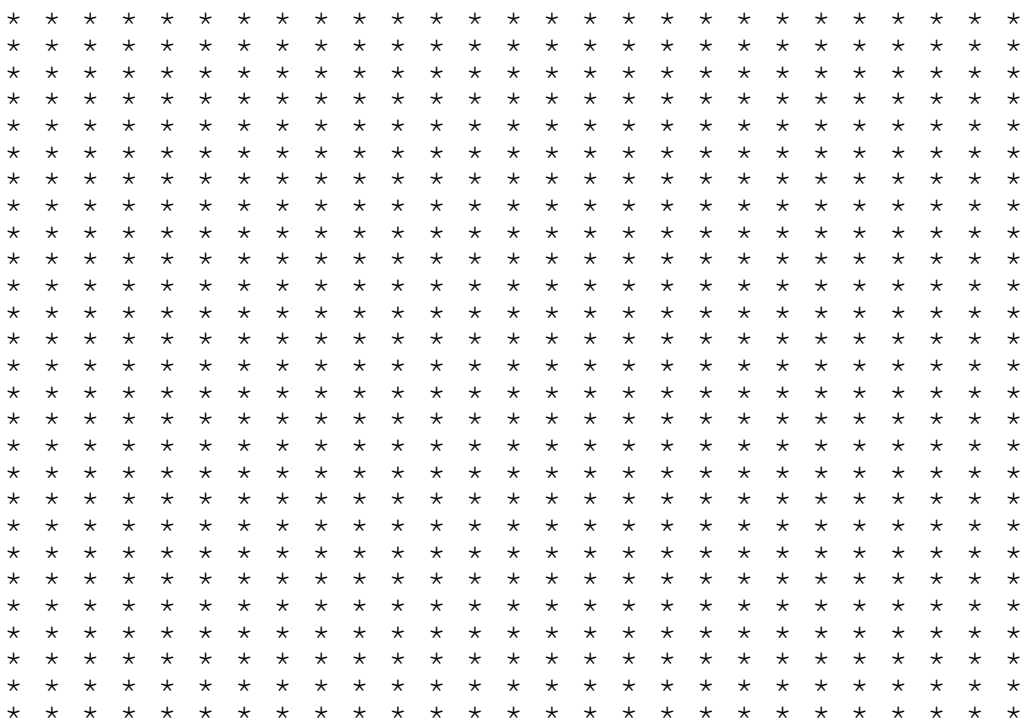
```
Before 02/03/2004, but not after December 19, 1999,
there was a rehash of the 55.34.02 meeting.  A date, like November 15,
95 cannot traverse two lines, nor can it be surrounded by alphabetics
or numerics like this:  78November 01, 88, or 6801/12/03, or 02/03/04x.
```

Problem 3
Divided Fractals

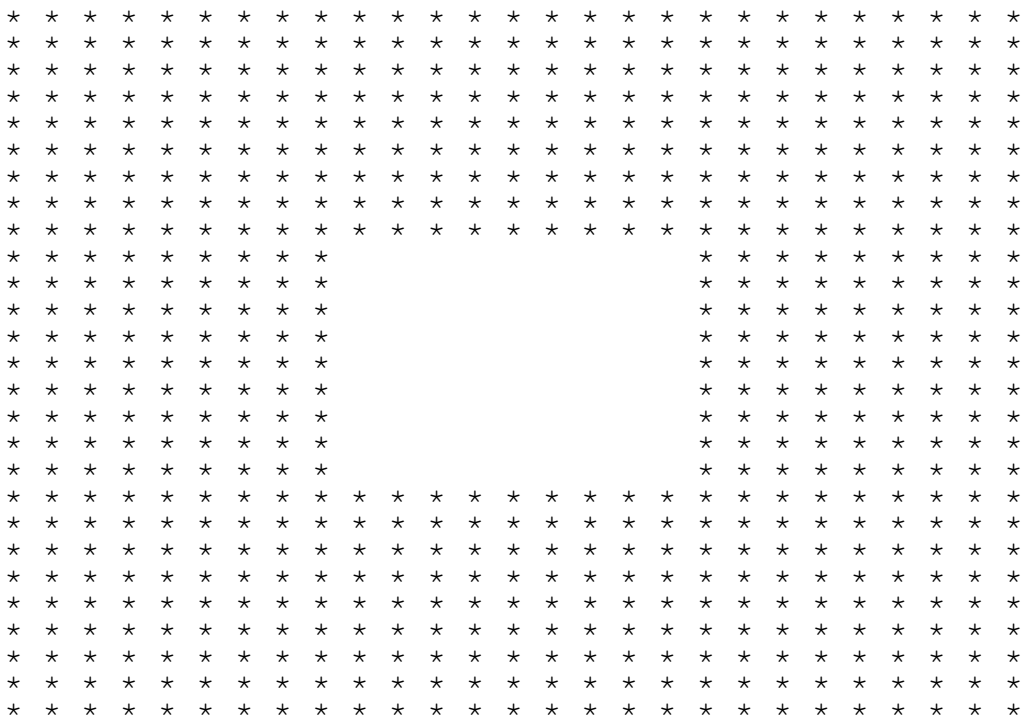
Input file : frac.in Output file : frac.out

A fractal is a geometrical object with the property that subsections of the object appear identical to (but smaller than) the whole object. Here we consider a specific fractal, which we will approximate by iterating a construction process.

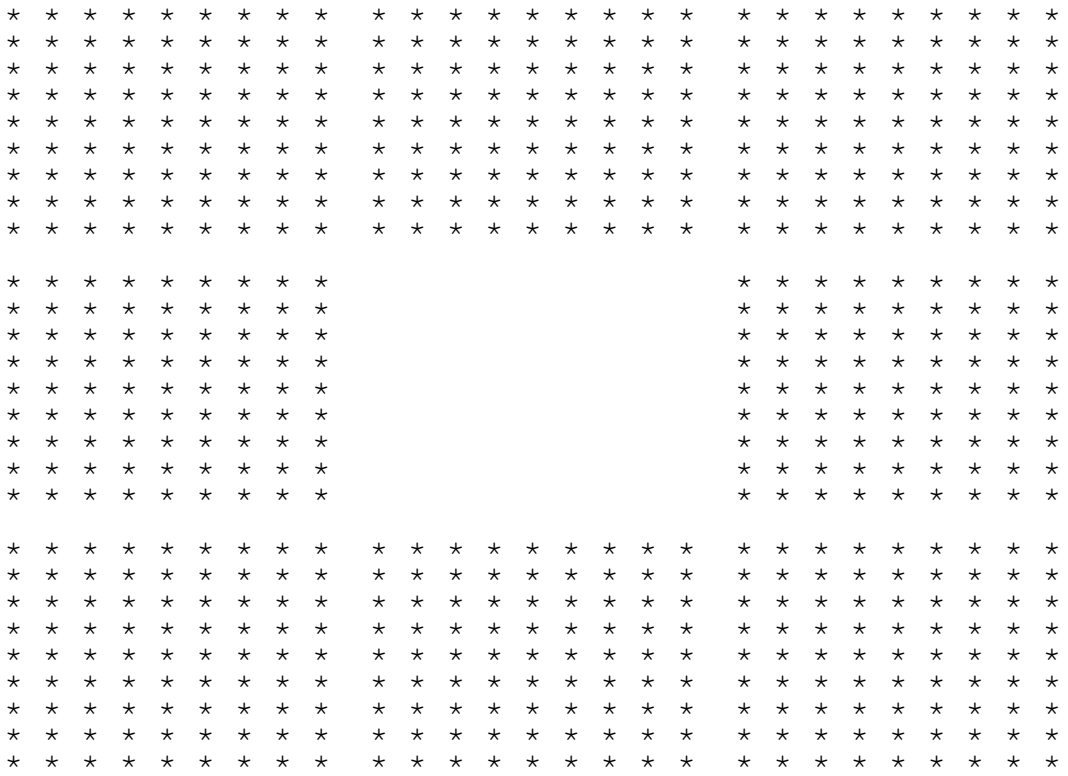
Start with a filled square whose side length is 1. For example:



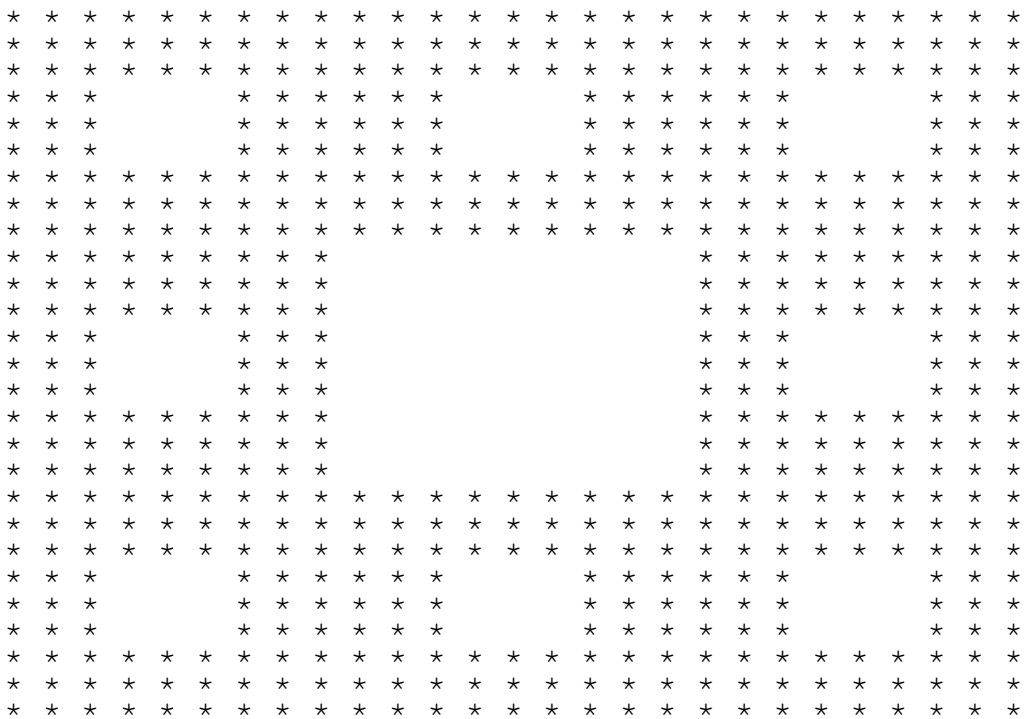
Remove a square of side 1/3 from the middle:



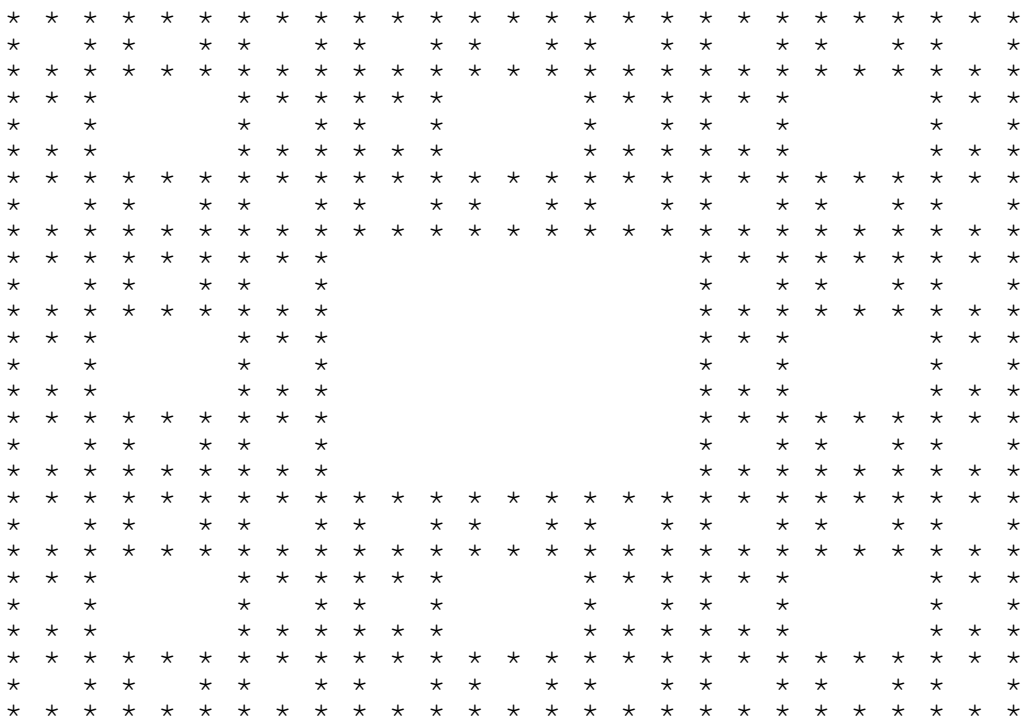
Note that this figure is equivalent to 8 squares of size 1/3, as illustrated below. *The spaces between squares are for illustration only and do not appear in the fractal.*



We can apply this process again to each of the squares. Thus after 2 iterations of the construction process, we have:



Each of the eight squares is now a copy of the first iteration. Each of these contains eight filled squares, for a total of 64. The process is applied to each of these squares. After 3 iterations we have:



The actual fractal is what we get when this process is iterated infinitely many times. As one might expect, each of the 8 subsections of this fractal is an exact copy of the fractal, scaled by a factor of $\frac{1}{3}$.

Question 3(a)

If the process is iterated n times $(n \geq 1)$, how many holes are there in the square?

Question 3(b)

After n iterations, what is the total filled area?

Question 3(c)

After an infinite number of iterations, what is the total filled area?

Question 3(d)

Write a program to compute the specified function after n iterations$(n \leq 5)$. To do this, represent the figure as a 3^n by 3^n grid, with '*' to indicate filled areas and ' ' to indicate unfilled areas. The figure will be too large to print on a single screen or sheet of paper so the input will specify a small rectangular portion of the figure to be printed.

The first line of input contains a positive integer *d*, indicating the number of test data sets to follow. Each data set consists of five lines containing:

- *n*, the number of iterations $(0 \leq n \leq 5)$
- *b*, the bottom row of the rectangle to be printed $\left(1 \leq b \leq 3^n\right)$
- *t*, the top row of the rectangle to be printed $\left(b \leq t \leq 3^n\right)$
- *l*, the left column of the rectangle to be printed $\left(1 \leq l \leq 3^n\right)$
- *r*, the right column of the rectangle to be printed $\left(l \leq r \leq 3^n\right)$

Note that rows are numbered from bottom to top and columns from left to right.

Compute the figure and print the specified rectangular portion, with one line of output per row. Print the top row first, and the bottom row last. To make the output appear square, leave a single horizontal space between row elements (as is done in the examples above). Leave a blank line in the output after every test case.

Sample Input

```
1
3
2
10
5
27
```

Output for Sample Input

```
* * * * *                         * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * *
  * *   * *   * *   * *   * *   * *   * *   * *
* * * * * * * * * * * * * * * * * * * * * * * * *
    * * * * * *       * * * * * *       * * * * *
    *   * *   *       *   * *   *       *   * *   *
    * * * * * *       * * * * * *       * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * *
  * *   * *   * *   * *   * *   * *   * *   * *
```

Problem 4
A Knightly Pursuit

Input file : knight.in Output file : knight.out

In chess, game pieces move about an 8×8 chessboard in a fashion defined by their type. The object of the game is to capture opposing pieces by landing on their squares, and eventually trapping the king piece.

In our version of the game, we shall use a variable sized board with only 2 pieces on it: A white pawn which moves relentlessly towards the top row of the chessboard one square at a time per move; and a black knight which can move from its current location in any of up to eight ways: two squares up or down *and* one square left or right, or one square up or down *and* two squares left or right. The knight must remain on the board at all times; any move that would take it off the board is therefore disallowed. In the diagram below, the knight's position is labelled K and its possible moves are labelled 1 to 8.

```
. . . . . . .
. . 8 . 1 . .
. 7 . . . 2 .
. . . K . . .
. 6 . . . 3 .
. . 5 . 4 . .
. . . . . . .
```

The pawn moves first; then the knight and pawn alternate moves. The knight tries to land either on the square occupied by the pawn (a *win*) or on the square immediately above the pawn (a *stalemate*). If the pawn reaches the top row of the board the game ends immediately and the knight loses (a *loss*).

Write a program to determine whether the knight can win and, if it can, the minimum number of moves it must make to do so. If the knight cannot win, your program should determine if it can cause a stalemate and, if it can, the minimum number of moves it must make to do so. Finally if the knight cannot win or cause a stalemate, your program should compute the number of moves the knight makes before the pawn wins.

The first line of input contains a positive integer, *n*, the number of games to analyze. For each game there are six lines on input:

- *r*, the number of rows in the chessboard $(3 \le r < 100)$
- *c*, the number of columns in the chessboard $(2 \le c < 100)$
- *pr*, the row of the starting position of the pawn $(1 \le pr \le r)$
- *pc*, the column of the starting position of the pawn $(1 \le pc \le c)$
- *kr*, the row of the starting position of the knight $(1 \le kr \le r)$
- *kc*, the column of the starting position of the knight $(1 \le kc \le c)$

The pawn and the knight will have different starting positions. Row 1 is at the bottom of the board and Row *r* is at the top of the board. Column 1 is at the left and column *c* is at the right.

Sample Input

```
3
99
99
33
33
33
35
3
3
1
1
2
3
99
99
96
23
99
1
```

Output for Sample Input

```
Win in 1 knight move(s).
Stalemate in 1 knight move(s).
Loss in 2 knight move(s).
```

Problem 5
Letter Arithmetic

Input file : letter.in Output file : letter.out

A popular form of pencil game is to use letters to represent digits in a mathematical statement. An example is

```
 SEND
+MORE
-----
MONEY
```

which represents

```
 9567
+1085
-----
10652
```

Your task is to read in sets of three "words" and assign unique digits to the letters in such a way as to make the sum of the first two words equal to the third word.

The input file begins with a line containing a positive integer *n* which is the number of test data sets contained in the file. Each data set consists of three lines, each of which contains one word with the third word being the sum of the first two. The words will contain no more than 20 upper case letters.

The output file is to consist of *n* sets of lines each containing the numeric representation of each word in the corresponding test data set. There will be exactly one correct solution for each data set. Leave an empty line after the output for each data set.

Sample Input

```
2
SEND
MORE
MONEY
MEND
COPE
CONEY
```

Output for Sample Input

```
9567
1085
10652

9567
1085
10652
```