1994 YSU-ACM High School Programming Contest

Problem #1 Key to Success

Pascal Source File: P1.PAS **BASIC Source File: B1.BAS** **Data file: D1.DAT**

Description: Any one-to-one mapping, f, of any alphabet to itself can be used to encode text by replacing each occurrence of any letter, c, with f(c). One such mapping could be the mapping of a letter to three positions beyond the letter in the alphabet. That is, a→d, b→e, c→f, d→g and so on. With this mapping, "The car is blue." will be encoded as "Wkh fdu lv eoxh."

Write a program that decodes the contents of its input file according to the following guidelines:

1. Upper case letters are mapped to upper case letters and lower case letters are mapped to lower case letters (A → D, Z → C, a → d, z → c).
2. The mapping that defines the encoding is one-to-one. That is, two different letters never map to the same letter of the alphabet (a→x and t→x is impossible).
3. The program will loop through the input until a line with a "0" offset is reached printing the output to the screen.
4. The first character of the encoded test will be a digit from 0 to 9 representing the offset by which the text is mapped.
5. The string of text will be no longer than 200 characters.
6. Non alphabetic characters are to be printed without encoding.

Input: A sequence of encoded strings. Input is terminated by a line beginning with "0" Do not print the message associated with the terminating line..

Output: Output from your program will be the decoded strings separated by at least one blank line.

Example: Input:

```
3Whvw gdwd lv DEF ABC def abc
9Rc'b j carlth xwn, cx bqxf qxf rc fxatb!
7Zabmm av zll ovd pa dvyrz HIJ EFG hij efg
0the rest of this line does not matter
```

Output:

```
Test data is ABC XYZ abc xyz

It's a tricky one, to show how it works!

Stuff to see how it works ABC XYZ abc xyz
```

1994 YSU-ACM High School Programming Contest

Problem #2 Overlapping Rectangles

Pascal Source File: P2.PAS BASIC Source File: B2.BAS Data file: D2.DAT

Description: While managing a collection of rectangular windows a window manager must determine whether two windows overlap, and if so, where on the screen the overlapping region lies.

Write a program to perform this function. Your program will accept as input the coordinates of two rectangular windows. If the windows do not overlap, your program should produce a message to that effect. If they do overlap, you should compute the coordinates of the overlapping region (which must itself be a rectangle).

All coordinates are expressed in "pixel numbers", integer values ranging from 0 to 9999 (with upper left-hand corner being (0,0) and the lower right-hand corner being (9999,9999)). A rectangle will be described by two pairs of (X,Y) coordinates. The first pair gives the coordinates of the upper left-hand corner (X_{UL},Y_{UL}). The second pair gives the coordinates of the lower right-hand coordinates (X_{LR}, Y_{LR}). You are guaranteed that $X_{UL} > X_{LR}$ and $Y_{UL} > Y_{LR}$.

Input: Input for your program consists of sets of eight lines. The first four contain the pixel numbers X_{UL}, Y_{UL}, X_{LR} and Y_{LR} for the first window. The second four contain the pixel numbers for the second window. Input is terminated by a four zeros.

Output: Output for your program will be to the screen. If the two windows do not overlap, print the message "No Overlap". If the two windows do overlap, print the 4 pixel numbers giving the X_{UL}, Y_{UL}, X_{LR} and Y_{UR} for the region of overlap. These values should be written in order pair notation (*i.e.* (X_{UL}, Y_{UL}), (X_{LR}, Y_{LR})).

Note that two windows that share a common edge but have no other points in common are considered to have "No Overlap".

Example:

```
Input:
  0
  20
  100
  120
  80
  0
  500
  60
  0
  0
  0
  0
```

```
Output:
  (80, 20) (100, 60)
```

1994 YSU-ACM High School Programming Contest

Problem #3 Word Search

Pascal Source File: P3.PAS **BASIC Source File: B3.BAS** **Data file: D3.DAT**

Description: A word search puzzle.

Input: Input for your program is as follows. The first line of input will contain the number of puzzles to be solved. Subsequent lines will be puzzle descriptions for each puzzle. The first two lines of each puzzle description will be the width w and height h of the puzzle to follow ($1 \leq w \leq 80$ and $1 \leq h \leq 80$). The next h lines of w characters will be the puzzle itself. Following the puzzle will be the number of words to search for n. The next n lines will contain the words to be searched for. No word will be longer then 20 characters.

Output: Output for each puzzle begins with an appropriately labeled number. Print the puzzle itself (each puzzle letter separated by a space). Print the word to be searched for, and its starting and ending coordinates. (The coordinate (1,1) represents the top left). Separate output from different puzzle records by at least one blank line. If a word appears more than once, only print the first one found. Words may appear, horizontally, vertically, forward, backward and diagonal.

Example:

```
Input:
  2
  6
  6
  ABCDEF
  GHIJKL
  MEOPQR
  SLUSYX
  YL1234
  5O7890
  1
  HELLO
  7
  7
  ABCDEFG
  KDICJEN
  AKEWNQM
  OAYTUEI
  ENAMAJD
  ZZCADWU
  EKSIAPI
  3
  AAAAA
  BYEBYE
  BORLAND
```

```
Output:
  Puzzle #1
    A B C D E F
    G H I J K L
    M E O P Q R
    S L U S Y X
    Y L 1 2 3 4
    5 O 7 8 9 0
   HELLO (2,2)->(2,6)


  Puzzle #2
    E Y B E Y B D
    K D I C J E N
    A K E W N Q A
    O A Y T U E L
    E N A M A J R
    Z Z C A D W O
    E K S I A P B
   AAAAA (1,3)->(5,7)
   BYEBYE (6,1)->(1,1)
   BORLAND (7,7)->(7,1)
```

1994 YSU-ACM High School Programming Contest

Problem #4 Pyramids

Pascal Source File: P4.PAS **BASIC Source File: B4.BAS** **Data file: D4.DAT**

Description: Display a solid triangle.

Input: A sequence of positive, odd integers n, one per line, terminated by 0.

Output: A solid triangle. An * should be used to denote the triangle. Separate output from different triangles by at least one blank line.

Example:

```
Input:
  5
  9
  0
```

```
Output:
    *
   ***
  *****

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

1994 YSU-ACM High School Programming Contest

Problem #5 Phone Numbers

Pascal Source File: P5.PAS **BASIC Source File: B5.BAS** **Data file: D5.DAT**

Description: Advertising agencies recommend that companies use phone numbers that are easy to remember. One way of accomplishing this is to use the alphabetic translations of numbers on a phone pad to spell out words.

Write a program that determines all possible alphabetic translations for a given phone number. If a non-translatable character appears in the input string it should be passed to the output string as a constant.

Input: A sequence of 7 digit strings, one per line. Input is terminated by an input string of all zeros.

Output: Output all character string, one per line, that can be generated by the given input string. Separate output from different input strings by at least one blank line.

Example:

```
Input:
  borla63
  0000000
```

```
Output:
  borlamd
  borlame
  borlamf
  borland
  borlane
  borlanf
  borlaod
  borlaoe
  borlaof
```

1994 YSU-ACM High School Programming Contest

Problem #6 Lucky Number

Pascal Source File: P6.PAS **BASIC Source File: B6.BAS** **Data file: D6.DAT**

Description: Determine the sum of the bottom faces of n dice given the values of the top faces.

Input: The first line will be n, the number of dice rolled, followed by those n values. This makes one set of die rolls which will be used to generate one solution. Input is terminated by $n = 0$.

Output: The solution number followed by the sum of the bottom faces of dice. Separate output from different sets by at least one blank line.

Notes: The sum of opposing faces on a die is a constant.

Example:

```
Input:                Output:
  3                    Solution 1
  6                    12
  2
  1
  4                    Solution 2
  1                    17
  2
  5
  3
  0
```

1994 YSU-ACM High School Programming Contest

Problem #7 Rabbit Populations

Pascal Source File: P7.PAS **BASIC Source File: B7.BAS** **Data file: D7.DAT**

Description: It is possible to predict the population of a group of rabbits based on a second order recurrence relation. The most popular of these is the Fibonacci sequence. Given two starting populations, one can compute the population of rabbits for a set number of generations resulting from the original populations.

Let $g(x)$ be a function of x which returns the population of rabbits for a generation x. Then,

$$g(x) = F_{x+1}$$

Here F_{x+1} is the (x+1) number in the sequence defined by

$$F_y = F_{y-1} + F_{y-2}$$

given initial values for F_0 and F_1.

Input: Input items are grouped in three-tuples, each of which pertain to one prediction of rabbit population for a given generation. The first number will represent F_0, the second number will represent F_1, and the third number will represent the generation, x, whose population must be computed based on the two starting populations F_0 and F_1. Input will be terminated when a three-tuple is read in consisting of all 0's.

Output: Output should consist of the number of rabbits (population) of the generation x.

Notes: A Fibonnaci sequence is a special case where $F_1 = F_0 = 1$.

Therefore,

$$\begin{aligned} F_2 &= F_1 + F_0 \\ F_2 &= 1 + 1 \\ F_2 &= 2 \end{aligned}$$

and

$$\begin{aligned} F_3 &= F_2 + F_1 \\ F_3 &= 2 + 1 \\ F_3 &= 3 \end{aligned}$$

After that, we can list the sequence as:

$$1\ 1\ 2\ 3\ 5\ 8\ 13\ 21\ 34\ 55\ \cdots$$

With different seed numbers of F_0 and F_1, obviously, a different sequence is generated.

Example:

```
Input:
  3
  5
  3
 22
 29
  4
  0
  0
  0
```

```
Output:
  Population:  21

  Population: 211
```