INF01056
Strings

Prof. João Comba

Baseado no Livro Programming Challenges

Strings: Character Codes

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	”	35	#	36	$	37	%	38	&	39	’
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	/	93	]	94	ˆ	95	_
96	‘	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	—	125	}	126	∼	127	DEL

- ASCII-TABLE

Strings: Representation

Null-terminated Arrays — C/C++ treats strings as arrays of characters. The string ends the instant it hits the null character "\0", i.e., zero ASCII. Failing to end your string explicitly with a null typically extends it by a bunch of unprintable characters. In defining a string, enough array must be allocated to hold the largest possible string (plus the null) unless you want a core dump. The advantage of this array representation is that all individual characters are accessible by index as array elements.

Array Plus Length — Another scheme uses the first array location to store the length of the string, thus avoiding the need for any terminating null character. Presumably this is what Java implementations do internally, even though the user's view of strings is as objects with a set of operators and methods acting on them.

Linked Lists of Characters — Text strings can be represented using linked lists, but this is typically avoided because of the high space-overhead associated with having a several-byte pointer for each single byte character. Still, such a representation might be useful if you were to insert or delete substrings frequently within the body of a string.

Strings: example

Corporate name changes are occurring with ever greater frequency, as companies merge, buy each other out, try to hide from bad publicity, or even raise their stock price – remember when adding a .com to a company's name was the secret to success!

These changes make it difficult to figure out the current name of a company when reading old documents. Your company, Digiscam (formerly Algorist Technologies), has put you to work on a program which maintains a database of corporate name changes and does the appropriate substitutions to bring old documents up to date.

Your program should take as input a file with a given number of corporate name changes, followed by a given number of lines of text for you to correct. Only *exact* matches of the string should be replaced. There will be at most 100 corporate changes, and each line of text is at most 1,000 characters long. A sample input is —

Strings: example

```
4
"Anderson Consulting" to "Accenture"
"Enron" to "Dynegy"
"DEC" to "Compaq"
"TWA" to "American"
5
Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Which should be transformed to —

```
Anderson Accounting begat Accenture, which
offered advice to Dynegy before it CompaqLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Solution

```c
#include <stdio.h>
#include <string.h>

#define  MAXLEN   1001  /* longest possible string */
#define MAXCHANGES 101/* maximum number of name changes */

typedef char string[MAXLEN];

string mergers[MAXCHANGES][2]; /* store before/after corporate names */
int nmergers;         /* the number of different name changes */

read_changes() {
   int i;/* counter */

   scanf("%d\n",&nmergers);
   for (i=0; i<nmergers; i++) {
      read_quoted_string(&(mergers[i][0]));
      read_quoted_string(&(mergers[i][1]));
    }
}

read_quoted_string(char *s) {
   int i=0;     /* counter */
   char c;         /* latest character */

   while ((c=getchar()) != '\"') ;
   while ((c=getchar()) != '\"') {
      s[i] = c;
      i = i+1;
   }
   s[i] = '\0';
}
```

Solution

```
main()
{
   string s;     /* input string */
   char c;       /* input character */
   int nlines;      /* number of lines in text */
   int i,j;      /* counters */
   int pos;      /* position of pattern in string */

   read_changes();
   scanf("%d\n",&nlines);

   for (i=1; i<=nlines; i=i+1) {
      /* read text line */
      j=0;
      while ((c=getchar()) != '\n') {
         s[j] = c;
         j = j+1;
      }
      s[j] = '\0';
      for (j=0; j<nmergers; j=j+1)
         while ((pos=findmatch(mergers[j][0],s)) != -1) {
            replace_x_with_y(s, pos,
               strlen(mergers[j][0]), mergers[j][1]);
         }
      printf("%s\n",s);
   }
}
```

Solution

```
int findmatch(char *p, char *t) {
   int i,j;        /* counters */
   int plen, tlen;         /* string lengths */

   plen = strlen(p);
   tlen = strlen(t);

   for (i=0; i<=(tlen-plen); i=i+1) {
      j=0;
      while ((j<plen) && (t[i+j]==p[j]))
         j = j+1;
      if (j == plen) return(i);
   }
   return(-1);
}

replace_x_with_y(char *s, int pos, int xlen, char *y) {
   int i;            /* counter */
   int slen,ylen;       /* lengths of relevant strings */

   slen = strlen(s);
   ylen = strlen(y);

   if (xlen >= ylen)
      for (i=(pos+xlen); i<=slen; i++) s[i+(ylen-xlen)] = s[i];
   else
      for (i=slen; i>=(pos+xlen); i--) s[i+(ylen-xlen)] = s[i];

   for (i=0; i<ylen; i++) s[pos+i] = y[i];
}
```

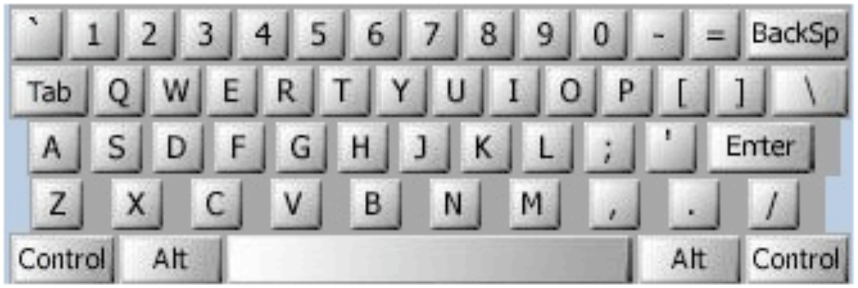
C String functions

```
#include <ctype.h>      /* include the character library */

int isalpha(int c);     /* true if c is either upper or lower case */
int isupper(int c);     /* true if c is upper case */
int islower(int c);     /* true if c is lower case */
int isdigit(int c);     /* true if c is a numerical digit (0-9) */
int ispunct(int c);     /* true if c is a punctuation symbol */
int isxdigit(int c);    /* true if c is a hexadecimal digit (0-9,A-F) */
int isprint(int c);     /* true if c is any printable character */

int toupper(int c);     /* convert c to upper case -- no error checking */
int tolower(int c);     /* convert c to lower case -- no error checking */


#include <string.h>      /* include the string library */

char *strcat(char *dst, const char *src);     /* concatenation */
int strcmp(const char *s1, const char *s2);   /* is s1 == s2? */
char *strcpy(char *dst, const char *src);     /* copy src to dist */
size_t strlen(const char *s);                 /* length of string */
char *strstr(const char *s1, const char *s2); /* search for s2 in s1 */
char *strtok(char *s1, const char *s2);       /* iterate words in s1 */
```

C++ String functions

```
string::size()             /* string length */
string::empty()            /* is it empty */

string::c_str()             /* return a pointer to a C style string */

string::operator [](size_type i)        /* access the ith character */

string::append(s)           /* append to string */
string::erase(n,m)          /* delete a run of characters */
string::insert(size_type n, const string&s) /* insert string s at n */

string::find(s)
string::rfind(s)            /* search left or right for the given string */

string::first()
string::last()              /* get characters, also there are iterators */
```

WERTYU


` 1 2 3 4 5 6 7 8 9 0 - = BackSp
Tab Q W E R T Y U I O P [ ] \
A S D F G H J K L ; ' Enter
Z X C V B N M , . /
Control Alt Alt Control

A common typing error is to place the hands on the keyboard one row to the right of the correct position. So "Q" is typed as "W" and "J" is typed as "K" and so on. You are to decode a message typed in this manner.

Input consists of several lines of text. Each line may contain digits, spaces, upper case letters (except Q, A, Z), or punctuation shown above [except back-quote (`)]. Keys labelled with words [*Tab, BackSp, Control,* etc.] are not represented in the input. You are to replace each letter or punction symbol by the one immediately to its left on the QWERTY keyboard shown above. Spaces in the input should be echoed in the output.

Sample Input

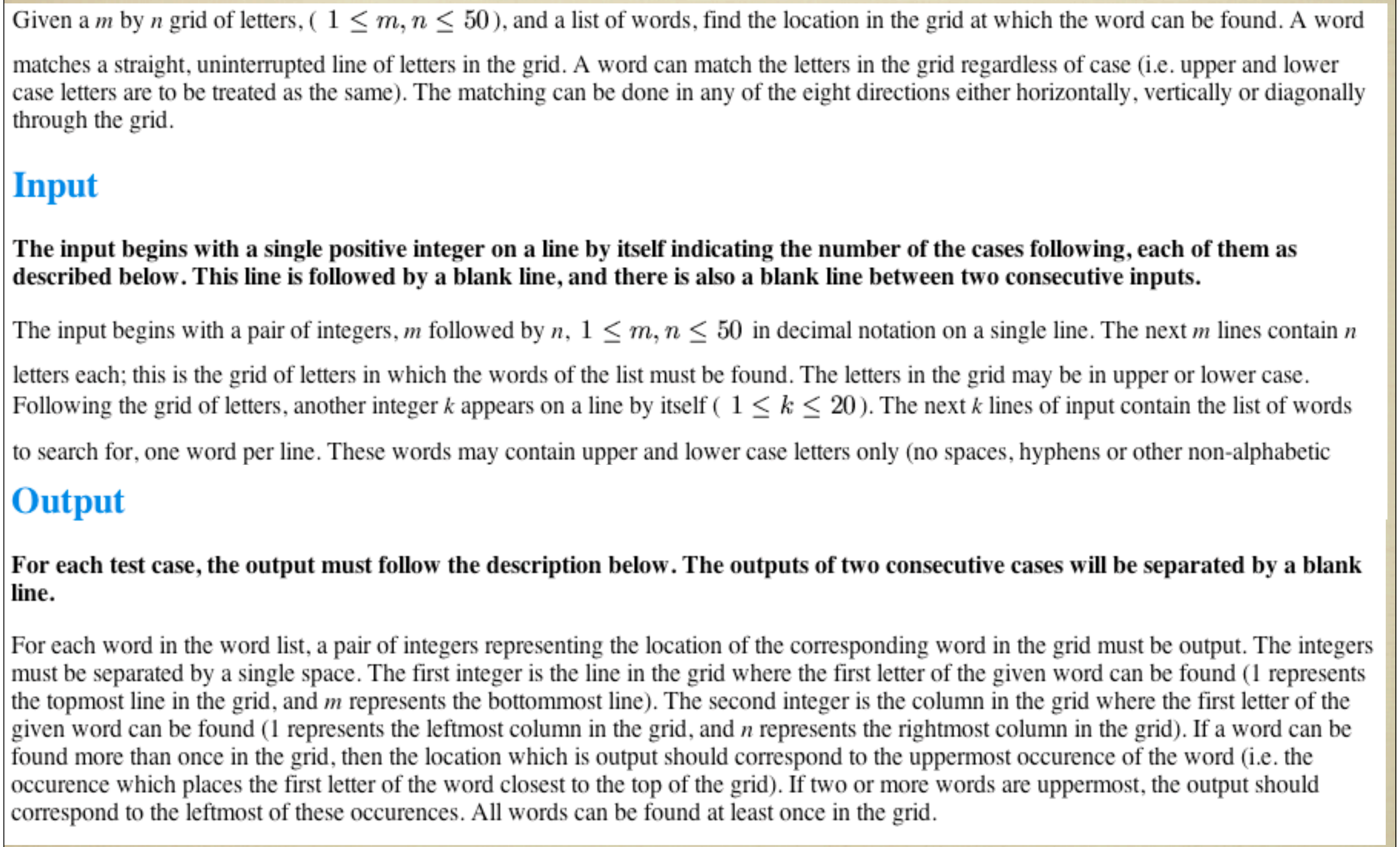
```
O S, GOMR YPFSU/
```

Output for Sample Input

```
I AM FINE TODAY.
```

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int
main() {
    vector<char> w;
    for(int i=0; i<128; i++)
        w.push_back((char)i);

    w['2'] = '1'; w['3'] = '2'; w['4'] = '3'; w['5'] = '4'; w['6'] = '5'; w['7'] = '6';
    w['8'] = '7'; w['9'] = '8'; w['0'] = '9'; w['-'] = '0'; w['='] = '-';

    w['W'] = 'Q'; w['E'] = 'W'; w['R'] = 'E'; w['T'] = 'R'; w['Y'] = 'T'; w['U'] = 'Y';
    w['I'] = 'U'; w['O'] = 'I'; w['P'] = 'O'; w['['] = 'P'; w[']'] = '[';w['\\'] = ']';

    w['S'] = 'A'; w['D'] = 'S'; w['F'] = 'D'; w['G'] = 'F'; w['H'] = 'G'; w['J'] = 'H';
    w['K'] = 'J'; w['L'] = 'K'; w[';'] = 'L'; w['\''] = ';';

    w['X'] = 'Z'; w['C'] = 'X'; w['V'] = 'C'; w['B'] = 'V'; w['N'] = 'B'; w['M'] = 'N';
    w[','] = 'M'; w['.'] = ','; w['/'] = '.';
    string line;
    while(getline(cin,line)){
        for(int i=0; i<line.size(); i++){
            cout << w[line[i]];
        }
        cout << endl;
    }
}
```

Where's Waldorf



Given a *m* by *n* grid of letters, ($1 \leq m, n \leq 50$), and a list of words, find the location in the grid at which the word can be found. A word matches a straight, uninterrupted line of letters in the grid. A word can match the letters in the grid regardless of case (i.e. upper and lower case letters are to be treated as the same). The matching can be done in any of the eight directions either horizontally, vertically or diagonally through the grid.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

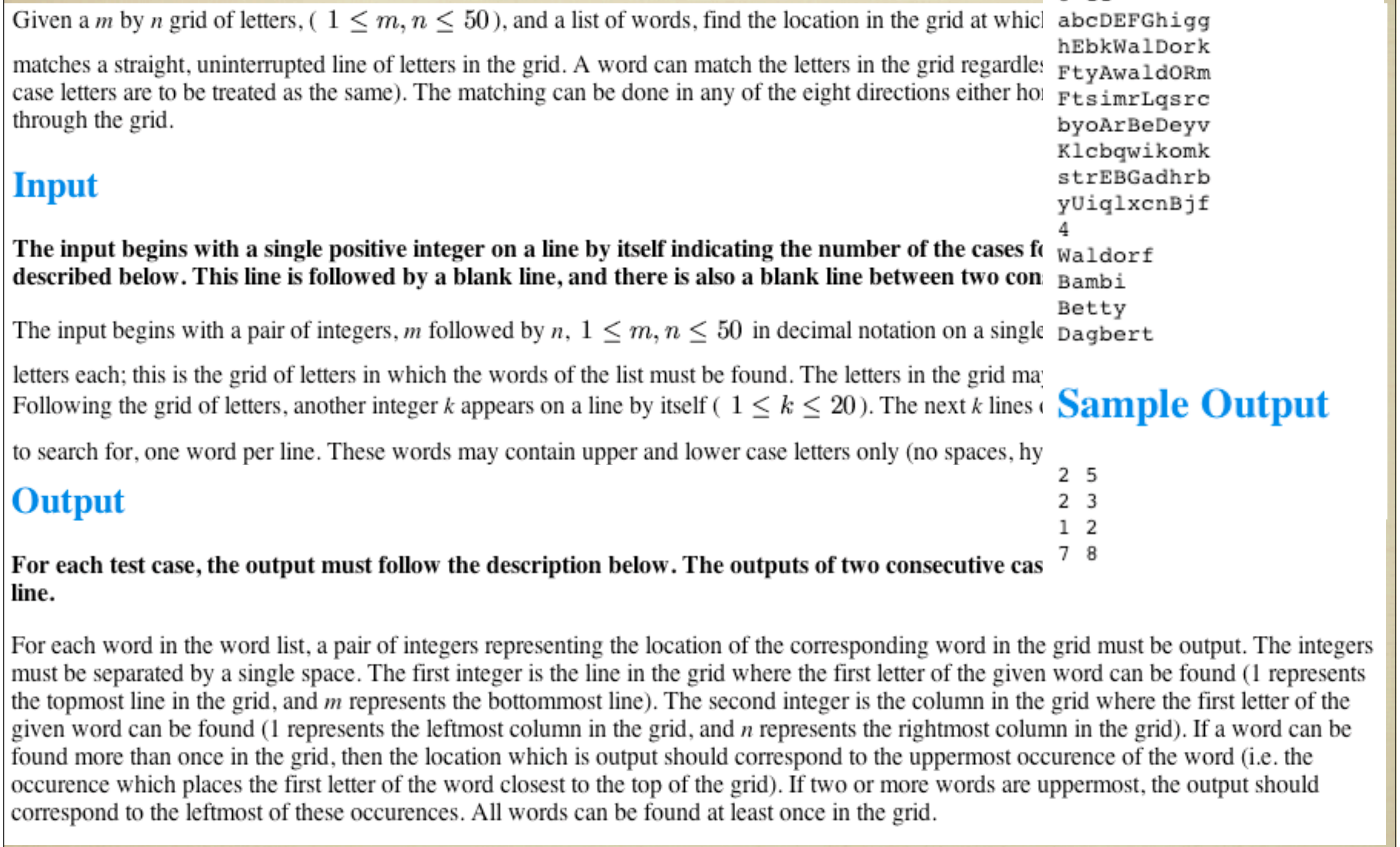
The input begins with a pair of integers, *m* followed by *n*, $1 \leq m, n \leq 50$ in decimal notation on a single line. The next *m* lines contain *n* letters each; this is the grid of letters in which the words of the list must be found. The letters in the grid may be in upper or lower case. Following the grid of letters, another integer *k* appears on a line by itself ($1 \leq k \leq 20$). The next *k* lines of input contain the list of words to search for, one word per line. These words may contain upper and lower case letters only (no spaces, hyphens or other non-alphabetic

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

For each word in the word list, a pair of integers representing the location of the corresponding word in the grid must be output. The integers must be separated by a single space. The first integer is the line in the grid where the first letter of the given word can be found (1 represents the topmost line in the grid, and *m* represents the bottommost line). The second integer is the column in the grid where the first letter of the given word can be found (1 represents the leftmost column in the grid, and *n* represents the rightmost column in the grid). If a word can be found more than once in the grid, then the location which is output should correspond to the uppermost occurence of the word (i.e. the occurence which places the first letter of the word closest to the top of the grid). If two or more words are uppermost, the output should correspond to the leftmost of these occurences. All words can be found at least once in the grid.

WHERE'S WALDORF



Given a m by n grid of letters, ($1 \leq m, n \leq 50$), and a list of words, find the location in the grid at whicl

matches a straight, uninterrupted line of letters in the grid. A word can match the letters in the grid regardle: case letters are to be treated as the same). The matching can be done in any of the eight directions either ho: through the grid.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases fc described below. This line is followed by a blank line, and there is also a blank line between two con

The input begins with a pair of integers, m followed by n, $1 \leq m, n \leq 50$ in decimal notation on a single

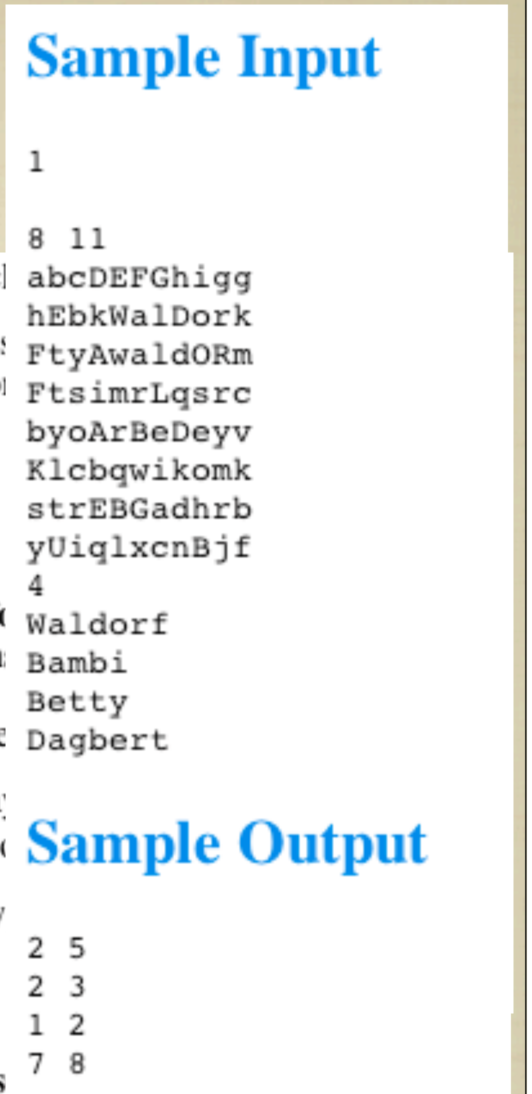
letters each; this is the grid of letters in which the words of the list must be found. The letters in the grid ma: Following the grid of letters, another integer k appears on a line by itself ($1 \leq k \leq 20$). The next k lines (

to search for, one word per line. These words may contain upper and lower case letters only (no spaces, hy

Output

For each test case, the output must follow the description below. The outputs of two consecutive cas line.

For each word in the word list, a pair of integers representing the location of the corresponding word in the grid must be output. The integers must be separated by a single space. The first integer is the line in the grid where the first letter of the given word can be found (1 represents the topmost line in the grid, and m represents the bottommost line). The second integer is the column in the grid where the first letter of the given word can be found (1 represents the leftmost column in the grid, and n represents the rightmost column in the grid). If a word can be found more than once in the grid, then the location which is output should correspond to the uppermost occurence of the word (i.e. the occurence which places the first letter of the word closest to the top of the grid). If two or more words are uppermost, the output should correspond to the leftmost of these occurences. All words can be found at least once in the grid.



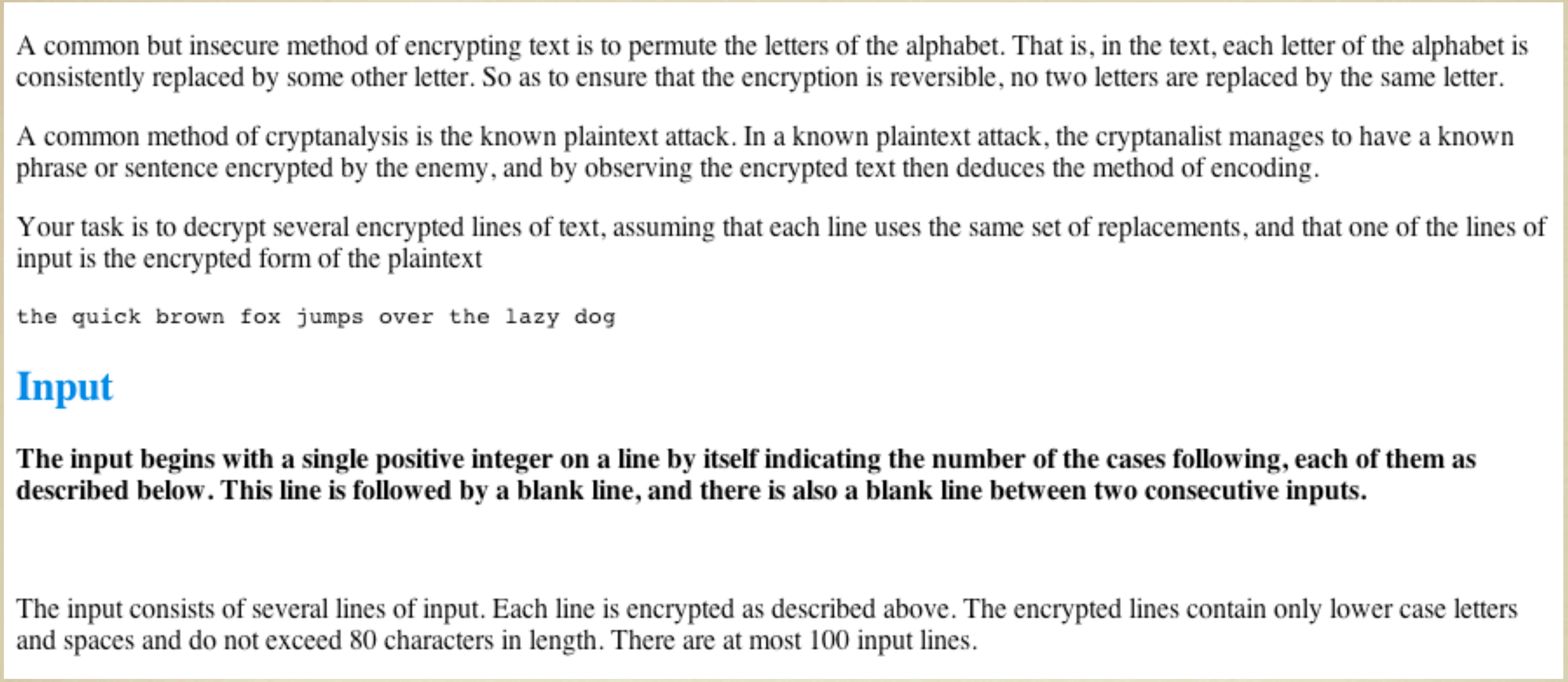
Sample Input

```
1

8 11
abcDEFGhigg
hEbkWalDork
FtyAwaldORm
FtsimrLgsrc
byoArBeDeyv
Klcbqwikomk
strEBGadhrb
yUiqlxcnBjf
4
Waldorf
Bambi
Betty
Dagbert
```

Sample Output

```
2 5
2 3
1 2
7 8
```

Crypt Cricker 2



A common but insecure method of encrypting text is to permute the letters of the alphabet. That is, in the text, each letter of the alphabet is consistently replaced by some other letter. So as to ensure that the encryption is reversible, no two letters are replaced by the same letter.

A common method of cryptanalysis is the known plaintext attack. In a known plaintext attack, the cryptanalist manages to have a known phrase or sentence encrypted by the enemy, and by observing the encrypted text then deduces the method of encoding.

Your task is to decrypt several encrypted lines of text, assuming that each line uses the same set of replacements, and that one of the lines of input is the encrypted form of the plaintext

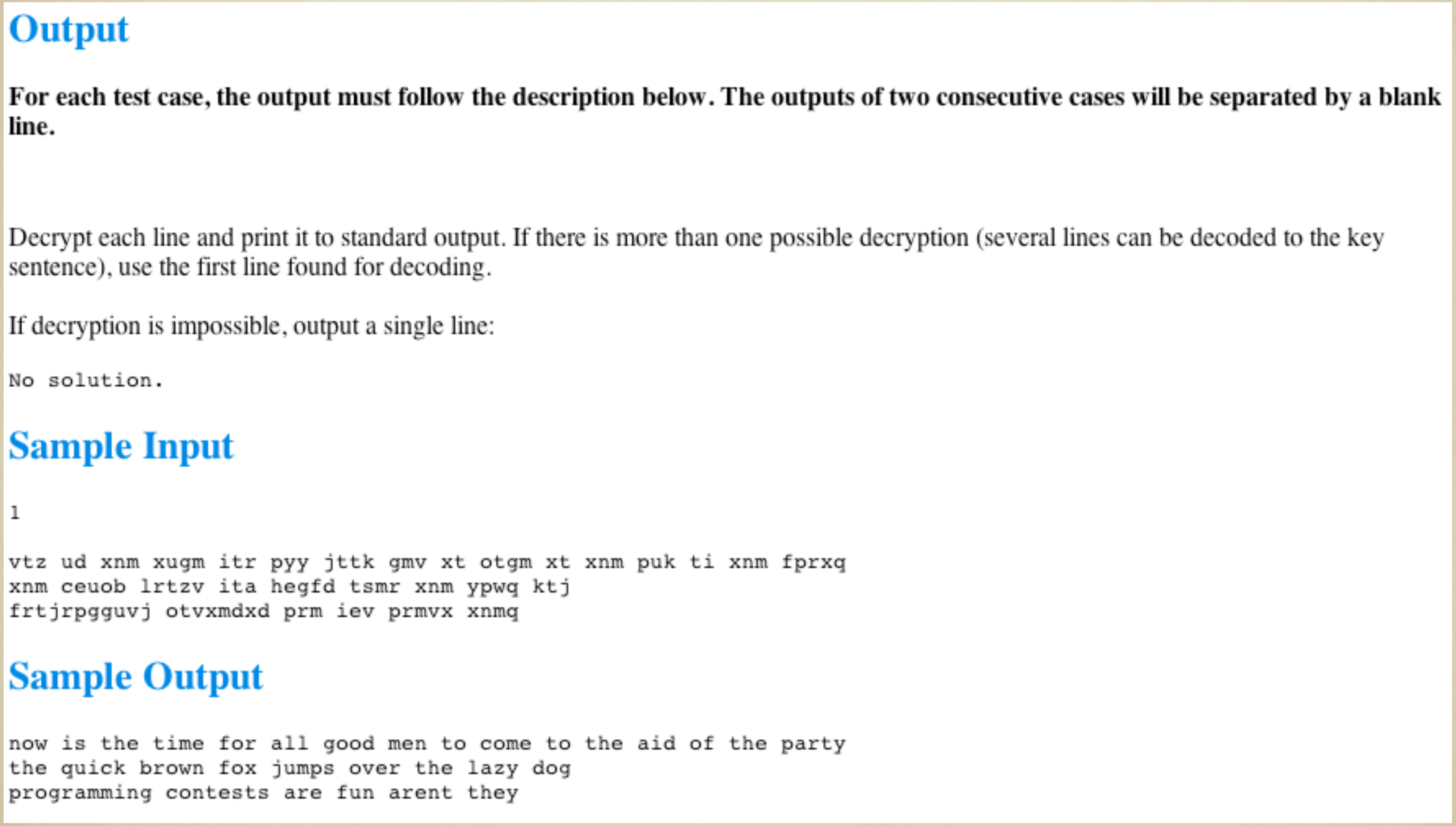
```
the quick brown fox jumps over the lazy dog
```

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The input consists of several lines of input. Each line is encrypted as described above. The encrypted lines contain only lower case letters and spaces and do not exceed 80 characters in length. There are at most 100 input lines.

Crypt Cricker 2



Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

Decrypt each line and print it to standard output. If there is more than one possible decryption (several lines can be decoded to the key sentence), use the first line found for decoding.

If decryption is impossible, output a single line:

```
No solution.
```

Sample Input

```
1

vtz ud xnm xugm itr pyy jttk gmv xt otgm xt xnm puk ti xnm fprxq
xnm ceuob lrtzv ita hegfd tsmr xnm ypwq ktj
frtjrpgguvj otvxmdxd prm iev prmvx xnmq
```

Sample Output

```
now is the time for all good men to come to the aid of the party
the quick brown fox jumps over the lazy dog
programming contests are fun arent they
```