acm International Collegiate Programming Contest

UC Programming Contest 2004
Problem Set
Universidad de Carabobo, Venezuela

October 30, 2004

This problem set should contain a rules section and six (6) problems on ten (10) numbered pages.
Please inform a runner immediately if something is missing from your problem set.

Rules

- There are six (6) problems for each team to be completed in four (4) hours.
- You may use any of the following programming languages: Pascal, C, C++ or Java. You may use different programming languages for different problems and even for different submissions of a problem.
- All problems require that you read test data from the standard input and write results to the standard output.
- All problems have a source file name shown below the title of the problem. The extensions you must use for your source files are: *.pas* for programs written in Pascal, *.c* for those written in C, *.cpp* for those in C++, and *.java* for those in Java.
- You may use any of the standard libraries that your chosen programming language provides. You may not use any other library that requires an extra flag to be passed to the compiler command. If you do this, judges will probably get a compilation-linking error in your program.
- Output corresponds exactly to the provided sample output format, including (mis)spelling and spacing. Multiple spaces will not be used in any of the judges' output, except where explicitly stated.
- Your solution to any problem should be submitted for judging using PC^2 software only. Once you have submitted a solution, it will reach the judges. The time it takes for your problem to be judged will depend on how busy the judges are. Once your submission has been judged, you will receive a message through PC^2 indicating the judgment. If your solution is accepted the message will be "Yes", if it's not then the message will be "No" along with the type of error the judges encountered, which may be: "Compilation Error", "Run-time Error", "Time-limit Exceeded", "Wrong Answer", "Excessive Output", "Incomplete Output" or "Output Format Error".

Problem A: Build Gates

Source File: gates.*

Statement of the Problem

Your cousin Bill has a patented process that produces AND, OR and NOT circuits. Unfortunately, OR gates are very expensive. He offers to wash your laundry for the next year if you can fix the problem.

Input Specification

A sequence of simplified logical expressions, one per line, which conform to the following BNF grammar:

E := '!' E | '(' E '&' E ')' | '(' E '|' E ')' | V
V := 'a' | 'b' | ... | 'z'

Here, the **bold** are nonterminals or parts of the BNF grammar, while the 'teletype' are terminals. Of course, '&', '|' and '!' are AND, OR and NOT operators, respectively.

There will be no whitespace in the input file.

Output Specification

For each expression in the input, you are to produce an equivalent simplified expression in the output, i.e., one that gives the same truth value for all possible combinations of the input, but without using the OR operation.

The generated expression must conform to the same grammar, with the same whitespace restrictions.

Sample Input

```
x
(x&!(y|!z))
!(x|y)
```

Expected Output for the Sample Input

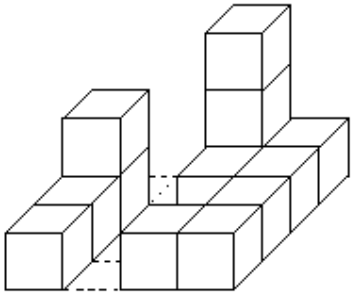
```
x
(x&(!y&z))
(!x&!y)
```

Problem B: Cube Chaos

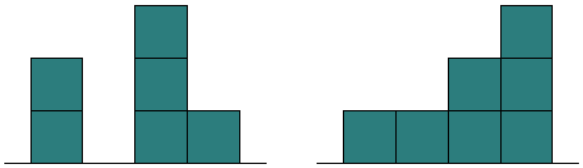
Source File: cube.*

Statement of the Problem

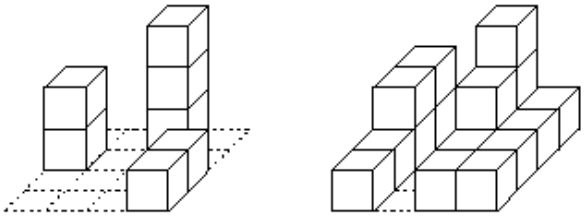
A construction is a number of cubes arranged on a square base as in the diagram below.



This is a four by four square supporting stacks of cubes. No stack will contain more than 8 cubes. The front and side projections of the construction are shown below.



Obviously there are a number of constructions that will satisfy both these projections. The constructions with the maximum and minimum number of blocks that will satisfy both projections are shown below.



Here the maximum number of blocks is 17, and the minimum is 7. Your job is to find the maximum and minimum number of blocks given the front and side projections.

Input Specification

The input file contains the front and side descriptions of several constructions. For each construction there will be three lines. The fist line is an integer $k(1 \leq k \leq 8)$, the length of the side of the square base. The next two lines contain the front and side projections. Each projection consists of k integers, which indicate the heights reflected in it. The end of the input will be signified by a $k = 0$.

Output Specification

For each construction you must output the maximum and minimum number of blocks required to obtain its projections, as shown in the sample output.

Sample Input

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

Expected Output for the Sample Input

```
Construction #1:
  The maximum number of blocks is: 17.
  The minimum number of blocks is: 7.
Construction #2:
  The maximum number of blocks is: 1.
  The minimum number of blocks is: 1.
```

Problem C: Magic Squares

Source File: magic.*

Statement of the Problem

Among the oldest and most popular forms of mathematical recreations are *magic squares*. A magic square of order n is an $n \times n$ array constructed out of the integers $1, 2, 3, \ldots, n^2$ in such a way that the sum s of the integers in each row, in each column, and in each of the two diagonals is the same. The number s is called the *magic sum* of the magic square. Examples of magic squares of order 3 and 4 are

$$\begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{pmatrix} \tag{2}$$

with magic sum 15 and 34 respectively. In medieval days there was a certain mysticism associated wirh magic squares, and they were worn for protection against evils. Benjamin Franklin was a magic square fan, and his papers contain many interesting examples.

We describe here a method found by de la Loubère in the seventeenth century for constructing magic squares of order n when n is odd. First a 1 is placed in the middle square of the top row. The successive integers are then placed in their natural order along a diagonal line which slopes upwards and to the right, with the following modifications:

1. When the top row is reached, the next integer is put in the bottom row as if it came immediately above the top row.
2. When the right-hand column is reached, the next integer is put in the left-hand column as if it immediately succeeded the right-hand column.
3. When a square is reached which has already been filled or when the top right-hand square is reached, the next integer is placed in the square immediately below the last square which was filled.

The magic square of order 3 in (1) was constructed using the de la Loubère's method, as was this magic square of order 5:

$$\begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix} \tag{3}$$

Your task is to write a program that constructs a magic square of odd order up to 255 using the de la Loubère's method described above.

Input Specification

The input will contain one or more test cases. Each test case consists of one line containing one integer, representing the order of the magic square. The end of input stream is marked by the number zero.

Output Specification

The output will consist of one magic square for each positive integer in the input. Each magic square must be separated by a blank line. The output must be formatted as shown in the example below:

- The numbers are outputted in columns of the same width.
- The numbers are right justified within columns.
- Each column of numbers must be separated by only one column of blank spaces.

Sample Input

```
3
5
7
0
```

Expected Output for the Sample Input

```
8 1 6
3 5 7
4 9 2

17 24  1  8 15
23  5  7 14 16
 4  6 13 20 22
10 12 19 21  3
11 18 25  2  9

30 39 48  1 10 19 28
38 47  7  9 18 27 29
46  6  8 17 26 35 37
 5 14 16 25 34 36 45
13 15 24 33 42 44  4
21 23 32 41 43  3 12
22 31 40 49  2 11 20
```

Problem D: Humble Numbers

Source File: number.*

Statement of the Problem

A number whose only prime factors are $2, 3, 5$ or 7 is called a *humble number*. The sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 25, 27, ... shows the first 20 humble numbers.

Write a program to find and print the nth element in this sequence.

Input Specification

The input consists of one or more test cases. Each test case consists of one integer n with $1 \leq n \leq 5842$. Input is terminated by a value of zero (0) for n.

Output Specification

For each test case, print one line saying "The nth humble number is number.". Depending on the value of n, the correct suffix "st", "nd", "rd", or "th" for the ordinal number nth has to be used like it is shown in the sample output.

Sample Input

```
1
2
3
4
11
12
13
21
22
23
100
1000
5842
0
```

Expected Output for the Sample Input

```
The 1st humble number is 1.
The 2nd humble number is 2.
The 3rd humble number is 3.
The 4th humble number is 4.
The 11th humble number is 12.
The 12th humble number is 14.
The 13th humble number is 15.
The 21st humble number is 28.
The 22nd humble number is 30.
The 23rd humble number is 32.
The 100th humble number is 450.
The 1000th humble number is 385875.
The 5842nd humble number is 2000000000.
```

Problem E: Knight Moves

Source File: knight.*

Statement of the Problem

A friend of you is doing research on the Traveling Knight Problem (TKP) where you are to find the shortest closed tour of knight moves that visits each square of a given set of n squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy.

Of course you know that it is vice versa. So you offer him to write a program that solves the "difficult" part. Your job is to write a program that takes two squares a and b as input and then determines the number of knight moves on a shortest route from a to b.

Input Specification

The input will contain one or more test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter (a-h) representing the column and a digit (1-8) representing the row on the chessboard.

Output Specification

For each test case, print one line saying "To get from xx to yy takes n knight moves.".

Sample Input

```
e2 e4
a1 b2
b2 c3
a1 h8
a1 h7
h8 a1
b1 c3
f6 f6
```

Expected Output for the Sample Input

```
To get from e2 to e4 takes 2 knight moves.
To get from a1 to b2 takes 4 knight moves.
To get from b2 to c3 takes 2 knight moves.
To get from a1 to h8 takes 6 knight moves.
To get from a1 to h7 takes 5 knight moves.
To get from h8 to a1 takes 6 knight moves.
To get from b1 to c3 takes 1 knight moves.
To get from f6 to f6 takes 0 knight moves.
```

Problem F: Syllabication in Spanish

Source File: syllable.*

Statement of the Problem

In Spanish, when we write some text, such as a tale, a novel or a math book, we sometimes need to divide words in smaller pieces called *syllables*. This is especially true when we write by hand and reach the right margin of the page. In this case, we must divide the word in two by putting a dash after the first part and writing the rest of the word on the next line. However, such a process of dividing words in syllables follows a set of systematic rules known as *syllabication patterns*. Before explaining these patterns, some phonologic aspects from the Spanish language must be taken into account:

- The Spanish language contemplates five vowels: *a*, *e*, *i*, *o*, *u*, which may be emphasized within a word. In that case an accent mark is placed on top of them: *á*, *é*, *í*, *ó*, *ú*. Likewise, in Spanish a word may contain one or more vowel sequences (accented or not) in the form of diphthongs (two consecutive vowels in the same syllable), triphthongs (three consecutive vowels in the same syllable), and hiati (two consecutive vowels in different syllables).
- The Spanish language contemplates 22 consonants: *b*, *c*, *d*, *f*, *g*, *h*, *j*, *k*, *l*, *m*, *n*, *ñ*, *p*, *q*, *r*, *s*, *t*, *w*, *x*, *y*, *z*. Some of these consonants form clusters that, in orthographic terms, are considered as an inseparable unit. These clusters are: *pr*, *pl*, *tr*, *cr*, *cl*, *br*, *bl*, *dr*, *gr*, *gl*, *fr*, *fl*. There are also the *double consonants*, which are a special case of clusters; these are: *ll* and *rr*. Finally, there is the letter *ch*, which is neither considered as a cluster nor as a double consonant. It is rather a special graph that represents an affricate sound typical for Spanish, which is produced by opposing the tip of the tongue against the alveolus.

Your task consists in writing a program that, given an arbitrary word in Spanish, divides it properly in syllables, according to the rules shown below. Here 'V' stands for 'vowel' and 'C' stands for 'consonant'.

1. A consonant between two vowels: each consonant is associated with each of the vowels:

   $VCV \rightarrow V - CV$

2. Two consonants between two vowels: each consonant is associated with each of the vowels:

   $VCCV \rightarrow VC - CV$

   In the case where both consonants form a cluster or the letter *ch*, both consonants are associated with the second vowel:

   $VCCV \rightarrow V - CCV$

3. Three consonants between two vowels: the first two consonants are associated with the first vowel, and the remaining consonant is associated with the other vowel:

   $VCCCV \rightarrow VCC - CV$

   In the case where the last two consonants form a cluster or the letter *ch*, both consonants are associated with the second vowel:

   $VCCCV \rightarrow VC - CCV$

4. Four consonants between two vowels: each pair of consonants is associated with each vowel:

   $VCCCCV \rightarrow VCC - CCV$

With the aim of making the program simpler to elaborate, you must consider the following conventions:

- Monosyllabic words will not be considered.
- Words with vowel sequences will not be considered, i.e., there won't be any diphthongs, triphthongs or hiatuses.
- Words with the letter *h* in an intermediate position will not be considered. Words starting with an *h* will be considered.

- Words with the letter x or w in any position will not be considered.
- Words with a Greek origin containing the cluster *ps* will not be considered.
- Derived words that contain the *sub-* or *ab-* Latin prefixes will not be considered.
- Every composed word will be subject to each one of the previous conventions.

Additionally, your program must indicate the type of word according to its accentuation, i.e., if it's oxytone, paroxytone, proparoxytone or with the stress on the syllable before the antepenult. The meaning of these types of words follows:

- A word is oxytone if it's accented on the last syllable. The *orthographic accent* of an oxytone word is written if only if the word ends with n, s, or a vowel.
- A word is paroxytone if it's accented on the next to last syllable. The *orthographic accent* of a paroxytone word is written if and only if the word does not end with n, s, or a vowel.
- A word is proparoxytone if it's accented on the antepenult syllable. The *orthographic accent* of a proparoxytone word is always written.
- The *orthographic accent* of a word accented on any syllable before the antepenult is always written.

Input Specification

The input will contain several lines. The first one shows a number $N \geq 1$ which indicates the number of words the input has. Then, N lines will follow, each of them containing just one word.

Output Specification

The output for each of the N words in the input consists of two lines, the first one of them containing the word itself enclosed between quotes and followed by the message *"has S syllables: "*, where S is the number of syllables the word has been divided into. Notice that a trailing blank is left behind the colon in the message. Immediately, you must place the current word properly divided into syllables such that each one of them is isolated from the other by two dashes —except for the first and last syllables—. The second line of the ouput will contain a message which indicates the type of the word according to the accentuation rules given above. If the word is oxytone, paroxytone or proparoxytone, you must write *"The word is ...."*, but if the word is accented before the antepenult syllable you must write *"The word is accented before the antepenult syllable."*. The output for each of the N words must be separated by one blank line. There won't be any line after the last output.

Sample Input

```
3
resultado
chícharos
últimamente
```

Expected Output for the Sample Input

```
"resultado" has 4 syllables: re-sul-ta-do
The word is paroxytone.

"chícharos" has 3 syllables: chí-cha-ros
The word is proparoxytone.

"últimamente" has 5 syllables: úl-ti-ma-men-te
The word is accented before the antepenult syllable.
```