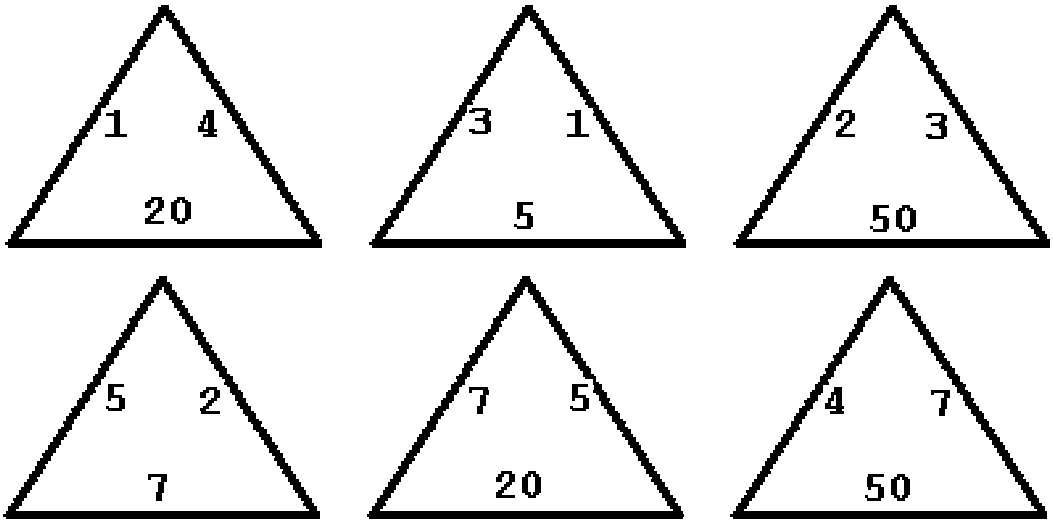
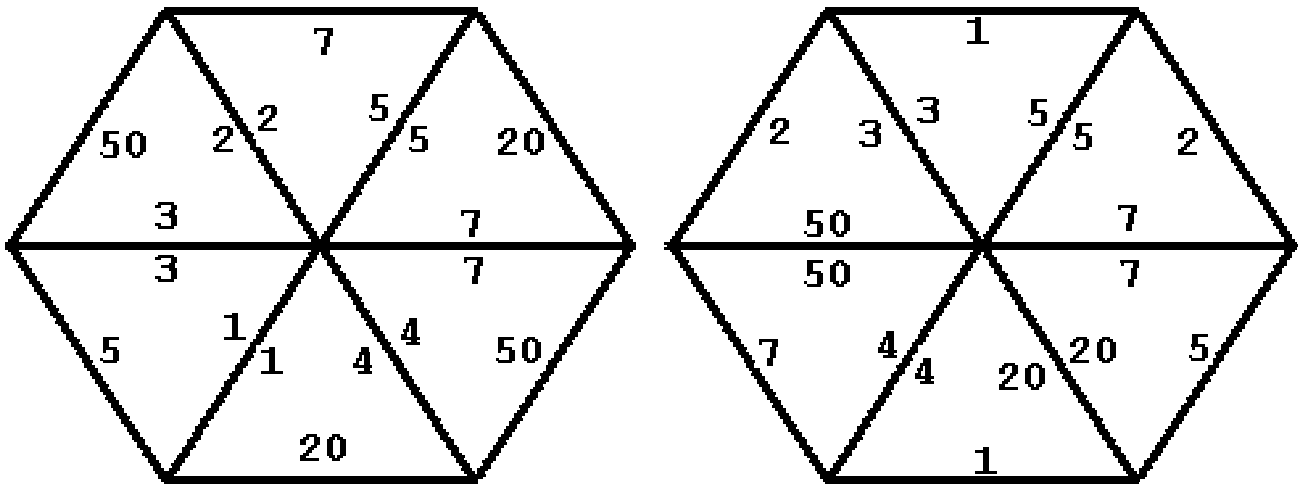
Problem A: The Triangle Game

Source file: `triangle.{c, cpp, java, pas}`
Input file: `triangle.in`
Output file: `triangle.out`



In the triangle game you start off with six triangles numbered on each edge, as in the example above. You can slide and rotate the triangles so they form a hexagon, but the hexagon is only legal if edges common to two triangles have the same number on them. You may not flip any triangle over. Two legal hexagons formed from the six triangles are illustrated below.



The score for a legal hexagon is the sum of the numbers on the outside six edges.

Your problem is to find the highest score that can be achieved with any six particular triangles.

The input file will contain one or more data sets. Each data set is a sequence of six lines with three integers from 1 to 100 separated by blanks on each line. Each line contains the numbers on the triangles in clockwise order. Data sets are separated by a line containing only an asterisk. The last data set is followed by a line containing only a dollar sign.

For each input data set, the output is a line containing only the word "none" if there are no legal hexagons or the highest score if there is a legal hexagon.

Example input:

```
1 4 20
3 1 5
50 2 3
5 2 7
7 5 20
4 7 50
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
60 6 10
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
10 6 60
$
```

Example output:

```
152
21
none
```

Last modified Tue Oct 24 20:38:19 2000

Problem B: Easier Done than Said?

Source file: `say.{c, cpp, java, pas}`
Input file: `say.in`
Output file: `say.out`

Password security is a tricky thing. Users prefer simple passwords that are easy to remember (like *buddy*), but such passwords are often insecure. Some sites use random computer-generated passwords (like *xvtpzyo*), but users have a hard time remembering them and sometimes leave them written on notes stuck to their computer. One potential solution is to generate "pronounceable" passwords that are relatively secure but still easy to remember.

FnordCom is developing such a password generator. You work in the quality control department, and it's your job to test the generator and make sure that the passwords are acceptable. To be acceptable, a password must satisfy these three rules:

1. It must contain at least one vowel.
2. It cannot contain three consecutive vowels or three consecutive consonants.
3. It cannot contain two consecutive occurrences of the same letter, except for '`ee`' or '`oo`'.

(For the purposes of this problem, the vowels are '`a`', '`e`', '`i`', '`o`', and '`u`'; all other letters are consonants.) Note that these rules are not perfect; there are many common/pronounceable words that are not acceptable.

The input consists of one or more potential passwords, one per line, followed by a line containing only the word '`end`' that signals the end of the file. Each password is at least one and at most twenty letters long and consists only of lowercase letters. For each password, output whether or not it is acceptable, using the precise format shown in the example.

Example input:

```
a
tv
ptoui
bontres
zoggax
wiinq
eep
houctuh
```

```
end
```

Example output:

```
<a> is acceptable.
<tv> is not acceptable.
<ptoui> is not acceptable.
<bontres> is not acceptable.
<zoggax> is not acceptable.
<wiinq> is not acceptable.
<eep> is acceptable.
<houctuh> is acceptable.
```

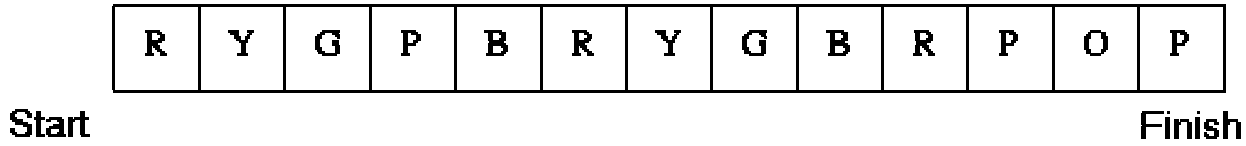
Last modified Fri Oct 27 15:16:38 2000

Problem C: Colorville

Source file: `colors.{c, cpp, java, pas}`
Input file: `colors.in`
Output file: `colors.out`

A simple matching game for children uses a board that is a sequence of colored squares. Each player has a game piece. Players alternate turns, drawing cards containing either one colored square or two colored squares of the same color. Players move their pieces forward on the board to the next square that matches the single color on the card, or forward to the second matching square if the card contains two colored squares, or forward to the last square on the board if there is no square matching the description above. A player wins if his piece lands on the last square of the board. It is possible for all the cards to be drawn and still not have a winner.

This problem represents colors with capital letters from A-Z. Below is a diagram of a sample board.



Consider the deck of cards: `R, B, GG, Y, P, B, P, RR`

For 3 players, the game proceeds as follows:

```
Player 1 draws R,  moves to 1st square
Player 2 draws B,  moves to 5th square
Player 3 draws GG, moves to 8th square
Player 1 draws Y,  moves to 2nd square
Player 2 draws P,  moves to 11th square
Player 3 draws B,  moves to 9th square
Player 1 draws P,  moves to 4th square
Player 2 draws RR, Wins! (no Rs in front of piece so it goes to last square)
```

Using the same board and the same deck of cards, but with 2 players, Player 1 wins after 7 cards. With 4 players, no one wins after exhausting the deck of 8 cards.

Input consists of information for one or more games. Each game starts with one line containing the number of players (1-4), the number of squares on the board (1-79), and the number of cards in the deck (1-200). This is followed by a single line of characters representing the colored squares on the board. Following this are the cards in the deck, one card per line. Cards can have only a single character, or two of the same character. The end of the input is signalled by a line with 0 for the number of players - the other two values will be present but indeterminate.

For each game, the output is either the winning player and the total number of cards drawn, or the number of cards in the deck, as shown in the sample output. Always use the plural "cards".

Example input:

```
2 13 8
RYGPBRYGBRPOP
R
B
GG
Y
P
```

```
B
P
RR
2 6 5
RYGRYB
R
YY
G
G
B
3 9 6
QQQQQQQQQ
Q
QQ
Q
Q
QQ
Q
0 6 0
```

Example output:

```
Player 1 won after 7 cards.
Player 2 won after 4 cards.
No player won after 6 cards.
```

Last modified Tue Oct 24 23:49:55 2000

Problem D: Falling Leaves

Source file: `leaves.{c, cpp, java, pas}`
Input file: `leaves.in`
Output file: `leaves.out`

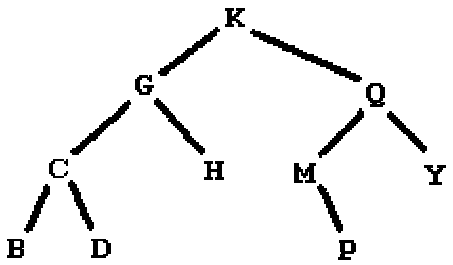


Figure 1

Figure 1 shows a graphical representation of a binary tree of letters. People familiar with binary trees can skip over the definitions of a binary tree of letters, leaves of a binary tree, and a binary search tree of letters, and go right to **The problem**.

A *binary tree of letters* may be one of two things:

1. It may be empty.
2. It may have a root *node*. A node has a letter as data and refers to a left and a right subtree. The left and right subtrees are also binary trees of letters.

In the *graphical* representation of a binary tree of letters:

1. Empty trees are omitted completely.
2. Each node is indicated by
   - Its letter data,
   - A line segment down to the left to the left subtree, if the left subtree is nonempty,
   - A line segment down to the right to the right subtree, if the right subtree is nonempty.

A *leaf* in a binary tree is a node whose subtrees are both empty. In the example in Figure 1, this would be the five nodes with data `B`, `D`, `H`, `P`, and `Y`.

The *preorder traversal of a tree of letters* satisfies the defining properties:

1. If the tree is empty, then the preorder traversal is empty.
2. If the tree is not empty, then the preorder traversal consists of the following, in order
   - The data from the root node,
   - The preorder traversal of the root's left subtree,
   - The preorder traversal of the root's right subtree.

The preorder traversal of the tree in Figure 1 is `KGCBDHQMPY`.

A tree like the one in Figure 1 is also a binary search tree of letters. A *binary search tree of letters* is a binary tree of letters in which each node satisfies:

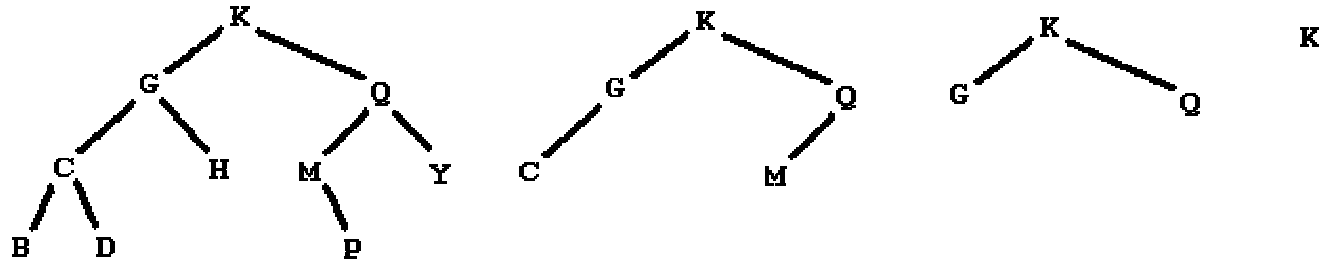
1. The root's data comes later in the alphabet than all the data in the nodes in the left subtree.
2. The root's data comes earlier in the alphabet than all the data in the nodes in the right subtree.

The problem:

Consider the following sequence of operations on a binary search tree of letters

Remove the leaves and list the data removed
Repeat this procedure until the tree is empty

Starting from the tree below on the left, we produce the sequence of trees shown, and then the empty tree



by removing the leaves with data

```
BDHPY
CM
GQ
K
```

Your problem is to start with such a sequence of lines of leaves from a binary search tree of letters and output the preorder traversal of the tree.

The input file will contain one or more data sets. Each data set is a sequence of one or more lines of capital letters. The lines contain the leaves removed from a binary search tree in the stages described above. The letters on a line will be listed in increasing alphabetical order. Data sets are separated by a line containing only an asterisk ('*'). The last data set is followed by a line containing only a dollar sign ('$'). There are no blanks or empty lines in the input.

For each input data set, there is a unique binary search tree that would produce the sequence of leaves. The output is a line containing only the preorder traversal of that tree, with no blanks.

Example input:

```
BDHPY
CM
GQ
K
*
AC
B
$
```

Example output:

```
KGCBDHQMPY
BAC
```

Last modified Tue Oct 24 00:17:27 2000