ODU/ACM
Programming Contest

Spring, 1993

1. Bridge Hand Evaluator

In the card game "Bridge", players must assess the strength of their hands prior to bidding against one another. Most players use a point-count scheme which employs the following rules:

1. Each ace counts 4 points. Each king counts 3 points. Each queen counts 2 points. Each jack counts one point.
2. Subtract a point for any king of a suit in which the hand holds no other cards.
3. Subtract a point for any queen in a suit in which the hand holds only zero or one other cards.
4. Subtract a point for any jack in a suit in which the hand holds only zero, one, or two other cards.
5. Add a point for each suit in which the hand contains exactly two cards.
6. Add two points for each suit in which the hand contains exactly one card.
7. Add two points for each suit in which the hand contains no cards.

A suit is *stopped* if it contains an ace, or if it contains a king and at least one other card, or if it contains a queen and at least two other cards.

During the opening assessment, the three most common possibilities are:

- If the hand evaluates to fewer than 14 points, then the player must pass.
- One may open bidding in a suit if the hand evaluates to 14 or more points. Bidding is always opened in one of the suits with the most cards.
- One may open bidding in "no trump" if the hand evaluates to 16 or more points *ignoring rules 5, 6, and 7* and if all four suits are stopped. A no trump bid is always preferred over a suit bid when both are possible.

Input: Standard input gets 13 cards on one line. Each card consists of two characters. The first represents the rank of the card: 'A', '2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K'. The second represents the suit of the card: 'S', 'H', 'D', 'C', standing for "spades", "hearts", "diamonds", and "clubs", respectively.

Output: The recommended bid, either "PASS", "BID *suit*", where *suit* is "S", "H", "D", or "C", or "BID NO-TRUMP".

Example: The input file contains:

```
KS QS TH 8H 4H AC QC TC 5C KD QD JD 8D
```

The evaluation starts with 6 points for the two kings, 4 for the ace, 6 for the three queens, and one for the jack. To this tally of 17 points, we add 1 point for having only two cards in spades, and subtract 1 for having a queen in spaces with only one other card in spades. The resulting 17 points is enough to justify opening in a suit.

The evaluation for no-trump is 16 points, since we cannot count the one point for having only two spades. We cannot open in no-trump, however, because the hearts suit is not stopped.

Hence we must open bidding in a suit. The two longest suits are clubs and diamonds, with four cards each, so the possible outputs are "BID C", or "BID D".

2. Polynomial Factorization

A polynomial is of degree k if the largest power of the variable in any term is no greter than k. For example,

$$x^2 + 4x + 2$$

is of degree 2. It is also of degree 3, 4, 5, ...

A polynomial with integer coefficients is "prime" if it cannot be expressed as the product of two lower-degree polynomials with integer coefficients.

Write a program to express a 4th degree polynomial with integer coefficients as the product of one or more prime polynomials.

INPUT 5 integers, representing the coefficients of a degree 4 polynomial in decreasing order of the variable power from 4 to 0.

OUTPUT Print the integer coefficients of each prime factor on a single line, in decreasing order of variable power.

EXAMPLE Given the input:

2 7 -1 -6 -2

The output could be

1 -1
2 1
1 4 2

(The three lines may appear in any order.)

In other words, the polynomial $2x^4 + 7x^3 - x^2 - 6x - 2$ has prime factors $x - 1$, $2x + 1$, and $x^2 + 4x + 2$.

3. Sentence/Phrase Generator

Write a program that generates English language phrases and sentences conforming to the following rules:

```
<sentence> ::= <trans-sentence>
<sentence> ::= <intrans-sentence>
<trans-sentence> ::= <subject> <verb-phrase> <object> <prep-phrase>
<intrans-sentence> ::= <subject> <intrans-verb-phrase> <prep-phrase>
<subject> ::= <noun-phrase>
<object> ::= <noun-phrase>
<noun-phrase> ::= <article> <modified-noun>
<modified-noun> ::= <noun> | <modifier> <noun>
<modifier> ::= <adjective> | <adverb> <adjective>
<verb-phrase> ::= <trans-verb> | <adverb> <trans-verb>
<intrans-verb-phrase> ::= <intrans-verb> | <adverb> <intrans-verb>
<prep-phrase> ::= <preposition> <noun-phrase> | <empty>
<noun> ::= man | dog | fish | computer | waves
<trans-verb> ::= struck | saw | bit | took
<intrans-verb> ::= slept | jumped | walked | swam
<article> ::= the | a
<adjective> ::= green | small | rabid | quick
<adverb> ::= nearly | suddenly | restlessly
<preposition> ::= on | over | through
<empty> ::= ""
```

For example, the first two lines say that. to generate a sentence, one may generate a "trans-sentence" or an "intrans-sentence". A transitive sentence, according to the third rule, consists of a "subject", followed by a "verb-phrase", followed by an "object", followed by a "prep-phrase". Similarly, the next-to-last rule indicates that a "preposition" can be any of the three words **on**, **over**, or **through**.

Your program should read from the input a number of requests for various kinds of phrases. Each request may be for any of the phrase names appearing on the left hand side of the above rules. It should then attempt to generate the requested phrase by applying these rules until all of the <...> have been replaced with appropriate words.

In many cases, you will face a choice of alternate rules for expanding a phrase name. In these cases, you should make a choice as follows: Suppose that this is the k^{th} such choice that you have faced since the start of execution of your program,and that you must choose one of n rules for expanding a given kind of phrase. Let the rules for that phrase be numbered from $1 \ldots n$ in the order of appearance above, and then choose rule number $(k \bmod n) + 1$.

INPUT The input will consist of an unspecified number of lines. Each line will contain, left-justified, a phrase name corresponding to one of the names appearing on the left-hand-side of the rules above (without the surrounding brackets).

OUTPUT For each phrase named in the output, print a single line containing the expansion of that phrase according to the above rules. Each word in the phrase should be separated from the others by a single space.

EXAMPLE If the input were

sentence
noun
sentence

the output could be

> the small dog restlessly jumped through the quick dog
> fish
> a dog took the quick computer

4. Overflow

Write a program that reads an expression consisting of two non-negative integer and an operator. Determine if either integer or the result of the expression is too large to be represetned as a "normal" signed integer (type `integer` if you are working Pascal, type `int` if you are working in C).

INPUT An unspecified number of lines. Each line will contain an integer, one of the two operators + or $*$, and another integer.

OUTPUT For each line of input, print the input folowed by 1-3 lines containing as many of these three messages as are appropriate: "first number too big", "second number too big", "result too big".

EXAMPLES For the input

```
300 + 3
99999999999999999999 + 11
```

the output would be

```
300 + 3
99999999999999999999 + 11
first number too big
result too big
```

5. Mirror, Mirror

A square pattern of light and dark cells is shown in its original state and a transformed state. Write a program that will recognize the minimum transformation that has been applied to the original pattern given the following list of possible transformations:

90 Degree Rotation: The pattern was rotated to the right 90 degrees.

180 Degree Rotation: The pattern was rotated to the right 180 degrees.

270 Degree Rotation: The pattern was rotated to the right 270 degrees.

Vertical Reflection: The pattern was reflected through a horizontal mirror positioned above the pattern.

Combination: The pattern was subjected to a vertical reflection followed by one of the rotations.

Preservation: The original pattern was preserved (the new pattern is identical to the original).

Improper: The new pattern was not obtained via any of these treansformations.

INPUT The input file will consist of an unknown number of pattern datasets on the standard input. Each pattern dataset will consist of an integer on a line by itself, which gives the dimensions of the square containing the pattern (the size will range from 1 to 10). The following lines will contain each line of the original and new (transformed) patterns in a side-by-side format, separated by a space. Light squares will be indicated by a dot (period), while dark squares will be represented with an x.

OUTPUT The output from your program will be a sentence describing the relationship that the new pattern bears to the original. Each sentence will begin with a pattern ID number (starting with 1) and end stating the relatinship representing the minimal amount of work necessary to derive the new pattern from the original. For the purpose of evaluating the amount of work needed, rotations are considered less work than reflections, and smaller rotations are less work than larger ones. Of course, "preservation" involves no work at all.

Note that only the above possibilities should be considered — there is no such thing as a "360 degree rotation" for this problem (such a transformation would "preserve" the pattern), nor is there a "horizontal reflection". Also, remember that when a single rotation or reflection is not sufficient, your program should next consider roated versions after a vertical reflection. Although a combination transformation might yield the same new pattern as one of the single transformations alone, the single transformation is the one you should output (the minimal transformation). Your output should be a complete sentence, ending with a period.

EXAMPLE If the input file contained

```
5
X...X ....X
.X... ...X.
...X. .X...
..X.X ..X..
....X XX..X
6
....XX X....X
```

```
...X.. X.X...
XX..X. .X..X.
..X... ...X.X
...X.. ..X...
..X..X ..X...
2
X. X.
.X .X
4
..X. ...X
XX.. ....
.... XX..
...X ..X.
5
X.... .X...
.X... ..X..
.X... ..X..
...X. ....X
....X X....
4
.X.. ..X.
.X.X X...
.... ..XX
..X. ....
2
.. XX
XX ..
```

your output would be something resembling:

```
Pattern 1 was rotated 90 degrees.
Pattern 2 was rotated 270 degrees.
Pattern 3 was preserved.
Pattern 4 was reflected vertically.
Pattern 5 was improperly transformed.
Pattern 6 was reflected vertically and rotated 270 degrees.
Pattern 7 was rotated 180 degrees.
```

6. Synching Signals

On your way to work wach morning you travel down a main traffic artery regulated by a number of traffic signals. On some mornings, you have noticed that all of the lights up ahead simultaneously turn green in your direction. But on other mornings, it seems that a random combination of red, yellow, and green is facing you.

After recently observing all of the lights ahead of you turn green simultaneously, you began to wonder how long after any one of them turns to yellow it would take before they all would be displaying green in your direction again. Write a program to figure out how long it will take such a set of traffic signals to all be displaying green again, given the cycle times for each traffic signal in the set.

INPUT The input file will consist of an unknown number of traffic signal datasets. Each signal dataset will consist of one line of integers (separated by spaces) giving the total cycle time (in seconds) for each signal. The *cycle time* is the total time that a signal will stay green and yellow in one direction and red in the opposite direction. In this problem, you may assume that a green signal will turn yellow for the last 5 seconds of its cycle. If one set of signals was given as:

30 25 35

your program would need to recognize that there are three signals in the set, that the first signal lasts 30 seconds, that the second signal cycles every 25 seconds, and that the thirs signal takes 35 seconds to start a new cycle. Specifically, the first signal will be red for 30 seconds in one direction, then green for 25 seconds, then yellow for 5.

Each set of signals will involve at least two and as many as 10 signals. Each signal will have a minimum cycle time of 10 seconds and a maximum cycle time of 90 seconds.

OUTPUT Your output will consist of a summary line for each set of signals. Your program should assign an ID number to each set of signals, beginning with set 1. The output line will begin with the signal set ID number and state the number of minutes (≤ 60) and seconds (< 60) it will take from when all of the signals simultaneously turn green initially to the first time they will all be showing green again in your direction after any of them has turned yellow. Note that this time may or may not be a time when all of the signals in the set simultaneously change back to gree — for this problem you need only indicate how long it will be before all signals in the set are once again simultaneously *showing* green in your direction after any of them has turned to yellow, even if this condition will only exist for a second or a very few seconds. If the signals will never simultaneously display green in your direction again within an hour, you should print a message that states that the signals in the set are not able to synchronize after one hour (note that an output of 60 minutes and 0 seconds should, however, be considered a successful synchronization).

EXAMPLE Suppose that the input file contained

```
30 25 35
25 25 25 25 25
15 30
20 21 30 23 29 25 27 22
19 20
```

The the output should resemble the following:

```
Set 1 synchs again at 5 minute(s) and 0 second(s) after all turning green.
Set 2 synchs again at 0 minute(s) and 50 second(s) after all turning green.
Set 3 synchs again at 1 minute(s) and 0 second(s) after all turning green.
Set 4 is unable to synch after one hour.
Set 5 synchs again at 0 minute(s) and 40 second(s) after all turning green.
```