COMPUTER AND INFORMATION SCIENCES

# UAB CIS High School Programming Contest

April 25, 2015

**THESE ARE THE OFFICIAL CONTEST QUESTIONS!**

Each problem in this packet contains a brief description, followed by three example test cases of a successful implementation.

The example input and output shown for each question should be regarded only as examples. We will test your programs on the examples provided with each problem, as well as several other test cases generated by the judges. Be sure to follow the input and output formatting exactly, printing each output answer on its own line. Extraneous or malformed output will result in a failed submission.

If your program fails, a result will be returned by the submission system stating the counterexample test case that caused your program to be judged incorrect. Each incorrect answer will be incur a 20 minute time penalty, but this penalty will only be applied to your score if the problem is eventually answered correctly. As noted in the rules, the overall ranks will be determined first by the number of problems completed, and then by your time score (including penalties).

Please pay close attention to the directions in each problem description. In some cases, assumptions are stated about the limitations of the input, which are designed so that you do not have to consider difficult cases or perform input validation.

If you get stuck on a problem, you are encouraged to jump around and try different problems, as an eventual correct answer on a problem is better than no answer at all.

## 1 COUNTER: County Counter

In Alabama, the county where a vehicle is registered can be deduced from the county ID on the license plate. For example, a license plate with the county ID 1 is registered in Jefferson County. To conduct road use analysis the Alabama DMV asks you to write a program that counts the number of vehicles from each county. The program should be able to distinguish the following counties: Jefferson, Shelby, Walker, and Cullman. Any vehicles registered outside these four counties is counted as other.
Registration Codes: 1 (Jefferson), 25 (Cullman), 58 (Shelby), 64 (Walker).

The input begins with one line indicating the number of test cases N that will be passed into the program, followed by N lines of county IDs extracted from observed vehicles. The program counts how many vehicles were registered in each county and prints the number of cars from each county in following county order: Jefferson, Cullman, Shelby, Walker, and Other for each of the N lines.

**Sample Input/Output**

Sample Input

```
3
25 1 1 25 58 77
64 12 58 64 1 88 3 2
58 58 58 58 58
```

Expected Output

```
2 2 1 0 1
1 0 1 2 4
0 0 5 0 0
```

## 2 TRIANGLE: Right Triangle Test

In a right triangle, the equation $a^2 + b^2 = c^2$ must hold (where $a, b, c$ are the lengths of each side, with $c$ being the longer side). Write a program that reads the coordinate of three points and tests whether it is a right triangle. The coordinates are represented as a pair of `int`. For example, the input 1 1 4 1 1 5 corresponds to the points $(1, 1); (4, 1); (1, 5)$. The input begins with one line indicating the number of test cases N that will be provided, followed by N lines containing 3 coordinate pairs. For each of the N lines, print 'yes' if it is a right triangle, or 'no' otherwise.

**Note** You will not need decimal precision when checking if the equation holds true (after calculations).

**Sample Input/Output**

Sample Input

```
3
1 1 4 1 1 5
2 3 3 2 6 6
3 2 14 2 3 62
```

Expected Output

```
yes
no
yes
```

## 3 ROMAN: Conversion from Roman Numerals

The history department at UAB asks you to develop a program that can automatically convert Roman numerals to Arabic. The Roman numerals are composed from the following seven symbols: $I$ (1), $V$ (5), $X$ (10), $L$ (50), $C$ (100), $D$ (500), $M$ (1000). Roman numerals combine these symbols in descending order. Arabic numbers can be computed by summing those values. For example, 11 is $XI$. An exception to this rule is the use of $I$ before $V$ and $X$ to create 4 and 9 respectively. For example, 119 is $CXIX$. Likewise, $X$ can precede $L$ (40) and $C$ (90), and $C$ can precede $D$ (400) and $M$ (900).

The input will begin with a line containing N, the number of test cases, followed by N Roman Numerals to convert into Arabic numbers. You may assume that the input consists only of valid Roman Numerals.

**Sample Input/Output**

Sample Input

```
3
CCXVI
MMMCMXVIII
MMXV
```

Expected Output

```
216
3918
2015
```

## 4 DIV7: Divisibility Test

To test whether a number is divisible by 7, one can multiply its digits by the repeating sequence $1, 3, 2, 6, 4, 5$. Start with the least significant digit of the input and multiply it with the first number of the sequence, etc. Then sum the products. The original number is divisible by 7, if the sum of products is divisible by 7.

For example, given the number 3456789 we would compute:

$$(9 * 1) + (8 * 3) + (7 * 2) + (6 * 6) + (5 * 4) + (4 * 5) + (3 * 1) = 126$$

. The result is 126, which is divisible by 7, indicating that 3456789 is divisible by 7.

Write a program that reads a line containing N, the number of test cases, followed by N sequences of very long numbers as string and determines whether each number is divisible by 7. Your program will output the sum of these numbers followed by a 'yes' or 'no' indicating if the original sequence is divisible by 7 (ex: "126 yes").

**Constraints**

- the input is given as string and the numeric value of the input may exceed bounds of available integral types.
- the sum of products fits within a `long`.

**Sample Input/Output**

| Sample Input |
|---|
| 3<br>7<br>10<br>6102752097204 |

| Expected Output |
|---|
| 7 yes<br>3 no<br>140 yes |

## 5 CAESAR: Deciphering Caesar

Caesar's cipher is one of the earliest encryption techniques. A message is encrypted by replacing each character with another character in the alphabet. An encoding with a fixed displacement $n$ replaces each character with the character + $n$ in the same alphabet (rollover at the end of the alphabet). One way to attack an encrypted message is to map the words in the message to a dictionary. If a consistent mapping with a fixed size $n$ can be found, the encryption can be broken.

Implement a decipher that breaks messages encoded with Caesar's cipher. The program first reads a line containing a dictionary of words separated by blanks, followed by a line containing N, the number of test cases, followed by N encrypted lines (in all lower case). Your program should use the provided dictionary and attempt to map each line's words to words in the provided dictionary. If successful, the program prints the number $n$ indicating character displacement used when the message was originally encrypted, followed by a blank and the decoded message for each of the N lines. If decryption fails, print $-1$.

**Sample Input/Output**

Sample Input

```
alabama blue birmingham color story street world
5
zcdlatsvt iwpi lxaa rwpcvt ndjg ldgas.
uyggv jqog cncdcoc.
dolyl aol zrplz hyl zv isbl.
blue street
oplioi iahkajs jaashsshh
```

Expected Output

```
15 knowledge that will change your world.
2 sweet home alabama.
7 where the skies are so blue.
0 blue street
-1
```

## 6 LADDER: Word Ladder

Lewis Carrol invented the game of word ladders. A word ladder is a sequence of words, where two adjacent words differ in a single character. Write a program that tests whether a list of words can be arranged so that they form a ladder. For example, the words “cord ward cold warm card” form a ladder in the following way: cold → cord → card → ward → warm.

The input begins with a line containing N, the number of test cases, followed by N lines containing word sequences (in lower case) separated by a blank. Each of words on a given line will be the same length (at least two characters long) and there will be at least two words on each line. For each of the N lines, print ‘yes’ if a word ladder is found, ‘no’ otherwise.

**Sample Input/Output**

Input

```
3
cord ward cold warm card
stone shone aloof chine chins coins
coins chine chins shone shine stone
```

Expected Output

```
yes
no
yes
```