

Programming Problems

16th Annual Computer Science Programming Contest

Department of Mathematics and Computer Science
Western Carolina University
April 5, 2005

Criteria for Determining Scores

Each program should:

1. execute without error, solve the appropriate problem efficiently, and satisfy all the conditions specified in the problem statement,
2. follow good programming style conventions such as avoiding confusing and unnecessary code,
3. be documented with comments and easily understandable variable names, and
4. start with an initial comment that includes
 - a) the name of the team's school,
 - b) the team's number, and
 - c) the names of the team's members.

An example such comment is

```
// Somewhere High School, Team 2, Susan Smith, Shelley Jones, and Sandy Hart
```

The score assigned to each program will be based upon the extent that the program satisfies the properties listed above. The total accumulated score for all four programs represents the team score. All the programs have the same weight in the scoring.

Administrative Notes

- 1) Each team may use two Java reference books. Each team may also look at the Java API on the web; however, no other site on the web can be used.
- 2) Each team may use up to **two** computers.
- 3) **Inappropriate code comments will result in a lower score and are grounds for disqualification for an award.**

Programming Notes

General

- Where appropriate, you should do input verification. In other words, if the user is supposed to input, for example, a positive number, an appropriate error message should be displayed when the user does not enter a positive number. The user should then get additional chances to enter valid input.
- The computers in the electronic classrooms have DeepFreeze on them. DeepFreeze deletes all new files when the computer is rebooted except for files that are in the My Documents folder. Consequently, you should save your BlueJ files in the My Documents folder on your computer.

Java

The version of Java installed in the electronic classrooms is the SDK 1.4.2. Java 5 (that is, JDK 1.5) is not installed.

Console I/O (i.e. keyboard input & screen output) is the expected form of input/output. Use `System.out.println` (or `System.out.print`, as appropriate) for output. The following information may be useful in implementing keyboard input:

- First, make sure that the following `import` statement is included at the very top of the code:

```
import java.io.*;
```

- Before doing any keyboard input, declare a `BufferedReader` object (named `br` in the following example) tied to `System.in`:

```
BufferedReader br = new BufferedReader( new
InputStreamReader( System.in ) );
```

- Use this object to read `Strings` (one line at a time) from the keyboard; for example:

```
System.out.println("Enter one line of input:");
String s = br.readLine();
System.out.println("Enter another line of input:");
String t = br.readLine();
```

The above fragment would read two lines from the keyboard; `s` references the `String` that is the first line of input, while `t` references the second line.

- Use `Integer.parseInt` or `Double.parseDouble` to convert any `Strings` to `ints` or `doubles`, respectively. Reading a line of input and converting it to a single numeric type can be done in one statement; for example:

```
int i = Integer.parseInt( br.readLine() );
```

- If a line of input has NUM_ELTS numeric values you can use a StringTokenizer to divide the line into tokens and then convert each token into the numeric value; for example:

```
import java.io.*;
import java.util.*;
...
StringTokenizer line = new StringTokenizer("");
try {
    line = new StringTokenizer(br.readLine());
} catch (Exception e) {}
for (i = 0; i < NUM_ELTS; i++)
    array[i] = Integer.parseInt(line.nextToken());
```

- Finally, any methods that contain a call to readLine (as well as methods that call a method that calls readLine, etc.) need a ``throws Exception" clause in the method header, unless you want to deal with Exception handling and place the readLine call in a try...catch block. (This type of Exception handling is not an expectation for the contest, so it is recommended that you just include the ``throws Exception" clause.) For example:

```
public static void main( String[] args ) throws Exception
{
    ... // call to readLine somewhere in here
}
```

- There is a bug in how BlueJ handles console I/O. Before your first readLine statement you must have a System.out.println statement. The System.out.println call tells BlueJ to open a console window; BlueJ is not smart enough to do this if a call is made to readLine first.

Problem 1: Encryption

A company wants to transmit data over the telephone line, but they are concerned that their lines are tapped. All of their data is transmitted as four-digit integers. They have asked you to write a program that encrypts their data so that it may be transmitted more securely. Your program should read a four-digit integer and encrypt it as follows:

Replace each digit by $((\text{digit} + 7) \bmod 10)$. Then, swap the first digit with the third, swap the second digit with the fourth, and print the encrypted integer.

An example session (input is italicized):

Enter a Four Digit Number: *6254*

The Encrypted Number is: **2139**

Problem 2: Degree of Inversion

Compute the extent to which an array of four integers is out of sorted order with sorted order meaning that the smallest index has the smallest value, the second index has the second smallest value, and so on until the largest index has the largest value. The metric for computing the degree of inversion is defined as follows. For each array index i count how many elements at larger indexes have values strictly smaller than the value of the element at index i . The degree of inversion is the sum of this count over all indexes. Notice that if the array is sorted, then the degree of inversion is 0. Your program output must look like the example output shown below. User input is shown in italics and program output is shown in boldface. You can assume that the array size is four.

Example Program Run 1

Please enter the array values: *4 3 2 1*

The degree of inversion is: **6**

Example Program Run 2

Please enter the array values: *1 2 3 4*

The degree of inversion is: **0**

Problem 3: Pseudo-Random Numbers

Computers normally cannot generate really random numbers, but frequently are used to generate sequences of pseudo-random numbers. These are generated by some algorithm, but appear for all practical purposes to be really random. Random numbers are used in many applications, including simulation.

A common pseudo-random number generation technique is called the *linear congruential method*. If the last pseudo-random number generated was L , then the next number is generated by evaluating $(Z \times L + I) \bmod M$, where Z is a constant multiplier, I is a constant increment, and M is a constant modulus. For example, suppose Z is 7, I is 5, and M is 12. If the first random number (usually called the *seed*) is 4, then we can determine the next few pseudo-random numbers are follows:

Last Random Number, L	$(Z \times L + I)$	Next Random Number, $(Z \times L + I) \bmod M$
4	33	9
9	68	8
8	61	1
1	12	0
0	5	5
5	40	4

As you can see, the sequence of pseudo-random numbers generated by this technique repeats after six numbers. It should be clear that the longest sequence that can be generated using this technique is limited by the modulus, M .

In this problem you will be given sets of values for Z , I , M , and the seed, L . Each of these will have no more than four digits. For each such set of values you are to determine the length of the cycle of pseudo-random numbers that will be generated. But be careful -- the cycle might not begin with the seed!

Input

Each input line will contain four integer values, in order, for Z , I , M , and L . The last line will contain four zeroes, and marks the end of the input data. L will be less than M .

Output

For each input line, display the case number (they are sequentially numbered, starting with 1) and the length of the sequence of pseudo-random numbers before the sequence is repeated.

An example session:

Sample Input

```
7 5 12 4
5173 3849 3279 1511
9111 5309 6000 1234
```

```
1079 2136 9999 1237
0 0 0 0
```

Sample Output

```
Case 1: 6
Case 2: 546
Case 3: 500
Case 4: 220
```

Problem 4: Zipper

Given three strings, you are to determine whether the third string can be formed by combining the characters in the first two strings. The first two strings can be mixed arbitrarily, but each must stay in its original order.

For example, consider forming "tcaete" from "cat" and "tree":

String A: cat
String B: tree
String C: tcaete

As you can see, we can form the third string by alternating characters from the two strings. As a second example, consider forming "catrtee" from "cat" and "tree":

String A: cat
String B: tree
String C: catrtee

Finally, notice that it is impossible to form "cttaree" from "cat" and "tree".

The Input

The first line contains a single positive integer from 1 through 1000. It represents the number of data sets to follow. The processing for each data set is identical. The data sets appear on the following lines, one data set per line.

For each data set, the line of input consists of three strings, separated by a single space. All strings are composed of upper and lower case letters only. The length of the third string is always the sum of the lengths of the first two strings. The first two strings will have lengths between 1 and 200 characters, inclusive.

The Output

For each data set, print:

Data set n: yes

if the third string can be formed from the first two, or

Data set n: no

if it cannot. Of course n should be replaced by the data set number. See the sample output below for an example.

An example session:

Sample input

3
cat tree tcræte
cat tree catrtee
cat tree cttaree

Sample output

Data set 1: yes
Data set 2: yes
Data set 3: no