ACM Programming Contest
Dixie State College
February 26, 2011

Do NOT turn this page until `http://time.gov/` says it is

10:00 am Mountain Standard Time!

Rules

1. Each team consists of up to three students.
2. Each team may use one computer.
3. Teams may use any printed references.
4. Teams may not use any electronic aids, including the internet. The help system built in to your programming environment is okay.
5. High school teams and college teams compete and are judged separately (unless otherwise requested).
6. All solution must be written in C++ or Java. Python is also permitted for high school teams, and for college teams where no member has completed CS 3005 or any other course that requires C++.
7. The team with the most correct solutions wins.
8. Ties will be broken using a time score:
   (a) Each time a team submits a correct solution, the number of minutes that have elapsed since the beginning of the competition is added to the time score.
   (b) For each incorrect solution submitted, a 20 minute penalty is added to the time score, but only if the team eventually submits a correct solution to that problem.
   (c) Multiple penalties will be added for multiple incorrect solutions to the same problem.
9. The input for each problem comes from standard in. This goes by the names `cin` and `STDIN` in C++, `System.in` in Java, and `sys.stdin` in Python. Other methods may also exist for receiving input from standard in for each language.
10. The output for each problem should be sent to standard out. This goes by the names `cout` and `STDOUT` in C++, `System.out` in Java, and `sys.stdout` in Python. Other methods may also exist for sending output to standard out for each language.
11. The output of submitted solutions must exactly match the output of the reference solution, down the the last character. Whitespace differences matter. Any other output, including debugging output, may cause an otherwise correct solution to be marked as incorrect. Each problem statement with example input and output shows exactly where newline characters are placed and where spaces are appropriate.
12. Solutions have a 10 second time limit. Any solution that runs longer than that will be considered incorrect.
13. If you need to handle last-minute registration details, please arrive at 9:00 am. Otherwise, please arrive at 9:30 am to find a computer, complete the sample problem, and sign in to the judging system.
14. The contest begins at 10:00 am and ends at 3:00 pm.

Contents

1 Abundant Numbers

Consider a number n, and take the sum of all of n's positive divisors, i.e., all numbers that divide evenly into n (including 1 and n itself).

If this sum is greater than $2n$, then we call n and *abundant* number. If the sum is equal to $2n$, we call n a *perfect* number. If the sum is less than $2n$, we call n a *deficient* number.

Read a list of positive integers, one per line, and write out the same number, labeling it as abundant, perfect, or deficient.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
1¶
2¶
3¶
4¶
6¶
12¶
15¶
18¶
20¶
28¶
```

Sample Output

```
1 is deficient¶
2 is deficient¶
3 is deficient¶
4 is deficient¶
6 is perfect¶
12 is abundant¶
15 is deficient¶
18 is abundant¶
20 is abundant¶
28 is perfect¶
```

2 Farmer Fox Fowl Feed

In a classic river crossing problem, a farmer is traveling with a fox, a fowl, and a bag of feed. The farmer reaches a river and must cross in a boat. The boat is so small that the farmer can carry at most one item on each crossing. If the farmer is not present, the fox will eat the fowl, and the fowl will eat the feed. The farmer must figure out a sequence of crossings to get everything across without anything being eaten.

In this problem you will write a program that checks a sequence of problem states and reports if it is correct or incorrect.

We represent the state of the system with four characters. The first character is "F" if the farmer is on the left side of the river, or "0" (zero) if the farmer is on the right side of the river. The second character is "F" or "0" depending on the location of the fox. The third and fourth characters are the same and represent the fowl and bag of feed respectively.

Examples:

- `F0F0` represents the farmer and fowl on the left side, and the fox and feed on the right side.
- `0FF0` represents the fox and fowl on the left side, and the farmer and feed on the right side. (The fox eats the fowl in this case.)

A sequence is a space separated list of states. For the sequence to be correct, it must begin with all items on the left side and end with all items on the right side. Also, each state must not allow the fowl or feed to be eaten. Finally, each transition between states must be legal. In other words, the farmer must move across the river and either take one item or no items with him. See the sample input below for examples.

The input consists of one sequence per line. Each line is a space separated list of states. Each line will have at most 12 states. There will be at most 20,000 lines in the input file. The input terminates with end of file.

The output will have one line per input line. The output line must begin with either "correct" or "incorrect" followed by a space, followed by the state sequence.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
FFFF 0F0F FF0F 000F F0FF 00F0 F0F0 0000¶
FFFF 0F0F FFFF 0F0F FF0F 000F F0FF 000F F0FF 00F0 F0F0 0000¶
FFFF 0FFF FFFF 0FF0 FFF0 0F00 FF00 0000¶
FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF 00FF¶
FF0F 0F00 FF00 0F00 FFF0 00F0 F0F0 0000¶
FFFF F00F FFF0 000F 0000¶
FFFF 0F0F FF0F 0F0F FF0F 0F00 FFF0 00F0¶
00FF F0FF 00F0 F0FF 00F0 F0F0 0000¶
FFFF F0F0 000F 00F0 000F 0000¶
```

Sample Output

```
correct FFFF 0F0F FF0F 000F F0FF 00F0 F0F0 0000¶
correct FFFF 0F0F FFFF 0F0F FF0F 000F F0FF 000F F0FF 00F0 F0F0 0000¶
incorrect FFFF 0FFF FFFF 0FF0 FFF0 0F00 FF00 0000¶
incorrect FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF 00FF¶
incorrect FF0F 0F00 FF00 0F00 FFF0 00F0 F0F0 0000¶
incorrect FFFF F00F FFF0 000F 0000¶
incorrect FFFF 0F0F FF0F 0F0F FF0F 0F00 FFF0 00F0¶
incorrect 00FF F0FF 00F0 F0FF 00F0 F0F0 0000¶
incorrect FFFF F0F0 000F 00F0 000F 0000¶
```

3 Interceptor

While working on constructing a space station in the midst of the intergalactic void your ship's sensors have detected another vessel moving at near the speed of light. You are extremely curious and concerned. This could be any number of possible new threats.

You have been ordered to get a better sensor reading, but the enemy vessel is moving so fast that your sensors cannot compensate and you must provide the ship's pilot with an appropriate course to parallel the other ship to get a reading. You are constrained by the fact that your ship is not carrying enough fuel to simply accelerate and wait for the ship to catch up, you must begin acceleration at the absolute latest moment to conserve fuel for the trip home.

Each line of input contains the parameters for a single problem. Each one supplies the speed of the other ship, its current distance, the magnitude of acceleration of which your own ship is capable, and the minimum speed required at interception to gain meaningful readings. For each problem, produce an output which is the truncated integer number of seconds to wait before beginning the acceleration.

All distances are in light-seconds, all speeds are given as percentages of the speed of light, all times are in integer seconds, and acceleration is measured as a percent of lights-peed per second. If it is not possible to accelerate in time to reach the necessary speed before the ship passes, print NA.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
.99 300 .01 .5¶
.80 10 .1 .4¶
.80 10 .001 .4¶
```

Sample Output

```
253¶
9¶
NA¶
```

4 Factorial Zeros

Write a program that determines the number of trailing zeros when taking the factorial of a given number. The factorial of n is the product of all integers from $1 \ldots n$.

The input has one number per line. The input numbers will be between 2 and 10 million inclusive.

The output is the number of trailing zeroes for the factorial of each of the input numbers.

Note: The factorial of 7 is 5040, which contains 2 zeros, but only 1 is trailing.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
2¶
7¶
10¶
24¶
```

Sample Output

```
0¶
1¶
2¶
4¶
```

5 Shortest Path

You are a member of a secret Air Force group dedicated to the study of an ancient alien device capable of creating artificial wormholes through space to identical devices placed throughout the galaxy and neighboring galaxies.

With recent budget cuts your superiors are looking to save on the project's power bill.

The device consumes power proportional to the square of the distance to the target. Specifically, the device consumes one terawatt for targets one light-year away, four terawatts for targets two light-years away and nine Terawatts for targets three light-years away and so on.

Given the network of devices in the galaxy, you realize that the power consumption could be optimized by jumping to a series of targets rather than jumping directly to a distant target.

Given an input file which describes the galaxy in the form

```
n
T0, x0, y0, z0
T1, x1, y1, z1
.
.
.
Tn, Xn, Yn, Zn
Ta
```

Where `n` is the number of targets in the universe less than or equal to twenty, `Tn` is the name of a target and `Xn`, `Yn`, `Zn` are the position of the target in light-years relative to Earth within a diameter of one hundred light-years. `Ta` is the name of one of the targets on a line by itself.

Find the most power-efficient route to `Ta`

Produce the output in the following format:

```
Ta, Pa, Ta1, Ta2, Ta3...
```

Where `Ta` is the label of the target, `Pa` is the total power of the trip to two decimals and `Ta1` ... `Tan` are the steps in the flight including the target, listed by label.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
4¶
Abydos, 0.00, 0.00, 14.38¶
Tolanna, 12.22, 38.21, 12.04¶
Chulak, -2.34, 52.60, 0.39¶
Taonas, 15.73, -23.3, -12.50¶
Chulak¶
```

Sample Output

```
Chulak, 2309.08, Tolanna, Chulak¶
```

6 Pretty Printer

You have been given a set of simple HTML files to work with, but they are poorly formatted and it is difficult to read them. Write a program to format them nicely.

Your input will consist of nested HTML tags, each of which is in one of two formats:

1. `<tagname>...</tagname>`
2. `<tagname />`

Each "`tagname`" can be any identifier consisting only of lower-case letters. Tags do not have attributes or contain anything except other tags, which may be nested where "..." is displayed. Tags that do not contain anything (after whitespace is removed) should be rewritten into the second form.

Note that tags must be nested correctly. The following is correct:

```
<ul>  <li><p><img /></p></li>  <li><p><br /></p></li>  </ul>
```

But the following is illegal:

```
<ul>  <li><p><img /></li>  <li><br /></p></li>  </ul>
```

The `<p>` tag opens inside the first `<li>` tag, but it it does not close inside the same `<li>` tag.

Your input consists of a single tag, with an arbitrary number of nested tags. Arbitrary whitespace can occur in the contents of a tag, though never inside the tag itself except the single space in a singleton tag (e.g., whitespace such as `< p>` is illegal). Whitespace in the contents of a tag should be ignored. Your task is twofold:

1. Determine if the input is valid. A tag name that is not a sequence of lower-case alphabetic characters is invalid, a malformed tag (something not matching one of the two forms given earlier), or incorrectly nested tags constitute invalid input. If you detect anything that makes the input invalid, your output should be the word "invalid" followed by a newline.
2. For valid input, format the output so that each tag (opening, closing, or singleton) is on its own line. Each time a tag opens, all tags inside it should be indented an additional two spaces, with the closing tag lining up with the opening. Singleton tags do not cause a change in the indentation.

Note that input lines may be arbitrarily long, as many HTML generators do not insert newlines. You should strip all whitespace that is present in the input, and reformat it exactly as described.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input I

```
<ul>  <li><p><img /></p></li>  <li><p><br /></p></li>  </ul>¶
```

Sample Output I

```
<ul>¶
  <li>¶
    <p>¶
      <img />¶
    </p>¶
  </li>¶
  <li>¶
    <p>¶
      <br />¶
    </p>¶
  </li>¶
</ul>¶
```

Sample Input II

```
<ul>  <li><p><img /></p>< /li>  <li><p><br /></ p></li>  </ul>¶
```

Sample Output II

```
invalid¶
```

7 Series of Numbers

In this problem you will be presented with a series of numbers. Your problem is to decide which series was used and then to give the next number in that series. There are five possible series that could be used:

1. Add a constant—in this series each number is generated by adding a fixed constant to the previous number, e.g. $1, 2, 3, 4, 5, 6$. In this example the constant is 1.
2. Subtract a constant—in this series each number is generated by subtracting a fixed constant from the previous number, e.g. $12, 10, 9, 8, 6, 4, 2$. In this example the constant is 2.
3. Multiply by a constant—in this series each number is generated by multiplying the previous number with a fixed constant, e.g. $4, 8, 16, 32, 64, 128$. In this example the constant is 2.
4. Divide by a constant—in this series each number is generated by dividing the previous number by a fixed constant, e.g. $128, 64, 32, 16, 8, 4$. In this example the constant is 2.
5. Fibonacci sequence—in this series each number is generated by summing the previous two numbers, e.g., $1, 1, 2, 3, 5, 8$.

The numbers and the constants used will not exceed $1,000,000,000$. There will be no numbers or constants less than or equal to 0. All numbers and constants will be integers.

The input consists of lines of text, each of which will have five numbers.

The output also consists of lines of text. The first five numbers on each line should be the series from the input, and the last number will be the sixth number in that series.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
1 2 3 4 5¶
2 4 6 8 10¶
1 1 2 3 5¶
50 45 40 35 30¶
512 256 128 64 32¶
3 9 27 81 243¶
32000 32003 32006 32009 32012¶
```

Sample Output

```
1 2 3 4 5 6¶
2 4 6 8 10 12¶
1 1 2 3 5 8¶
50 45 40 35 30 25¶
512 256 128 64 32 16¶
3 9 27 81 243 729¶
32000 32003 32006 32009 32012 32015¶
```

8 Stock Visualization

Assume that you are given a list containing the historical valuation of a stock's price for some number of days. The stock price for each day is on a single line of input. Write a program to display the stock values graphically instead of just printing the values. Round each stock price to the nearest integer, and print that number of *'s, separated by spaces. In order for us to see things easier, for each multiple of 10 you should print a '|' rather than a '*'.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
35.36¶
26.69¶
25.08¶
22.40¶
23.51¶
24.29¶
27.18¶
22.31¶
21.79¶
23.30¶
```

Sample Output

```
* * * * * * * * * | * * * * * * * * * * | * * * * * * * * * * | * * * * *¶
* * * * * * * * * | * * * * * * * * * * | * * * * * *¶
* * * * * * * * * | * * * * * * * * * * | * * * * *¶
* * * * * * * * * | * * * * * * * * * * | * *¶
* * * * * * * * * | * * * * * * * * * * | * * * *¶
* * * * * * * * * | * * * * * * * * * * | * * * *¶
* * * * * * * * * | * * * * * * * * * * | * * * * * * *¶
* * * * * * * * * | * * * * * * * * * * | * *¶
* * * * * * * * * | * * * * * * * * * * | * *¶
* * * * * * * * * | * * * * * * * * * * | * * *¶
```

9 Clueless

A popular logic game pits players against each other to be the first to find out who committed the crime (from a set of suspects), which weapon was used (from a set of possible weapons), and which room was the location of the crime (from a set of possible rooms).

The game uses a deck of cards where there is one card per suspect, possible weapon, and possible room. One of the suspect cards, one of the possible weapon cards, and one of the possible room cards are selected at random and stored in the "cellar". The rest of the cards are randomly distributed to the players of the game. The first player to correctly identify the three cards in the cellar is the winner.

For this problem you will create a program that creates propositional logic statements that help to deduce the cards that are in the cellar. There are two types of statements you need to create.

1. Each card in a group must be held by at least one of the players, or be located in the cellar.
2. Each card in a group must be held by at most one of the players, or be located in the cellar.

For example, if we have two players, `PlayerA` and `PlayerB`, and two suspects `SuspectD` and `SuspectE`, then you would represent the first type of rule with the following two statements.

```
PlayerA_SuspectD | PlayerB_SuspectD | Cellar_SuspectD
PlayerA_SuspectE | PlayerB_SuspectE | Cellar_SuspectE
```

Note that there is one statement per suspect, with 3 elements because there are 3 possible locations for the suspect card.

The second type of rule is represented with these statements.

```
PlayerA_SuspectD => (!PlayerB_SuspectD & !Cellar_SuspectD)
PlayerB_SuspectD => (!PlayerA_SuspectD & !Cellar_SuspectD)
Cellar_SuspectD => (!PlayerA_SuspectD & !PlayerB_SuspectD)
PlayerA_SuspectE => (!PlayerB_SuspectE & !Cellar_SuspectE)
PlayerB_SuspectE => (!PlayerA_SuspectE & !Cellar_SuspectE)
Cellar_SuspectE => (!PlayerA_SuspectE & !PlayerB_SuspectE)
```

Note that there are 3 statements per suspect, because the card may be located in one of 3 possible locations.

The same kinds of statements can be generated for weapons and for rooms.

The input contains 4 lines of text. The first line is a space separated list of players in the game. The second line is a space separated list of suspects. The third line is a space separated

list of weapons. The fourth line is a space separated list of rooms. There is only one problem in the input file. There will be at most 26 players, 26 suspects, 26 weapons, and 26 rooms.

The output contains text for each of the statements generated. The order of the lines and whitespace is important. The suspect statements come first, followed by the weapons statements, followed by the room statements. In each group of statements, the "must be held by at least one" statements come before the "must be held by at most one" statements. Within the statements the order of the items in the input lines should be preserved in creating output. For example, `PlayerA` comes before `PlayerB` because `PlayerA` was first on the line of the input file.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
PlayerA PlayerB¶
SuspectD SuspectE¶
WeaponG WeaponH¶
RoomJ RoomK¶
```

Sample Output

```
PlayerA_SuspectD | PlayerB_SuspectD | Cellar_SuspectD¶
PlayerA_SuspectE | PlayerB_SuspectE | Cellar_SuspectE¶
PlayerA_SuspectD => (!PlayerB_SuspectD & !Cellar_SuspectD)¶
PlayerB_SuspectD => (!PlayerA_SuspectD & !Cellar_SuspectD)¶
Cellar_SuspectD => (!PlayerA_SuspectD & !PlayerB_SuspectD)¶
PlayerA_SuspectE => (!PlayerB_SuspectE & !Cellar_SuspectE)¶
PlayerB_SuspectE => (!PlayerA_SuspectE & !Cellar_SuspectE)¶
Cellar_SuspectE => (!PlayerA_SuspectE & !PlayerB_SuspectE)¶
PlayerA_WeaponG | PlayerB_WeaponG | Cellar_WeaponG¶
PlayerA_WeaponH | PlayerB_WeaponH | Cellar_WeaponH¶
PlayerA_WeaponG => (!PlayerB_WeaponG & !Cellar_WeaponG)¶
PlayerB_WeaponG => (!PlayerA_WeaponG & !Cellar_WeaponG)¶
Cellar_WeaponG => (!PlayerA_WeaponG & !PlayerB_WeaponG)¶
PlayerA_WeaponH => (!PlayerB_WeaponH & !Cellar_WeaponH)¶
PlayerB_WeaponH => (!PlayerA_WeaponH & !Cellar_WeaponH)¶
Cellar_WeaponH => (!PlayerA_WeaponH & !PlayerB_WeaponH)¶
PlayerA_RoomJ | PlayerB_RoomJ | Cellar_RoomJ¶
PlayerA_RoomK | PlayerB_RoomK | Cellar_RoomK¶
PlayerA_RoomJ => (!PlayerB_RoomJ & !Cellar_RoomJ)¶
PlayerB_RoomJ => (!PlayerA_RoomJ & !Cellar_RoomJ)¶
Cellar_RoomJ => (!PlayerA_RoomJ & !PlayerB_RoomJ)¶
```

10 Count Rectangles

Given a two dimensional array of squares, count how many rectangles can be formed inside it.

For example, 9 rectangles can be found inside a 2×2 array of squares—four 1×1 rectangles, two 1×2 rectangles, two 2×1 rectangles, and one 2×2 rectangle.

Likewise, 18 rectangles can be found inside a 2×3 array of squares.

Write a program that determines how many rectangles can be found inside a set of $m \times n$ arrays of squares, where m and n are between 1 and 100 inclusive

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
2 2¶
2 3¶
3 2¶
```

Sample Output

```
9¶
18¶
18¶
```

11 Next Prime

In this problem you will be presented with a number n. Your job is to produce the next prime number greater than n. A prime number is a number that is evenly divisible only by itself and 1. The number 4 is *not* prime since it can be evenly divided by 2. The following are examples of prime numbers: $2, 3, 5, 7, 11, 13, 17, 19, \ldots$

The numbers used will not exceed 32767. There will be no numbers less than or equal to 0. All numbers will be integers.

The input consists of lines of text. Each line will have one number.

The output also consists of lines of text. The first number is the number given. The second number is the next prime greater than the given number.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
1¶
2¶
3¶
5¶
100¶
32000¶
```

Sample Output

```
1 2¶
2 3¶
3 5¶
5 7¶
100 101¶
32000 32003¶
```