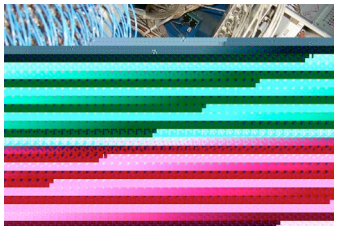
Baldwin-Wallace College

2004 Programming Competition

Contest Problem Set



PROBLEM 1 - ERROR CHECKING

Network Communications are essential in the modern day age. Small businesses and large, wealthy corporations all rely on their networks to keep their customers and employees informed. Often, a breakdown in the network can cost the company dearly. That is why companies and other important institutions try to do their best to protect their networks, whether it be from hackers or equipment failure.

At the bottom of any communication over any network is the transmission of a series of electrical signals over a particular media, which are then at the receiving node interpreted into 0's and 1's, the machine language. However, not every transmission goes unhindered. For example, EMI, or electro-magnetic interference can cause the transmission to go bad. How do we know (or in this case, how does the receiving node know) if the message is good or bad? Several methods have been developed over the years to check the transmission for errors. Some of them are Parity Checks, Longitudinal Redundancy Checks, Cyclic Redundancy Checks, and Checksums.

In particular, the Checksum calculation begins by adding the ASCII decimal face value of every character in the message block. For example, "Hello World" will have an ASCII value of 1052 once the individual characters' values are added. Then, this number is divided by 255. Finally, the remainder is determined and sent along with the rest of the message. It is this remainder that is to be verified by the receiving computer.

The input will begin with a number that tells you how many message blocks you have to process. Then, each two lines of input represent one case. The first line contains the actual message and the second is the checksum remainder. You are to process each message and determine weather the message was properly transmitted.

SAMPLE INPUT
3
tool
191
dinosaurs
245
Stop
87

SAMPLE OUTPUT
tool was transmitted properly.
dinosaurs was not transmitted properly.
Stop was not transmitted properly.

PROBLEM 2 - TIC-TAC-TOE

Tic-Tac-Toe is a relatively simple game played by two competing players on a 3x3 grid. Both players have a unique symbol – an 'x' or an 'o' – and their goal is to fill an entire row, column, or diagonal of the board with their symbol. Either player may initiate a game of Tic-Tac-Toe. Play continues until either one player succeeds in filling a row, column, or diagonal OR the board is filled and neither player has succeeded - in other words, the game ends in a tie.

The history of Tic-Tac-Toe (previously called Noughts and Crosses in the UK) may date back all the way to the Roman Empire. Tic-Tac-Toe boards have been found engraved into various surfaces throughout the empire. Mathematically, there are 255,168 possible legal games of Tic-Tac-Toe (if you count all the possible legal orders of symbols being placed on the board rather than just final board configurations). However, there are a mere 19,683 final board configurations (including legal, illegal, and incomplete games with or without blanks on the board). Your job is to read in a certain number, *n*, of Tic-Tac-Toe boards found in ancient Rome and determine the status of each game.

For our purposes, the boards will consist of x's, o's, and _'s, where an _ represents a position on the grid with no symbol on it. You will separate the boards into 5 categories and report into which category each board fell:

1. "Game *k* was invalid." -- This occurs when it is clear the rules in the first paragraph above were not followed during the playing of game *k*.
2. "Game *k* was won by X."
3. "Game *k* was won by O."
4. "Game *k* was incomplete." -- This occurs when a legal game is still in progress.
5. "Game *k* ended in a tie." -- This occurs as explained in the first paragraph.

The first line of the input will consist of a single integer, *n*, which represents the number of instances of the problem (how many boards will follow). Each set of three lines following the first line makes up one non-empty Tic-Tac-Toe board as described above (with x's, o's, or _'s). Each line of the output file should match one of the five possible categories as listed above, with the board number inserted for *k*.

The sample input and output appear on the next page.

PROBLEM 2 - TIC-TAC-TOE (Continued)

SAMPLE INPUT
3
xxo
xoo
x__
x_x
_x_
___
oox
xxo
oxo

SAMPLE OUTPUT
Game 1 was won by X.
Game 2 was invalid.
Game 3 ended in a tie.

PROBLEM 3 - TAX

Just like American citizens, the businesses and the corporations of America have to pay annual income taxes. For this problem, assume that the amount of the tax liability depends solely on the income level of a business. Thus, if a business sells $100,000 worth of products and the income tax rate is 15%, the tax liability for this particular business will be $15,000.

You are to write a program that calculates the annual income taxes of some businesses. The first line of the input will contain a positive integer, *n*, indicating the number of data sets (businesses) that follow. The first line of each data set contains three values: a single word, *name*, that identifies the particular business or firm, the tax percent rate, *r*, and the number of different products, *p,* that the business produced in the past year. These last two values are positive integers. Each of the next *p* lines will contain a string representing the name of a product and an integer stating the revenue earned from selling that product. You are to calculate the tax liability, *l*, of the business and report the result in the following format:

name income tax liability is $*l*

The tax liability should be truncated to two decimal places.

SAMPLE INPUT
```
2
John 15 3
Bolts 23500
Nuts 18765
Hammers 7890
L.P.T. 12 4
Flags 100000
Towels 15345
Sheets 47351
Napkins 31321
```

SAMPLE OUTPUT
```
John income tax liability is $7523.25
L.P.T. income tax liability is $23282.04
```

PROBLEM 4 - BINARY NUMBERS

"Restate my assumptions: 1. Mathematics is the language of nature. 2. Everything around us can be represented and understood through numbers. 3. If you graph these numbers, patterns emerge. Therefore: There are patterns everywhere in nature." Thus says Maximillian Cohen a character from a mathematical movie Pi, made in 1998. While a small percentage of us can cope with numbers, most people are terrified by mathematics. The most common numeric system used throughout the world for everyday applications is called the decimal number system. So what does this system mean? This simply means that decimal system uses a base 10 number system.

The decimal system is not the only system by which numbers can be represented. In fields such as computer science, a binary numbering system is used, which is a base 2 system. It is called binary because all the numbers using this system can be represented by two digits: 0 and 1. A binary number looks like this 010011101001. To you this number may mean nothing, but entire computer's architecture is based on binary system. As an example take a look at number 1010. Reading the number from right to left the powers of 2 increase by 1, where each digit represents some power of 2 and the rightmost power of 2 is 0. We then add up all the numbers to get a final solution, which in this case is 10.
The input consists of a number n, meaning the number of computations that you will be asked to do. It is the followed by n pairs of two numbers. The first number is given in binary and the second in decimal. In the first example, the binary number 1010 is equal to $(0 * 2^0) + (1 * 2^1) + (0 * 2^2) + (1 * 2^3) = 2 + 8 = 10$. Now, add this to the decimal number "10" to get 20… which in binary converts back to 10100 $(0 * 2^0) + (0 * 2^1) + (1 * 2^2) + (0 * 2^3) + (1 * 2^4)$

SAMPLE INPUT
3
1010 10
1011101001 350
101010101 94

SAMPLE OUTPUT
10100
10001000111
110110011

PROBLEM 5 - PRIME FACTORIZATION

Every number can be sub divided into its prime factors. This process is called prime factorization. A prime is a number that cannot be divided by any other number but it self. For example number 234 can be written as 2x3x3x13. Two, three and thirteen are all prime numbers. Your job is to create an algorithm that will perform prime factorization for a given number.

Input will consist of one number n, followed by n numbers that need to be converted into their prime factors. The output consists the prime numbers in ascending order; between each number there is an 'x' to represent multiplication.

Sample input:
3
18
457
812

Sample output:
2x3x3
457
2x2x7x29

PROBLEM 6 - 2D CELLULAR AUTOMATA

An automaton (plural automata) is a self moving abstract machine. A cellular automaton is a specific type of automaton which is a dynamic system consisting up of a number of "cells" which can either be alive or dead. You will be writing a program that will produce the next generations for a two dimensional automaton given a certain set of rules. The rules for this system are as follows:

Rules:
1. If the current cell is dead and there exists 3 living cells surrounding the dead cell (in any of the eight directions) then it comes to life in the next generation.
2. Otherwise the cell remains dead.
3. If the current cell is alive and there exist more than 3 living cells surrounding the cell then the cell dies from being overcrowded.
4. If the current cell is alive and there exist less than 3 living cells surrounding the cell then the cell dies of loneliness.

The input will consist of an integer no greater than 75 representing the number of rows and columns in the automaton and then the initial state of the system. A living cell will be represented by a '1' and a dead cell will be represented by a '0'. There will be multiple sets of data and will end when the number of rows and columns equals 0.

The program will output the two dimensional automaton with a blank line between each sets of data.

The sample input and output appear on the next page.

PROBLEM 6 - 2D CELLULAR AUTOMATA (Continued)

SAMPLE INPUT

```
10
0101011010
1100000000
0101000100
1111001110
1111000100
1010101010
1010101010
1010101010
1010101010
1010101010
4
0111
1111
1111
1111
0
```

SAMPLE OUTPUT

```
1010000000
1100101100
0000001110
0000101010
0000110000
1000000000
0000000001
0000000001
0000000001
0000000000

1001
0000
0000
1001
```