ACM Programming Contest
Dixie State College
Spring 2010

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

10:00 am Mountain Daylight Savings 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.
5. 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 2420.
6. The team with the most correct solutions wins.
7. 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.
8. 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.
9. 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.
10. 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.
11. Solutions have a 10 second time limit. Any solution that runs longer than that will be considered incorrect.
12. The contest begins at 10:00 am and ends at 3:00 pm.

Contents

1 Plot holes

You have recently been through a series of barely sensical events which have resulted in your possession of a powerful and advanced alien computer. You find a recording of many of the events stored in the alien computer in a "plot"-centric data format. The data is represented as a stream of hexadecimal data in which a set bit represents an event which is congruent and sensical in context.

Your task is to determine the number and severities of "holes" in the plot data which are defined as strings of OFF bits. (Remember, `1` in hexadecimal is made up of the bits `0001` and `F` is `1111`).

Input will consist of one or more lines of hexadecimal digits (capital letters) representing plot data up to 16,384 bits long. Each line represents one recording and should have one line of output.

Output will consist of four numbers space delimited. "Minor" plot holes are 1 off bit. "Major" plot holes are at least 8 continuous off bits. "Huge" plot holes are at least 256 continuous off bits. "Gargantuan" plot holes are 1,024 continuous off bits. A single hole may be made up of multiple types of holes, so 9 off bits is represented by a major plot hole and a minor plot hole together.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
FF00F00FFEFF00F$
0000000000000000000000000000000000000000000000000000000000000000E$
```

Example Output

```
1 3 0 0$
1 0 1 0$
```

2 A Slice of π

π (pronounced "pie") is a special numerical constant that is used commonly in mathematical formulas, particularly those that relate to circular shapes and motion. The actual value of π is irrational, which means that it consists of an infinite sequence of non-repeating digits. However, for most pratical purposes, an approximation of π will suffice. In this problem, you will create a program to calculate an approximation of π.

One formula for calculating π is as follows:

$$\pi = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1} = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11}\cdots$$

Your program will need to be able to calculate an approximate value of π, by using the first n terms of the summation displayed above. For example, if n is 0, then your approximation would be just the first term ($k = 0$), 4/1. If n is 4, then your approximation would be the first 5 terms ($k = 0$ to $k = 4$),

$$\frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9}$$

which has the value of 1052/315.

The input for your program will be several lines, with one number per line. The input number will be in the range $0 <= n <= 9$. The input number of -1 will indicate the end of the file, and your program should not process the -1.

The output for your program will be one line for each input number that you process. Each line will consist of a fraction written in the form "a/b" where "a" and "b" are integer numbers. Note that there are no spaces between the numbers and the "/". Also note that the output fraction must be reduced.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
0$
4$
7$
-1$
```

Example Output

```
4/1$
1052/315$
135904/45045$
```

3 Biased Coin

Suppose you are given a biased coin, i.e., one that is more likely to land on one side than the other when flipped. You can use it to make a fair, 50/50 decision even if you do not know how biased it is.

To do so, flip the coin twice in a row. If you get the same result each time, ignore the result and repeat (flip it twice again). If the two flips are different, take the first one as your answer.

You are given a sequence of potentially biased coin flips as input. Use them to produce an unbiased sequence of output flips. Output as many fair flips as you can. If you extra input that does not yield another fair flip, ignore it.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
heads$
heads$
tails$
heads$
heads$
heads$
tails$
heads$
heads$
tails$
tails$
heads$
tails$
tails$
tails$
```

Example Output

```
tails$
tails$
heads$
tails$
```

4 Unmitigated Emo

By feats of fantastic stupidity on the parts of others you have been trapped on an intergalactic alien space ship bound for parts unknown. The spectacular triteness of your companions is only matched by their angst. In order to prevent yourself from going postal and attempting to destroy the ship just to shut them up, you have been desperately searching the ships computers for something to distract you.

Through this search you have found a well written alien program with an interesting purpose. You have determined that given an audio recording of a conversation the program will classify each sentence into emotional categories and output a list of statements tokenized and tagged with a unique speaker identifier.

Given this information, you have decided to parse the output of the voice processor and hold a "most emo" contest with the remote hope of convincing your imbecile fellows to shut up.

Therefore, you have created the following equation based on the output of the program to calculate an "emo factor."

$$\frac{E \times E + W + S + M}{T + B + A \times A}$$

Where E is the number of `EMO` tokens for the speaker, W is the number of `WHI` tokens for the speaker, S is the number of `SYM` tokens for the speaker, and M is most `EMO+SYM+WHI` tokens for a speaker in one sentence. Also, T is the total number of sentences for the speaker, B is the number of sentences not containing a `EMO` or `SYM` and A is the number of `SAR` tokens for the speaker.

The input file will consist of at least three and no more than 1000 lines. Each line represents one sentence and will contain the user's unique id (you may assume at least three speakers) followed by all tokens for that sentence, comma delimited. A sentence may have multiple tokens and may have multiple instances of the same token and at most 100 tokens.

Valid Tokens:

- `EMO` (emotive statement)
- `SYM` (sympathetic statement)
- `CMD` (command statement)
- `DEC` (declarative statement)
- `REQ` (request statement)
- `INS` (insult statement)
- `CMP` (complement statement)
- `UNI` (unintelligible statement)
- `STU` (stupid statement)
- `SAR` (sarcastic statement)
- `WHI` (whine statement)

The output file will contain the top 3 speakers sorted by their emo factor, highest first, rounded to two decimal places of accuracy.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
X001, EMO, WHI$
X002, CMD, SAR$
X001, EMO, EMO, WHI, EMO, REQ$
X002, SAR, INS$
X001, WHI, WHI, EMO, EMO, EMO$
X003, EMO, EMO, EMO$
```

Example Output

```
X001 19.33$
X003 12.00$
X002 0.00$
```

5 Word Value

One can measure the "value" of a word by summing up the value of each letter, where 'a' is worth 1, 'b' is worth 2, through 'z' is worth 26. Capital letters also have values 1 through 26. All other characters in a word should be ignored.

The value of "cat" is thus $3 + 1 + 20 = 24$.

Your job is to read a bunch of words, 1 per line, and echo back all the words that have value 50 or 100, along with their respective values.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

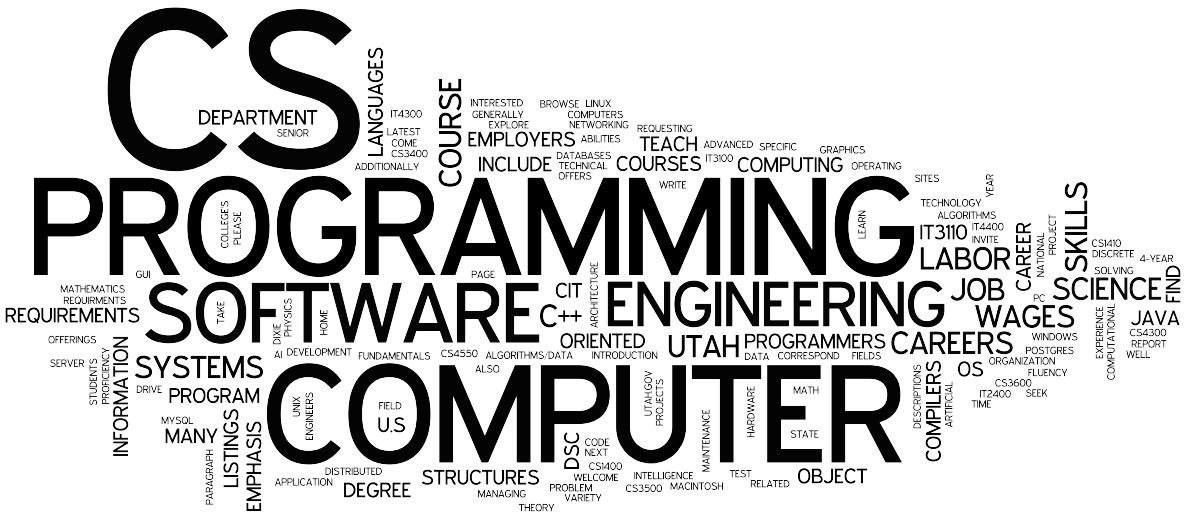
```
after123!$
cat$
Elephants$
easy$
garbage$
```

Example Output

```
after123! 50$
Elephants 100$
easy 50$
```

6 Word Clouds

"Word clouds" or "tag clouds" are visual representations of the frequency of words in a document. Size and color are used to make the display of the most frequent words more visible that the less frequent words. Here is an example of a cloud created from the computer science web page:



The first step in creating a word cloud is to count the number occurrences of each word in the document. For this task, you will create a word counting program.

The input for the program will be an ASCII text file. The output for the program will be an alphabetized list of all words that appear in the ASCII text file the count. The output will be one line per word in the format:

```
word 123
```

Where "word" is the word and "123" is the count. They are separated by a single space.

To determine the unique words, use the following rules:

1. All letters must be converted to lower case.
2. All digits must be left in place (they are treated like letters).
3. The apostrophe character (') must be removed. For example, "can't" becomes "cant".
4. All other punctuation and white space characters are used to separate words.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
The quick brown fox jumped over the lazy brown dog.  The$
dog couldn't figure out why.$
```

Example Output

```
brown 2$
couldnt 1$
dog 2$
figure 1$
fox 1$
jumped 1$
lazy 1$
out 1$
over 1$
quick 1$
the 3$
why 1$
```

7 Lexicographic Permutations

A permutation is an ordered arrangement of objects. For example, 3124 is one possible permutation of the digits 1, 2, 3, and 4. If all of the permutations are listed numerically or alphabetically, we call it lexicographic order. The lexicographic permutations of 0, 1 and 2 are:

$$012, 021, 102, 120, 201, 210$$

What is the nth lexicographic permutation of the digits 1, 2, 3, 4, and 5? Your program should take a single line as input (n) and return that lexicographic permutation. Each input number should be followed by a newline character. Sample input and output from 3 program runs are as follows.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input 1

```
1$
```

Example Output 1

```
12345$
```

Example Input 2

```
2$
```

Example Output 2

```
12354$
```

Example Input 3

```
100$
```

Example Output 3

```
51342$
```

8 Getting to Work

Bob has a great job, but he has a long drive to work. To keep things interesting he likes to take a different route to work each day. It is still important that he is on time to work, so he never backtracks.

City blocks in Bob's town are all perfect squares, and the city is a perfect square. Bob lives at the top left corner of the city and works in the bottom right corner of the city.

If the city was 2 square blocks (2×2 grid), there would be 6 routes Bob could take to work (without backtracking).

Problem

Write a program to calculate how many different routes Bob can take to get to work if his city was n square blocks ($n \times n$ grid). Each line of input represents a different city, and the output should contain one answer for each line of input.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
2$
3$
4$
```

Example Output

```
6$
20$
70$
```

9 Hit or Miss

Through a series of preposterous circumstances you have come to be on board an ancient alien ship hurdling through a distant galaxy. The ships power supplies are quickly failing and it drops into a solar system headed relatively close to a gas giant. You have a suspicion about the course the semi-intelligent computer has plotted. You must write a program to determine if the ship will impact the local star by simulating the next sixty days in one second intervals.

You may disregard all forces except the gravitational acceleration from the star and planet, including solar wind, atmospheric drag and other gravitational influences. You may disregard the motion of the planet relative to the star. You may disregard any interaction beyond the period of sixty days. The ship is assumed to be without propulsion. The star may be assumed to be a regular sphere with all of its mass located at the exact center. Margin of error for impact is to two decimal places. You may assume that the mass of the ship is irrelevant compared to the masses of the planet and star. You may assume that there is no impact with the planet. All motion is on the plane of the solar system, therefore there is no third dimension. All distances are in the same unit, all masses are in the same unit, all times are in seconds. Speed is measured in distance units per second.

The combination of all forces acting on a ship is equivalent to the sum of the vectors of those forces for that second.

The magnitude of gravitational force acting on the ship by each object can be determined by the following formula:

$$\text{units/second/second} = \frac{6.67438 \cdot \text{Mass}}{\text{Distance}^2}$$

Which is the gravitational constant multiplied by the mass of the object and divided by the square of the distance between the object and the ship.

The distance between two objects is:

$$\text{distance}(P1, P2) = \sqrt{(P2_x - P1_x)^2 + (P2_y - P1_y)^2}$$

Given a line defined by two points, A and B and a third point P you can find the closest point on the AB line to P with the following formula:

$x = A_x + u \cdot (B_x - A_x)$ and $y = A_y + u \cdot (B_y - A_y)$

where u is:

$$u = \frac{(P_x - A_x)(A_x - B_x) + (P_y - A_y)(A_y - B_y)}{(A_x - B_x)^2 + (A_y - B_y)^2}$$

The input consists of some number of lines of the following comma delimited values: the x, y coordinates of the star, the radius of the star, the mass of the star, the x, y coordinates of the planet, the mass of the planet, the initial x, y of the ship, the x, y velocity vector of the ships initial motion.

Each line of input has one corresponding line of output containing the strings "yes" or "no".

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
0,0,1,1000,20,20,1,30,30,-0.1,-0.1$
0,0,1,1000,20,20,1,10,30,1000,1000$
```

Example Output

```
yes$
no$
```

10 Last Names

Write a program that reads a list of names then outputs just the last names. Each line of the input contains a full name. It could contain any of the following combinations:

1. First and Last name
2. First, Middle, and Last name
3. First, Middle, Maiden, and Last name
4. First, Maiden, and Last Name

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
Jill Peterson$
Bill Gates$
Fred Brooks$
George W Bush$
Mary Joe Willams Green$
```

Example Output

```
Peterson$
Gates$
Brooks$
Bush$
Green$
```

11 Numeric Symbols

Given a string representing a new character set for a number system of base n, where n is the length of the string and the first character represents 0 and the last character represents $n - 1$, write a program to calculate the value of each number from decimal to base n (using the new character set).

The first line of input gives the symbols for the new numbering system, and each remaining line is a base 10 number to be converted to the new numbering system.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input 1

```
01$
1024$
384$
511$
9876$
```

Example Output 1

```
10000000000$
110000000$
111111111$
10011010010100$
```

Example Input 2

```
@#*%^&$
36$
612$
4248$
```

Example Output 2

```
#@@$
*&@@$
%#^@@$
```

12 Prime Triangles

A triple of numbers (a, b, c) is called a k-digit prime triangle if:

- a, b, and c are k-digit integers, and
- each concatenation $a \circ b$, $b \circ a$, $a \circ c$, $c \circ a$, $b \circ c$, $c \circ b$ is a prime number. (Formally, if x and y are k-digit integers, $x \circ y = 10^k \cdot x + y$.)

For example, the numbers $(1, 3, 7)$ form a 1-digit prime triangle because 13, 31, 17, 71, 37, and 73 are all prime numbers. The numbers $(4, 5, 6)$ do not form a prime triangle because 4 and 6 are not prime, and neither are concatenations of each. $(67, 3, 37)$ is a prime triangle.

For a given triple of numbers, determine if it is a prime triangle or not. Your program should take 3 numbers as input, separated by a single space, followed by a newline character. It should print out "True" if the inputs result in a prime triangle, otherwise it should print out "False".

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input 1

```
1 3 7$
```

Example Output 1

```
True$
```

Example Input 2

```
4 5 6$
```

Example Output 2

```
False$
```

Example Input 3

```
67 3 37$
```

Example Output 3

```
True$
```

13 Morse Code

Before the digital age, the most common "binary" code for radio communication was the Morse code. In Morse code, symbols are encoded as sequences of short and long pulses (called dots and dashes, respectively). The following table reproduces the Morse code for the alphabet, where dots and dashes are represented by ASCII characters "`.`" and "`-`":

A	`.-`	B	`-...`	C	`-.-.`	D	`-..`
E	`.`	F	`..-.`	G	`--.`	H	`....`
I	`..`	J	`.---`	K	`-.-`	L	`.-..`
M	`--`	N	`-.`	O	`---`	P	`.--.`
Q	`--.-`	R	`.-.`	S	`...`	T	`-`
U	`..-`	V	`...-`	W	`.--`	X	`-..-`
Y	`-.--`	Z	`--..`				

There needs to be a pause between each letter in order to decode the message.

Problem:

Suppose John finds a message written in Morse code and John wants to see what it says. Unfortunately, the person who recorded the message did not mark the pauses between letters. They did, however, separate each word. John needs some way to list all the possibilities for each word. For example, if one of the words was "`-.-.`", it could be decoded as "C", "KE", "NN", "NTE", "TAE", "TEN", "TETE", or "TR". If John had a way to make this list he could easily see that the word must be "TEN".

Help John by writing a program that reads a Morse code string and outputs all the possible ways of decoding the string. The different options must be listed in alphabetical order and in all capital letters.

Note: $ in the examples below represents a newline character, not a dollar sign character.

Example Input

```
--.-.$
```

Example Output

```
GN$
GTE$
MAE$
MEN$
METE$
MR$
QE$
TC$
TKE$
TNN$
TNTE$
TTAE$
TTEN$
TTETE$
TTR$
```

14 Monetary Decay

Las Vegas Casinos often boast their average return rates, such as 90%, or as high as 97%. A return rate of 90 percent means that if you played 1000 coins in their slot machine, on average, you'd leave with only 900. Of course, this is only an average. It is remotely possible that you could play 1000 coins and actually walk away with more than 1000.

A certain leading edge casino decides to simplify the experience of its gamblers, while at the same time cutting some of their own costs on purchasing expensive slot machines. When a gambler enters the casino, they ask him how many coins he intends to play, then they calculate his expected loss of money, based on their advertized rate of return. They then remove that number of coins (rounding up to the nearest coin) from the gamblers purse, and send him on his way.

A certain gambler decides this is a far more efficient use of his time than spending all day in the casino, and he hopes this new approach will help pacify his wife, who is constantly nagging him that he spends too much time gambling.

The gambler intends to take his monthly paycheck (all in coins), and return to the casino as often as possible, until all his money is gone.

Given an initial number of coins and a return rate as a percent, determine how many of these quick casino visits the gambler can enjoy before his purse is empty.

The input consists of several scenarios, one per line. The first number on the line is how many coins that gambler is able to get from his paycheck. The second number is the advertized return rate of a particular slot machine. The output should contain the number of visits that scenario allows before the gambler goes broke.

The number of coins will be more than zero, and the return rate will be between 0 and 100%, exclusive.

Sample Input

```
100 95.2$
1000 50$
```

Sample Output

```
45$
10$
```