Seventh Annual Quinnipiac University Programming Competition for High School Students

Quinnipiac University

April 30, 2011

Instructions

1. Submit a program by attaching all necessary files to an e-mail to `cscjudge`. Include the number of the problem you are solving in the subject line of your e-mail.

   Questions may also be addressed to `cscjudge`; put the word "Question" in the subject line, along with the number of the problem you are asking about if your question pertains to a particular problem.

2. You may assume the input is formatted as shown in the problem, and your program must work on such input. You may assume there are no tabs in the input.

3. Your feedback on a submitted problem will be e-mailed in a reply and will be limited to one of the following:

   (a) COMPILE ERROR: The program did not compile

   (b) PROGRAM NOT RUNNABLE: The program compiled, but did not run as a stand-alone program.

   (c) CRASH: The program crashed or threw an exception when it was run

   (d) TIMED OUT: The program did not finish in a reasonable amount of time

   (e) INCORRECT OUTPUT: Either the answer is incorrect, or it is not presented in the form specified in the problem.

   (f) ACCEPTED

   If your program crashes, produces incorrect output, or times out, no information will be given as to the input that caused the problem.

1 Wedding Gifts

Bob Cat was invited to a certain royal wedding in a foreign country (actually, he never received the invitation, but he assumes that was a mere oversight due to the busy schedules of the wedding planners).
The bride and groom requested that donations be made to charity in lieu of wedding gifts, and Bob has decided to follow this route. Unfortunately, the gifts are in a foreign currency, which Bob finds confusing. He wants to put the donation on a credit card, but the credit card also charges a fee, given by the larger of a percentage of the transaction cost, or a fixed amount. For example, the credit card company might charge the larger of 2.5% or $5.00. That charge is assessed after rounding the gift amount to the nearest cent.
Write a program which converts currency and tells Bob the amount that would be charged to his credit card. Your program should take one line of input in the format:

`<Foreign Currency> <Exchange Rate> <Percentage> <Minimum>`

and display the amount charged to the nearest cent. In the input, `<Foreign Currency>` represents the amount of the donation, `<Exchange Rate>` is the cost in dollars for one unit of the foreign currency, and `<Percentage>` and `<Minimum>` refer to the additional charges for the credit card company.
For example, the line

```
100.00 1.66648 2.5 5
```

means he gives 100.00 units of the foreign currency, each unit of that currency costs $1.66648, and the credit card company charges the larger of 2.5% of the gift or $5.00. In this case, the cost of the currency would be $166.65, and 2.5% of this amount is $4.17 (rounded off), so the company would charge an additional $5.00, since that amount is larger than 2.5%. The total that would appear on Bob's credit card would then be $171.65.
The following table gives sample inputs, along with the corresponding output for the program.

Input	Output
100.00 1.66648 2.5 5	171.65
100.00 1.66648 2.5 3	170.82
1000.00 1.66648 3 5	1716.47
1000.00 0.600067 5 5	630.07

2 Shopping List

Write a program which manages a shopping list. The input to the program is a series of lines, each beginning with either a plus sign (+) or a minus sign (−), followed by optional spaces (which should be ignored), followed by the name of an item to be added or removed from the list, and the last line of the input is simply an asterisk (*) on a line by itself.
The program then displays the remaining items on the list, **or the words "`Empty List`"** if the list contains no items. All items are added to the end of the list. Attempts to add an item already on the list, or to remove items not on the list are ignored, and you may assume there will never be more than 20 items on the list at any one time. You may also assume that the format of an item being removed (letter case, spacing, etc.) exactly matches the format used when the item was first added. For example, if we add chocolate chip cookies using the line

```
+ Chocolate Chip Cookies
```

the following lines would not appear later in the input:

```
- CHOCOLATE CHIP COOKIES
- chocolate chip cookies
- Chocolate  Chip   Cookies
```

because either the case or the spacing between words in the item have been changed. However, we could have any of these lines:

```
-Chocolate Chip Cookies
- Chocolate Chip Cookies
-   Chocolate Chip Cookies
```

in which the only difference from the line that added the cookies (other than changing the plus sign to a minus sign) is the number of spaces that are ignored in front of the initial 'C'.
For example, if the input is:

```
+ Chocolate Chip Cookies
-big sandwich
+ apples
+ oranges
+  apples
+   apples
+big sandwich
-   oranges
+Chocolate Chip Cookies
-   Chocolate Chip Cookies
*
```

only the boldfaced lines above have an effect, so the corresponding output is

```
apples
big sandwich
```

3 Mean and Standard Deviation

Write a program which takes a collection of numbers as input and prints out the *mean* (also called the *average*) and the *standard deviation* of those numbers, rounded off to two decimal places.
If there are n numbers, and they are $\{x_1, x_2, \ldots, x_n\}$, the mean of these numbers, usually denoted by the Greek letter μ (mu) is given by the formula:

$$\mu = \frac{x_1 + x_2 + \cdots + x_n}{n}.$$

The standard deviation, usually denoted by the Greek letter σ (sigma), is found by first computing the *variance* σ^2, the square of the standard deviation. The variance is computed by the formula

$$\sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_n - \mu)^2}{n}.$$

For example, if the numbers are 1, 2, 3, and 5, the mean is

$$\mu = \frac{1 + 2 + 3 + 5}{4} = 2.75,$$

and the variance is

$$\sigma^2 = \frac{(1 - 2.75)^2 + (2 - 2.75)^2 + (3 - 2.75)^2 + (5 - 2.75)^2}{4} = 2.1875,$$

so the standard deviation is $\sigma = \sqrt{2.1875} \approx 1.479$.
The input to your program consists of the number n on a line by itself, specifying how many numbers are to be involved in the computation, followed by n numbers, each on a line by itself. Your program should print as output a single line that begins with the mean, rounded off two decimal places, followed by a space, then the symbols `+/-`, then another space, and finally the standard deviation, also rounded to two decimal places.
The following input represents the example above

```
4
1.0
2.0
3.0
5.0
```

and the output of your program given this input should be

```
2.75 +/- 1.48
```

You may assume $n \leq 20$, and all numbers in the input are between -100 and 100, but they are not necessarily integers. Note that while you only print the mean and standard deviation rounded to two decimal places, you should use the exact mean (not rounded) in your computation of the standard deviation.

4 Mastermind

In the game of Mastermind, one player (the *codemaker*) chooses a secret code, usually consisting of four colored pegs, and the other player (the *codebreaker*) tries to guess the code as quickly as possible. After each guess, the codemaker tells the codebreaker how many pegs in the guess match a peg in the same position in the code, and how many of the remaining pegs in the guess match a peg in a different position in the code. For example, if the code is

```
RED WHITE BLUE RED
```

and the codebreaker guesses

```
BLUE GREEN RED RED
```

then one peg is in the correct position (the last RED peg), and two other pegs are the correct color, but in a different position (the BLUE peg and the other RED peg).
Write a program which takes a code on the first line and a sequence of guesses on each additional line. In each case, we use a single letter for the color of the peg, and the possible colors are (R)ed, (Y)ellow, (G)reen, (B)lue, and (W)hite, and the letters for each code are separated by a single space.
For each guess, the program prints two numbers on a line by themselves: the number of pegs in the correct position and the number of other pegs that can be matched with a peg in the code in a different position. At the end, the number of guesses that were used to break the code, followed by the word "`guesses`" is displayed.
For example, if the input to the program is

```
R W B R
B G R R
B R G Y
R B R G
R B W R
R W B R
```

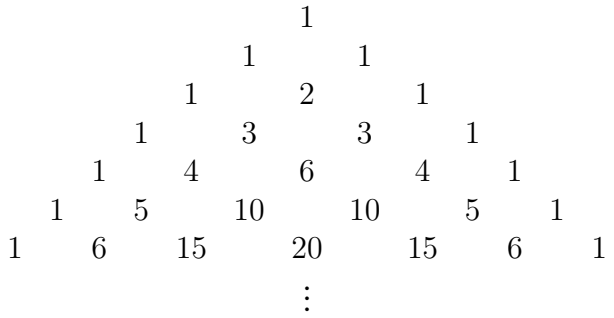
the corresponding output would be

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

You may assume that both a code and a guess consist of four pegs, and the last line of the input is a guess that matches the original code.

5 Pascal's Triangle

Consider the following triangle of integers, called Pascal's triangle. Each number in the interior of the triangle is the sum of the two numbers immediately above it.



Write a program which takes as input a sequence of lines, each with two numbers r and k on one line separated by a space, and prints out for each pair of numbers the kth entry of row r. You may assume that $1 \leq k \leq r \leq 60$ in every line except the last. This constraint guarantees that each number you compute is less than 2^{60}, since the largest number in each row cannot be more than twice the largest number in the previous row, *but the answer might be larger than* 2^{50}. The last line consists of two zeroes to signal the end of the program, and requires no computation.
For example, running the program with the input

```
7 1
7 2
7 3
7 4
7 5
7 6
7 7
0 0
```

produces the following output, corresponding to the last row printed above.

```
1
6
15
20
15
6
1
```

As another example, the 30th and 31st entries of the 60th row are both 59,132,290,782,430,712.

6 No Repeated Digits

We would like to investigate integers whose decimal representation is of a fixed length, and contains no repeated digits. Write a program which takes as input a sequence of lines in which each line except the last includes two parts. The first part is a string consisting of between 1 and 10 digits between 0 and 9, listed in order, with no repeats, and the second part is a number k. Your program then prints for each line the kth smallest integer that contains exactly the digits listed in the first part. If the first part contains n digits, there are $n! = n(n-1)(n-2)\cdots 1$ different numbers using those n digits, so we have $1 \leq k \leq n!$. The last line consists of a pair of zeroes, which signal the end of the input, and which should not be processed.
For the purposes of this problem, a number may begin with a single leading zero, i. e. 0 is simply a digit, just like any other.
For example, if the input is

```
0 1
035 1
035 2
035 3
035 4
035 5
035 6
1234 6
0123456789 3628800
0 0
```

the corresponding output is

```
0
035
053
305
350
503
530
1432
9876543210
```

Note that the first line of the input asks for the smallest number using the single digit 0, whereas the last line of the input simply ends the program. Also, in the line above the final pair of zeroes, 3,628,800 is $10!$, so that line asks for the largest ten-digit number in which no digit is repeated.

7 Approximate Phases of the Moon

You have developed an interest in calculating the phases of the moon. Unfortunately, exact dates are difficult to calculate, since the cycle of the phases varies from one cycle to the next, based on factors such as exactly how close the moon is to the earth. However, the average cycle is known to be approximately 29 days, 12 hours, and 44 minutes.
Write a program that takes input representing any date between January 1, 2001 and December 31, 3000, and prints the phase of the moon on that date, using the following assumptions:

- Every cycle lasts exactly 29 days, 12 hours, and 44 minutes.
- The last full moon of the year 2000 occurred on December 25 at 12:23pm EST.
- The elapsed time from the new moon to each phase is listed in the following table:

Phase	Time since new moon			Phase	Time since new moon		
	days	**hours**	**mins.**		**days**	**hours**	**mins.**
New Moon	0	0	0	Full Moon	14	18	22
Waxing Crescent	3	16	35	Waning Gibbous	18	10	58
First Quarter	7	9	11	Third Quarter	22	3	33
Waxing Gibbous	11	1	46	Waning Crescent	25	20	9
				Next New Moon	29	12	44

The input for the program consists of three numbers, representing the month, date in the month and year, in that order. Your program then calculates if one of the above phases occurs during that date in the Eastern time zone. If so, the program prints the name of that phase. If not, the program prints a line of the form:

`Between <previous phase> and <next phase>`

where `<previous phase>` and `<next phase>` are replaced by the appropriate phases.
The number of days in each month other than a leap year is as follows:

Month num.	1	2	3	4	5	6	7	8	9	10	11	12
Month	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
Days	31	28	31	30	31	30	31	31	30	31	30	31

Leap years, in which February has a 29th day, occur every year that is a multiple of four, except for the years 2100, 2200, 2300, 2500, 2600, 2700, 2900, and 3000.
The following table gives some examples of dates and the corresponding output.

Input	Output
`1 1 2001`	`First Quarter`
`1 24 2001`	`New Moon`
`2 29 2984`	`First Quarter`
`12 28 3000`	`Waxing Gibbous`
`12 31 3000`	`Between Waxing Gibbous and Full Moon`

8 Evaluating an Expression

Write a program that takes as input an expression consisting of integers, spaces, parentheses, and the characters `+`, `-`, `*`, `/`. Your program should then evaluate the expression and print out the result, either as an integer or as a fraction if the result is not an integer.
Your program should follow the standard order of operations. That is, all operations in parentheses should be evaluated first, then multiplications and divisions should be performed from left to right, and finally, additions and subtractions should be performed from left to right. You may assume there is at least one space before and after each parenthesis and each operator.
If the result is an integer, your program should print that integer. Otherwise, print the result as a fraction in lowest terms, with the numerator followed by a slash followed by the denominator. If the result is negative, print a single minus sign in front of your entire answer.
The following give examples of input and the corresponding expected output.

Input	Output
`5`	`5`
`2 + 2`	`4`
`2 - -3`	`5`
`3 * 4 + 2`	`14`
`2 / -3`	`-2/3`
`( 10 - -2 - 4 * 5 ) + ( 5 * 2 ) / 3`	`-14/3`
`1 / 2 + 1 / 2`	`1`
`( ( 1 - 2 * ( 3 * 4 + 5 ) ) )`	`-33`
`1 / 2 / 3 / 4`	`1/24`
`( 1 / 2 ) / ( 3 / 4 )`	`2/3`

You may assume all input is valid, no numerators or denominators are larger than 1000, and there is no division by zero.