2013 Widener University Programming Contest for High School Students
Widener University
Thursday, November 14, 2013

Rules:

1. Name your programs as follows: TeamNProblemK.ext, where N is a team number, K is a problem number, and **ext** is a language extension. For example, if you are in Team 1 and you are working on problem 1 and you are writing in C++ you will save your program as: Team1Problem1.cpp

2. You have to use standard input and standard output for ALL problems. This means that input will be entered from the keyboard and the output will be displayed on the screen. If you have language specific questions related to this requirement, please ask all questions during the orientation time.

3. All programs must produce the correct output within 30 seconds.

4. How to save your work:
   - Teams working on their own laptops will store their work on their own laptop.
   - Teams that are working in the main Computer Science department lab will save their work on the desktop of the computer in a separate folder that has the name of the team. For example, Team 1 will create a folder team1 and will save all programs in this folder.

5. Submission:
   - When you are ready to submit the solution for a problem you have to raise your hand to call the judge.
   - The judge will write the submission time for the problem and check your program.
   - **Carefully check your program before submission.**

IMPORTANT: All submissions are FINAL.
You will NOT be allowed to resubmit a solution.

Problem 1:

Write a program that reads 3 positive numbers, the value of the first number is the cost of some item, the value of the second number is a discount percentage (integer) and the value of the third number is a tax percentage (integer). The program calculates the final cost of the item after the discount is taken and the tax is applied to the discounted cost

.

You can assume that the input is valid. Use whatever floating point format your language provides by default for the output.

Example 1:
Input: `100 30 7`
Output: `74.9`

Example 2:
Input: `199.99 25 6`
Output: `158.99`

```
price, discount, tax = map(int, raw_input().split())   # float?
print (price * ((100-discount)/100)) * ((100+tax)/100)
```

Problem 2:

Write a program that reads a sequence of positive integers greater than 10. The first negative integer terminates the input. The program finds the average of the last digits of the even numbers and the first digits of the odd numbers. You can assume that the input is valid and not empty – there is at least one positive integer in the input. Use whatever floating point format your language provides by default for the output.

For example, if the user enters the positive numbers **23**, **4**55, 66**8** and 2**4**, the program will compute the average of the first digits of the odd numbers (2 and 4) and the last digits of the even numbers (8 and 4): (2 + 4 + 8 + 4) / 4 = 4.5

Example 1:
Input: `23 455 668 24 -1`
Output: `4.5`

Example 2:
Input: `56 -1`
Output: `6.0`

Example 3:
Input: `10 20 340 -1`
Output: `0.0`

```
data = map(int, raw_input().split("-1")[0].split())   # float?
numbers = []
for item in data:
    if item%2==0: # if even                            # check
        numbers.append(float(str(item)[:1]))
    else: # if odd
        numbers.append(float(str(item)[-1:]))
print sum(numbers)/len(numbers)
```

Problem 3:

In cryptography we use the following terminology:
a. plaintext is the original message
b. ciphertext is the encrypted message
c. encryption is the process of converting plaintext into ciphertext
d. decryption is the process of converting ciphertext into plaintext
e. cipher is the algorithm that was used to encrypt and decrypt the messages

The ADFGVX cipher can encrypt all 26 letters and 10 digits.
The key is the square below.

	A	D	F	G	V	X
A	P	H	0	Q	G	6
D	4	M	E	A	1	Y
F	L	2	N	O	F	D
G	X	K	R	3	C	V
V	S	5	Z	W	7	B
X	J	9	U	T	I	8

To encrypt, each letter/digit is replaced by the pair of correspondent letters in the ROW and COLUMN. For example, letter O will be encrypted by the pair FG and the digit 0 will be encrypted by the pair AF.
To decrypt, the reverse procedure is applied.

Write a program that can do both encryption and decryption. The program must take two inputs. The first input must be either 1 or 2, with 1 signaling encryption and 2 signaling decryption. The second input is a string of upper-case characters that will be either encrypted or decrypted, depending on the first input.

Example 1:
Input: `1 YANA`
Ciphertext Output: `DXDGFFDG`

Example 2:
Input: `2 AVFG`
Plaintext Output: `GO`

Problem 4:

A **happy number** is defined by the following process. Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are **happy numbers**, while those that do not end in 1 are **unhappy numbers** (or **sad numbers).**

If *n* is not happy, then its sequence does not go to 1. What happens instead is that it ends up in a cycle such as:

4, 16, 37, 58, 89, 145, 42, 20, 4, 16, 37, 58, 89, 145, 42, 20, 4, 16, 37, 58, 89…

Write a program that reads one integer and outputs the following: if the number is happy, it outputs YES and the number of steps it took to get from the input to 1 (including the step that resulted in 1), otherwise, the program outputs NO and the number of steps it took to get to a repeated number (including the step that resulted in the repeated number).

For example if n = 44, we get
Step 1: $4^2 + 4^2 = 32$
Step 2: $3^2 + 2^2 = 13$
Step 3: $1^2 + 3^2 = 10$
Step 4: $1^2 + 0^2 = 1$ – terminates the process and the output is: YES, 4

For example, if n = 43, we get
Step 1: $4^2 + 3^2 = 25$
Step 2: $2^2 + 5^2 = 29$
Step 3: $2^2 + 9^2 = 85$
Step 4: $8^2 + 5^2 = \mathbf{89}$
Step 5: $8^2 + 9^2 = 145$
Step 6: $1^2 + 4^2 + 5^2 = 42$
Step 7: $4^2 + 2^2 = 20$
Step 8: $2^2 + 0^2 = 4$
Step 9: $4^2 = 16$
Step 10: $1^2 + 6^2 = 37$
Step 12: $3^2 + 7^2 = 58$
Step 13: $5^2 + 8^2 = \mathbf{89}$

```
i=1
num = int(raw_input())
sequence =[num]
while True:
digits=
map(int, list(str(sequence[i-1])))
new=sum(function(x){return x**x;},
digits)
if sequence.find(new):
print "NO, %d" %i
break
if new==1;
print "Yes, %d" %i
break
```

In this example, the number 89 is encountered twice, which means we have detected a cycle and the process should be stopped. The program should output the following: NO, 13

Example 1:
Input: `1`
Output: `YES, 1`

Example 2:
Input: `2`
Output: `NO, 9`

Example 3:
Input: `7`
Output: `YES, 5`

(More examples on next page.)

Example 4:
Input: `1663`
Output: `YES, 4`

Example 5:
Input: `2147483648`
Output: `NO, 14`

Problem 5:

The advice to "buy low" is half the formula to success in the stock market. But to be considered a great investor you must also follow this problem's advice:

"Buy low, buy lower"

That is, each time you buy a stock, you must purchase more at a lower price than the previous time you bought it. The more times you buy at a lower price than before, the better! Your goal is to see how many times you can continue purchasing at ever lower prices.

You will be given the daily selling prices of a stock over a period of successive days. You can choose to buy stock on any of the days. Each time you choose to buy, the price must be lower than the previous time you bought stock. Write a program that reads a sequence of positive integers. The first negative integer terminates the input. Each positive integer represents the price of the stock on that day: the first element in the sequence is the price of the stock on day 1, the second element in the sequence is the price of the stock on day 2, etc. The program identifies on which days you should buy stock in order to maximize the number of times you buy. The program outputs the total number of all different longest possible sequences of decreasing prices and lists all such sequences

Assume that the input is valid. Pay attention, if two different sequences of "buy" days produce the same string of decreasing prices, they are considered ONE solution. Two solutions are considered the same, and need to be output only once, if they repeat the same string of decreasing prices, that is, if they "look the same" when the successive prices are compared.

For example, suppose on 12 successive days a stock is selling like this:

```
 Day    1  2  3  4  5  6  7  8  9 10 11 12
Price 68 69 54 64 68 64 70 67 78 62 98 87
```

In the example above, the best investor (by this problem, anyway) can buy at most four times if they purchase at a lower price each time. There are two possible solutions, one four-day sequence of acceptable buys is: 69 68 64 62 and the second four day sequence of acceptable buys is 69 68 67 62. So the output will be:

```
2
69 68 64 62
69 68 67 62
```

(More examples on next page.)

```
outputs = []
for i in range(0, len(days)):
    j = i+1
    outputs.append()
    while j <= len(days):
        if data[j] < data[i]:
            outputs[i].append(
        j++
```

```
def iter(list):
```

Example 1:
Input: `5 -1`
Output:
```
1
5
```

data = map(int, raw_input().split(" -1")[0].split())
solutions = []

Example 2:
Input: `1 2 3 4 5 6 7 8 9 10 -1`
Output:
```
10
1
2
3
4
5
6
7
8
9
10
```

Example 3:
Input: `5 4 3 2 1 -1`
Output:
```
1
5 4 3 2 1
```

Example 4:
Input: `10 10 6 6 4 4 10 4 6 10 6 4 -1`
Output:
```
1
10 6 4
```

Problem 6:

There are some chickens and some cows in Farmer John's yard. John's daughter Susie counted that all the animals in the yard have a total of 3 heads. John's son Billy counted their legs and got a total of 8. Using their answers, Farmer John easily determined that there have to be exactly 2 chickens and 1 cow.

Write a program that will solve a general version of Farmer John's problem. The program reads two positive integers: the number of **heads** and the number of **legs**. The program computes and outputs the number of chickens and the number of cows in this order. If there is no solution, the program outputs the following message "No Solution"

(Examples on next page.)

heads, legs

cows = 0
chickens = legs/2
while heads > cows+chickens:
    chickens -= 2
    cows += 1
if chickens*2 + cows*4 == legs:
    and chickens+cows == heads:
    print "%d chickens %d cows" %(chickens, cows)
else: print "No Solution"

Example 1:
Input: 4 12
Output: 2 chickens 2 cows

Example 2:
Input: 30 74
Output: 23 chickens 7 cows

Problem 7:

You are packing books into some boxes, packing as many books as you can into each box without exceeding a given weight limit.

Write a program that reads first a positive integer that represents the maximum weight that can fit into each box, and then reads a sequence of positive integers representing the weights of the books. You can assume that none of the book weights exceeds the maximum weight, meaning each book can fit into a box. The first negative value terminates the input. The program outputs the minimum number of boxes you will need and one optimal distribution of the books among boxes.

Example 1:
Input: 10 4 3 4 1 7 8 -1
Output: 3 {8} {4, 4, 1}, {3, 7}

Example 2:
Input:8 4 3 4 1 7 8 -1
Output: 4 {4, 4} {1, 7}, {8}, {3}

10 none
9 none
8 yes
2 nope
1 yes
7 yes, 3 yes
6 no
5 no
4 yes

data=map(int, raw_input().split(" -1")[0].split())

data = data.split("-1")[0]
data = data.split(" ")
data = [ ]

max = data.pop(0)

check if any depth = max {
    pop it if so
}

subtract 1 from max and see if there is their is that num
if so, find if their is a num that equals the difference
if so pop them
else continue

> check if exact
> check if max-1 and a 1
check if max - 2 and a 2
~~check if max - 2 and a 1~~
Check if max - 3 and a 3
Check if max - 4 and a 4

max - 5 and a 5
max - 6 and a 6
max - 7 and 7
... until max - (max - 1)
max - 2 and 1
max - 3 and 1