

# **ACM Programming Contest**

## **Fall 2005 Questions**

### **Northeast Preliminary Site**

**SUNY Oswego**

**October 29, 2005**

# 1. Complex Arithmetic

Write a program that performs the usual binary arithmetic operations (addition, subtraction, multiplication and division) on *complex numbers*. Complex numbers are represented by ordered pairs of floating point numbers, e.g., (1.2,-3.4). The first value in each pair is the *real part* and the second value is the *imaginary part*. Thus, in (1.2,-3.4), the real part is 1.2 and the imaginary part is -3.4.

Recall that if  $(a,b)$  and  $(x,y)$  are complex numbers, then:

- $(a,b) + (x,y) = (a+x,b+y)$
- $(a,b) - (x,y) = (a-x,b-y)$
- $(a,b) * (x,y) = (ax-by,ay+bx)$
- $(a,b) / (x,y) = ((ax+by)/(x^2+y^2), (bx-ay)/(x^2+y^2))$

For example:

- $(1.2,-3.4) + (2,1) = (3.2,-2.4)$
- $(1,2) * (3,4) = (1(3)-2(4),1(4)+2(3)) = (3-8,4+6) = (-5,10)$

Each input line specifies an operation on two complex numbers, as follows:

$$(\langle real_1 \rangle, \langle imag_1 \rangle) \langle operator \rangle (\langle real_2 \rangle, \langle imag_2 \rangle)$$

For example:

$$\begin{aligned} &(1.2, -3.4) + (2, 1) \\ &(1, 2) * (3, 4) \\ &(4, 2) / (2, 1) \\ &(2.2, 4) - (0, 1) \end{aligned}$$

There is no limit to the number of lines of input. Your output should give the results of the specified operations, one per line. For example, given the input shown above, the output should be:

$$\begin{aligned} &(3.2, -2.4) \\ &(-5, 10) \\ &(2, 0) \\ &(2.2, 3) \end{aligned}$$

---

## 2. Who's the Smartest?

We'd like to write a list of people, ordered so that no one appears in the list before anyone he or she is less smart than. The input will be a list of pairs of names, one pair per line, where the first element in a pair names a person smarter than the person named by the second element of the pair. That is, each input line looks like:

```
<smarter-person> <less-smart-person>
```

For example:

```
Einstein Feynmann  
Feynmann Gell-Mann  
Gell-Mann Thorne  
Einstein Lorentz  
Lorentz Planck  
Hilbert Noether  
Poincare Noether
```

(We don't mention computer scientists for obvious reasons.) There is no limit to the number of lines of input. Your output should be a list of all the distinct input names, *without duplicates*, one per line, ordered as described above. For example, given the input shown above, one valid output would be:

```
Einstein  
Feynmann  
Gell-Mann  
Thorne  
Lorentz  
Planck  
Hilbert  
Poincare  
Noether
```

Note that the "smarter than" relation supplied as input will *not*, in general, specify a total order that would allow us to write out the desired list in strictly decreasing order of smartness. For example, the following output is also valid for the input shown above:

```
Hilbert  
Einstein  
Feynmann  
Gell-Mann  
Poincare  
Thorne  
Lorentz  
Planck  
Noether
```

### 3. Turing Machine Simulator

Write a program that simulates a Turing machine (Tm). A Tm is a primitive model of a general purpose computer. Programs for a Tm consist of a collection of quadruples, each of the following form:

```
<state> <symbol> <action> <nextstate>
```

where

```
<state>      ::= a non-negative integer
<symbol>    ::= 1 | B
<action>    ::= <symbol> | L | R
<nextstate> ::= a non-negative integer
```

A Tm stores data on a tape divided into squares, each of which contains one symbol, either 1 or B. Only one square can be inspected at a time. Initially, a sequence of symbols is written on the tape (this is the input to the Tm), the Tm's "read/write head" is positioned over the leftmost symbol, and the state of the Tm is set to some "start state".

At any point in a Tm computation, the Tm is in a particular state with its read/write head scanning the symbol in a particular square. What happens next depends on whether the Tm's program contains a quadruple whose *<state>* is the Tm's current state and whose *<symbol>* is the symbol contained in the square currently being scanned. If such a quadruple exists (*there cannot be more than one such*), the *<action>* part of this quadruple controls what happens next:

- if *<action>* specifies a symbol, then that symbol replaces whatever symbol is in the square currently under scan
- if *<action>* is L, the Tm's read/write head is moved one square to the left (the symbol in the square previously under scan is not changed)
- if *<action>* is R, the Tm's read/write head is moved one square to the right (the symbol in the square previously under scan is not changed)

After one of the above actions is done, the Tm changes to state *<nextstate>* and we repeat the process of searching for an appropriate quadruple to "execute". This is continued until no quadruple matches the current state and symbol under scan, in which case the Tm halts and we consider its output to be the sequence of symbols between the first and last 1's on the tape, inclusive (if no 1's remain, we consider the output to be just B).

The input is a sequence of Tm instruction quadruples followed by a "start" line that specifies the initial state for the Tm, followed by the sequence of symbols that make up the initial value stored on the Tm's tape. Generically:

```
<state> <symbol> <action> <nextstate>
<state> <symbol> <action> <nextstate>
<state> <symbol> <action> <nextstate>
<state> <symbol> <action> <nextstate>
...
start <state>
```

```
<symbol>  
<symbol>  
<symbol>  
<symbol>  
<symbol>  
...
```

For example:

```
0 1 R 0  
0 B 1 1  
start 0  
1  
1  
1  
1
```

Given the above input, the output should be:

```
11111
```

### Note

If a quadruple requires moving left from the leftmost symbol on the tape, or right from the rightmost symbol, a square containing the symbol B must be added to the appropriate end of the tape (which is thus "finite, but unbounded").

## 4. The Perimeter

### The Setup

A peculiar type of "agent" randomly walks around the Cartesian Plane by repeatedly taking one-unit steps in one of four directions: north, east, south, or west. Suppose that three of these agents start at the origin and that each agent independently takes 20 steps. The points at which these three agents end up at will define a (possibly degenerate) triangle. An example showing 20-step histories of random walks of three agents should clarify all of this.

Example:

Agent 1 History

```
[1:(0,1)][2:(-1,1)][3:(0,1)][4:(1,1)][5:(0,1)]
[6:(0,2)][7:(0,3)][8:(0,4)][9:(0,3)][10:(1,3)]
[11:(1,4)][12:(2,4)][13:(2,5)][14:(2,4)][15:(1,4)]
[16:(2,4)][17:(2,5)][18:(2,4)][19:(1,4)][20:(0,4)]
```

Agent 2 History

```
[1:(0,1)][2:(1,1)][3:(1,0)][4:(1,-1)][5:(2,-1)]
[6:(3,-1)][7:(3,-2)][8:(3,-1)][9:(3,-2)][10:(2,-2)]
[11:(2,-1)][12:(2,0)][13:(2,1)][14:(2,2)][15:(1,2)]
[16:(1,3)][17:(2,3)][18:(3,3)][19:(2,3)][20:(2,2)]
```

Agent 3 History

```
[1:(-1,0)][2:(-2,0)][3:(-2,-1)][4:(-3,-1)][5:(-4,-1)]
[6:(-4,-2)][7:(-3,-2)][8:(-3,-1)][9:(-3,0)][10:(-3,-1)]
[11:(-2,-1)][12:(-1,-1)][13:(-2,-1)][14:(-1,-1)][15:(0,-1)]
[16:(0,0)][17:(-1,0)][18:(0,0)][19:(0,1)][20:(1,1)]
```

The perimeter of the triangle defined by the points at which these three agents ended up happens to be roughly 7.405 units in length. (That is, the perimeter of the triangle defined by points (0,4) (2,2) and (1,1) is roughly 7.405 units.) Of course, give three other agents a chance to define their own triangle by taking 20-step random walks from the origin and you will very likely get a different triangle with a different perimeter!

### The Problem

Write a program, subject to the two constraints given below, to determine the **expected perimeter length** of a triangle defined by the "destination" positions of three of these peculiar agents -- assuming that each agent begins at the origin and takes a random walk of exactly 20 steps.

Constraints:

1. The estimated perimeter length should be computed on the basis of 100 simulated "trials", where each trial involves three agents taking 20 random steps in the Cartesian Plane from the origin.
2. The program should run in less than 10 seconds.

## 5. Counter-Productive

A lunch counter has 6 stools for customers. Six patrons have reservations for time  $T_0$ . Six more are scheduled for time  $T_1$ . And so forth. All customers show up exactly on time; and it is known exactly how long each customer will stay. When customers arrive, any one of them may be seated if a prior customer is finished eating, with the proviso that any earlier arriving customer has been seated first. Determine the order of seating and, hence, the shortest time to serve all the customers.

Input will be in the form of  $N$  pairs of lines. The first of the pair will be a single integer denoting the current time of this set of arrivals. The second line of the pair will contain 6 integers denoting the service time required for the 6 arriving diners.

Output should be some number of lines with 6 elements per line showing the service times for each stool. If a column is shorter than the longest, fill in the missing elements with an underscore (`_`). These lines are followed by a final line with a single number denoting the time after the first arrival when the final customer has finished eating.

For example, with  $N$  equal 2 and the following service times:

```
12
3 7 5 3 6 1
15
1 2 1 4 6 5
```

Output would be in the following form (this is in fact an incorrect solution, as there are answers with a lower time until the last customer is served)

```
3 7 5 3 6 1
1 _ _ 1 _ 2
4 _ _ 5 _ 6
11
```

(Note that the 11 at the end of output says the last customer finished at time 23, 11 units after the first arrival.)

## 6. Sorta in order

Sort a given set of strings based on a unique collating sequence for each position in a string. Given  $N$  collating sequences, to sort strings of length greater than  $N$ , sequence  $i \bmod N$  is used at character position  $i$ .

For example, consider the three collating sequences:

collating sequence 0 is: ASCII-order-ignore-case

collating sequence 1 is: reverse-ASCII-order

collating sequence 2 is: a-z 0-9 ASCII-order A-Z

In this example the strings

The Cat in the Hat

the Rain in Spain

The RAIN in Spain

Beavis and Butthead

would be sorted into the list:

Beavis and Butthead

the Rain in Spain

The RAIN in Spain

The Cat in the Hat

Note that the last ordering says lower case comes before digits; and digits before everything not upper case; and upper case follows all.

The allowable notations for collating sequences are:

ASCII-order

ASCII-order-ignore-case

reverse-ASCII-order

reverse-ASCII-order-ignore-case

a-z

A-Z

0-9

These can occur in any order without repetition.

### Input will be in the form:

$N$

description of collating sequence 1

:

description of collating sequence  $N$

line 1

line 2

:

line unknown number

So for the given example, the input would look like:

3

ASCII-order-ignore-case

reverse-ASCII-order

a-z 0-9 ASCII-order A-Z

The Cat in the Hat

the Rain in Spain

The RAIN in Spain

Beavis and Butthead