2000 Stanford Local Programming Contest

October 7, 2000

Read these guidelines carefully!

Rules

1. You may use resource materials such as books, manuals and program listings. You may *not* use any machine-readable versions of software, data or existing electronic code. All software submitted during the contest *must* be typed in during the contest. No cutting and pasting of code is allowed.
2. Students may not collaborate in any way with each other or anybody else, including people contacted via Internet.
3. You are expected to adhere to the honor code. You are still expected to conduct yourself according to the rules, even if you are not in Gates B02.

Guidelines for submitted programs

1. All programs must be written in C, C++, or Java. To judge the programs, I will compile programs in the following way:

   - `.c`: using `gcc -lm`
   - `.cc`: using `g++ -lm`
   - `.java`: using `javac`

   I will use the Leland system to do this, so if your programs compile on a Leland host, they should compile for me. Please use only the listed filename extensions.
2. **Java users: Please place your `public static void main()` function in `public class Main`.** I want to be able to run your program with the command `java Main`.
3. Your solution may have as many files as necessary. However, please use the same language (C, C++, or Java) for a given run. Make sure each problem's code files are in a separate directory.
4. All programs should accept their input on **stdin** and produce their output on **stdout**. They should be batch programs in the sense that they do not require human input other than what is piped into stdin.
5. Be careful to follow the output format described in the problem. I will be judging programs based on a **`diff`** of your output with the correct solution.

How will the contest work?

1. From 12:00 to 1:00 students will pick a computer, set up their workspace and complete a test problem. The purpose of this test problem is to send me information about who you are; you won't be registered until you do this. This will also make sure the submission script is working for you.
2. At 1:00 the contestants will gather in the **outside Gates B02** and I will hand out the problems. You should then return to your workspace and begin solving the problems.
3. The test problem description explains how to submit your solutions. **Make sure that you run the submission script from the correct directory, and that only the files for your solution are in that directory!**

4. For every run, your solution will either be accepted or rejected for one of the following reasons: *compiler error, run time error, time limit exceeded, wrong answer, presentation error*. The time limit for a run is **one minute**.

5. Watch your **email on Leland**. I will be sending out any necessary messages as well as notification if your programs are accepted or rejected via email, and the emails will go to

   `your_leland_id@leland.stanford.edu`.

6. If you want to follow the progress of the contest, go to the **standings web page**. A link to this page exists on the main contest page, http://www-db.stanford.edu/~cooperb/acm.

7. At 4:00 the contest will end. No more submissions will be accepted. The contestants will be ranked by the number of solved problems. Ties will be broken based on the total time used for correct solutions (the time elapsed since 1:00 plus 20 minutes per rejected solution).

8. The top six contestants will advance to the regionals, subject to the constraint of a maximum of one graduate student per three person team.

If you have any questions, come talk to me in **Gates B02**, or send email to **cooperb@leland**.

0 Test Problem

0.1 Description

This is a test problem to make sure you can compile and submit programs. Your program for this section will print your email address, name and whether you are a graduate student or an undergraduate.

You must submit your solutions via the Leland system and the submission script I have created. You can develop your programs on whatever platform you want, but the submission must be via Leland.

Your solution to a given problem should be in its own directory, separate from the other problems. For example, you may have a `p0` directory, a `p1` directory, and so on. When you submit your solution, everything in this directory will be tarred and sent to the judge, so be careful that the right things are in the directory.

To submit a solution, run the `~cooperb/submit` script with a single argument: the problem you are submitting. For example, to submit problem 0, type:

```
$ ~cooperb/submit 0
```

Make sure you are in the directory with the solution for that program.
C/C++ users: Make sure only one of your C/C++ code files has a `main()` method.
Java users: Make sure there is a `public static void main()` method in `public class Main`.

When you submit your solution, the judge will receive an email and will judge your solution as soon as he can (be patient!) Once this is finished, you will receive an email stating that you have completed the problem or that you have not, and a reason why not. You can resubmit rejected solutions as many times as you like (though incurring a 20 minute penalty for each rejected run of a problem you eventually get right). Once you have submitted a correct solution, future submissions of that problem will be disregarded.

0.2 Input

This program will take no input.

0.3 Output

The program should output your email address, name and student status. This information should be separated by commas, and all on the same line. The line should be terminated with a newline. For example:

```
cooperb@leland,Brian Cooper,Graduate
```

1 The Circumference of the Circle

1.1 Description

To calculate the circumference of a circle seems to be an easy task - provided you know its diameter. But what if you don't?

You are given the cartesian coordinates of three non-collinear points in the plane. Your job is to calculate the circumference of the unique circle that intersects all three points. The value of pi is approximately 3.141592653589793.

1.2 Input

The input will contain one or more test cases. Each test case consists of one line containing six real numbers $x_1 y_1 x_2 y_2 x_3 y_3$, representing the coordinates of the three points. The diameter of the circle determined by the three points will never exceed a million. Input is terminated by end of file. For example:

```
0.0 -0.5 0.5 0.0 0.0 0.5
0.0 0.0 0.0 1.0 1.0 1.0
5.0 5.0 5.0 7.0 4.0 6.0
0.0 0.0 -1.0 7.0 7.0 7.0
50.0 50.0 50.0 70.0 40.0 60.0
0.0 0.0 10.0 0.0 20.0 1.0
0.0 -500000.0 500000.0 0.0 0.0 500000.0
```

1.3 Output

For each test case, print one line containing one real number telling the circumference of the circle determined by the three points. The circumference is to be printed accurately rounded to two decimals. For the sample input above:

```
3.14
4.44
6.28
31.42
62.83
632.24
3141592.65
```

2 The Monkey and the Coconuts

2.1 Description

A well known problem in recreational mathematics is the following:

> Five men and a monkey were shipwrecked on a desert island, and they spent the first day gathering coconuts for food. Piled them all up together and then went to sleep for the night.
>
> But when they were all asleep one man woke up, and he thought there might be a fight over dividing the coconuts in the morning, so he decided to take his share in advance. So he divided the coconuts into five piles. He had one coconut left over which he threw back into the jungle for the monkey. He hid his pile and put the rest all back together.
>
> By and by the next man woke up and did the same thing. One after the other took a fifth of the coconuts in the pile having one left over for the monkey. In the morning they divided what coconuts were left, and they came out in five equal shares. Of course each one must have known there were coconuts missing; but each one felt guilty like the others, so they didn't say anything. How many coconuts were there in the beginning?

Your task is to write a program for solving generalizations of the problem given above. Consider N men and the monkey shipwrecked on an island, with $1 \leq N \leq 10$. When dividing the remaining coconuts in N equal shares, the k^{th} man has a_k coconuts left over for the monkey, where $0 \leq k < N$. In the morning, there are a_N coconuts left over for the monkey after the pile is divided the final time.

Write a program to find the smallest non-negative number of coconuts that satisfies these conditions. Numbers greater than 10,000,000 need not be considered, because there are never as many coconuts on a single island.

2.2 Input

The input consists of multiple scenarios, each in a single line. The first number in each line gives the number N of men on the island, followed by the $N+1$ numbers $a_0, a_1, ..., a_N$ of coconuts that subsequently are given to the monkey. The input file is terminated by a line containing a single zero. For example:

```
5 1 1 1 1 1 0
2 1 0 1
0
```

2.3 Output

For each scenario print a line containing the smallest non-negative integer satisfying the conditions stated above. If there is no such number smaller than or equal to 10,000,000, print "No small solution.". For the sample input above:

```
3121
5
```

3 Call Forwarding

3.1 Description

Thanks to computer technology the functionality of phone systems has been greatly enhanced in the last ten years. We have automated menus, sophisticated answering machines, conference call capabilities, group addressing and so on. A common feature of a company's phone system is the ability to set call forwarding. For example, if Bob at the Nobody's Home Company (NHC) goes on vacation, he sets things up so that all calls coming to him are forwarded to his associate Jane. This problem addresses how phone systems might keep track of call forwarding.

The phones at the NHC all have four digit extensions. Employees can set call forwarding by entering the appropriate information through their telephone interface. If an employee is going to be away they enter the following information: their extension, the time they are leaving, how long they will be away, and the extension that their calls should be forwarded to, with the following constraints:

- All extensions consist of four digits.
- The extensions 0000 and 9999 are reserved for special use and will not be entered as information by an employee.
- Times are recorded in increments of 1 hour and are based on a clock that begins at 0000 at midnight every New Year's Eve. Therefore, when describing the time they are leaving, employees always use an integer between 0000 and 8784 (which is 366*24). The call forwarding system is completely reset at the beginning of the year.
- A call forward set to start at time X for a duration of Y will be in effect from time X to time $X + Y$ inclusive.

Users are "good" about the requests they enter. They follow the format rules. They do not enter a request such that the duration of the request would go past the end of the year. They do not enter two requests for their extension that overlap in time. Even though the users enter correct, clear, non-overlapping information from their own point of view, a degenerate situation can still occur in a call forwarding system, if requests have been made in such a way as to forward a call back to the original target of the call. For example if Bob forwards his calls to Sue, and Sue forwards her calls to Joe, and Joe forwards his calls to Bob then when somebody calls any of these three people their calls would be forwarded forever. To prevent this situation the call forwarding system uses the "dead end" number 9999. Any calls made to an extension involved in such a degenerate situation will be forwarded to the special 9999 extension.

3.2 Input

The first line contains an integer N between 1 and 10 describing how many call forwarding systems will be simulated by your program. Each call forwarding system will be represented by 0 to 100 *source time duration target* lines. These lines represent the requests by the users to set up a call forwarding from the *source* to the *target* starting at the *time* for a length of *duration*, and will be in the form `dddd dddd dddd dddd`. A line with `0000` in the *source* position indicates the end of this portion of the input. The call forwarding requests are listed in the order received. They will be followed by 1 or more *time extension* lines, in the form `dddd dddd`, in non-decreasing order by *time* representing calls made into the system at *time* to *extension*. A line with `9000` in the *time* position indicates the end of this portion of the input. For example:

```
2
1111 0100 0200 2222
1111 0301 0500 4444
2222 0200 0200 3333
3333 0250 1000 1111
7777 1000 2000 7777
```

```
0000
0050 1111
0150 1111
0200 1111
0225 2222
0270 1111
0320 1111
0320 3333
0900 3000
1250 3333
1250 7777
9000
0000
3000 1111
9000
```

3.3 Output

The first line of output must read `CALL FORWARDING OUTPUT`. This will be followed by sections of information about each of the call forwarding systems being simulated. Each of these sections should be headed by the line `SYSTEM` N, where N is the number $(1, 2, ...)$ of the system. Within the section there should be a line describing the result of each of the calls made into the system, with the format `AT dddd CALL TO dddd RINGS dddd`. The final line of output should read `END OF OUTPUT`. For the sample input above:

```
CALL FORWARDING OUTPUT
SYSTEM 1
AT 0050 CALL TO 1111 RINGS 1111
AT 0150 CALL TO 1111 RINGS 2222
AT 0200 CALL TO 1111 RINGS 3333
AT 0225 CALL TO 2222 RINGS 3333
AT 0270 CALL TO 1111 RINGS 9999
AT 0320 CALL TO 1111 RINGS 4444
AT 0320 CALL TO 3333 RINGS 4444
AT 0900 CALL TO 3000 RINGS 3000
AT 1250 CALL TO 3333 RINGS 1111
AT 1250 CALL TO 7777 RINGS 9999
SYSTEM 2
AT 3000 CALL TO 1111 RINGS 1111
END OF OUTPUT
```

4 Visible Squares

4.1 Description

You are given n squares in the coordinate plane whose sides are parallel to the coordinate axes. All the corners have integer coordinates and the squares do not touch or overlap.

A square is *visible* from a point P if there are two distinct points A and B on one of its sides, such that the interior of the triangle PAB has no common points with any of the remaining squares.

You are required to count the number of squares visible from the origin point $(0, 0)$.

4.2 Input

The input file may contain several instances of the problem. Each instance is described as follows:

1. The first line contains the integer n, $1 \leq n \leq 1000$, representing the number of squares.
2. Each of the following n lines describes a square: it contains a triple of integers xyl, $1 \leq x, y, l \leq 10000$, where coordinate (x, y) is the lower left corner (the corner with the least x and y coordinates) and l is the side length of that square.

The input file is terminated by a line containing only the integer 0 (zero). For example:

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

4.3 Output

For each instance of the problem, your program should print just one line, containing the number of squares visible from the origin. For the sample input above:

```
3
2
```

5 Biorhythms

5.1 Description

Some people believe that there are three cycles in a person's life that start the day he or she is born. These three cycles are the physical, emotional and intellectual cycles, and they have periods of lengths 23, 28, and 33 days, respectively. There is one peak in each period of a cycle. At the peak of a cycle, a person performs at his or her best in the corresponding field (physical, emotional or mental). For example, if it is the mental curve, thought processes will be sharper and concentration will be easier.

Since the three cycles have different periods, the peaks of the three cycles generally occur at different times. We would like to determine when a triple peak occurs (the peaks of all three cycles occur in the same day) for any person. For each cycle, you will be given the number of days from the beginning of the current year at which one of its peaks (not necessarily the first) occurs. You will also be given a date expressed as the number of days from the beginning of the current year. Your task is to determine the number of days from the given date to the next triple peak. The given date is not counted. For example, if the given date is 10 and the next triple peak occurs on day 12, the answer is 2, not 3. If a triple peak occurs on the given date, you should give the number of days to the next occurrence of a triple peak.

5.2 Input

You will be given a number of cases. The input for each case consists of one line of four integers p, e, i and d. The values p, e, and i are the number of days from the beginning of the current year at which the physical, emotional and intellectual cycles peak, respectively. The value d is the given date and may be smaller than any of p, e or i. All values are non-negative and at most 365, and you may assume that a triple peak will occur within 21252 days of the given date. The end of input is indicated by a line in which $p = e = i = d = -1$. For example:

```
0 0 0 0
0 0 0 100
5 20 34 325
4 5 6 7
283 102 23 320
203 301 203 40
-1 -1 -1 -1
```

5.3 Output

For each test case, print the case number followed by a message indicating the number of days to the next triple peak, in the form:

```
Case 1:  the next triple peak occurs in 1234 days.
```

Use the plural form "days" even if the answer is 1. For the sample input above:

```
Case 1:  the next triple peak occurs in 21252 days.
Case 2:  the next triple peak occurs in 21152 days.
Case 3:  the next triple peak occurs in 19575 days.
Case 4:  the next triple peak occurs in 16994 days.
Case 5:  the next triple peak occurs in 8910 days.
Case 6:  the next triple peak occurs in 10789 days.
```