# 2010 Stanford Local ACM Programming Contest

Saturday, October 2nd, 2010

**Read these guidelines carefully!**

**Rules**

1. You may use resource materials such as books, manuals, and program listings. You may *not* search for solutions to specifics problems on the Internet, though you are permitted to use online language references (e.g., the Java API documentation) and digital versions of textbooks or course notes. However, you may not electronically copy from machine-readable versions existing code or data. That is, all programs submitted must be *manually typed in their entirety during the contest.* No cutting and pasting of code is allowed!

2. You may bring and use your own laptop to write your code for this contest. To keep the computing environments as fair as possible across contestants, we ask that you restrict your use of software during the contest to text editors, IDEs, document readers, and a browser for the sole purpose of connecting to the contest site (and perhaps a language reference). In other words, please refrain from using programs like Mathematica, Maple, or other specialized tools if you have them installed. You may also use a simple electronic calculator (or the software one that comes pre-installed on your OS), but no fancy graphing calculators please!

3. You may not collaborate in any way (verbally, electronically, in writing, using gestures, telepathically, etc.) with other contestants, students, or anyone else during the contest.

4. 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 participating on site in the Gates building.

**Guidelines for submitted programs**

1. All programs must be written in C, C++, or Java. For judging, we will compile the programs in the following way:

   - `.c`: using `gcc -O2 -lm` (GCC version 4.1.2)
   - `.cc`: using `g++ -O2 -lm` (GCC version 4.1.2)
   - `.java`: using `javac` (Sun Java version 1.6.0)

   All programs will be compiled and tested on a Leland `myth` machine. The `myth` machines are Intel Core2 Duo 3.16 GHz machines with 4 GB RAM running CentOS Linux 5.3. Compilation errors or other errors due to incompatibility between your code and the `myth` machines will result in a submission being counted incorrect.

2. Make sure you `return 0;` in your `main()`; **any non-zero return values may be interpreted by the automatic judge as a runtime error**.

3. **Java users:** Please place your `public static void main()` function in a public class with the same name as the base filename for the problem. For example, a Java solution for the `test` program should be submitted in the file `test.java` and should contain a `main()` in `public class test`.

4. All solutions must be submitted as a single file.

5. 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.

6. Be sure to follow the output format described in the problem exactly. We will be judging programs based on a `diff` of your output with the correct solution, so your program's output must match the judge output **exactly** for you to receive credit for a problem. As a note, each line of an output file must end in a newline character, and there should be no trailing whitespace at the ends of lines.

**How will the contest work?**

1. If you chose to work remotely from a home computer, we recommend that you test out your account on the online contest system by submitting a solution for the test problem shown on the next page. We will do our best to set up the contest host to accept test problem submissions Saturday morning until approximately 1:45 pm.

2. For those who choose to participate onsite, from 1:00 to 1:45 pm, you should select a computer (or find a place to plug in your laptop), set up your workspace and complete a test problem. Space in Gates B02 and the PUP cluster is limited, and will be available on a first-come first-served basis. You may also choose to work directly on one of the `myth` machines in Gates B08, although technically we do not have that room reserved for the contest.

3. At 2:00 pm, the problems will be posted on the live contest page in PDF format, all registered participants will be sent an e-mail that the problems have been posted, and we will distribute paper copies of the problems to contestants competing in either Gates B02 or the PUP cluster.

4. For every run, your solution will be compiled, tested, and accepted or rejected for one of the following reasons: *compile error*, *run-time error*, *time limit exceeded*, *incorrect output*, or *presentation error*. In order to be accepted, your solution must match the judge output exactly (according to `diff`) on a set of hidden judge test cases, which will be revealed after the contest.

    - Source code for which the compiler returns errors (warnings are ok) will be judged as *compile error*.
    - A program which returns any non-zero error code will be judged as *run-time error*.
    - A program which exceeds the time allowed for any particular problem will be judged as *time-limit exceeded* (see below).
    - A program which fails a `diff -w -B` will be judged as *incorrect output*.
    - A program which passes a `diff -w -B` but fails a `diff` (i.e., output matches only when ignoring whitespace and blank lines) will be judged as *presentation error*.
    - A program which passes a `diff` and runs under the time constraints specified will be judged as *accepted*.

5. For each problem, the time allowed for a run (consisting of multiple test cases) will be 10 seconds total for all test cases. The number of test cases in a run may vary from 20 to 200 depending upon the problem, so be sure to write algorithmically efficient code!

6. You can view the status of each of your runs on the live online contest site. Please allow a few minutes for your submissions to be judged. The site also provides a live scoreboard for you to watch the progress of the contest as it unfolds.

7. At 6:00 pm, the contest will end. No more submissions will be accepted. Contestants will be ranked by the number of solved problems. Ties will be broken based on total time, which is the sum of the times for correct solutions; the time for a correct solution is equal to the number of minutes elapsed since 2:00 pm plus 20 penalty minutes per rejected solution. No penalty minutes are charged for a problem unless a correct solution is submitted. After a correct submission for a problem is received, all subsequent incorrect submissions for that problem do not count towards the total time.

8. The results of this contest will be used in part to select team members for representing Stanford at the forthcoming ACM regional competition. Six or more contestants have been customarily invited to the Stanford ACM ICPC teams in previous years.

**Helpful hints**

1. **Make sure your programs compile and run properly on the `myth` machines.** If you choose not to develop on the Leland systems, you are responsible for making sure that your code is portable.

2. **Read (or skim) through all of the problems at the beginning to find the ones that you can code quickly.** Finishing easy problems at the beginning of the contest is especially important as the time for each solved problem is measured from the beginning of the contest. Also, check the leaderboard frequently in order to see what problems other people have successfully solved in order to get an idea of which problems might be easy and which ones are likely hard.

3. If you are using C++ and unable to get your programs to compile/run properly, try adding the following line to your `.cshrc` file

   ```
   setenv LD_LIBRARY_PATH /usr/pubsw/lib
   ```

   and re-login.

4. The `myth` machines in Gates B08 are not officially reserved for the contest, but these will be the machines used for judging/testing of all programs. You may find it helpful to work on these machines in order to ensure compatibility of your code with the judging system.

5. If you are a CS major and have a working `xenon` account, please work in the **PUP cluster** rather than Gates B02; the PUP cluster has UNIX machines, which may be a more convenient programming environment if you intend to use Emacs, etc. If you don't know where the PUP cluster is, just ask!

6. If you are working on a PC in Gates B02, it may be helpful to run a VNC session if you don't like coding from a terminal. Check out the IT services page on using VNC, which can be found at `http://unixdocs.stanford.edu/moreX.html`. If you wish to use an IDE (e.g., Visual Studio or Eclipse), please make sure that you know how to set this up yourself beforehand. We will not be able to provide technical support related to setting up IDEs during the contest.

7. If you need a clarification on a problem or have any other questions, post an clarification request to the live contest page, or just come talk to us in **Gates B02** or the **PUP cluster**.

*The directions given here are originally based on those taken from Brian Cooper's 2001 Stanford Local Programming Contest problem set, and have been updated year after year to the best of our ability. The contest organizers would like to thank the problem authors of 2010, in alphabetical order, Andy Nguyen, Jaehyun Park, Jeffrey Wang, and Sonny Chan.*

# 0 Test Problem (`test.{c,cc,java}`)

## 0.1 Description

This is a test problem to make sure you can compile and submit programs. Your program for this section will take as input a single number $N$ and return the average of all integers from 1 to $N$, inclusive.

To submit a solution, navigate your browser to the live contest page, which this year resides at:

`http://cs.stanford.edu/group/acm/slpclive/`

You must log in using your assigned user ID and password for the contest. If you have registered for the contest, but have not received an email message containing your login information, then it's time to contact one of the contest organizers in panic! Students in Gates B02 or the PUP cluster will also be given their login information on paper before the contest.

Use the "Submissions" tab on the contest page to submit your solution to the problem. Detailed instructions can be found in the "Help" section online if necessary.

After submitting your solution, please allow a few minutes for it to be judged. 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 still be graded but will not count towards your final score or total time.

Note that you do not need to submit this problem during the actual contest!

## 0.2 Input

The input test file will contain multiple test cases. Each test case is specified on a single line containing an integer $N$, where $-100 \leq N \leq 100$. The end-of-file is marked by a test case with $N = -999$ and should not be processed. For example:

```
5
-5
-999
```

## 0.3 Output

The program should output a single line for each test case containing the average of the integers from 1 to $N$, inclusive. You should print numbers with exactly two decimal places. For example:

```
3.00
-2.00
```

### 0.4 Sample C Solution

```
#include <stdio.h>

int main() {
  int n;
  while (1) {
    scanf("%d", &n);
    if (n == -999) break;
    if (n > 0) printf("%.2lf\n", (double)(n * (n + 1) / 2) / n);
    else printf("%.2lf\n", (double)(1 + n * (1 - n) / 2) / (2 - n));
  }
  return 0;
}
```

### 0.5 Sample C++ Solution

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
  cout << setprecision(2) << setiosflags(ios::fixed | ios::showpoint);
  while (true) {
    int n;
    cin >> n;
    if (n == -999) break;
    if (n > 0) cout << double(n * (n + 1) / 2) / n << endl;
    else cout << double(1 + n * (1 - n) / 2) / (2 - n) << endl;
  }
  return 0;
}
```

### 0.6 Sample Java Solution

```
import java.util.*;
import java.text.DecimalFormat;

public class test {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        DecimalFormat fmt = new DecimalFormat("0.00");
        while (true) {
            int n = s.nextInt();
            if (n == -999) break;
            if (n > 0) System.out.println(fmt.format((double)(n * (n + 1) / 2) / n));
            else System.out.println(fmt.format((double)(1 + n * (1 - n) / 2) / (2 - n)));
        }
    }
}
```

# A AAAAHH! Overbooked! (`aaaahh.{c,cc,java}`)

## A.1 Description

Elaine is excited to begin the school year—so excited, in fact, that she signed herself up to attend several events today (This programming contest, sadly, is not one of them). She may have overdone it, though; she didn't bother to check whether the events she signed up for have conflicting times. While you're sitting here in this contest, why not check for her?

## A.2 Input

The input consists of multiple test cases. Each test case begins with an integer $N$, $1 \leq N \leq 100$, on a line by itself denoting the number of events. After that follow $N$ lines giving the start and end times of each event, in `hh:mm-hh:mm` 24-hour format. The end time is guaranteed to be strictly after the start time. Input is followed by a single line with $N = 0$, which should not be processed. For example:

```
3
09:00-09:50
13:00-17:00
09:50-10:30
2
10:00-11:00
09:00-12:00
0
```

## A.3 Output

For each test case, print out a single line that says "`conflict`" (no quotes) if Elaine's events have conflicting times, and "`no conflict`" (no quotes) otherwise. Assume that Elaine can travel around campus instantaneously, so if an event starts at the same time another event ends, the two events do not conflict. For example:

```
no conflict
conflict
```

# B Betting Sets (`betting.{c,cc,java}`)

## B.1 Description

Your local casino (which here in California is a fair distance away) has decided to try out a new betting game. The player is given a table on a card with $N$ rows and $M$ columns. Each entry $ij$ of the table corresponds to a coin flip with a probability $p_{ij}$ of coming up heads. All the coin flips are independent of each other. The player must choose $N$ sets of $M$ entries, with each set having exactly one entry from each column, and each entry being used exactly once. For each chosen set where all $M$ coins come up heads, the player wins 1 dollar. The casino is feeling uncharacteristically generous, and so as part of the promotion of this new game, they have been handing out one card to each patron. Since you have nothing to lose, you're going to give this game a shot. How can you maximize your expected winnings from this card?

## B.2 Input

The input consists of multiple test cases. Each test case begins with a line with two integers $N$ and $M$, $1 \leq N \leq 100$, $1 \leq M \leq 10$, separated by a space. This is followed by $N$ lines with $M$ space-separated numbers each, denoting the entries $p_{ij}$ of the table. Input is followed by a single line with $N = M = 0$, which should not be processed. For example:

```
2 3
1.0 1.0 1.0
0.5 0.4 0.3
0 0
```

## B.3 Output

For each test case, print out a single line with the maximum expected winnings from the given table, accurate to 4 decimal places. For example:

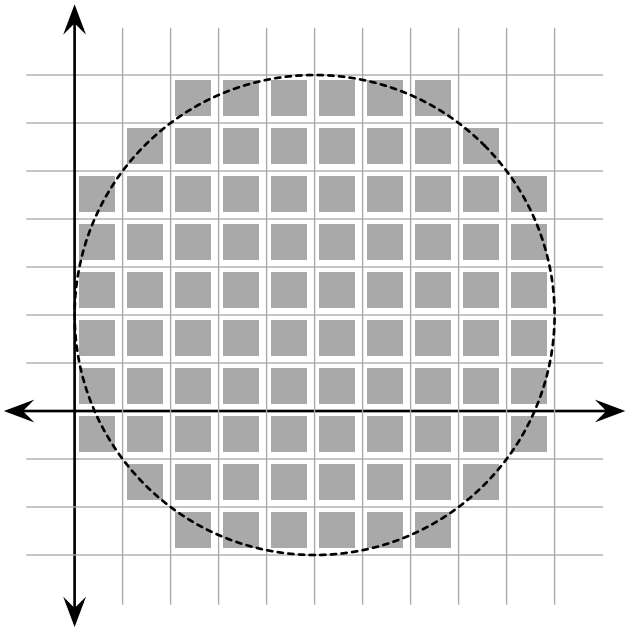
```
1.0600
```

# C Counting Pixels (`counting.{c,cc,java}`)

## C.1 Description

Did you know that if you draw a circle that fills the screen on your 1080p high definition display, almost a million pixels are lit? That's a lot of pixels! But do you know *exactly* how many pixels are lit? Let's find out!

Assume that our display is set on a Cartesian grid where every pixel is a perfect unit square. For example, one pixel occupies the area of a square with corners $(0, 0)$ and $(1, 1)$. A circle can be drawn by specifying its center in grid coordinates and its radius. On our display, a pixel is lit if any part of is covered by the circle being drawn; pixels whose edge or corner are just touched by the circle, however, are not lit.



Your job is to compute the exact number of pixels that are lit when a circle with a given position and radius is drawn.

## C.2 Input

The input consists of several test cases, each on a separate line. Each test case consists of three integers, $x$, $y$, and $r$ ($1 \leq x, y, r \leq 10^6$), specifying respectively the center $(x, y)$ and radius of the circle drawn. Input is followed by a single line with $x = y = r = 0$, which should not be processed. For example:

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

## C.3 Output

For each test case, output on a single line the number of pixels that are lit when the specified circle is drawn. Assume that the entire circle will fit within the area of the display. For example:

```
4
88
```

# D Matryoshka Dolls (`dolls.{c,cc,java}`)

## D.1 Description

Adam just got a box full of Matryoshka dolls (Russian traditional) from his friend Matryona. When he opened the box, tons of dolls poured out of the box with a memo from her:

*Hi Adam, I hope you enjoy these dolls. But sorry, I didn't have enough time to sort them out. You'll notice that each doll has a hollow hole at the bottom which allows it to contain a smaller doll inside.*
*...*
*Yours,*
*Matryona*

Adam, who already has so many things in his showcase already, decides to assemble the dolls to reduce the number of outermost dolls. The dolls that Matryona sent have the same shape but different sizes. That is, doll $i$ can be represented by a single number $h_i$ denoting its height. Doll $i$ can fit inside another doll $j$ if and only if $h_i < h_j$. Also, the dolls are designed such that each doll may contain at most one doll right inside it. While Adam is stacking his gigantic collection of Matryoshka dolls, can you write a program to compute the minimum number of outermost dolls so that he can figure out how much space he needs in his showcase?

## D.2 Input

The input consists of multiple test cases. Each test case begins with a line with an integer $N$, $1 \leq N \leq 10^5$, denoting the number of Matryoshka dolls. This is followed by $N$ lines, each with a single integer $h_i$, $1 \leq h_i \leq 10^9$, denoting the height of the $i$th doll in cm. Input is followed by a single line with $N = 0$, which should not be processed. For example:

```
4
5
7
7
3
3
10
10
10
3
10
999999999
100000000
0
```

## D.3 Output

For each test case, print out a single line that contains an integer representing the minimum number of outermost dolls that can be obtained by optimally stacking the given dolls. For example:

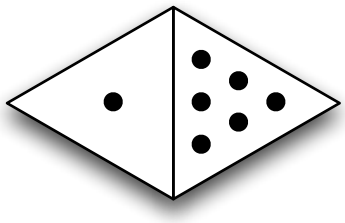
```
2
3
1
```

# E Equilateral Dominoes (`equilateral.{c,cc,java}`)

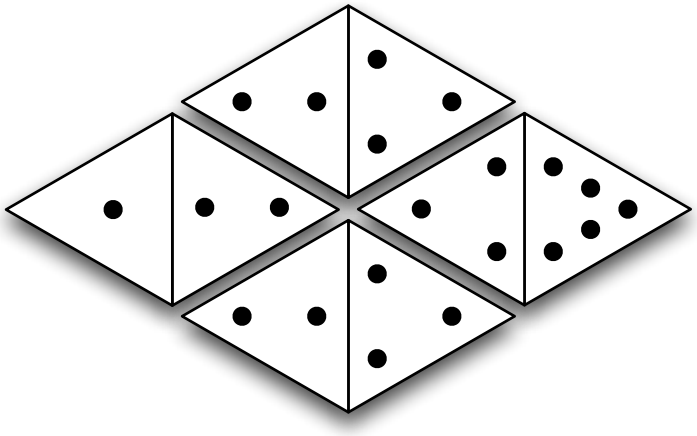
## E.1 Description

We've all seen regular dominoes before—they look like rectangular tiles, twice as long as they are wide, with pips designating a number between 1 and 6 on each of their two ends. The object of a dominoes game involves some variation of tiling the dominoes so that the numbers on adjacent dominoes match up.

Now what if we were to make the dominoes out of equilateral triangles, like this:



An equilateral domino is shaped like a quadrilateral made up of two equilateral triangles. Again, the triangle at each end depicts a number from 1 to 6. Two equilateral dominoes can be tiled next to each other if the domino ends on either side of their shared edge have the same value, and if neither domino overlaps another. Once you have begun a tiling, additional dominoes may only placed adjacent to one or more of the dominoes already on the table. In other words, two more disjoint groups of dominoes does not constitute a valid tiling. For example, equilateral dominoes may be tiled like this:



Your task is to find a best tiling, in some sense, of some equilateral dominoes. If you score one point for every edge that is shared between two dominoes, what is the best way to tile a given set of equilateral dominoes to achieve the highest score?

## E.2 Input

The input consists of multiple test cases. The first line of each test case contains an integer $N$, $1 \leq N \leq 6$, the number of equilateral dominoes in that set. This is followed by $N$ lines with two integers each (values between 1 and 6 inclusive), with each line indicating the pip values on each of the dominoes in the set. Input is followed by a single line with $N = 0$, which should not be processed. For example:

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

## E.3 Output

For each test case, output a single line containing the highest tiling score that can be achieved with the given set of equilateral dominoes. If there is no valid tiling of the dominoes, output a zero. For example:

```
4
0
1
```

# F Four Gate Push (`fourgate.{c,cc,java}`)

## F.1 Description

You are working hard on your Protoss builds in StarCraft II, especially the 4 Gate Push. You've come upon a tough problem, however, which is how to determine the distribution of zealots, stalkers, and sentries to maximize your army strength. Recall (although you should already know!) that zealots cost 100 minerals and no gas, stalkers cost 125 minerals and 50 gas, and sentries cost 50 minerals and 100 gas. Given your current economy and how much each unit increases your army strength, determine the maximum army strength you can obtain.

## F.2 Input

The input consists of multiple test cases, one on each line. Each test case has 5 integers $M$ ($0 \leq M \leq 50{,}000$), the amount of minerals you have, $G$ ($0 \leq G \leq 50{,}000$), the amount of gas you have, $Z$ ($0 \leq Z \leq 1{,}000$), the strength of a zealot in your army, $S$ ($0 \leq S \leq 1{,}000$), the strength of a stalker in your army, and $E$ ($0 \leq E \leq 1{,}000$), the strength of a sentry in your army. Input is followed by a single line with $M = G = Z = S = E = 0$, which should not be processed. For example:

```
500 400 10 20 15
0 0 0 0 0
```

## F.3 Output

For each case, output a single line containing the maximum army strength you can obtain. For example:

```
95
```

# G Game Rigging (`game.{c,cc,java}`)

## G.1 Description

You've decided to host a StarCraft II tournament to decide who is the very best StarCraft II player at Stanford. A lot of your friends are entering, and you want one of them to win because the prize is two tickets to the next StarCraft II World Finals! Luckily, you get to arrange the games and you have some information about which players will beat which other ones, i.e. match-ups which have a guaranteed outcome. The tournament is played as a series of games where in each game two players (of your choice) who have not yet been eliminated play each other. This continues until only one player remains, and he is the winner. Can you set the tournament up so that one of your friends wins?

## G.2 Input

The input consists of multiple test cases. Each test case will start with two integers $N$ ($2 \leq N \leq 100{,}000$), the number of players in the tournament, $K$ ($1 \leq K \leq N$), the number of your friends in the tournament, and $M$ ($0 \leq M \leq 100{,}000$), the number of match-ups for which you know the outcome. The next line will contain $K$ integers between 1 and $N$, indicating which of the players are your friends (indices corresponding to the order of the input). The following $M$ lines will contain two integers each; a line containing `A B` indicates that if players `A` and `B` play each other, player `A` will always beat player `B`. Input is followed by a single line with $N = K = 0$, which should not be processed. For example:

```
4 1 3
1
1 2
1 3
2 4
0 0 0
```

## G.3 Output

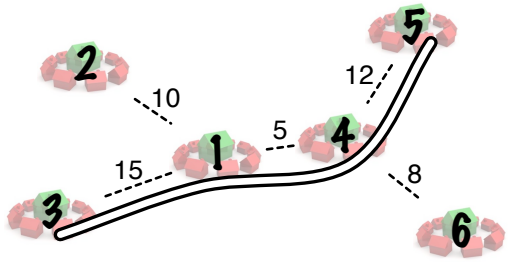
For each test case, output one line which contains either "`yes`" or "`no`" (without quotes) indicating whether you can guarantee that one of your friends wins the tournament. For example:

```
yes
```

# H Highway Construction (`highway.{c,cc,java}`)

As head of the Accessible Commuting Movement (ACM), you've been lobbying the mayor to build a new highway in your city. Today is your lucky day, because your request was approved. There is one condition though: *You* must provide the plan for the best highway artery to construct, or else it's not going to happen!

You have a map that shows all communities in your city, each with a unique number, where you may place highway on-ramps. On the map are a set of roadways between pairs of communities, labelled with driving distances, which you may choose to replace with your highway line. Using this network of roadways, there is exactly one route from any one community to another. In other words, there are no two different sets of roadways that would lead you from community A to community B.




You can build a single highway that runs back and forth between any two communities of your choosing. It will replace the unique set of roadways between those two communities, and an on-ramp will be built at every community along the way. Of course, residents of communities that will not have an on-ramp will have to drive to the nearest one that does in order to access your new highway.

You know that long commutes are very undesirable, so you are going to build the highway so that longest drive from any community to the nearest on-ramp is minimized. Given a map of your city with the roadways and driving distances, what is the farthest distance from any community that someone would have to drive to get to the nearest on-ramp once your new highway is complete?

## H.1 Input

The input consists of multiple test cases. Each test case is a description of a city map, and begins with a single line containing an integer $N$ ($2 \leq N \leq 100{,}000$), the number of communities in the city. Then $N-1$ lines follow, each containing three integers, $i, j$ ($1 \leq i, j \leq n$), and $d$ ($1 \leq d \leq 10{,}000$). Each line indicates that communities $i$ and $j$ are connected by a roadway with driving distance $d$. Input is followed by a single line with $N = 0$, which should not be processed. For example:

```
6
2 1 10
3 1 15
1 4 5
4 5 12
4 6 8
0
```

## H.2 Output

For each city map, output on a single line the farthest distance from any community to the nearest on-ramp of the new highway. For example:

```
10
```