The Internet Programming Contest: A Report and Philosophy

Owen Astrachan
Duke University
ola@cs.duke.edu

Vivek Khera
Duke University
khera@cs.duke.edu

David Kotz
Dartmouth College
David.Kotz@dartmouth.edu

(listed aphabetically)

Abstract

Programming contests can provide a high-profile method for attracting interest in computer science. We describe our philosophy as it pertains to the purpose and merits of programming contests as well as their implementation. We believe that we have successfully combined the theoretical and practical aspects of computer science in an enjoyable contest in which many people can participate.

The contests we describe have distinct advantages over contests such as the ACM scholastic programming contest. The primary advantage is that there is no travel required—the whole contest is held in cyberspace. All interaction between participants and judges is via electronic mail.

Of course all contests build on and learn from others, and ours is no exception. This paper is intended to provide a description and philosophy of programming contests that will foster discussion, that will provide a model, and that will increase interest in programming as an essential aspect of computer science.

1 Introduction

Programming contests can provide a way of attracting interest in computer science, of honing analytical skills, and of having fun. Contrary to methodologies espoused in [8] and debated in [7], we hold the view that programming computers is an integral part of computer science accessible to beginning students. Programming can be used to develop theoretical concepts in an environment that provides immediate and useful feedback.

The type of problem used in a contest and the method of administering a contest have a significant impact on the contest's success both from the point of view of those participating in the contest and from the validity of the contest as viewed by computer scientists. Of course the difficulty of the problems in a contest should depend on the level of expertise of the contestants; one expects harder problems in a contest in which graduate students compete than in a contest for high school students.

In this paper we report on a philosophy of programming contests and its realization in a global contest open to any individual and group with access to electronic mail. Participation in this contest has grown considerably since its inception two years ago. The first contest [2] was only "advertised" on one Usenet news group (`comp.edu`) one week prior to the contest date in October, 1990. Even with this short notice the contest was truly global; teams participated from the United States, Canada, Sweden, Australia, and New Zealand. Although we had anticipated a handful of teams, over 60 competed in the three hour contest which was held in real-time over the Internet. There were 330 program submissions processed semi-automatically in this three hour period. The second contest, held in November 1991, was eagerly anticipated by many participants of the first contest. This contest was announced much earlier and to a wider group of people. This resulted in a drastic increase in the number of participants, again more than we expected. Additional countries represented included Finland, the United Kingdom, India [1], Guatemala, and Belgium. There were over 200 teams from more than 100 sites (including one high school). This contest lasted three and a half hours during which there were 713 program submissions. The third contest, held in November 1992, involved approximately 290 teams — a roughly 40% increase in participation compared to the 1991 contest. Teams came from at least 14 countries on five continents. New countries included Chile, Brazil, Hong Kong, and Lithuania; there were also two high schools fielding teams. During the three hour contest, 890 program submissions were judged.

2 A Contest Rationale and Manifesto

The Internet Programming Contest (IPC) was inspired by and modeled after the ACM scholastic programming contest [6, 4, 3, 5], but designed with a different philosophy than that contest. Our primary purpose was to foster interest in the contest, to allow anyone to compete, and to have fun. We wanted our programming problems to be challenging and to test knowledge of fundamentally important topics in computer science, but to be entertaining as well. Our goal was not to test typing speed, but thinking speed: all of our problems have solutions of less than 200 lines of code. Because the IPC awards no prizes other than bragging rights, we do not have the same worries inherent in a contest such as the ACM contest which awards prizes of considerable value. For example, we make our solutions and our test data available immediately upon conclusion of the contest, which the ACM contest has not been able to do. Perhaps this is because we are free to be second-guessed by contestants when the contest is over since not as much is at stake.

Although the description of the 1988 ACM contest [6] included a methodological description of problem selection, such descriptions have been absent from the reports on the most recent contests [4, 3] (no report on the 1989 contest appeared in a SIGCSE bulletin). Our philosophy closely matches the description given for problem design in the 1988 ACM contest:

> Problems requiring solutions of more than about 150 statements are immediately suspect ...problems should be interesting and novel (or at least exhibit some novel twist), should require solutions demonstrating knowledge of computer science, and should allow comprehensive and definitive testing of purported solutions.

We also wanted to encourage participation at all levels. The ACM contest has, properly, restricted the level of the contestants in the past three years. Whereas the 1990 ACM contest permitted contestants at any level of graduate study, the past two contests have required teams to consist primarily of undergraduate students with graduate students restricted to those in their first two years of study. In addition, the number of contestants per team has decreased from four to three. We agree with the principles motivating these changes: the contestants should be on a roughly equal footing. Again, because only bragging rights are awarded in the IPC we have an open division in which anyone may compete. In the 1991 contest this division included several faculty and industrial teams. If we have erred, perhaps it has been on the side of difficulty; many teams were not able to solve any problem in the 1991 contest. In 1993 we plan to offer another division intended for those in their first years of study. The problems in this category will be easier than those we have used in the past. Since we depend only on an honor system to enforce proper team placement, this should be of great value in increasing participation and satisfaction.

Finally, as far as we are aware, this is the first global programming contest held in cyberspace. The term *Internet* Programming Contest is a slight misnomer, because the participants do not really need to be on the Internet, they just need fast electronic mail access. This permits many teams that would be unable to participate in some contests for financial reasons to participate in our contest. Although at least one ACM regional contest is run in a distributed manner (the Mountain Region permits teams to travel to one of several regional sites), this kind of distributed solution is problematic since, for example, problem clarifications need to be coordinated in some manner. Holding the contest in cyberspace using electronic mail eliminates travel expenses, increases the level of automation, and considerably decreases the computer and personnel resources needed by the contest host.[1] This allows broader participation by the teams, and allows the organizers to put more effort into producing a quality problem set.

3 A Closer Look at Problem Sets

The problems used in the contests must also be teaching tools. We feel that good problems will challenge people to be problem solvers, not just programmers. To this end, we design all of our problems with the goal that the contest should reward problem-solving speed rather than typing speed. Each of our problems also includes a background section describing the relevance of the task to real-world problems or computer science theory.

As is often the case with problems from the ACM regional and national contests, many of our problems are veiled descriptions of common computer science algorithms, such as finding the shortest path in a graph, determining the minimum spanning tree, or finding the convex hull of a set of points. Other problems involve solving algebraic or geometric problems, financial calculations, simplified VLSI tasks, robotics, automatic program generation, and artificial intelligence. None of our problems require tricky input or output processing. We do have some problems that are of the "just-for-fun" variety such as "The Cat in the Hat" [9] problem, which appeared in the 1991 contest in memory of the late Theodore Seuss Geisel.

[1]The Fall 1992 Mid-Atlantic regional contest was conducted in a distributed manner using the software we developed for the Internet Programming Contest.

We also design the problems so that the choice of a naïve algorithm would likely result in a program that would not work on all possible test cases. This requires the participants to read the specifications carefully and take into consideration the problem size when deciding on algorithms and in making design decisions.

Finally, from a practical standpoint, the problems are rigorously proofread by several other programming contest veterans, and a solution is written by at least one person who is not the problem's author. This process refines the problem specification to eliminate most of the clarifications in advance.

4 Logistics

Each participating site is expected to have a contest administrator to install the needed software and to make printed copies of the problem set. The administrator is the contact person to whom all pre-contest mailings are sent. The initial mailing includes full contest rules, contest guidelines, and software for submitting problems. This is sent out several days prior to the contest so that the software can be tested and the materials distributed. The day before the contest, the problem set is sent to the administrator, with instructions to keep it secret until the start of the contest. The contestants only know that there will be between three and ten problems.

Due to the global nature of the contest, any starting time would inconvenience people in at least one timezone. Some sites would start the contest at 3AM local time, while others would start at 3PM local time. We therefore select a starting time that is convenient for us (early evening).

Starting one hour prior to the contest start time, each team is required to run a registration program. This program prompts them for some information (names, education level, electronic mail address, *etc.*) and then sends an electronic mail message to contest headquarters. Once the message is received and processed, an acknowledgment is sent to the team with their assigned team number. All further correspondence relies on this team number for identification. Each team may have up to four team members, and is placed into one of three categories for scoring purposes:[2]

I All undergraduates or high school students.

II Mixed:

- At least two undergraduates
- No member with more than two years in graduate school
- No member with a Ph.D. degree.

III Open (everyone else)

When local time coincides with the contest starting time, each team is given a copy of the problems by the local administrator. The teams then have three hours from this time to complete and submit all work. All timing is done using the local clock on the team's computer to avoid problems with clock skew and mailing delays.

Once a team has decided that a problem is completed, they run a submission program which prompts them for their team number, the problem number, the language used,[3] and the name of the file containing the program. This information is packaged and sent via electronic mail to Duke where the program is compiled and run with our secret test cases. One of the following results of judging are sent back to the submitting team: (a) Correct, (b) Incorrect Output, (c) Incorrect Output Format, (d) Incomplete Output (e.g., no output for some test cases), (e) Compiler Error, (f) Runtime Error, (g) Runtime Limit Exceeded (one minute wall-clock time on the judges' machine).

Any questions about the problems are submitted by running another program called `clarify`. This program is like the submission program, but sends a question rather than a program.

5 Results

The results of the 1990 contest are presented in Table 1 for the three divisions. Only the top four teams in each division are listed. Only one team was able to solve all six problems.

Final standings from the 1991 contest are presented in Table 2 by division. No team solved all six problems, nor did any one team attempt to solve them all. Based on this data and post-contest comments from the contestants, we believe that the problems in this contest may have been a little bit too mathematically oriented for many participants.

The final results from the third programming contest are summarized in Table 3. Of the six problems presented, 127 teams were able to solve at least one. Each of the problems was solved by some team and two teams solved four problems. Scheme was introduced as a new language in 1992. On an experimental basis, a few teams used Haskell and Ada. Next year, these will be officially supported, along with some other new languages.

[2]We plan to divide division I into two divisions for the 1993 contest.

[3]We currently accept submissions in Classic or ANSI C, Pascal, Haskell, Ada, and Scheme; the languages used in the future may change to include, for example, C++.

Rank	Solved	Institution	Team Name
		Division I	
1	5	Carnegie-Mellon University	Brand X
2	3	Williams College	Brain Lecture
3	2	St. Olaf College	
		Division II	
1	4	Duke University	Team 1
2	3	Michigan State University	Students
3	3	University of Virginia	
		Division III	
1	6	University of Maryland	Defending National Champs!
2	4	University of Tennessee	
3	4	University of Maryland	Terrapins

Table 1: 1990 contest results

Rank	Solved	Institution	Team Name
		Division I	
1	4	University of Linkoping, Sweden	Team Lysator
2	4	University of Maryland	7-11 cashiers
3	3	Rice University	The Screaming Schemers
		Division II	
1	4	University of Pennsylvania	Dining Philosophers
2	4	Virginia Tech	Fighting Gobblers
3	4	University of Waterloo	WatBrains
		Division III	
1	4	University of Central Florida	Old Timers
2	4	Univ. of Sydney Australia	LUD Industries
3	4	Sun Microsystems	Rocky Mountain Technology Center

Table 2: 1991 contest results

Rank	Solved	Institution	Team Name
		Division I	
1	4	Virginia Tech	Virginia Tech 1
2	3	Rice University	Screamin' Schemers
3	3	Oberlin College	Macho Hacker Geeks
		Division II	
1	3	Stanford University	Cardinal Sin
2	3	University of Waterloo	WatTheHell
3	3	Linkoping University, Sweden	Team Proj-P +
		Division III	
1	4	University of Maryland	Battle-Scarred Hackers Anonymous
2	3	University of Hildesheim	Joghurt
3	3	Carnegie Mellon University	CHEM

Table 3: 1992 contest results

6 Implementation details

The contest software used by the contestants, consisting of three programs called `register`, `submit`, and `clarify`, is very Unix-centric, but we have been able to cope with submissions from a few non-Unix sites. We assume the existence of some of the basic tools that are traditionally standard under Unix (e.g., `sed`, `awk`, and `Mail`). Almost all of the judging software was written using Perl scripts.

At the judges' machine, the receipt, unpacking, compiling, running, and initial testing of program submissions is completely automatic. We call this process *RoboJudge*. Careful specification of the problems allows for our software to easily test a submitted program's output for correctness. Only when a submission appears to be incorrect is it necessary for a human judge to intervene and decide on a score. Recording and reporting the score is also automated.

Scoring the problems also involves taking into account the amount of time a team took to solve a problem. Therefore we use the submission time (as recorded by the `submit` program), rather than receipt time, in our calculations. Since we assume that that each team starts the contest when their local clock reaches the appointed start time, the use of submission time is a valid measure of the elapsed time taken to solve a problem. To handle multiple time zones, we used Universal Time (i.e., Greenwich Mean Time) for all time values.

Most of the automation relied on the coordination between the submit program and RoboJudge. The submit program carefully formats the email messages it sends so the judging programs can identify the problem being attempted, the compiler to use, and the team being judged. The `register` and `clarify` programs also use fixed-format email messages to automate and speed their processing.

When a message arrives at the judging machine, it is identified and placed in an appropriate bin: registration request, clarification request, judge request, or a special bin for unrecognized mail. RoboJudge picks problems to judge from the judge request bin. Similarly, the contest registration program uses the registration request bin. The clarification requests are picked up by a clarification response program run by one of the judges. Because of the existence of RoboJudge, one human judge can monitor and handle several judging sessions at once. Five human judges are able to process several hundreds of submissions during a three-hour contest, with a typical submission requiring less than two minutes processing time, from receipt to response.

7 Conclusion

We have received many positive comments about the contests and are pleased that they have been successful. We thank the Department of Computer Science at Duke University for allowing us to use the computing resources that we needed to run these contests.

We believe the contests were fun for all involved. We certainly enjoyed organizing and running them. The tension level at Contest Central (the room in which judging takes place) was intense during evenings that the contest was held. Post-contest enthusiasm was high, and several people have volunteered to help in future contests.

We hope that the 1993 contest, and future contests, will continue to elicit the same degree of enthusiasm as have past contests.

Finally, we hope that other contest administrators will use our contest as inspiration for new contests and as an impetus towards developing a contest philosophy. An explicit problem philosophy is an important part of ensuring a successful contest, though we do not anticipate that all will share ours.

7.1 Contest materials

The contest materials for the three contests held so far are available via anonymous ftp from the host `cs.duke.edu` in the directory `dist/misc/acm_contest`. Materials found in this directory are the problem sets, copies of complete results, and the software needed to run the contest.

Acknowledgments

We would like to thank the following people for their support and help in running these contests: Lars Nyland, Steve Tate, Dave Reed, Amr Fahmy, Ólafur Gudmundsson, and Rob Kolstad. We also want to thank our advisors for realizing that time spent away from research in developing this contest was not time lost, but time well-spent.

References

[1] KSR Anjeneyulu. Report on the 2nd internet programming contest. *The Software Bulletin*, Number 1, January 1992. Published by the National Centre for Software Technology (NCST) in Bombay, India.

[2] Owen Astrachan, Vivek Khera, and David Kotz. The duke internet programming contest. Technical Report CS-1990-21, Duke University, Durham, NC 27706, December 1990.

[3] J. Comer, J. Perry, B. Poucher, R. Rinewald, and S. Wileman. Results and problems from the 1991 acm scholastic programming contest finals. *SIGCSE Bulletin*, 24(2):48–54, 1991.

[4] J. Comer, J. Perry, B. Poucher, R. Rinewalt, and S. Wileman. Results and problems from the 1990 acm scholastic programming contest finals. *SIGCSE Bulletin*, 22(4):15–22, 1990.

[5] J. Comer, R. Rinewalt, P. Ryan, and W.B. Poucher. The ACM scholastic programming contest — 1977 to 1990. In *The Papers of the Twenty-first Technical Symposium on Computer Science Education*, page 256. ACM Press, February 1990. SIGCSE Bulletin V. 22 N. 1.

[6] L.E. Deimel. Problems from the 12th annual ACM scholastic programming contest. *SIGCSE Bulletin*, 20(4):19–28, 1988.

[7] Edsger Dijkstra, David Parnas, William Scherlis, M.H. van Emden, Jacques Cohen, Richard Hamming, Richard M. Karp, and Terry Winograd. A Debate on Teaching Computer Science. *Communications of the ACM*, 32(12):1397–1414, December 1990.

[8] Edsger W. Dijkstra. On the Cruelty of Really Teaching Computer Science. In *SIGSCE Technical Symposium on Computer Science Education*, pages xxv–xxxix, 1989.

[9] Dr. Seuss. *The Cat in the Hat Comes Back*. Basic Books. Random House, 1958.