Strategies for Encouraging Individual Achievement in Introductory Computer Science Courses

Eric Roberts
Department of Computer Science
Stanford University
eroberts@cs.stanford.edu

Abstract

Students in introductory computer science courses often vary widely in background and ability. As a result, some students are bored by the pace of presentation, while others struggle to keep up. This paper describes our experience using open-ended assignments and programming contests to capture the interest of our strongest students without adversely affecting the educational experience for the other students in the class. This approach has been markedly successful, particularly for highly motivated students, who are often able to work well beyond the level of the class. The paper also includes a survey of student reactions to the various extra-credit opportunities, which indicates that many student value this component of the class even if they do not participate directly in these activities.

1 Introduction

A major challenge in teaching introductory computer science to a large, diverse audience is the fact that the wide variation in student capabilities makes it hard to find the appropriate level of instruction. If the level of presentation is too high, slower students find it difficult to keep up, which is disastrous in a course that is highly cumulative in nature. On the other hand, pitching the course too low often means that the strongest students are bored by the lack of challenge. As a result, these students—many of whom are ideal candidates for the computer science major—do not perform up to their potential and may be discouraged from future study.

Over the past decade, I have experimented with several strategies designed to capture the interest and enthusiasm of the most capable, highly motivated students without compromising the value of the introductory courses for the remainder of the class. The principal components of that strategy are as follows:

• *Take whatever steps are appropriate in your institution to minimize the range of background and interest level you find in each class.* In institutions facing large introductory enrollments, it is probably wise to offer multiple tracks in the introductory sequence. At our institution, for example, we offer both a standard CS1-CS2 sequence and an accelerated introduction that covers that material in a single term. Allowing students to select courses that match their ambition level reduces the variability within each class, making them easier to teach.

SIGCSE 2000 3/00 Austin, TX, USA

• *Provide extensive opportunities for extra-credit work.* In my introductory courses, students have many opportunities to go beyond the course requirements and undertake more challenging tasks. The primary forms for these extra-credit opportunities are open-ended extensions on assignments and programming contests.

• *Develop an incentive system that offers tangible, long-term benefits for student initiative and achievement.* In our department, students who rise to the challenges provided by the introductory courses have more opportunities to become involved in teaching and research as undergraduates.

Individually, none of these ideas is new. Tracking is used in many programs and goes a long way toward solving the problem of keeping the best students excited [4]. Contests and other similar incentives have often been exploited for their ability to motivate students [1, 16]. This paper illustrates how these techniques can be applied in an integrated way.

As with any pedagogical technique, the design of contests and other extra-credit incentives must take into account the nature of the environment. Some strategies work better in large universities than they do in smaller colleges. On the other hand, the characteristics of small colleges sometimes make it possible to use these strategies in a way that would be impossible in a larger institution. I have used variations on these strategies at liberal-arts colleges as well as large universities. In both settings, these techniques have proved highly successful as a way to increase student motivation.

2 Providing extra-credit opportunities

Even though our institution offers both a standard and an accelerated version of the introductory course, students in each course vary considerably, both in their level of motivation and in the rate at which they can assimilate new programming concepts. Because we are committed to making the introductory course accessible to a wide range of students, the pace at which we can proceed is limited. As a result, some students are capable of moving much more quickly than the class as a whole. Because this group contains the students we would most like to attract as majors, it is essential to provide additional challenges that keep them interested in and excited about the material.

In our introductory courses, we use two strategies to encourage extra effort and capture the interest of the most highly motivated students: open-ended extensions to assignments and optional programming contests. Each of these strategies is discussed in more detail in the individual sections that follow.

2.1 Open-ended extensions to assignments

The programming assignments used in our introductory courses include specifications that enumerate in detail the requirements students are expected to meet. Students who satisfy those requirements—which include criteria for judging both programming style and correctness—receive a grade of √+, which corresponds to an A in the computation of the final grade. The grading system, however, includes two higher grades, + and ++, that are reserved for students who go beyond the nominal requirements by incorporating extensions of their own design. The ++ grade is awarded very rarely and cannot be given by the grader alone. Any ++ grades require the approval of the instructor and are reserved for the highest level of student work—a program so good that "it makes you weep."

Because this grading system is described in a 1995 SIGCSE paper [15], its details are not a central focus here. The important point is that the system provides an incentive for extra work without making students feel compelled to undertake it. A grade of √+ on an assignment represents an A, pure and simple. The assignment and final course grades are assigned without any adjustments to make the distribution fit an arbitrary curve. Thus, a student who does well on the individual assignments will do well in the course, no matter how many people are undertaking the extra-credit work.

The optional extensions to assignments have proven to be very popular with students. In my introductory class in 1998-99, 64 percent of the students received extra credit on at least one assignment. Five students out of the class of 332 received extra credit on every assignment. Almost all of these premium grades were of the + variety. Only three ++ grades were awarded during the entire term, two of which went to the same student.

Designing programming assignments so that they offer exciting opportunities for extra-credit work requires a certain amount of planning. Over the last few years, I have discovered that my approach to developing assignments has changed dramatically in view of the success I have had in using open-ended extensions to encourage extra work. A self-contained assignment that offers little or no room for extension no longer seems like an interesting assignment. I instead look specifically for problems that are likely to encourage students to go beyond the minimal bounds. Judging from student comments, the assignments have become much more interesting as a result.

2.2 Optional programming contests

Although many students invest the effort required for + and ++ grades, we also hold several programming contests each term, which give students a more public opportunity to demonstrate their skill and creativity. We announce the contests with considerable fanfare and demonstrate the winning entries in class. The top prize is a perfect score on the final exam, which has proven to be an exceptionally good motivator, mostly because it eliminates so much pressure at the end of a term. In most cases, the prize itself is meaningless in terms of the final course grade, because the students who win the contests are usually those who would get A's in any case.

There are three contests in a typical course: one at the beginning, one near the middle, and one at the end. The first contest establishes the pattern and encourages students to engage in extra-credit projects from the beginning of the term. The second maintains the momentum and is usually scheduled to fill the break in homework assignments that occurs around the midterm exam. The third serves as a capstone project for the students who undertake it, and allows them to exercise all the skills they have acquired.

Holding a contest near the end of the term, however, requires a certain amount of care. For most students, the pressure of approaching finals and work for other courses makes it impossible to devote much energy to an optional contest. Moreover, because students by this time have a sense of how much work goes into the winning entries, few students are willing to commit the necessary time if the work is unrelated to other course requirements. The best way to encourage participation in an end-of-term contest is to link it to one of the final assignments. Because students must do the assignment anyway, the idea of putting in extra effort to create a contest entry is not nearly so daunting.

As with the extensions to the assignments, contests are completely optional. Because students can get an A in the course without undertaking any extra-credit work, they are not pressured to enter the contests and do not feel threatened by them. Students who choose not to enter can simply sit back and enjoy the contests, which are often described—even by nonparticipants—as one of the best features of the course.

In my experience, contests are particularly effective in large classes. For one thing, having a large number of students helps ensure that the number of contest entries reaches a critical mass. Large classes also increase the incentive for students to enter contests by allowing them to compete for recognition in an environment in which they might otherwise be anonymous. The contests offer students a chance to stand out above the mass, which has proven to be a powerful incentive.

3 Providing continuing incentives

The programming assignments and contests described in the preceding sections all include an incentive in the form of extra credit for the course. It is important, however, that students get something back for their work that is more lasting. For our students, one of the important motivations for undertaking such challenges is that distinguishing themselves in the introductory course gives them additional long-term opportunities in the department. Some students, for example, have the chance to undertake independent research with faculty members. A substantial number later take part in teaching the introductory course as section leaders [15]. Competition for these positions is stiff, and students who have shown initiative in their early courses have a definite advantage.

4 Selected contest descriptions

This section describes some of the most successful programming contests we have used in our introductory courses in recent years. My hope is that the descriptions supply enough detail about these projects for other instructors to use them in their own courses.

4.1 Karel the Robot contest

For more than 15 years, the first week of our CS1 course has been devoted to Karel the Robot, a simulator for a simple robot that provides, in the words its creator, Richard Pattis, a "gentle introduction to programming" [11]. We have found Karel extraordinarily useful in getting students to understand basic programming concepts and the more general issue of algorithmic problem solving. Moreover, Karel's simple structure and delightful presentation help students overcome any fears they might bring to the course.

Using Karel also makes it easy to introduce a contest at the beginning of CS1. To enter, students simply program Karel to solve a problem of their own choosing. They can, for example, program Karel to produce artistic patterns, illustrate a story, or tackle some conceptually difficult task. The entries are judged by the entire course staff, who award a prize for each of two categories:

• *Algorithmic sophistication.* This prize is awarded to the Karel program that solves the most challenging task in the most interesting way.

• *Aesthetic merit.* This prize is awarded to the program that, in the opinion of the judges, has the greatest literary, artistic, or entertainment value.

In a typical year, I get entries from approximately 10 percent of the class. Although the weakest entries are no more ambitious than the homework assignment, the overall quality of the submissions is quite high. The winning algorithmic entries have included an implementation of J. H. Conway's game of Life [7], a calculator program capable of extracting square roots, and a 2500-line program to calculate the determinant of a 3×3 matrix, written by a student whose previous programming experience was limited to BASIC in elementary school.

Offering the aesthetic prize has the effect of opening the contest to students who do not see themselves as hot-shot programmers. Many of the entries in this category are extremely creative. One of my favorites in recent years was a program entitled "Karel—Prince of Dotmark," written by a student in her second week of college, in which Karel acts out the story of Shakespeare's *Hamlet*. The program was accompanied—as encouraged in the contest rules—by a delightful comic narrative, written "with deepest apologies to Shakespeare," that was extremely well received when I read it to the class.

4.2 Graphics contest

Our CS1 course makes extensive use of a portable graphics library [14], which allows students to create engaging, interactive programs. The graphics library makes an ideal foundation for the second contest in CS1, in which students are asked to submit a graphical application of their own design. The prize categories are the same as those in the earlier Karel contest. The algorithmic prize is based on the difficulty of the programming and the sophistication of the image it generates; the aesthetic prize is awarded on the basis of the artistic value, as evaluated by the staff.

The algorithmic prize in the graphics contest typically goes to students who use the library to display Mandelbrot and Julia sets, which they have seen in *Scientific American* [6]. Because I encourage students to undertake research, it is not surprising that the algorithms are derived from published sources. I insist, however, that the students code the algorithms themselves so that they conform to the software engineering principles emphasized in the course.

The entries that have won the aesthetic prize in the graphics contest are among the most impressive student programs I've seen. One of the most memorable was a 12-minute animated sequence about an evil tyrant who forced his unfortunate slaves to undertake a set of Herculean labors that bore an uncanny resemblance to the previously assigned homework. The artwork was beautifully designed, and the demise of the tyrant was a big hit with the class.

4.3 Adventure contest

For reasons discussed in the earlier section on "Optional programming contests," I usually link the final contest of any term to the last programming assignment. As with all assignments, I publish a detailed set of requirements so that students know exactly what they have to do, but leave open the possibility of adding extended features on top of the required functionality. I then make the assignment into a contest by announcing that the student who implements the most ambitious extensions wins the standard contest prize of a perfect score on the final exam.

This strategy means that it is important to design the last assignment so that it offers considerable opportunities for exciting extensions. Starting in 1995–96, I introduced a new assignment at the end of our CS1 course, which was to implement a simple version of Adventure, a text-based exploration game developed by Willie Crowther in 1975. Although the Adventure assignment is quite challenging, its basic requirements are easy to explain. At the same time, it allows students to exercise considerable creativity as they compete to design the most intriguing game.

Students were given the opportunity of working in pairs for this assignment, which I think helped build enthusiasm for the contest. The students poured an enormous amount of time and energy into the assignment; several students told me they had put over 100 hours into the project. That effort was clearly visible in the quality of the contest entries. Some students added graphics and sound, while others designed elaborate puzzles that showed considerable imagination and thought.

4.4 Sorting contest

The first contest in our CS2 course focuses on algorithms and computational complexity. After several lectures and an assignment on sorting, students are given the challenge of implementing a function that sorts an array of integers as efficiently as possible. The entries are judged entirely on how much time they require to sort arrays of 10,000 randomly chosen integers.

The sorting contest works well for the following reasons:

• *It encourages students to conduct their own research into algorithmic strategies.* The winning student three years ago consulted several books on algorithms and invested an estimated 30 hours tuning his program to the point at which it ran almost 20 percent faster than the second-place entry. In each of the last two years, the winning entry has run even faster.

• *The contest requires students to think hard, not only about writing efficient code, but also about the complexity of the algorithms they have seen.* Students who enter experience directly what it means for an algorithm to be $O(N^2)$ as opposed to $O(N \log N)$. They also learn that complexity analysis is not the whole story, and that it is critical to consider the constant factors.

• *The optimal algorithm is not at all obvious, given the specifics of the problem.* Most contestants adopt a hybrid strategy that uses Quicksort until the size of the array falls below a certain threshold, followed by an insertion sort pass to complete the sorting operation [3]. On the other hand, students can exploit the specific characteristics of the problem—sorting 10,000 two-byte integers—to derive a more efficient solution.

• *Students quickly learn the importance of good testing.* In a surprising number of cases, students submit contest entries that fail to work correctly for certain input arrays.

4.5 Darwin contest

Our second contest in CS2 is closely tied to the fifth programming assignment, in which students write a simulator for a dynamic evolution game called *Darwin* [10]. Darwin was invented several years ago by lecturer Nick Parlante and has proven highly effective because

• The program involves both graphics and animation, which keep the students excited.

• The implementation is complex enough for students to learn the value of modular decomposition and abstract data types, both of which are used extensively in Darwin.

• The game is instructive in its own right because it gives students a feel for assembly language programming.

The Darwin game is played in a two-dimensional world divided into small squares and populated by creatures of various species. Each creature lives in one of the squares and is controlled by a program particular to its species. These programs are written in a simple assembly language with primitives for moving forward, turning left and right, and infecting a creature in the next square so that it becomes a member of your own species. The language also includes control instructions that allow a creature to examine the square it faces and to change the order of program execution.

The game is played by cycling through the individual creatures in a random order, giving each a turn. A turn consists of executing some instructions from the program for that species. A creature can execute any number of control instructions without relinquishing its turn; executing any other instruction ends the turn and gives the next creature a chance. When all the creatures have had a turn, the cycle is complete, and the process begins all over again. Between turns, each creature maintains its own program counter and starts up again from the point at which the last turn ended. The ultimate goal is to infect other creatures, which turns them into your own kind. Play ends either when one species has completely taken over the world or when 1000 complete cycles have occurred, in which case the species with the greatest population is declared the winner.

After implementing the Darwin simulator an assignment, students have the option of writing a program for a new species that is adapted for survival in the Darwin world, which they then submit as a contest entry. Over the years, students have come up with intricate creature designs that are remarkably clever and creative.

The entries in the Darwin contest are initially evaluated by a program that conducts a round-robin tournament in which every contest entry plays three games against each of the other entries. The top two contenders from the round-robin tournament are then pitted against each other for an in-class, best-of-five showdown.

4.6 BASIC contest

As their sixth assignment, students in our CS2 course write a simplified interpreter for BASIC [8]. The requirements of the assignment itself are quite specific. Students must devise a data structure for storing program statements and then implement simple forms of common statements, including **LET**, **PRINT**, **INPUT**, **GOTO**, **IF**, **RUN**, and **LIST**. Because they have already seen a simple recursive-descent parser for arithmetic expressions in class, the program is not as difficult as it might at first appear; in fact, students find it easier than the Darwin assignment, which has the additional complications of graphics and two-dimensional arrays.

Even though the assignment itself has well-defined requirements, the underlying problem of building a BASIC interpreter is open-ended. For the BASIC contest, students are invited to extend their assignment to implement as much of standard BASIC as they can. The assignment is graded on the basis of how well it meets the stated requirements. Any extra features make the program eligible for consideration in the contest. The section leaders nominate their best programs, which are then reviewed by a selection committee to select the most sophisticated interpreter.

Enthusiasm for this contest is high, in part because it was a BASIC interpreter that gave the Microsoft corporation its start. The contest handout begins with the following epigraph, which appears in the book *Gates*, by Stephen Manes and Paul Andrews [9]:

> Allen rushed to the dorm to find Bill Gates. They had to do a BASIC for this machine. They had to. If they didn't, the revolution would start without them.

In each of the last five years, the winning entries have implemented most of the features in standard BASIC, including arrays, strings, subroutines, and library functions. As a result, these student-designed implementations have been able to execute large BASIC programs, including a complete blackjack simulator taken from a programming text one of the students had used in high school.

5 Evaluating student reactions

Over the years, my subjective impression has been that providing extra-credit opportunities has increased the level of excitement—not just for the students who take part—but for everyone connected with the course. Such subjective impressions, however, are subject to considerable bias, particularly in a large class. While it is certainly clear that students who enter the contests seem to enjoy them, there is

the possibility that other students may find these activities intimidating, reducing their satisfaction with the course.

To evaluate student perceptions more broadly, I conducted a survey at the end of the course in 1998-99. A copy of the survey form appears in Figure 1. The final question, which asks students to describe their reaction to the presentation of contests in class, deliberately sought a narrative response to avoid suggesting reactions to students. Asking directly whether students had been "intimidated" would likely elicit a large number of positive responses, but so would asking them whether they were "inspired." The responses were read independently by three readers, who rated each response as positive, negative, or neutral (which includes no response). The identifications were highly consistent among the three readers; in particular, there were no responses that were identified as positive by one reader but negative by another. The results of this evaluation for the 133 responses look like this:

positive	negative	neutral
84 (64%)	11 (8%)	37 (28%)

These data make it clear that, on the whole, reaction to the contests is positive. There were, however, 11 students whose responses were interpreted as negative. The tone of the negative responses ranged from mild expressions of intimidation ("I could never do that") to a couple of angry comments about how the contest winners were clearly overqualified for the class. Most reactions, however, were more along the lines of one respondent who wrote "Cool! Put me in awe of my fellow students, but it made me realize that even I was capable of the work that these people do."

I undertook a regression test to determine whether student reaction correlated with any of the other responses on the questionnaire, such as sex or prior programming background. The strongest correlation was with the number of contests entered, but none of the correlations were statistically significant at the standard $p = 0.05$ level.

There is also some evidence that the use of extra-credit incentives does encourage students to major in computer science. Of the 63 students who applied for a highly selective program for sophomore majors in 1998-99, 13 specifically identified the programming contests as a reason for their decision to choose this field.

Figure 1 Contest survey

Year (sophomore, junior, graduate, etc.) _____

Sex ____

Prior programming background before taking this course:

None ____ A little ____ Quite a bit ____

On how many assignments did you go beyond the minimal requirements?

How many contests did you enter?

If you did enter contests, answer the following questions:
In how many contests were you a winner or a runner-up?
In how many contests did you receive an honorable mention?

Whether or not you entered contests, what was your reaction to the presentation of contest entries in class?

6 Conclusions

As the descriptions of the submissions in section 4 show, students who take advantage of extra-credit opportunities are often able to accomplish amazing things. Moreover, the survey data in section 5 suggests that inclusion of these activities as part of the course, whether or not they actually take part. The other students in the class get caught up in the energy of the contests to the point that everyone seems to move faster as a result. The course staff can watch the students create remarkably sophisticated programs and take pride in the results. And, as instructor, I have the pleasure of seeing highly talented students get so turned on by computer science and the challenge of working on open-ended problems that they learn more in a week than many of my students do in the entire term.

References

[1] Owen Astrachan, Vivek Khera, and David Kotz. The Internet programming contest: a report and philosophy. *SIGCSE Bulletin*, March 1993.

[2] Richard Austing, Bruce Barnes, Della Bonnette, Gerald Engel, and Gordon Stokes. Curriculum '78: Recommendations for the undergraduate program in computer science. *Communications of the ACM*, March 1979.

[3] Jon Bentley. Programming pearls: how to sort, *Communications of the ACM*. April 1984.

[4] Kim Bruce. Attracting (and keeping) the best and the brightest: an entry-level course for experienced introductory students. *SIGCSE Bulletin*, March 1994.

[5] James S. Coan. *Basic BASIC.* New York: Hayden Book Company, 1970.

[6] A. K. Dewdney. Computer recreations. *Scientific American*,, February 1989.

[7] Martin Gardner. Mathematical games. *Scientific American*, February 1970.

[8] John G. Kemeny and Thomas E. Kurtz. *BASIC Programming* (second edition). New York: John Wiley and Sons, 1971.

[9] Stephen Manes and Paul Andrews. *Gates.* New York: Simon and Schuster, 1993.

[10] Nick Parlante, et al. Nifty assignments panel. *SIGCSE Bulletin*, March 1999.

[11] Richard E. Pattis, Jim Roberts, and Mark Stehlik. *Karel the Robot: A Gentle Introduction to the Art of Programming* (second edition). New York: John Wiley and Sons, 1995.

[12] Eric Roberts. Using C in CS1: Evaluating the Stanford experience. *SIGCSE Bulletin*, March 1993.

[13] Eric Roberts. *The Art and Science of C: A Library-Based Approach.* Reading: Addison-Wesley, 1995.

[14] Eric Roberts. A C-based graphics library for CS1. *SIGCSE Bulletin*, March 1995.

[15] Eric Roberts, John Lilly, and Bryan Rollins. Using undergraduates as teaching assistants in introductory programming courses: An update on the Stanford experience. *SIGCSE Bulletin*, March 1995.

[16] Connie C. Widmer and Janet Parker, Programming contests: what can they tell us?, *Journal of Research on Computing in Education*, Spring 1988.