

# Plagiarism by Student Programmers

Neal R. Wagner

The University of Texas at San Antonio  
Division Computer Science  
San Antonio, TX 78249, USA

Copyright 2000, Neal R. Wagner. All rights reserved.

Many forms of academic dishonesty are found in programming courses. This article looks at *plagiarism of programs*: when students copy all or part of a program from some source and submit the copy as their own work. This includes students who collaborate and submit similar work. Such plagiarism is felt to be common, though the true extent is hard to assess. Quoting from [12]: ``As with Nessie of Loch Ness there does not seem to be a clear picture of this beast, only a variety of eyewitness reports."''

This article focuses on two events, referred to here as the *UTSA experiment* and the *MIT incident*. During the fall 1990 and spring 1991 semesters at the University of Texas at San Antonio, an experiment detected 29 students in a data structures course involved in plagiarism. In the spring 1990 semester, the Massachusetts Institute of Technology disciplined 73 students for cheating in a beginning programming course [1]. In both cases there was a special check for copying that was not common practice at the school and was unanticipated by the students. These events span the spectrum from a smaller state university to a prestigious private school, and in both cases students involved ranged from freshmen to seniors. The events suggest a significant plagiarism problem in programming courses everywhere -- perhaps more of a problem than many realize.

Plagiarism of student programs has fundamental differences from ordinary plagiarism -- making it easier to carry out and harder to detect. In contrast with a student writing an English term paper, beginning programming students typically work in the same environment on the same problem, and one expects similar resulting programs. The output data may have an exact required form. Students can copy and exchange programs in machine-readable form, and can use an editor to make extensive changes that do not affect program execution.

The terms ``copying" or ``plagiarism" refer to unequivocal cases of copying an entire program or most of a program. Even if there are many edited changes, there must be a structural similarity between the two versions. This article uses the criterion that someone *knowledgeable in computer science* should be convinced *beyond reasonable doubt* that the similarities result from *copying*. In the event of suspected copying, the school or department would have to apply their own criteria for plagiarism. What seems obvious to a computer scientist might not be convincing to someone else.

This article first considers what students do when they copy and why they do it, including an exhaustive list of reasons for copying. Next the two events mentioned above are examined in detail. Finally the paper presents extensive recommendations. The focus is on prevention, rather than detection and punishment. To quote from the excellent article [12]: ``Preventing the incidence of cheating is far preferable to expending effort establishing that cheating has occurred and taking action against the students involved. When it is possible to do so without compromising educational quality, courses and supporting computer usage [should] be organized to avoid situations in which students might be pressured or tempted to cheat." The present article covers some of the same material as [12], though with a focus on two striking and sobering instances of plagiarism. One conclusion is that an unannounced and unexpected check for plagiarism in any programming course might uncover a surprising amount.

As the author of this paper, I am not trying to claim some self-righteous moral high ground. Students who plagiarize must be dealt with firmly but with compassion and with an understanding of their right to a presumption of innocence. In the end plagiarism must be resisted one way or the other because it undermines learning.

## What Students Do

Academic dishonesty is not limited to plagiarism -- consider the various ways of cheating on exams. By definition the most successful episodes go undetected. Certainly students can show incredible ingenuity. For example the author encountered a student in a large section computer course, who, after an exam was passed back, retyped it on new ditto masters, ran off a single copy, and filled in mostly correct answers from the answer sheet. As a final twist, the student ``graded" it, arriving at a grade of 84, when the actual grade had been 48. The student later produced this new exam, claiming that the teaching assistant had transposed digits in recording the grade. This student was only caught because the new exam, though typed with amazing accuracy, was in the wrong typeface.

Students commonly copy all or part of a program for an assignment. They copy from other current students, from past students, from files of old programs, and from textbooks and other sources of programs. Some of this copying might be acceptable to certain instructors.

Students also get help from others in writing or debugging their programs. The amount of help given and the level of help acceptable to an instructor varies greatly.

As another common practice, students edit a program's output before printing. Even a reasonable student may get 90% of the code complete, producing most of the correct output, and then make up 10% of the code along with the correct output. This practice argues in favor of students mailing their program source for compilation and execution by the instructor.

## How Students Copy

Some students submit a copy of another student's program with almost no changes -- perhaps only the name changed. (In the UTSA experiment, one student forgot to change the name.) Other students go to great effort to make the copied program appear different from the original. Some of the changes they make are listed in [Table 1](#).

Some students obtain program source in electronic form, usually from someone cooperating with them, or less frequently by breaking into a computer account. Others will copy from a listing, either being given the listing, stealing it, or finding a discarded listing. Collaborating students may work together on a single source program.

As an example, two separate programs turned in from the UTSA experiment had many similarities, including the executable statement `count := count` as the target of an if-statement and `Counts := Counts` as the target of the corresponding if-statement in the other program. This is a bit of redundant insanity that has no effect on program execution. Most likely the students were worried about negating the if expression. The same unusual construct in both programs is what one might call a "smoking gun" -- an unequivocal indication of copying.

## Why Students Copy

Students copy a program because they have trouble writing it themselves (lack of ability), or because they do not have time to finish it (lack of time), or because they want a better result.

[Table 2](#) presents a surprisingly large number of factors that add to the chances of copying. Each item is followed by typical justifying comments. There seem to be enough factors here for every student to find some reason to copy -- perhaps that is part of the problem. [Table 5](#) examines these factors with a goal of altering them to lessen the likelihood of copying.

Many motivations from [Table 2](#) were found with MIT's students. The article [1] refers to a "sense of entitlement" developed by some students -- they felt they had put in so much time that they should get a good grade. Students are quoted as saying: "You could check for cheating in any class and you'd certainly find a significant portion of the people cheating -- I think it's one way of getting through MIT." One senior mechanical engineering student said he "felt under intense pressure because everyone else was getting 100's and his program did not quite work." One official is quoted as saying: "What is emerging from our experience is a sense that many MIT students see the institute as an obstacle course set up by the faculty. Many feel that the required work is clearly impossible by straightforward means, and that any means that makes survival possible is allowed. We found students who felt that the major problem was getting caught."

## The UTSA Experiment

The author devised an experiment at UTSA to look for copied programs in all sections of a data structures course over two semesters. Students received the following written rules about copying: "In practice, for this course, you may discuss assignments in general terms, but you are not allowed to share any details of actual algorithms or of program code. You may help someone else debug their program as long as you do not start substituting in your own code when there are problems. Turning in a copy of someone else's program, even a copy with extensive changes made to it, is a very serious offense in this course." Students were told to hand in a program listing and to E-mail the program source. They were *not* told about the experiment but were told in writing that mailing the source was "another basic course requirement, to resolve any possible questions or problems."

The author suggested to the instructors that they exercise "normal" vigilance in checking for copying, i.e., no elaborate checking, and no checking at all across sections. The instructors knew about the experiment, but understood that no results would be available until after they turned in grades. The author decided in setting up the experiment to use results only for statistical purposes and not to initiate any disciplinary action.

## The MIT Incident

During the spring 1990 semester, 239 students finished the course "CE 1.00: Introduction to Computers and Problem Solving" at the MIT. This course, described as "popular but difficult" in [1], was a requirement of the six civil engineering students enrolled, but is taken electively by about 40% of MIT undergraduates. The enrollment figures in [Table 4](#) show that many students take the course early in their studies. Non-civil engineering majors take the course because it satisfies a science distribution requirement and helps get employment.

The instructor, Professor Nigel Wilson, did not give written rules for the course (he does now), but said [1] he had "spelled out what he felt were clear guidelines on acceptable collaboration.... It would be all right [for students] to work together in the initial stages of each assignment .... But ... it would not be acceptable for students to collaborate on writing the programs." Help was available from teaching assistants, and students were encouraged to seek this help when they got stuck. They could also get help from other students to get around specific bugs -- they just could not code jointly.

Late at night a student gave Professor Wilson a tip that there was a problem with plagiarism in the course. Professor Wilson is quoted as saying [1]: "The student had worked very hard and was very frustrated that others were getting more credit than they deserved." Source for assignments had been submitted both in hard copy and electronically. Professor Wilson and his teaching assistant first monitored one assignment and seeing the results, extended the monitoring to the four remaining ones.

## Software to Detect Plagiarism

Ideally it should be harder to copy successfully than to write the program from scratch, and software

detection should work even if students know the particular algorithm.

The problem at issue is a special case of file or string comparison. There is a large body of literature on such problems [11], most of which is not of use here because of the changes that are often made to copied programs.

Reference [3] gathers statistics about program features, but this approach does too weak a comparison and generates too many false alarms. The reference [6] considers a complex comparison of the structure of programs as trees of procedures.

At MIT, the software used a statistical approach similar to [3], with an objective function based on the number of for's, and's, if's, else's, and or's. Possible duplicates identified by this function were examined visually.

The author and two undergraduate students at UTSA used the approach of [5], along with a preprocessing stage that removed comments, made letters lowercase, deleted names except for reserved words, replaced constant names by the constant, prettyprinted, and removed material in quote marks. Then for each pair of resulting files, the software counted the number of lines that occur exactly once in each file.

Dick Grune of Vrije Universiteit in Amsterdam has written a sophisticated program to test for plagiarism in a variety of languages. This software does a drastic transformation of each source program to a much shorter character string, and then in pairs finds matching substrings of decreasing lengths. It does a good job of detecting plagiarism and is available via anonymous ftp [4]. The author used this tool to recheck all the programs in the UTSA experiment.

## Results of the UTSA Experiment

The UTSA experiment uncovered an alarming amount of copying -- far more than the author expected (see [Table 3](#)). There were 8 students in the fall and 21 students in the spring involved, where this might not mean that they knew copying occurred.

Two "A" students reported four incidents of apparently stolen listings. In each case a student involved in other episodes of copying later submitted a copy of the stolen program.

Two collaborating students who turned in copied programs mailed their source at nearly identical times -- in one case the timestamps differed by 1.5 seconds.

From the stored data it would be possible to retrieve the names of the students involved. There are students who did not earn their grade, though sorting out who copied from whom would be a daunting task, even involving students who have graduated and left town. Also two instructors gave permission to carry out the experiment with the understanding that only statistics would ever be released. For these

reasons the author has decided not to follow up on any students. Some individuals who reviewed this article expressed misgivings about course failures caused by hidden detection systems. Others were upset that nothing was done to discipline any dishonest students.

## Results of the MIT Incident

At MIT an astounding amount of copying occurred -- 80 out of 239 students enrolled were referred to the MIT Committee on Discipline. In the end, 73 students were disciplined. Quoting the instructor, Nigel Wilson [1], ``... an eight-month investigation determined that [two] students stole others' [listings], and at least one student [obtained another's password] and stole the program electronically. But the majority of the 73 students worked together in some fashion, and it was difficult to tell who did what work. A great majority felt what they had done was inappropriate. But some felt it was O.K., and some had not understood my instructions. They had genuinely engaged in joint coding." The largest collaborating group was 8, but 2 was the most typical size. About 10% of the submitted assignments were copies with no changes (except the name).

Probation was the dominant punishment, but several students were suspended for one semester. No students were expelled. About half of the reduced grades were to zero and remaining ones were reduced by 50%. [Table 4](#) gives additional statistics related to the incident.

## General Recommendations

There are two ways to reduce plagiarism. One is the *hardline* strategy: the instructor must catch students who copy and make sure they never copy again. Such approaches involve vigilance, no concessions, and strong punishment. These people might object to elements of a course that lessen the need to copy, and might favor ``entrapment" strategies, where apparent opportunities to copy are clever traps for copiers. Many people feel this hardline approach does not create a good atmosphere for learning. Reference [12] recommends avoiding ``an oppressive environment in which students are *constantly aware* that they must not even give the appearance of cheating."

Instead, this article promotes a *compassionate* strategy: try to eliminate copying by changing factors that induce students to copy. Make no mistake, the author is not proposing to *condone* copying. One needs firm policies for dealing with instances of copying, including a graduated scale of punishments. Also one ought to be able to explain to students why they should not copy.

The real issue is learning to program, and copying is one of many things that gets in the way of learning. In catching and punishing copiers, one is not directly promoting any learning. The summary recommendation: emphasize prevention rather than punishment.

## Strategies to Lessen Copying

This section suggests specific actions to lessen the student's perceived need to copy or to lessen the likelihood of copying. The items of [Table 5](#) mirror the reasons for copying given in [Table 2](#). [Table 6](#) lists additional actions. Many of these recommendations would improve the course independent of the copying issue. [Table 5](#) and [Table 6](#) can be used as exhaustive checklists.

Several items in Tables 5 and 6 are controversial or deserve further comment. The issue of deadlines and late programs is complex. No deadlines at all or too lenient a policy toward late programs is detrimental -- students have many demands on them from other courses. If deadlines are not tight, a late program can start a domino effect, where each program suffers in quality or is late due to the work required by the previous late program. Some people argue that tight deadlines with no late programs accepted is best for the students. They say this is like the situation in industry. If this is the policy, it should be clearly stated. There should also be a policy for students with unusual problems. One compromise allows students to work past the deadline, but they must turn in *something*, their best effort, by the deadline.

Surely all programmers have encountered a persistent bug that was difficult to isolate. Students have particular problems with such bugs -- finding them and recovering from them is an important part of their education. But one should provide help in debugging and should be flexible in special circumstances.

The level of difficulty of programming assignments is another problem area. Some instructors add features until an assignment is quite hard. They may add artificial features that serve only to require a more difficult program. The author recommends the opposite: taking a substantial task and adding artificial simplifying assumptions that make the program easier to complete. Instructors should also consider assignments that focus on a single topic or goal.

Getting the rules disseminated, understood, and agreed to is hard. This was a problem in the MIT incident [1], where a student said Professor Wilson's instructions on what was acceptable collaboration and what was cheating were unclear, whereas Professor Wilson felt he had given clear (verbal) guidelines. Rules must be carefully crafted to be clear, complete, and unambiguous, and they must be written.

A colleague of the author once gave a class written rules, writing in part: "You are to complete your assignments individually .... The work you turn in must be your own work." He also said verbally: "Do not work together. In case of any doubt about what is acceptable, come talk to me first." At a university hearing the lawyer for a student accused of collaborating argued successfully that the rules were not clear. The student *completed* his work individually, it was argued, even though he didn't *start* it individually. The work he handed in *was* his own work -- and also someone else's.

This last anecdote illustrates the increasing fear of litigation. Students and faculty now share a new set of rules, where a student's case might be defensible, *not* because the student actually cheated or not, *not* because the student *knew* he or she was cheating or not, but because the instructor could not *prove* that the student was undeniably cheating and because the instructor could not *prove* that the student *knew* that the cheating was against the rules.

The need for ethics education has been repeatedly recognized, both by the CSAB accrediting agency and in the new *Computing Curriculum 1991* [13]. The author recommends a short segment (several lectures introducing ethical issues) in the first or second programming course and a short segment in an advanced course, such as software engineering. Students in such segments carry on a surprisingly lively discussion. In fact they often expect no controversy initially -- they expect agreement with their position and are themselves surprised by the opinions of their peers. Good texts for segments on ethics are [2], [7], and [8]. See [9] for a short article that would stimulate discussions. The ACM Self-Assessment Procedure dealing with ethics in computing [14] contains the ACM Code of Ethics and case studies. See also [10] for case studies.

Unfortunately some computer scientists regard ethics instruction as a waste of time. They argue that there is no room in the syllabus for ethics, and that it would be hard to present in a uniform and responsible manner. They also argue that teaching ethics will not make students behave ethically. All these reasons conspire to keep ethics out of a curriculum. Ideally, a department should have one or two faculty members responsible for helping the students discuss ethics. Then students and faculty alike would realize that no one is trying to make them behave according to some ethical norm. The goal is for students to think about these issues and form their own opinions.

## What to do in Case of Copying

Now suppose you are an instructor faced with copying. [Table 7](#) gives actions to consider. Some instructors feel the best outcome is to avoid a formal complaint -- perhaps a zero grade on the assignment. This is often a quick and convenient resolution, but not necessarily the best course -- a student may later contest the outcome. There is no record of the offense and no possibility of an escalated penalty in case of a repetition.

A common outcome of an interview with two students who submit copied programs is for them both to deny any wrongdoing. One could ask the students to explain their code, as recommended in [12], but such a small "oral exam" puts a lot of pressure on students.

A second common outcome of an interview is for students to admit to some collaboration but to deny line-by-line copying, even in the face of clear line-by-line similarity. One should be careful here: for example a teaching assistant may have helped several students with exactly the same suggested code.

The discovery of plagiarism can be disconcerting and frustrating, even demoralizing. Support from peers, university discipline committees, and administrators may be weak. Student reactions can be unsettling, as they deny any plagiarism or justify their actions with no sign of remorse, with faked remorse, or with remorse only at getting caught -- as they bring their own lawyer to a hearing. The actions listed in [Table 7](#) often come when the instructor has many other commitments.

## Why Not Copy?

If one feels strongly that students should not copy, then one ought to give them reasons. [Table 8](#) attempts an answer. In the end one hopes that each student finds reasons for not copying. These issues could also be discussed in an ethics segment in the computer science curriculum.

## Sample Forms

[Table 9](#) presents a sample rules form, asking students to sign indicating their understanding. One could use a similar form that asked them to *agree to abide* by the rules, or a form with no signature. In contrast, [Table 10](#) promotes more of a humane strategy involving a *commitment*, almost like a contract.

## Conclusions

This article has described two significant plagiarism events: an experiment at the University of Texas at San Antonio used to detect cases of copying of programs, and an unexpected plagiarism incident at the Massachusetts Institute of Technology. A great deal of copying occurred during the two semesters of the experiment at UTSA, and even more took place in the MIT incident. These results suggest that copying may be widespread. The article has also presented reasons why students copy and strategies to lessen this copying, suggesting positive alternatives to threats of punishment.

In summary, the author recommends that schools make educational goals foremost and that they try to emphasize prevention. A school should study its plagiarism problem carefully, interviewing faculty and students, and looking over all the items of [Table 5](#) and [Table 6](#). The author particularly suggests various improvements -- to the hardware, to the working environment, to assignments, and to the curriculum, including the addition of ethics instruction. Finally, a school should use software to detect plagiarism.

## Acknowledgements

The author first thanks Walt Moore and Chris Kanute, who worked on the software in the UTSA experiment. Nigel Wilson provided information about the MIT incident. Jeff Popyack read several revisions and gave many useful suggestions. Additional help came from George Butchee, Dave Eberly, Bob Hazy, Dennis Kern, Hugh Maynard, Myles McNally, and Holly Roe.

## References

1. Butterfield, F. Scandal over cheating at MIT stirs debate on limits of teamwork, *The New York Times*, (May 22, 1991), A23.
2. Erman, M.D., Williams, M.B., and Gutierrez, C. (editors) *Computers, Ethics, & Society*. Oxford Univ. Press, New York, N.Y., 1990.

3. Grier, S. A tool that detects plagiarism in Pascal programs. *SIGCSE Bulletin* 13, 1 (1981), 15-20.
  4. Grune, D. SIM: software similarity tester, Version 2.1, Vrije Universiteit, Amsterdam (June 18, 1991). Available via anonymous ftp at ``star.cs.vu.nl" in the directory ``/pub/dick/similarity\_tester".
  5. Heckel, P. A technique for isolating differences between files. *Comm. ACM* 21, 4 (April 1978), 264-268.
  6. Jankowitz, H.T. Detecting plagiarism in student Pascal programs. *The Computer Journal* 31, 1 (1988), 1-8.
  7. Johnson, D.G. *Computer Ethics*. Prentice-Hall, Englewood Cliffs, N.J., 1985.
  8. Johnson, D.G., and Snapper, John W. (editors) *Ethical Issues in the Use of Computers*. Wadsworth, Belmont, Calif., 1985.
  9. McFarland, M.C. Urgency of ethical standards intensifies in computer community. *IEEE Computer* 23, 3 (March, 1990), 77-81.
  10. Parker, D.B., Swope, S., and Baker, B.N. *Ethical Conflicts in Information and Computer Science, Technology and Business*. Q.E.D. Information Sciences, Wellesley, Mass., 1990.
  11. Sankoff, D., and Kruskal, J.B. (editors) *Time Warps, String Edits, and Macromolecules: the theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, Mass., 1983.
  12. Shaw, M. *et al.* Cheating policy in a Computer Science Department. *SIGCSE Bulletin* 12, 2 (July 1980), 72-76.
  13. Tucker, A.B. Computing Curricula 1991. *Comm. ACM* 34, 6 (June 1991), 68-84.
  14. Weiss, E.A. Self-Assessment Procedure IX: A self-assessment procedure dealing with ethics in computing. *Comm. ACM* 25, 3 (March 1982), 181-195. (See also letters in the December 1982 issue.)
- 

## **Table 1. Changes Students Make to Copied Programs.**

- *Change comments.*
  - *Change names.*
  - *Change case.*
  - *Reformat the text.*
  - *Reorder the text.*
  - *Add or delete redundant elements.*
  - *Redo standard constructs, such as loops.*
  - *Rewrite.*
- 

## **Table 2. Factors Contributing to Copying.**

## Factors That Increase the Likelihood of Copying.

(Refer to Table 5 for strategies to lessen the need and the opportunities to copy.)

### I. Course organization.

- a. *Tight or short deadlines. No late programs accepted. Little or no partial credit.* "I had to copy to meet the deadline." "I had the program 90% finished, but I still get a zero if I turn it in."
- b. *Dull, boring assignments. No connection with real world problems. Lots of busywork.* "Why not copy -- I wouldn't get anything out of this assignment even if I did it myself."
- c. *Assignments that seem impossibly hard.* "You have to copy in this course -- it's the only way to finish."
- d. *Course rules not disseminated, understood, and agreed to.* "I didn't know there was anything wrong with the copying I was doing. And besides I didn't agree not to copy."
- e. *Assignments made and due near the end of the term.* "The instructor piled on work at the end."
- f. *The course poorly taught.* "What a rotten instructor! I didn't feel bad at all about copying."
- g. *Failing students who don't turn in most programs.* "The instructor requires 70 percent of the programming points to pass, regardless of exams."
- h. *Old assignments reused. Assignments taken from reference books.* "My friend did this assignment last semester."
- i. *Large class. Different instructors using the same assignments.* "My friend and I submitted the same code to two different teachers."
- j. *Accepting late assignments, after other students' work has been returned.* "I just copied my friend's program after he got it back."
- k. *A slow or easy start in a course.* "By the time I saw how hard the course was it was too late to drop."

### II. School environment, CS curriculum, computer consultants.

- a. *An environment where students copy without being caught.* "The others are copying -- why not me? If I don't copy I'm at a big disadvantage."
- b. *A history of leniency toward copying when it is detected.* "Even if I'm caught they won't fail me."
- c. *School rules not disseminated, understood, and agreed to.* "I didn't know our school had any rules against copying."
- d. *No ethics in the curriculum. No other discussion of the ethics of copying.* "As far as I'm concerned, there's nothing wrong with plagiarism. Just don't get caught."
- e. *Inadequate or unavailable consultants.* "It took me a week to learn to use the school's computer system with no help, and meanwhile a program was due."
- f. *A terminal course, or a course required of non-CS majors.* "This course is just a silly degree requirement. I don't need to know this material for any other course or for employment."
- g. *Poor advising. Rigid drop policies. Poor course descriptions.* "I got into the wrong course-in over my head and it was too late to drop."

- h. *Students putting off a required programming course.* ``I already had a job contingent on finishing my degree."

### III. Hardware environment.

- a. *Poor hardware, i.e., slow, hard to use.* ``It took five minutes just to get a program compiled."  
b. *Poor working environment, i.e., restricted hours, poor access, no phone access.* ``You wait in line for an hour to get a terminal and then wait for the system to respond."  
c. *Overloaded machines or poor access toward the end of the term.* ``The last week of classes was crazy -- hordes of students all night long."  
d. *Requiring that campus machines be used.* ``I couldn't use my own machine, and I just couldn't make it to campus often enough to finish the programs."  
e. *Listings can be stolen or retrieved from trash cans.* ``I was tempted by a listing sitting there by me."  
f. *Program source easily exchanged.* ``I couldn't believe how easy it was to have a friend mail me his program."

### IV. Factors related to the student.

- a. *External factors requiring a certain performance, i.e., probation, visa status, scholarship.* ``I lose my scholarship if I don't get a `B'." ``I'll be deported if I fail this course."  
b. *External factors interfering with the student's course work, i.e., course overload, employment, emotional problems, family problems.* ``Because of personal problems [health problems, etc.] I didn't have time for the programs."  
c. *Copying or poor preparation in the previous course.* ``I copied in the beginning course, and now there's no way to catch up."  
d. *Bug in a program. Difficulty getting it to work at all or getting it 100% correct.* ``I never did get my version working."  
e. *A sense of entitlement, say from hard work or high tuition.* ``With the tuition I pay [the work I did] I deserve to get through this course and finish my degree."  
f. *Lack of time at end of term.* ``I had two take-home finals and a term paper due the last week. It was physically impossible to finish a program too."  
g. *Desire to impress the instructor.* ``I wanted the teacher to think I was a good student."

---

## Table 3. Results of the UTSA Experiment.

Data Structures I (CS 1723), University of Texas at San Antonio  
Fall 1990 (3 sections), Spring 1991 (3 sections)

Numbers of Students in Various Categories (*italics* = those involved)

Semester	Programs mailed	Students Withdrawn (involved)	Students Passed (involved)	Students Failed (involved)	Total Students (involved)	Episodes of copying
Fall 90	232	27 (1)	52 (5)	7 (2)	86 (8)	4 pairs
Spring 91	271	33 (2)	39 (14)	13 (5)	85 (21)	17 pairs, 2 triples, 1 quad
Both (total)	503	60 (3)	91 (18)	20 (8)	171 (29)	21 pairs, 2 triples, 1 quad

**Details about groups that copied during the Spring 1991 semester**

Student	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Grade	C	C	C	A	F	F	F	F	C	C	C	C	W	A	C	W	F	C	C	B	C
Assign1		1	1		1							1				2	2				
Assign2	1	2	2	3	3	4	4	1		5	5		3								
Assign3	1	2	2	1	2	3	3											4	4		
Assign4	1			1		2	2		3											3	
Assign5	1	2										2		1							
Assign6	1	2						3	3	4	4				2						1

*Explanation:* On each assignment line, students submitting similar programs have the same digit as entry. Thus students 4, 5 and 13 submitted similar programs for assignment two, and they received course grades of ``A'', ``F'', and ``W''. Italics = copying detected during the term.

*Further notes:*

- Student 5 submitted a copy of assignment 3 without changing the name, but the similarity between students 2 and 3 on the same assignment was not noted at the time.
- Student 8 mailed assignment 6 many days after student 9, and in an interview admitted to copying.
- Students 10 and 11 were caught copying assignment 6, and received a ``C'' grade with a zero for that assignment. The copying of assignment 2 was not noted until later.
- There are at least 9 cases of two students copying who then both passed the course.

## Table 4. Results of the MIT Incident.

Introduction to Computers and Problem Solving (CE 1.00)  
 Massachusetts Institute of Technology  
 Spring 1990 (1 section)

### Numbers of students enrolled and involved, broken down by class

	Freshmen	Sophomores	Juniors	Seniors	Graduates	Totals
<b>Students enrolled</b>	92	57	48	35	7	239
<b>Students involved</b>	34	16	17	13	0	80

### Distribution of problem sets duplicated

Number of students with	
One duplicate set	26
Two duplicate sets	22
Three duplicate sets	20
Four duplicate sets	11
Five duplicate sets	1
<b>Total students involved</b>	<b>80</b>

#### Notes:

- Only five problem sets were investigated, so five is the maximum possible number of duplicated problem sets for any one student.
- 33.5% of students were involved (includes stolen listings and stolen programs).
- 30.5% of students were disciplined.
- 15% of all possible problem sets were part of duplicates.
- The largest group that worked together was 8, though 2 was by far the most common.
- There were no cases in which both students involved in a copying episode denied all knowledge.
- There were some equivocal cases which were resolved in the students' favor.

## Table 5. Strategies to Lessen the Need and the Opportunities to Copy (Based on Table 2)

### I. Course organization.

- Review policies on deadlines, late programs, and partial credit:* Don't make deadlines too short. Consider a graduated late penalty. Use a tight deadline that doesn't slip. The author prefers liberal partial credit, especially for minor problems and in beginning courses.

- b. *Create interesting assignments:* Some students will *claim* an assignment is dull-they need motivation.
- c. *Create reasonable assignments:* Don't get carried away. Supply part of the code for a more complex program. Use harder optional or extra credit parts.
- d. *Review course rules:* Supply clear, complete, and unambiguous rules in writing. Decide how much help students can get. Don't forget the issues of lifting code from books.
- e. *Lessen work at term's end:* Set up assignments so students finish programming early. Ease up with the last assignment.
- f. *Improve the quality of teaching.*
- g. *Consider carefully any policy of failing students who don't turn in most of the programs:* Some instructors believe in this requirement, and count exams more than programs, but this increases the chances of plagiarism.
- h. *Don't use repeated or standard assignments:* Even small changes help.
- i. *Avoid large classes or different instructors giving the same assignment:* There is an increased sense of anonymity. Plagiarism detection software may help.
- j. *Review the policy on accepting late assignments:* Again plagiarism detection software may help.
- k. *Don't get off to a slow or easy start:* Indicate the course level early in the term.

## **II. School environment, CS curriculum, computer consultants.**

- a. *Avoid a copying environment:* The strategies of Table 6 may help here.
- b. *Avoid a history of leniency:* The author is *not* recommending no punishment at all-just no emphasis on punishment.
- c. *Review school rules:* Involve faculty and administration in formulation of uniform rules.
- d. *Include a discussion of ethics.*
- e. *Provide consultants:* Tell consultants how much help to give.
- f. *Give non-CS majors a choice of required courses.*
- g. *Check on advising, drop policies, and course descriptions:* Check at the beginning that students belong in the course. Start the programming right away to indicate the course level (and to help finish the programming early).
- h. *Don't allow students to delay taking required programming courses.*

## **III. Hardware environment.**

- a. *Improve the hardware:* Even if you can't change the hardware, be willing to make concessions because of poor quality or availability.
- b. *Improve the working environment:* Be at least sympathetic to problems resulting from a bad environment.
- c. *Help with overloaded machines or poor access at the end of the term:* If this is known to be a problem, schedule due dates well before the term's end.
- d. *Try not to require that campus machines be used.*
- e. *Keep listings from being stolen or found in the trash:* Control the security of student listings including pickup, and disposal as trash. ``Removing a listing should be either difficult or

awkwardly obvious to the others. [12]"

- f. *Keep programs from being mailed:* Turn off the mail feature for beginning students.

#### IV. Factors related to the student.

- a. *Help with external factors requiring performance:* One can be flexible about deadlines and still maintain standards and require the same performance.
  - b. *Help with external factors interfering with course work:* Most schools have some sort of *crisis* option for unusual hardships. Be sympathetic.
  - c. *Avoid poor preparation in previous courses:* Use tight standards in prerequisite courses and perhaps a minimum grade to take the next course.
  - d. *Help with program bugs:* Be understanding in case of a bug.
  - e. *Counter a sense of entitlement:* Attitudes like these might be addressed in an ethics segment added to the CS curriculum.
  - f. *Help with lack of time at end of term:* Make programs due before the end of the term. Recommend that students try hard to finish the last assignment on time.
  - g. *Desire to impress the instructor:* Perhaps ethics discussion would help with such a misguided desire.
- 

## Table 6. Additional Strategies to Lessen the Need and the Opportunities to Copy.

#### V. Recommended.

- a. *Warn:* Warn students that they might be caught. Unusual programs or, even worse, ones with unusual output are particularly noticeable.
- b. *Use detection:* Have students mail source. Use the date and time of mailing. Use the software of [4] to detect copying, and let the students know about it. Pair-wise compare any *incorrect* output data.
- c. *Punish:* Use clear, uniform, unambiguous policies. Include a graduated scale of punishment. Reference [12] suggests that "for a first offense, the penalty should always be more severe than the penalty for failing to turn in the assignment."
- d. *Initiate an honor code:* Honor codes often work because they have been in place for many years and are a given when a student enters. Starting a new honor code is difficult, but some form should be workable at every school.
- e. *Publicize the problem:* Use a student newspaper, a campus radio station, handouts, or posters to say there is a problem that the department will not condone.

#### VI. Reasonable possibilities.

- a. *Count exams for more of the grade:* Many instructors do this. Some use a minimum score on exams to pass the course. This also covers the case of excessive help on programs. The problem is that programming may represent a major portion of the student's effort, and exams are not good assessment tools.
- b. *Use multiple assignments:* This makes copying more conspicuous. Assign separate programs to different subsets of the class.
- c. *Create buffers between assignments:* Provide a zone of time not dedicated to a program, so that late programs don't eat into the time for the next program.
- d. *Use linked assignments:* Link assignments together, a later one depending on earlier ones. A copying student may not be able to modify the program.
- e. *Maintain an audit trail:* Insist that all work on a program make use of "scripts" that create a timestamped log of compiles, executes, and the amount of CPU and clock time needed to write and debug the program.

## VII. Extra Possibilities.

- a. *Ask for an explanation of code.*
- b. *Write and debug a program for the final exam:* This places a lot pressure on beginning students, though it might work when all the students in the class are good.
- c. *Bluff:* Have students mail source in the absence of any comparison program. Warn them that they may be subject to sophisticated detection approaches (that are in fact not in place).
- d. *Allow copying:* Openly allow copying, with programs not counting for much of the grade. This might actually help students realize that they learn by completing the programs.

---

## Table 7. What to do in case of copying -- addressed to the instructor.

- *Decide if clearcut.* Even if the case is unequivocal, decide if you can convince the necessary administrators.
- *Determine the rules at your school.*
- *Discuss with others, particularly the department chairman or college dean.*
- *Respect the students.* Don't try to trick or pressure them. Let them know their appeal rights.
- *Get the facts straight.* Have a witness present at any interview.
- *Check if this is a repeat offense.* Only if the school rules allow it. Don't use any anecdotal evidence.
- *Get as much as possible in writing.*
- *Proceed with formalities.* After the case has left your hands, don't worry about it.
- *Don't do anything hasty.*

---

## Table 8. Why not copy -- addressed to the student.

- *It's not dignified.*
  - *It cheapens and weakens the degree.*
  - *You're not learning.*
  - *You're missing the fun of programming.*
  - *It's not fair to other students.*
  - *It's demoralizing to the instructor.*
  - *It's not playing by the rules.*
  - *It's not professional.*
  - *You might get caught and punished.*
- 

## Table 9. Rules for CS xxxx.

I understand that the following rules hold:

1. The computer account is only for my use and for this course. I must not misuse the account or the computer equipment.
2. Programs handed in by me must not be *direct copies* of programs obtained from some other source. I understand that the term "direct copy" refers to copying work that is not my own, and that making simple changes to an existing program still results in a direct copy.
3. I must not let any other CS xxxx student make a direct copy of one of my programs.
4. In case I use a *segment* of code that I have not written myself, I must cite the source of this code segment and call attention to its existence.
5. I understand that submitting a direct copy as my own work is a serious act of academic dishonesty that can result in significant disciplinary action.
6. I understand that consultations with other students and help in debugging are encouraged, as long as these do not involve the direct copying of each others programs. (The instructor and teaching assistants are also available for consultation and help.)
7. I understand that there will be a substantial penalty for late programs.

Signature

---

## Table 10. Commitment

**Between the instructor and the student regarding the Course CS xxxx.**

**As a student in CS xxxx, I commit to work hard for the course, to do my own work, and to abide by the following conditions:**

1. I will not abuse or misuse my account or the computer equipment.
2. I will not copy programs or written homework from someone else's work. (You are encouraged to consult with other students and help one another in debugging.)
3. I will not let any other student copy my work.
4. I will cite the source of any code segments I use that I have not written myself.

Student's Signature

**In exchange as the instructor for CS xxxx, I commit to work hard to get you successfully through this course. In particular I commit to the following goals, to the best of my ability:**

1. To prepare careful, worthwhile lectures.
2. To prepare fair exams, typed and covering the material from the course.
3. To assign interesting, worthwhile programs in writing.
4. To grade programs and exams promptly, fairly, and carefully, with partial credit for programs or exam answers that are not perfect.
5. To help you write and debug programs, both personally and through assistants.
6. To create reasonable course rules that are given out along with this document.
7. To listen to complaints about the course and correct defects where possible.

Instructor's Signature

---