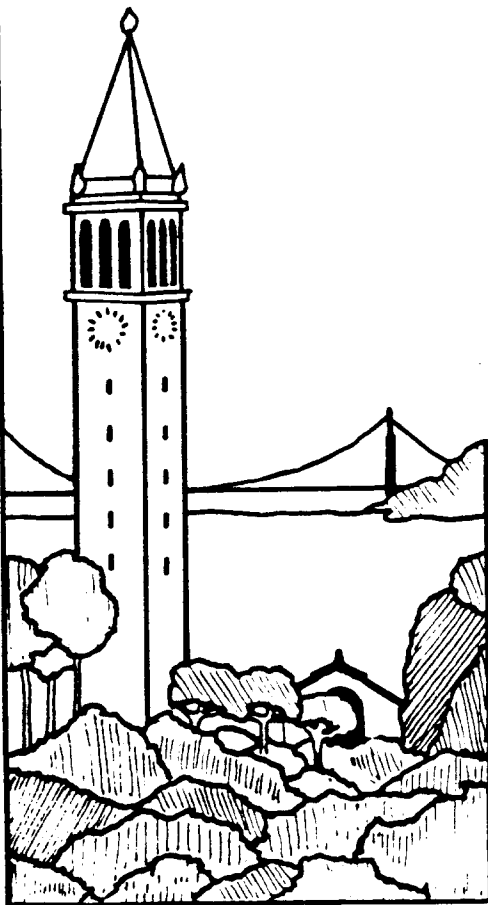


**Everything You Ever Wanted to Know About
"AN INTRODUCTION TO COMPUTERS"**

David A. Patterson



Report No. UCB/CSD 86/288

March 1986

**Computer Science Division (EECS)
University of California
Berkeley, California 94720**



Everything You Ever Wanted to Know About "An Introduction to Computers"

David A. Patterson
February, 1986

Abstract

A new course to introduce computers to liberal arts students was taught for the first time at U.C. Berkeley in the Spring 1985. This course is neither a computer literacy course nor an introductory programming course; it emphasizes the application of computers to solving problems in the liberal arts. It relies on Macintosh personal computers, starting with applications and then Pascal programming before finishing with projects that solve real problems in the students' majors. The goal is to convert traditionally computerphobic students into computer fanatics. This report describes this non-traditional approach, giving the rationale for each decision, and documents its successes and failures.



Table of Contents

Chapter	Page
Chapter 1. Introduction	1
Computer Literacy?	1
Goals of Course	1
Audience of this Report	1
Overview of Report	3
Chapter 2. Key Ideas and Rationale	4
TAs from Outside of Computer Science	4
Some Major Laboratory Sections	5
Faculty Section	6
Labs Using Personal Computers	6
Macintosh versus IBM PC	7
Emphasis on Direct Personal Impact of Computers	7
Applications and then Programming	8
Modify Programs versus Creating from Scratch	8
Pascal versus Basic	9
A Final Project	9
Chapter 3. The Course Organization	11
Labs	11
Lectures	12
Traditional Lectures	16
Demonstrations	17
Exams	17
Guest Lectures	18
Text Books	18
Teaching Programing	19
Grading	19
Chapter 4. The Project	21
Picking a Project	21
Picking A Team	21
Computer Faire	23
TA's Choice	23
Chapter 5. The People Involved with the Course	26
Part I. Responsibilities	26
Lecturer	26
Lab Instructor	26
Head TA	26
TAs	27
Student Aides	28
Part II. Selection of the People	29
Faculty	29
Head TA	30
TAs	30
Student Aides	33
Students	33
Part III. Esprit de Corps	34

Chapter 6. Conclusions	35
Rolling Your Own Course	36
"Freeware"	36
Videotapes	37
Let Us Know What Happens	37
Impact of the Course	38
The Future	39
Acknowledgements	40
Appendix I. History of IDS 110	42
Appendix II. Summary of Student Surveys	44
Statistics on students in class	44
Time spent on class	45
Project	45
Programming	46
Miscellaneous	47
Students quotes before and after IDS 110	48
Appendix III. Guest Lecture Letters	52

Table of Figures

Figure	Title	Page
1.	Our view of "Computer Literacy."	2
2.	List of IDS 110 TAs and their majors for 1984-85.	4
3.	List of section major and TA major for 1985-86.	5
4.	Recommended order of scheduling lectures.	13
5.	Early lecture schedule with labs (Spring 1985).	14
6.	Final lecture schedule with reading assignments.	15
7.	Lecture organization flowchart.	16
8.	Sample projects from Spring 1985.	22
9.	May 6, 1985 Daily Cal article on "TA's Choice" lecture.	25
10.	Academic year schedule of events to prepare IDS 110.	28
11.	Prima Donna sign at entrance of IDS 110.	29
12.	Example questions to ask applicants.	31
13.	20 Popular majors at U.C. Berkeley.	32
14.	Advertisement for IDS 110.	34
15.	Quotes from students before and after IDS 110.	35

Chapter 1. Introduction

Computer Literacy?

Goals of Course

Audience of this Report

Overview of Report

Interdisciplinary Studies 110/110L, "Introduction to Computers," is an attempt to bring the computer revolution to the liberal arts. Our goal is to turn traditionally computerphobic students—those not involved in engineering, mathematics, or physical sciences—into computer fanatics.

Computer Literacy?

This course is **not** a computer literacy course. To me computer literacy is a catch phrase with no semantic content. To illustrate my opinion of that phrase, I draw an analogy to "Bicycle Literacy" in my first lecture. (See Figure 1.) Students in a bicycle literacy class would read books about bicycles, memorize bicycle buzzwords ("a derailleur is ..."), see video tapes of bicycle races, discuss the psycho-sociological impact of bicycles, have guest lectures by people who have actually ridden bicycles, and so on. Students from such a class would sound knowledgeable about bicycles, would surely have strongly held opinions, and would be fascinating conversationalists at cocktail parties, as long as no one asked if they had ever ridden a bicycle.

Goals of Course

Instead of producing brilliant conversationalists, this course asserts that computers are simple, and that college students in any major are able to learn "to ride the mental bicycle." Learning to ride means hands-on experience with a computer twice a week every week, and at the end a project to demonstrate their new found ability.

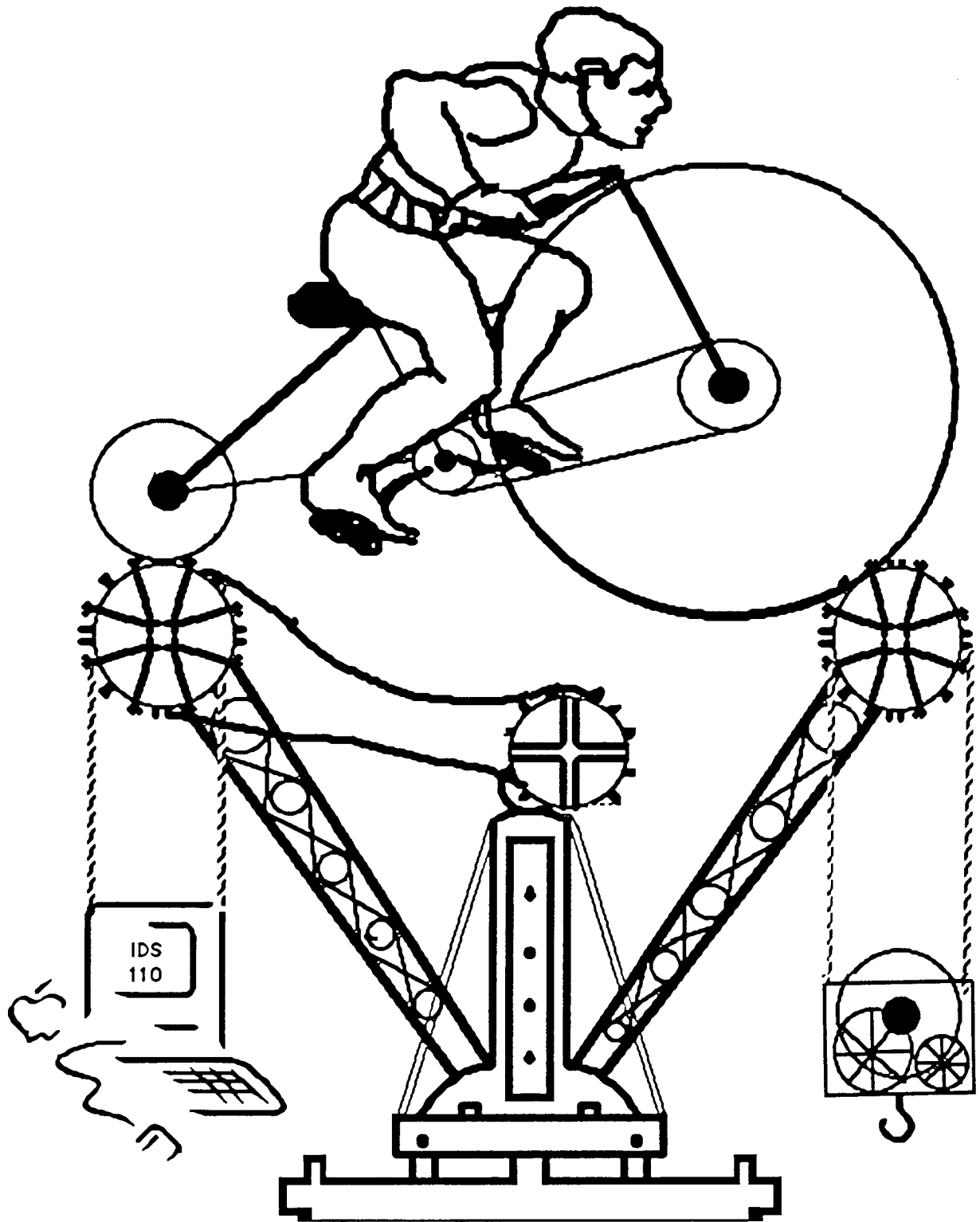
A secondary goal is for this course to be fun for everyone involved: instructors, teaching assistants, and students. Let me warn you that I have an unusual definition of fun: fun is learning a great deal by working hard on a project which is considered valuable. We expect a lot from the students, but they learn a lot in return. To make the course fun for the instructors and assistants, we have reduced the tedious grading chores (see Chapter 3) to allow concentration on teaching, which all of us enjoy and find rewarding.

Audience of this Report

My purpose in writing this report is to document everything I have learned creating and teaching this course. I have two groups in mind as I write this report:

- (1) Instructors, teaching assistants, and student aides who teach IDS 110/110L at Berkeley.
- (2) Faculty at other institutions who might want to use some or all of these ideas in a similar course.

Riding a Computer is as easy as Using a Bicycle . . .



Literacy Doesn't Get You Anywhere !

Figure 1. *Our view of Computer Literacy.*

Overview of Report

As a guide to the rest of the report, I recommend that everyone read Chapter 2, since it contains the philosophy and reasons behind most of our ideas, Chapter 3, since it describes the course organization and also Chapter 4, since it describes the project, probably the keystone of the course. If you are involved in IDS 110, then read about the responsibilities of your job in Chapter 5. If you are thinking of starting a course on your own, be sure to read the section "Rolling Your Own Course" in Chapter 6 and pay attention to the advice in Chapter 5 on selecting the people for the course. Just in case this is not enough reading material, the appendices contain a history of the origins of the course, the results from surveys of students in the first course offering, and copies of letters sent to guest lecturers.

Chapter 2. Key Ideas and Rationale

Teaching Assistants from Outside of Computer Science

Same Major Laboratory Sections

Faculty Section

Labs Using Personal Computers

Macintosh versus IBM PC

Emphasis on Direct Personal Impact of Computers

Applications and then Programming

Modify Programs versus Creating from Scratch

Pascal versus Basic

A Final Project

This chapter describes the most unusual aspects of the course, and the reasons behind them.

Teaching Assistants from Outside of Computer Science

The lab sections are staffed by graduate students from the same majors as the students.

Since this course is not for Computer Science students, there are no Computer Science TAs.

The original reason for seeking external graduate students was quite practical. Berkeley already has more teaching assistantships for Computer Science courses than Computer Scientists to fill them, so I was sure that we would have problems finding another dozen C. S. graduates who were interested in this course.

I also suspected that other fields were not so fortunate, and hoped that talented graduate students in the liberal arts would be available for this course. This was certainly the case. There are talented people all over the Berkeley campus, so our main difficulty was selecting only a dozen TAs from a pool of 70 capable applicants. (See Chapter 5 for TA selection.) Figure 2 lists the TAs for 1984-85.

I would have to count this as one of our most successful ideas, for this was the best group of TAs I have ever worked with. The Berkeley Committee on Teaching agreed with me, for each TA received an Outstanding Teaching Assistant award.

<i>TA Major</i>	<i>TA Name</i>
Anthropology	Rand Miyashiro
Classics	Neel Smith
Education	Denise Kiser
English	Henry Chen
History	Noel Cary
Library Science	Geri Bunker
Music	Joshua Kosman
Near Eastern Studies	Anne Marchant
Philosophy	Mark Bedau
Psychology	Zoe Kersteen
Sociology	Maurilia Flores

Figure 2. *List of IDS 110 TAs and their majors for 1984-85.*

Same Major Laboratory Sections

I thought that it might be interesting to associate these excellent TAs with students in the same major. Since many more students applied to IDS 110 than we could accommodate, we hoped to create homogeneous lab sections by first admitting students with the same major as the TA. We felt this would make the labs more effective—the TAs would be on the same "wavelength" as the students—plus it would allow TAs to create projects that would tie the common subject area to computers.

This was a qualified success. There were some majors that had so many students that it was easy to fill a lab section. English, History, and Psychology were examples. Other TA majors were close enough to popular subject areas that we could fill the sections with similar students. For example, the Anthropology TA took Biological Sciences and the Sociology TA covered the Social Sciences major. Some majors, however, were so small and isolated that they couldn't fill a section—Music and Near Eastern Studies come to mind here—and some graduate programs, such as Education and Library Science, were so demanding that students did not have the time to take the class. Philosophy was an unfortunate example of a major that had plenty of students, but they were not interested in this class. Classics, on the other hand, was an example of a small major that had such an enthusiastic following that it filled one lab section.

In 1985-86 we therefore chose more TAs from the large majors. Figure 3 shows our finalists. Note that we also decided to offer a few multi-major sections so that students from the less popular majors could still take the class. See Chapter 5 for more information about selecting students for the class.

<i>Section Major</i>	<i>TA Major</i>
Architecture	Architecture
Biological Sciences	Entomology
Economics	Business
English	English
History	History
Political Science	Mathematics
Social Sciences	Sociology
Multi-Major	Classics
Multi-Major	Education
Multi-Major	Near Eastern Studies

Figure 3. *List of Section Major and TA major for 1985-86.*

For the sections that were filled with students of the same major, things worked as we had hoped. The TAs were able to better explain concepts to students in their major, and the TAs helped students create excellent projects in the common major.

Faculty Section

We set aside one section just for faculty who wanted to be introduced to computers. When I first thought about the course I realized that in addition to neglected subject areas, there were neglected age groups. On a campus these age groups correspond to faculty and graduate students. We solved the problem for graduate students by making it an upper division course, thereby allowing them to get academic credit. I hoped faculty would learn about computers in this course and then either teach this course or teach a follow-up course in their departments.

My expectations for the faculty were too high. Faculty are so busy that it was hard for them to attend the lab throughout the course. We kept a majority until the project, but the end-of-semester crunch waylaid most of them. We had only 2 projects from the 30 faculty.

I still believe in the faculty section, as it seems better to teach existing faculty about the power and possibilities of computers rather than waiting for the older generation to retire. However, we must come up with better ideas to attract and keep the faculty involved in the course. Next time we will try giving preference to faculty who have won teaching awards, trying to let faculty know we missed them if they skip a lab, encouraging them to use computers outside of the lab times, and offering them slots in student labs to accommodate their busy schedules.

Labs Using Personal Computers

Every student attends two 2-hour labs each week, and each has his or her own personal computer for the full lab. I was surprised that many professionals thought this was not a good idea. They raised two objections:

- (1) You should use a drop-in schedule, not scheduled lab sessions.
- (2) You can't teach a large class on a flock of microcomputers.

The first suggestion allows students to spend as much time as they need to complete an assignment, and allows the hot shots to experiment as much as they want. (Most traditional introductory computer classes are run this way.) We use assigned labs because this allows the TAs to work with students as they are doing the labs, it encourages students to be prepared before doing the assignment, and it makes effective use of the computer facilities. It also forces us to keep the labs short so that students can finish in two hours.

The second objection was that you could never get a class doing the same thing when everyone had their own computer, and that distribution of the software and lab assignments would be awful. Taking last things first, the distribution problem forced us to be more prepared than necessary with a timeshared computer, where the instructor has the luxury of completing an assignment and making it available on the computer the week before it is due. Instead, the TAs created a lab manual—with all the assignments for the semester—that was available the first day of class. In the first week we distributed floppy disks containing all software needed for the labs. The one pain in the neck is that third party software such as Multiplan and MacPascal must be passed out and collected at the beginning and end of each lab. When a personal computer network is installed in our lab, I hope software distribution will be as easy as timesharing.

Macintosh versus IBM PC

Once you have decided to use personal computers, you have to decide which one to use. IDS 110 students rely on the Macintosh computer, using the IBM PC only 2 of the 15 weeks.

It is hard to believe now, but the MacIntosh computer had not been announced at the time I agreed to do this class. After I had used one, I had to use all my persuasive powers to get 40 Macs installed in the Tolman lab. Since the IBM PC dominated the marketplace and since students were much more likely to use a PC after this course, some thought it was silly to use Macintoshes. Fortunately for my reputation, the Macintosh was a big success. First, the select-from-a-menu user interface of the Mac is clearly better for beginners than the memorize-a-list-and-recall user interface of the PC. Second, the Mac encourages all applications programs to stick to the same style of user interface, so students don't have to remember different commands for similar events in different applications. Third, the MacPascal programming environment is the best programming environment I have ever seen on any computer. Finally the high quality graphics, variable font word processing, sound generation, mouse, and picture-input digitizers collectively make the Mac an exciting and inspirational machine. In my opinion it is the perfect place to start apprehensive beginners from the liberal arts.

If I feel so strongly about the Macintosh, why do we use the IBM PC at all? Prof. Richard White, who is the faculty member responsible for creating the Tolman Microcomputer Lab that we use for IDS 110, argued strongly that students must be exposed to more than one machine. Although I initially doubted him, after trying it in the class I am convinced he was right. Without some exposure to another machine, some students would conclude that the only computer they can use is the Mac. By using two computers, students gain confidence that they can now use any computer. I also suspect it helps to see more than one way to do things. A survey suggests that students liked our approach, as 91% were glad that they used the PC as well as the Mac (See Appendix II).

Emphasis on Direct Personal Impact of Computers

With the organization of the labs settled, I then turned my attention to the content of the labs. A criticism we heard of some traditional introductory computer classes was that students don't make the connection between what they learn in class and how they can apply what they learn. We were determined to make that connection. Thus our word processing lab required students to make their own résumé, our spread sheet lab had students predict their future income using 1980 census data, and our data base lab had students query a data base of Bay Area restaurants. I believe some of our programming labs were less successful in showing the direct impact of programming, and the TAs worked on that during the summer of 1985.

Students in this course really made the connection between computers and themselves. First, it was a complete surprise to me that so many students picked their own projects instead of a project suggested by the TA. Virtually all the best projects were the ones students created themselves. Second, in a survey taken at the end of class, 5% had already purchased a computer by the end of class, 13% planned to buy one after class, and 78% would like to buy one but

couldn't afford it (see Appendix II). This adds up to 96%, which is a good batting average considering that only one person had written programs before this class and that 80% initially indicated some form of computer anxiety.

Applications and then Programming

Given that we wanted to show the direct, positive, personal impact of computers on each student, it made sense to start with applications. Thus the first third of the labs teach applications of computers such as word processing, spread sheets, and data bases. We then introduce students to programming.

The original reason for showing applications was to convince students that computers were useful by showing them applications that they could use immediately. Since many of our students have poor math and science backgrounds, it was also important to allow them to gain confidence with computers before they tried to tackle programming. A secondary reason was to build up skills in dealing with a computer—using the mouse, keyboard, disks, and so on.

Serendipity then visited our course, for many of the concepts learned in the applications proved to be useful in introducing programming. For example, the distinction between name and value, a difficult concept for beginners, is clearer to students who have used spread sheets and seen both the name and the value of a cell in a spread sheet. The two dimensional nature of spread sheets is also useful in introducing arrays. The record data structure found in programming seems natural for students who have seen records in data bases. Even the skills and commands learned in word processing are reused when students create programs on the Mac.

Based on the results of our class, I believe showing applications before programming is definitely a good idea. We replace some of the time that would be spent discussing abstract concepts with hands-on use of applications that illustrate them. For students with poor math and science backgrounds, I think you must show applications to build confidence before they try programming. Most of these students wouldn't even enroll in a traditional introductory programming course.

I have assumed that it is not controversial to teach programming to liberal arts students. The argument against teaching programming is that few students in such a course will ever need to program. I am not sure I buy that, but in any case most computer scientists believe that you will never really master a computer if you have never tried programming one. Algorithmic thinking and the mental discipline of creating and debugging programs is the only way I can see to give real insight into the possibilities as well as the limitations of computers. How else can a novice know what to expect from future applications of computers?

Modify Programs versus Creating from Scratch

Given that you are going to teach programming, the question is where to start. Programming is traditionally taught by having students create a program completely on their own. Since many programmers start by modifying a program, I thought it would be better to give students a program and then have them modify it in the lab.

This idea had mixed results. It is a good way to get students started, giving them positive

experiences to help them get over anxieties about programming. The problem is that if they never write a program from the beginning, many will never fully understand how to program. Drawing an analogy to English composition, it is as if they can read and edit a paper but they can't write one. A final project that involves programming forces students to overcome this difficulty, but we have made other changes to the labs to move gracefully from modifying programs to writing them from scratch. Examples include the freedom of choice programming lab and the graded programming quiz.

I would conclude that if students' **only** experience is modifying programs, they will have great difficulty ever programming on their own and they may not understand key programming concepts.

Pascal versus Basic

Once I had an approach to teaching programming, I needed a language in which to teach. Outsiders may be surprised at how provocative this subject is for computer scientists, as it appears to generate as much adrenaline as discussions of evolution vs. creationism.

To illustrate this point, when I first told my Computer Science colleagues that I was thinking of creating and teaching an introductory computer class, I saw a look on their faces that said

"Poor Dave, he must have hit his head. Why would any sane person want to teach such a class?"

I could live with the reputation of being a slightly demented professor, but when I mentioned I was thinking of teaching programming in Basic, my colleagues all did their best steely-eyed Clint Eastwood impressions and offered opinions about my family ancestry. (Many computer scientists believe that learning Basic will lead to such bad habits that nothing could overcome such an initial disability.) I said that I would stick to my guns, but I would base my decision on which language had the best programming environment. I was afraid that an elegant language with a woeful programming interface and debugger would make it impossible for students to get very far in 5 weeks of 2-hour labs.

Once again, the Mac came to my rescue. Not only was MacPascal better than Microsoft Basic 1.0 on the Mac, it was the best programming environment for beginners I had ever seen on any computer of any price. MacPascal was a major success in this course. I had hoped that once someone learns Pascal they could pick up Basic very rapidly, and this has been our experience. We allocate only two 1-hour lectures to Basic and then students write two 2-hour labs in Basic. The surveys indicated that students were as confident of programming in Basic as they were in Pascal. Just as the case of seeing the PC as well as the Mac, I believe seeing two languages helps teach fundamentals and will enable students to learn other languages.

A Final Project

In the last three weeks of the class students are expected to complete a final project of their choice that applies computers to their majors. (See Chapter 4.) My original plan was that students would choose between one project available to all sections and another project suggested by each TA for their section. (I convinced one TA to build a checkers playing program for students to

improve as an alternative to projects suggested by the TA.) If students couldn't decide from those two, they would be allowed to pick a project of their own with the approval of their TA. I expected maybe one or two students in each section to pick their own projects, since they only had 3 weeks in which to complete it.

This was probably my biggest surprise of the course. Of the more than 100 student projects, only one group tried checkers. Furthermore, at least half of the projects were formulated by the students themselves, and these were the best projects: of the 14 exemplary projects selected to be shown in lecture in the final week, 12 were suggested by students.

Although my expectations were way off base, the final project was very successful and is crucial to the course. It allows students to pull together several aspects of the course, work hard and learn a great deal by working hard to solve a problem in their major using computers. I suspect this is the event that really gets them hooked on computers. It appears that students who select their own projects are the most highly motivated, and thus work harder and learn more. Now that I know what we can expect, I will ask the TAs to create several candidate projects for their sections and get students thinking about projects very early in the course.

I was frankly astounded by several of the projects that the students were able to create in just three weeks. This led me to a tentative hypothesis that a good way to learn computers is to first learn a subject area, and then learn about computers so as to solve problems in that area.

Chapter 3. The Course Organization

Labs

Lectures

Text Books

Teaching Programming

Grading

This chapter covers in exhaustive detail the content of the course. As a reminder, I am documenting all our ideas in a frank and informal manner so that others can learn from our successes and failures. I believe the most important aspect of a computer course is using computers, so I start this chapter with a discussion of the labs.

Labs

Our main goal was making the labs as strong a learning experience as possible. The problems we faced were:

- (1) making sure students do all the labs,
- (2) designing labs that are effective for all students, and
- (3) reducing anxiety in labs that have rigid time constraints.

Normally you use grades to ensure lab participation, but grading can increase anxiety. Our solution was to go to a "negative" grading system: for every missed lab, we lower the final grade by one third grade point. This addresses points (1) and (3), but we found that some students will complete a lab twice as fast as others. Our solution was to have a kernel in each lab that everyone should finish, and then have optional lab work for the faster students. We tell the faster students that the optional lab work will help when it comes time for the final project. The TAs just grade the labs pass or fail, but keep track of the students doing optional lab work in case these students are near the borderline on their final grade. By removing competition for grades from the lab, we also encourage students to help each other.

To maximize students' time with computers in the labs, we demonstrate new ideas and techniques in the lectures rather than at the beginning of every two-hour lab. Some students would still squander lab time by coming to labs unprepared, spending much of the first hour reading the labs. To discourage procrastination, every lab has a prelab and questionnaire that must be filled out before the student receives the application disks needed for the lab.

There is one TA in charge of each lab, but there is a second TA to help. The reason for this organization was to help transfer information and good ideas between TAs, and also so that one TA could cover for another in case of an illness. We later found that it was useful to have two TAs familiar with each student when determining final grades.

Several of these ideas worked well. Negative lab grading worked like a charm: attendance remained high, and if students missed a lab, they almost always made it up. And the prelab questionnaire does reduce procrastination.

A few ideas needed more work. Having the lab demos in the lectures was a good idea, but I

think the execution of the idea was poor. My demos covered the features I liked instead of the features students needed for the labs and the problems they were likely to encounter in the lab. This year we used TAs familiar with the type of difficulties students have in the labs to give the demos, and I think this worked much better.

The primary problems were the running of the labs and the creation of the lab material. Two TAs are not enough if 40 students believe they need to talk to a TA every time something unexpected happens. We used the following techniques to reduce pressure on TAs in the labs:

- Students who asked questions that were answered in the lab manual were encouraged to read the manual;
- We formed student teams that would sit together every lab so that they would help each other when they had problems;
- We announced in class that computing is a gregarious and cooperative enterprise, and that helping your neighbor is part of the culture.

Nevertheless we still had troubles. Our solution this semester was to borrow an idea from the introductory Computer Science courses and enlist student volunteers to help with the labs. These students have taken IDS 110 and receive pass/fail course credit for helping with the labs. (See Chapter 5 on selecting student aides.) I think everyone—students, TAs, faculty, and student aides—are happy with this arrangement.

Interestingly, we also had the problem of giving too much help to students in the labs. The danger in answering a question is that you may solve the student's problem rather than helping a student discover his or her own solution. In the graded programming lab it appeared some good students did poorly because TA help was part of their debugging process. We now warn TAs and student aides of the danger of being too helpful.

The final problem of the labs is the surprising difficulty of making a good lab: it must teach important ideas, fit within the 2-hour limit, have interesting optional work to keep the fast students interested, and must be clearly written. Our process was to try out labs on the TAs and faculty, then a new version on a small group of students, and then a final version on a large group of students. I think it is virtually impossible to make a new lab without going through at least three versions. Let me redundantly reemphasize this point: no matter how much time you spend on the original lab, it is useless until several people have tried it and given you feedback, and the revision will be a problem unless you go through one more iteration. You may even have to throw the lab away and start over. As an aside, this has been the experience with all computer software, and it is remarkable how difficult it is to make something useful. The software industry has formalized this process, and the reliability and role of "alpha" and "beta" versions of software are well understood.

Lectures

In this course the lectures are not as obvious as in traditional courses, in part because there is no textbook that you can follow. As I said above, I believe it is the use of computers that really teaches the ideas, so the lab material determines the contents of the lectures. The place to start is the

lab manual, and you start by making a list of the ideas that must be learned for each lab and when students will do the lab. To simplify lecture preparation, I have prepared my lecture notes on the Mac so that others can use those notes. Ideally each lecturer will improve those notes and they will be passed back and forth until the notes are in very good shape.

Once you make this list of ideas, you then schedule the lectures. Follow Figure 4 in forming the schedule, and Figures 5 and 6 show the schedule for Spring 1985. I found it useful to classify lectures in four categories:

1. *Traditional Lecture*
2. *Demonstrations*
3. *Exams*
4. *Guest lectures*

Schedule the lectures in this order:

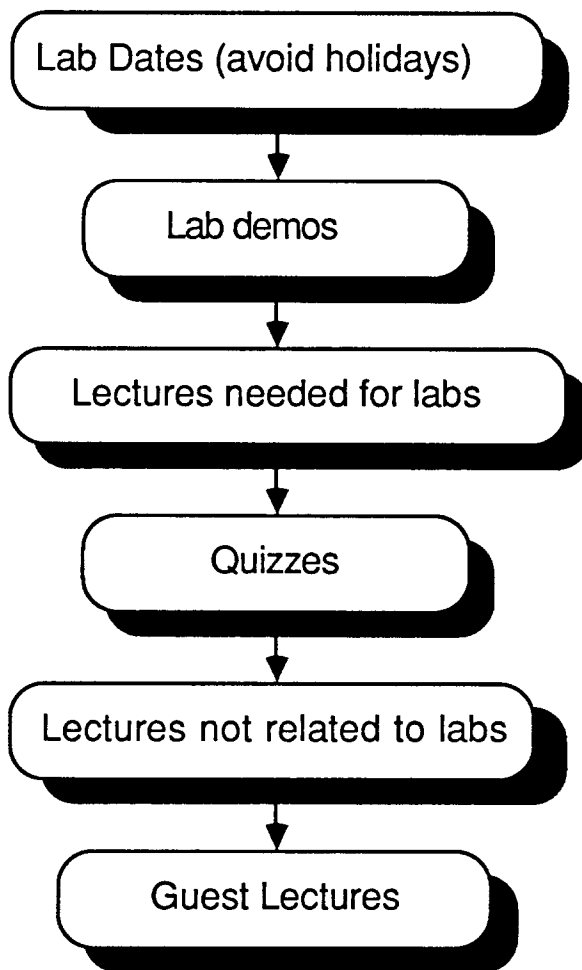


Figure 4. *Recommended order of scheduling lectures.*

Lecturers should check their travel plans to see which dates must be covered by others, and then follow this flowchart until the schedule is complete.

<i>Date</i>	<i>Day</i>	<i>Lecture Title (3 per week)</i>	<i>Week</i>	<i>Lab Title (2 per week)</i>
1/23	W	Course Administration	1	Meet your TA/enroll in course
1/25	F	Computers are Simple		
1/28	M	Desk-top Demo	2	Getting Your Feet Wet(MacPaint)
1/30	W	Word Processing		Desk-Top Management
2/1	F	What is a K?		
2/4	M	FileSystem,Database&SprSht	3	Revising Documents(Word Processing)
2/6	W	Data Bases/Demo		Résumé (Word Processing)
2/8	F	Spreadsheet Demo		
2/11	M	User Interface	4	Earnings I (Spread Sheets)
2/13	W	What's in a name?		Earnings II (Spread Sheets)
2/15	F	QUIZ #1		
2/18	M	HOLIDAY(Presidents' Day)	5	"Free" Lab (Explore,Makeup) Sec. 2,4,5,7,8,10
2/20	W	Why Program? (guests)		"Free" Lab (Explore,Makeup Missed Labs) S.1-10
2/22	F	Prog. on blackboards #A		
2/25	M	Prog. on blackboards #B	6	Eating Out (Data Bases on IBM PC)
2/27	W	Prog. on blackboards #C		Food and Drugs (Data Bases on IBM PC)
3/1	F	Prog. on blackboards #D		
3/4	M	Prog. on computer Demo	7	Filevision (Data Bases on MacIntosh)
3/6	W	QUIZ #2		Bull's-eye (Programming#A)
3/8	F	Pascal		
3/11	M	Strings/Arrays	8	Flower (Programming#B)
3/13	W	Programming Style		MouseSketch (Programming#C)
3/15	F	Pascal Syntax		
3/18	M	Guest(Atkinson on MacPaint)	9	MacMusic (Programming#D)
3/20	W	Parameters/Arguments		Freedom of Choice (Programming#E)
3/22	F	Guest(Engelbart on Mouse)		
3/25	M	Tour of Checkers	10	MacTrivia (Programming#F)
3/27	W	Intro to Basic		Checkers (Programming#G)
3/29	F	Basic Demo		
4/1	M	Networks/Timesharing	11	Basic Basic (Basic Programming#A)
4/3	W	Meet TA in Lab Section		Doctor(Basic Programming#B)
4/5	F	Review of Programming		
4/8	M	Guest(Parker on Computer Crime)	12	Graded Programming Lab
4/10	W	Guest(Ousterhout on C.S. Sermons)		Project Lab#A Select Project/Partner
4/12	F	Solutions to Graded Lab		
4/15	M	Review of last projects	13	Project Lab #B M/C homework due
4/17	W	Computer Graphic Movies		"Free" Lab (Explore,Makeup) Sec. 1,6
4/19	F	HOLIDAY(Spring Recess)		
4/22	M	TBD	14	Project Lab #C:
4/24	W	Guest(Stefik on Expert Systems)		Project Lab #D
4/26	F	TBD		
4/29	M	Software Maintenance	15	Project Lab #E
5/1	W	Guest(Alan Kay on the Future)		Computer Faire: Th 5/2 8am-6pm
5/3	F	TA's Choice (presentations of interesting projects as selected by the TA jury)		
5/6	M	Course Review	16	(no lab)
5/10	F	Final Exam: 8-11am		

Figure 5. Early Lecture Schedule with Labs (Spring 1985).

<i>Date</i>	<i>Day</i>	<i>Lecture Title</i>	<i>Reading Per Lecture</i> (<i>JC=Joy of Computers, MP=MacPascal</i>) (<i>LM=IDS 110 Lab Manual, MMM=Mythical Man Month</i>)
1/23	W	Course Administration	Start "Soul of a New Machine"
1/25	F	Computers are Simple	JC: 6-7,14-15,18-19,168-170
1/28	M	Desk-top Demo	LM: Getting You Feet Wet,Desk-Top Management
1/30	W	Word Processing	JC:98-99; LM: Revising Documents:Finish "Soul"
2/1	F	What is a K?	JC:12-13; Start "Fire in the Valley"
2/4	M	FileSys,DB&SprSheet	JC: 94-98, 100, 158
2/6	W	Data Bases/Demo	LM: Eating Out
2/8	F	Speadsheet Demo	LM: Earnings I: Finish "Fire in the Valley"
2/11	M	User Interface	JC: 44-47
2/13	W	What's in a name?	
2/15	F	QUIZ #1	
2/18	M	HOLIDAY (Presidents' Day)	
2/20	W	Guests:Why Program?	JC: 54-55,MMM(Mythical Man Month): 7-9
2/22	F	Prog. on blackboards #A	JC: 56-57
2/25	M	Prog. on blackboards #B	
2/27	W	Prog. on blackboards #C	
3/1	F	Prog. on blackboards #D	
3/4	M	Prog. on computer Demo	LM: Bull's-eye MP(Mac Pascal): Ch. 1
3/6	W	QUIZ #2	
3/8	F	Pascal Loops/Arrays	MP: 60-72, 289-302
3/11	M	Strings/Read/Write/Case	MP:308-18/91-92,241-42/97-98/232-34
3/13	W	Pascal Syntax	MP: 79-84,86 LM:Summary of Pascal Commands
3/15	F	Programming Style	JC: 77-79; MP 207-214; MMM 141-150
3/18	M	Guest:Atkinson on MacWrite (Tape)	
3/20	W	Parameters/Arguments/Functions	MP: 191-206,333-343
3/22	F	Guest:Engelbart on Mouse	
3/25	M	Tour of Checkers	LM: Checkers
3/27	W	Intro to Basic	JC: 58-61,186-187; LM: Basic Appendix
3/29	F	Basic Demo	LM: Basic Basic, Doctor
4/1	M	Project Preview (Demos)	
4/3	W	Meet TA in Lab Section	
4/5	F	Review of Programming	
4/8	M	Networks/Timesharing	JC: 152-157, 178-179 (Graded Prog. Lab)
4/10	W	Guest:Parker on Computer Crime	JC: 159 (Select Project & Partner)
4/12	F	Solutions to Graded Lab	
4/15	M	Guest:Ousterhout on C.S. Sermons	(Multiple Choice Homework due in labs)
4/17	W	Guest:Computer Graphics (Tape)	JC: 32-33, 110-115
4/19	F	HOLIDAY (Spring Recess)	
4/22	M	Review of last projects	MMM(Mythical Man Month): 164-175
4/24	W	Guest:Stefik on Expert Systems	JC: 86
4/26	F	Optional Lecture: Records,types	MP: Ch. 11(Optional: not on final exam)
4/29	M	Software Maintenance	MMM 13-26, 53-58, 153-160
5/1	W	Guest:Alan Kay on The Future	
5/2	Th	8am-6pm Computer Faire	(Final project demos in lab;turn in reports)
5/3	F	3-5pm: TA's Choice (presentations of most interesting projects as selected by TAs)	
5/6	M	Course Review	
5/10	F	Final Exam: 8-11am	Covers Everything(Including guest lectures)

Figure 6. *Final lecture schedule with reading assignments .*

Traditional Lectures. Almost all college material is presented in this format, and unfortunately it is not an effective format. It has been shown that the first 20 minutes is the most important, so I start lecturing at the bell and then stop after 20 minutes to make announcements: reminder of the next exam, reading assignments, lab announcements, and so on. This breaks the monotony of lecture format, and student surveys suggest that most students prefer this format.

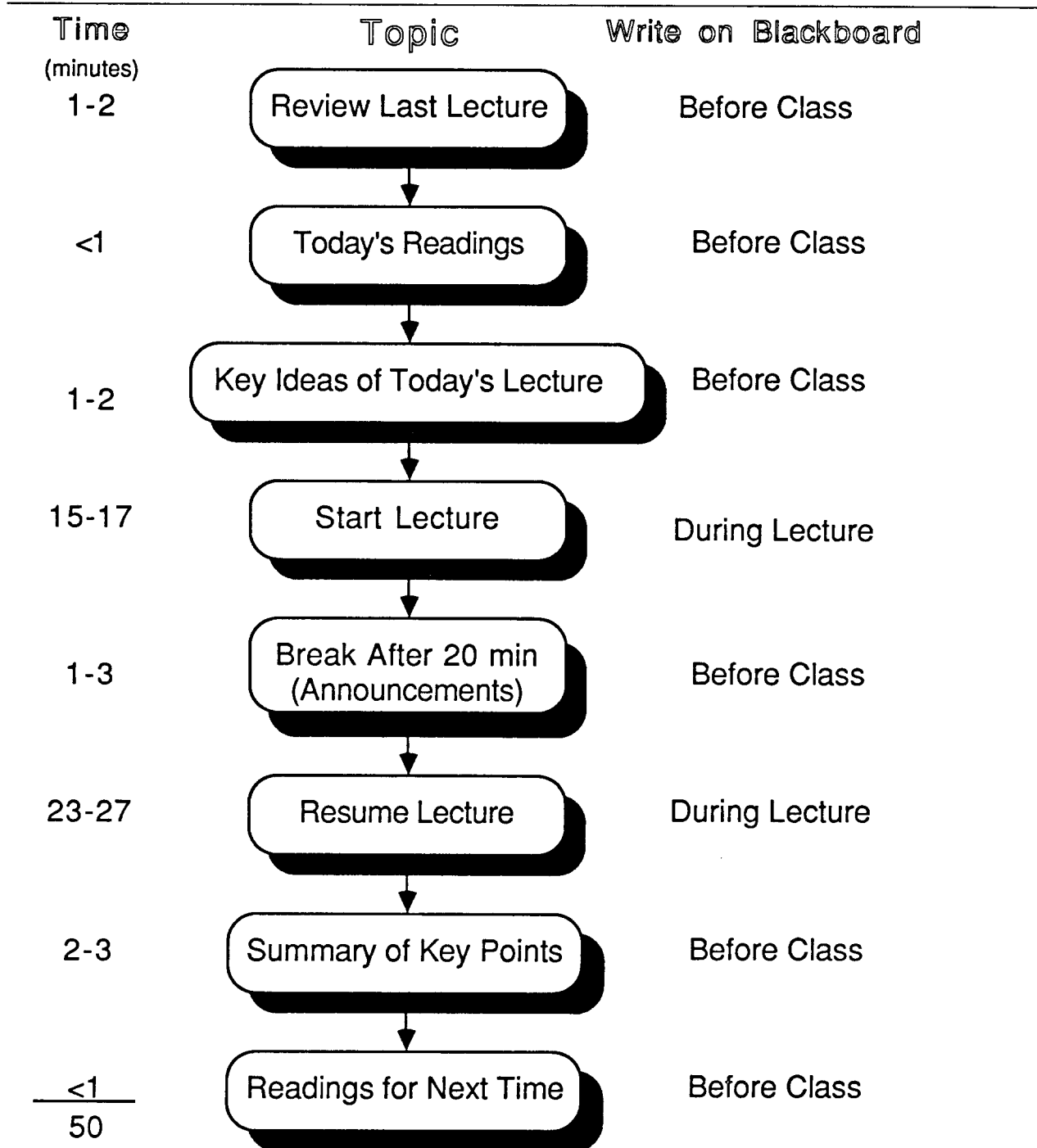


Figure 7. Lecture organization flowchart.

As part of each lecture I try to review the last lecture, mention the important ideas that will be covered in this lecture, and then quickly summarize the key points at the end of the lecture. (See Figure 7.)

I have a few pointers to pass along when lecturing in Physical Sciences Lecture Hall Room 1 (PSL 1), the home of IDS 110. The room is huge, so you first have to learn writing on the blackboard in large letters using extra thick chalk. There are eight blackboards in the room, so there is plenty of space to wander about the stage. Since a projectionist is recording your lectures and projecting it on a large screen, if you wander it lessens the value of the projection system. I try to stay in one place, writing on both the lower and upper boards before moving positions. I meet with the projectionist before the lecture to let the person know what I plan to use in the lecture—overheads, the Macintosh, blackboards—to smooth the delivery of the lecture.

It takes a long time to write information on the board in large letters and thick chalk, so I try to arrive early to get as much information on the board as possible. (See Figure 7.) PSL 1 has a rotating stage with three sets of blackboards, so you can have the projectionist let you in backstage as early as you like. I found that writing programs on the blackboard takes forever, especially when you want to show several versions of the program. I now use overhead transparencies for programs, and using a smaller font, make handouts so that students can concentrate on the ideas rather than transcribing the blackboard.

Demonstrations. As I mentioned in the previous section, to save time in the labs we give demos once to all the students in the lecture rather than several times at the beginning of each lab. The only other comment is, while preparation is needed to give a good demo, you should not give a perfect demo. We need to instill the idea that you don't always know what to do with a computer, and when you don't, you try things to see what happens. Planning to make a few small mistakes and then showing how to recover from them should be part of every demo.

Exams. Every course has exams, but we are trying a few unusual ideas. In interviewing TAs I found that while many enjoyed teaching, almost no one liked grading. We decided to reduce the burden of grading hundreds of exams by using multiple choice questions that are automatically graded by a Scantron system. We later added a wrinkle to reduce anxiety about multiple choice questions. If students think there is no proper answer to the question, they are allowed to write their own answers for up to two questions on the back of the Scantron form.

Like preparing the lab material, it takes a great deal of work to prepare good multiple choice questions. We assign a committee of TAs to prepare the exam, and then everyone takes the exam in time to make revisions. We also try the exam out on a few students who need to take the exam early and incorporate their feedback. By the way, TAs who are graduate students in education tell us that multiple choice exams are just as good a test as any other form of exam, so you don't have to feel like you are cheating by reducing the pain of grading.

Speaking of cheating, you have to be prepared when you have hundreds of students that there will be a few bad apples. We make two versions of each multiple choice exam by exchanging

the order of the questions and then mix the two versions so that every other student gets a different exam. Also we get all the TAs to come to the exam so that we have many sets of eyes to discourage copying. Finally, I recently realized that PSL 1 has a blue dot on the right corner of every other chair, so if you make sure students only sit in chairs with blue dots you can further reduce the opportunity of cheating.

We also tried one more novel idea, and it had mixed results. As a student I would get some ideas wrong on an exam and sometimes I would then "learn" the wrong concept because I wouldn't find out I had made a mistake until the exam was returned. We give 30 minute exams and then spend the last 15 minutes going over the answers. The good news is that it is a chance to correct misconceptions by giving a 45 second lecture about difficult subjects. But pity the poor lecturer who must explain the answer to a poorly worded question to hundreds of agitated students. I now know what it is like to be an umpire, for only approval is silent.

Guest Lectures. I also reserve some lectures for guest speakers, primarily to show students what is happening in computing in the local area. It also serves as a way to keep the course interesting to the lecturer and TAs, since they can use this opportunity to meet new people and learn new ideas. Finally, they preserve space for new material in later versions of the course. My only suggestion is that the lecturer select the guest speakers early in the semester and ask them to send papers to be given out in the class either before or after the guest lecture, for some students will be very interested. Appendix III contains two letters I sent to recruit a guest lecturer.

Text Books

I suspect no lecturer is ever happy with a textbook unless he or she is the author, and I am no exception. As I said earlier, I believe the labs are where much of the education occurs, so the key is having a good lab manual. Since our TAs wrote the lab manual, we have at least one book we are happy with. We have realized that we must revise the manual every summer to take advantage of new software and to continue our quest for perfect labs.

Actually, we are happy with two of the text books. *Macintosh Pascal* by Robert Moll and Rachel Folsom (Houghton Mifflin Company, 1985) is an excellent introductory Pascal programming text for the MacPascal system. We feel no need to make further changes.

After those two we have problems. We tried *Joy of Computers* by Peter Laurie (Little, Brown and Company, 1984), which has some excellent short and accurate descriptions of computer technology, but it is not really a textbook and it does use a few examples that women might find offensive. This year we are trying *The Computer Annual* by Robert Blissmer (John Wiley and Sons, 1985), but it does not include anything on programming. I understand that there is a programming supplement for this text, and it may help.

My summary is that we are happy with (our) lab manual and the Pascal book, but we would like to find a better general text.

Teaching Programming

In Chapter 2 we said that we cover applications before programming and that this pays off in that several ideas needed for programming are introduced as part of the applications. Our next step is to introduce programming on the blackboard. Our idea is that it is better to introduce the concepts of assignments, variables, loops, and procedures without being concerned about syntax errors or the problems of debugging programs. To encourage students to learn programming before they face a computer, we use two techniques:

- (1) We have two home work assignments on programming that we go over in class. I found that if you don't collect the assignments, many students won't do the work.
- (2) We also have a quiz on programming concepts before students start programming on computers.

Grading

As I mentioned earlier, in discussions with TAs, the least popular part of the job is assigning grades. We have simplified the grading by making all tests multiple choice except the programming quiz. The lab assignments are graded pass/fail. In recognition of the hard work, we give mostly A's and B's on the project. The only difficult grade is to come up with the final course grade.

How do you give final grades in a course like this? There is no campus-wide policy on grading, so I adopted the policy of our department, which is that the average class grade point average of an upper division course should normally be between 2.7 and 3.1 (4 is an A, 3 being a B, and so on).

First the TAs and the instructor meet after the final exam is graded to assign the course grades. We use the computer to accumulate the total points for each student, and then plot the scores from highest to lowest. We use Multiplan to divide the total score into three point categories, then count the number of students in each category, and use Chart to plot the grades. Then we make the initial association of grades and scores. We found that just dividing between A's, B's, and so on was good enough to keep class grade point average within bounds.

The next step is for the TAs to go off in pairs to assign grades for each student. TAs consider questions such as:

Did the student do optional lab work?

Did the exam scores improve over the semester?

Did it appear that the student didn't do much of the work on the project?

Did the student miss any labs? (If so, subtract 1/3 grade for each missed lab.)

The TAs may want to discuss with the instructor when assigning grades, but they know the students so much better that I give TAs a great deal of leeway. The main role of the instructor is to set policy. After the TAs adjust the grades, including the assignment of pluses and minuses (e.g., B+, A-), we record the grades on the final forms. By the way, graduate students must receive a B- or better to get a Satisfactory if they take the course S/U, and undergraduate students must get a C- or better if they take the course P/NP.

We have several tasks to perform to record the grades, so we have four TAs act as scribes, and when a student name is read by one of the scribes the remaining TAs will find the name on their lists and tell the grade . The roles of the four scribes are:

1. *Record the grades on the IDS 110 report form.*

Be sure to record the grade in the comment section if someone is taking the course S/U or P/NP in case there is some change.

2. *Record the grades on the IDS 110L report form.*

IDS 110 and IDS 110L are separate courses at Berkeley.

3. *Record the grades on the postcards that students submit.*

4. *Count the number of grades in each category .*

Calculate the class grade point average.

The only other advice is to record the names of any unusual cases and give them to the instructor and the IDS secretary. Examples of unusual cases are students whose names are not on the grading form but have taken the class, students whose names are on one report but not the other, and students whose names are on the class list but we do not have them in our files. In addition you should write these special cases at the end of the grade report form. These cases need to be resolved sometime, so it is a good idea to write down all the details while you know them.

The final step is to ask the IDS secretary to make copies of the reports to be posted outside of 230 Bechtel. It is courteous to remove the names from the reports, just leaving the student ID numbers. Since it is hard to find a match with a list of hundreds of numbers, we ask the secretary to identify the first letter of the last names.

I then recommend going out and celebrating the end of the course!

Chapter 4. The Project

Picking a Project
Picking a Team
Computer Faire
TA's Choice

The project is probably the most important part of the course. Students work in small groups during the last three weeks of the semester to use computers to solve an interesting problem in their field. Much of the learning in the course occurs here, for students practice what they have seen with little handholding by the TAs. If the projects are successful, students really see that computers are useful and that they can master them. I think even if the project is not so successful, students still learn a great deal, if only the importance of keeping a project small.

Picking a Project

As I mentioned in Chapter 2, my biggest surprise was the number of students who picked their own projects. We found that if students pick their own project they will work very hard, and the more they work with computers the more they learn. The choice of project is up to the ingenuity of the students under the guidance of the TAs. My only advice is that the project be "real stuff"--that is, if the project is successful some people not related with the course will be interested in seeing or using it, ideally a professor in some discipline. It is easy to pick projects that no one cares about, but if you are clever you can find a project that is interesting, educational, and "real". For example, a text editor is educational but it is not "real stuff" since no one in this course can build a better text editor in three weeks. Besides, a text editor is hardly a solution to some unsolved problem in their major. Figure 8 shows some of the interesting projects from the Spring 1985 semester.

I was pleasantly surprised at how many students picked programming projects last Spring. We found that students who did such projects did well on the final. This semester we are asking that there be some programming as part of every project, in an attempt to get more students to practice programming. Our only other restriction from last time is that we prohibit students from programming computer games, unless the game teaches an important idea in their field. A wonderful example of such a game is "Yellow Fever," a game that teaches the player about the importance of herd immunity: you can defeat a disease if you vaccinate 80% of the population. "Yellow Fever" is fun to watch and to play, and it teaches an important concept in Public Health.

Picking a Team

The original reason that the projects are done in teams is that it is a good experience for students to work in groups, as that rarely happens in academia and I believe it is better to learn about group dynamics by making mistakes at school rather than after you start a career. Team projects also allow people to tackle larger problems. Project teams of two or three students are ideal, and we require justification for a larger team. In addition to limiting the size, an important

-
- "Gel Analysis of DNA Sequences"
by Kevin Nash & Mehdi Ganjeizadeh (Genetics) [MacPascal]
- "Chinese Typewriter"
by David Kelly (Oriental Language) [MacPascal]
- "Yellow Fever: Immunization Game"
by Susan Cummins, Tom Smith, & Jack Yang
(Public Health) [MacPascal]
- "Piano Playing/Recording Scores"
by Yvonne Brovard, Tom Deering, & Malinda Lai
(Music/Public Health) [MacPascal]
- "Mac The Pirate: Board Game"
by Maureen Kamiya & Jennifer Sotto (Math/Psych.) [MacPascal]
- "Medical Statistical Paradox"
by Nancy Coe (Psychology) [MacPascal]
- "Eastern European Jewry: Turn of the Century Immigration"
by Mark Dollinger & Jocelyn Berner
(History) [MacPascal, Multiplan, Filevision]
- "Analysis of 'Who Voted For Hitler?' "
by Noel Cary (TA) et al (History) [Multiplan]
- "Archaeology Dig Map/Database"
by Richard Lindstrom (Anthropology) [Filevision]
- "Invisible Knee/Sports Injuries"
by Donna Burden, Caryn Salisbury, Susanne Marx, & Lora Wong
(Physical Education) [Filevision, MacPascal]

Figure 8. *Sample Projects from Spring 1985.*

These projects are the result of 3 weeks of programming by students from traditionally "computerphobic" majors who had no experience with computers 15 weeks before. The students majors are in parentheses and the applications are in square brackets.

step in making a successful team is selecting the right people. I tell students to ask the following questions when forming a team with common goals and interests:

(a) Are you all taking the course for a grade or for P/INP?

Students taking the course pass / no pass may not have the same commitment to a project as someone taking it for a grade.

(b) Do you have compatible schedules?

(c) Are you all committed to working at the same intensity?

(d) Do you get along?

A few labs are done in pairs. Check for personality conflicts in these labs.

(e) Can you agree on a project?

My only other general advice is to try to keep stronger students and slower students on different teams. In unbalanced teams the stronger students will feel like they are doing all the work and the weaker students will feel the others are hogging the glory and not giving the weaker students a chance.

The TAs found a very important reason for using teams. It frees up half the Macintoshes, allowing students from other sections to drop in and use the idle computers. This extra time allows the dedicated students to get more access to computers and thus learn more.

Computer Faire

On a specified date at the end of the three weeks assigned to the project, students demonstrate their project to the TAs and the instructor. We call it a "Computer Faire," inviting students to visit the other projects to see what everyone has done. Because students work very hard it is important that the instructor and the TAs give everyone a pat on the back. As we mentioned in Chapter 3, we tend to give high grades to projects of students who try hard.

TA's Choice

Since not every student project is successful and students may not see projects done in other sections, we reserve one lecture to present some of the best projects. I think this presentation is crucial so that all students can share in the successful applications of computers to the liberal arts, and they can see that computers can be useful, no matter how well their project turned out.

We call that lecture "TA's choice" because the TA's and the instructors see the projects at the faire and pick representative projects for presentation the next day. We don't say they are the best projects, just that they are representative. Thus the TAs and instructor should collect the phone numbers of the the students whose projects look like they may be chosen for presentation so that the students can be informed later that day.

We learned a bit from the presentation last time that I would like to pass along. First, early in the semester you need to reserve the room and the projectionist for more than the normal class time. We tried 2 hours in the spring but I felt that was too long, and I recommend you only go for 1.5 hours. It is important for the audience to be able to ask questions, so I tell students they have 8

minutes and I give them 10. I kept a wireless microphone so I could keep the presentations moving and gave the other microphone to the students.

We had all presenters and TAs meet an hour before the normal class time so that we had everything ready when class started: the speakers, the Macs, and the software. We had two Macs set up so that we could switch immediately rather than pause while the next is being set up. The basic order should be least to most impressive, but I would start with one excellent project, two good but drier projects, and then work your way up from there. I recommend you have a TA write the project name, students names, and presentation order on the blackboard so that everyone knows what is happening. By the way, these students are likely candidates for student aides, and I would try to recruit them afterwards.

Finally, a new course needs to become known on the campus, so I recommend inviting everyone you can, including the campus press to publicize the class. Figure 9 is an article that appeared on the front page of the Daily Californian after the presentation.

Computer course helps explain Bach, genetics and knee pain Course

By MARGARET SUH

For many of the 250 people in Interdepartmental Studies 110, "Introduction to Computers," the initial thought of computer programming aroused fear and loathing.

In a survey soon after the course began, more than 80 percent expressed "extreme anxiety" about using the machines.

Fifteen weeks later, the students, most of whom had no computer background before taking the class, were writing elaborate programs to analyze DNA, teach basic immunization, diagnose knee injuries, and play music. These and other programs were displayed at a "computer fair" held Friday afternoon in the Physical Sciences Laboratory lecture hall.

Kevin Nash and Mehdi Ganjeizathe, two genetics students, created a program that analyzes DNA se-

quences, giving the correct nucleotide and amino acid sequences in a piece of DNA, the molecule that holds the genes in all living creatures.

The program can analyze in three to four minutes what would usually take three to four hours, the students said.

A program developed by physical education students Donna Burden, Suzanne Marx, Karen Salsburg and Laura Wong gave brief descriptions of common knee ailments, complete with detailed graphics of the knee.

Public Health students Susan Cummin and Jack Yang and Latin American studies major Tom Smith devised the "Yellow Fever Immunization Game." The purpose is to "immunize," using the computer, 80 percent of the computer-simulated "population" before it succumbs to yellow fever.

"This is a simple and fun way to

SEE PAGE 2

FROM FRONT PAGE

teach immunity concepts in Third World countries," the students said.

The Apple MacIntosh, the computer used in IDS-110, played Bach, as well as simpler tunes, in a program created by public health student Yvonne Brovard, music major Tom Deering and psychology student Malinda Lai.

The course, taught by computer science Professor David Patterson, was targeted at "computer-leery" students with little technical background.

Patterson said he wanted to show that computers were "simple and general-purpose tools . . . which are interesting and fun."

In three weekly lectures, Patterson addressed questions like "What is a K?" and "Why program?" Guest lecturers also covered computer crime and computer-generated movies.

Students had four laboratory hours per week. During this time, they wrote their own resumes, did their own lifetime earning projections, and worked on other projects.

Most laboratory assignments were geared to take no more than the four hours. "You didn't spend nights hacking away," said molecular biology senior Debbie Jan.

Juniors, seniors, and graduate students are eligible to take IDS-110. Although ACE scheduling is preferred, Patterson said all interested students should show up for the first class meeting next fall.

Figure 9. May 6, 1985 Daily Californian article on "TA's Choice" lecture.

Chapter 5. The People Involved in the Course

Part I. Responsibilities

Part II. Selection of the People

Part III. Esprit de Corps

PART I. RESPONSIBILITIES

The first part of this chapter describes the responsibilities of everyone involved with this course. By reading the appropriate section, everyone will know what they must do to make the course successful. The second section gives advice on selecting new people for the course.

Responsibilities of Lecturer

The lecturer's first job must be completed long before the first day of class. As mentioned in Chapter 3 and shown in Figure 5, the lecturer must make a class schedule that:

- (1) assigns lectures and demos associated with a lab before that lab;
- (2) identifies the dates that the lecturer must miss because he or she is out of town;
- (3) fills in schedule with the more flexible lectures, demos, exams, and guest lecturers.

Since the demos are given by the TAs, the lecturer must coordinate with the head TA to schedule the demos and guest lectures given during travel periods.

Once the schedule is set, the next step is to invite the guest lecturers. We have video taped several guest lectures, so you might start with those tapes. Alternatively, you could also save them as backup in case something happens to a guest. I found guest lectures to be an enjoyable part of the course, and I encourage you to invite guests you want to hear, keeping the course fun for you as well as interesting to beginning students. If you need some suggestions, ask the TAs what they are interested in. I recommend trying to find ties with your campus to help students appreciate the local impact of computers.

The rest of the responsibilities are more straightforward: giving lectures, helping create and debug the exams, attending the weekly TA meetings, and preparing and duplicating handouts. The lecturer may also help with the selection of a new round of TAs.

Responsibilities of Lab Instructor

It was Dean Alan Portis's idea to separate the lectures (IDS 110) from the labs (IDS 110L). Without someone dedicated to modernizing the lab, he feared the lab would soon become obsolete. In addition to seeing that the software and hardware track technology changes, the lab instructor must help smooth the interface between the TAs and the staff in charge of maintaining the lab. It is important to attend the weekly TA meetings to be sure to solve problems as they occur. The lab instructor may also help with the selection of new TAs.

Responsibilities of the Head TA

The head TA may be the most important person, since he or she is the only person involved with the course over several semesters. I think of the head TA as the manager. His or her primary

responsibility is to make the course successful. The head TA must recognize potential disasters before they happen, playing an activist role in the course.

The lab manual is also the responsibility of the head TA. The head TA must keep track of errors and suggestions for the manual during the year, and normally is employed during the summer to correct and upgrade the manual. The head TA is also given one section to lead, normally the faculty section. (To reduce the load a bit the Head TA does not help run a second section.) In addition the head TA runs the weekly TA meeting, setting the agenda and suggesting topics that need to be covered. The head TA is also the person who meets with visitors who would like to find out more about the course.

Finally, as well as being involved in the selection of TAs, the head TA must groom a successor from among the current TAs.

Responsibilities of TAs

The TAs determine the success of the labs, the most important part of the course. This means the TAs must be very familiar with the labs before students try them. We ask new TAs to spend time just before the Fall semester to familiarize themselves with the labs, and to help uncover mistakes in the labs. Like TAs in other courses, the cardinal sins for TAs are missing a lab, missing an office hour, or missing a weekly TA meeting. If it looks like such a disaster might happen, the TA must find someone to cover for him or her and contact the head TA so he or she is informed.

The primary work of running labs is answering student questions. An additional responsibility is distributing and collecting the proprietary software. The TA must also grade the labs, record those grades, and return graded labs to the students. The one other task, which should be a lot of fun, is preparing sample projects for students in the same major as the TA. (We have several examples of projects leading to research results of interest to the TA.) We want to get as many students as we can to select their own projects, but we need very good examples to inspire them.

The course is run cooperatively, so there are some tasks that the TAs must all share so that no one gets stuck with all the work. These include:

- (1) making exams,
- (2) pretesting exams,
- (3) grading exams,
- (4) making floppies for labs,
- (5) giving demos in class, and
- (6) running review sessions.

We also expect each TA to recruit students to act as students aides. Figure 10 shows the work that must be done for the course over the year.

Responsibilities of Student Aides

The primary responsibility of the student aide is to help the TAs run the lab. Our style is to help students discover the joys of computing for themselves rather than to just give the answer as soon as someone is stuck. The fun part is helping students learn what the aides have learned, and I expect some aides will discover they enjoy teaching.

<i>Summer</i>	<ul style="list-style-type: none"> Revise Lab Manual Write TA Handbook Revise exam and quiz questions Update reader Write student information sheet (grading policy, books, lab sections etc.) Check new software Write to faculty for faculty section Start contacting guest speakers
<i>August</i> (before) (school) (starts)	<ul style="list-style-type: none"> Meet with TAs Organize lab sections, student aides, TA handbooks Make list of TAs to attend lectures Post TA office hours in Wheeler Send semester software requests to Sarah in Tolman Call George Huega in Educational TV Office to videotape the lectures using VHS and bill IDS at the end of the semester Book PSL for Computer Presentation Put books and reader on reserve in libraries (Moffitt, Education-Psychology and according to the majors).
<i>August</i> (after) (school) (starts)	<ul style="list-style-type: none"> Post student lists by section Get quiz 1 ready, TA help for day of quiz and grading quiz
<i>September</i>	<ul style="list-style-type: none"> Get Quiz 2 ready
<i>October</i>	<ul style="list-style-type: none"> Publicity for Spring enrollment (Oct. 21 - Nov. 8) ASUC bookstore - requests for Spring TAs help on day of quiz2 and grade quiz2 via Scantron
<i>November</i>	<ul style="list-style-type: none"> Get final exam ready
<i>December</i>	<ul style="list-style-type: none"> Call Daily Cal about Computer Faire

Figure 10. *Academic Year schedule of events to prepare IDS 110.
(The Spring is the same schedule as August through December)*

PART II. SELECTION OF THE PEOPLE

This next section is my advice on how to get the best people, surely the most important part in making a successful course. Figure 11 shows the one piece of advice I would keep in mind when selecting anyone for the course.

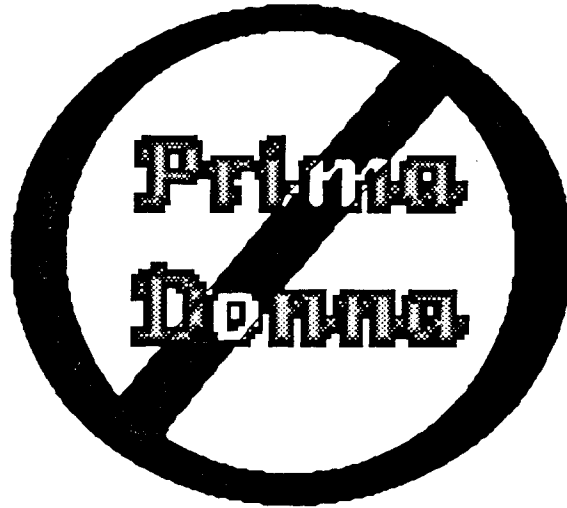


Figure 11. *Prima Donna* sign at entrance of IDS 110.

They may be useful in operas, but Prima Donnas have no role in an introductory computer course.

Selecting Faculty

As mentioned above, the faculty burden of IDS 110 is shared between the lecturer and the lab instructor. I think the lab instructor needs to be someone like Prof. Richard White, a person who sincerely believes in the educational potential of microcomputers and is willing to spend the time and energy to bend bureaucracies to his will. I think this person must be willing to take on this responsibility for several years.

The lecturer, on the other hand, need be willing to take on the course for only one semester. The ideal lecturer will have a contagious enthusiasm about computing as well as a deep understanding so that he or she will be able to give simple explanations to confusing issues. I think it will be healthy to rotate the course through several faculty, with each passing his or her lecture material on to the next victim. Presumably once several people have taught the course, it will be easy to find instructors who are willing to teach it again.

Finding the first few pioneers is more difficult. Alan Portis came up with the idea of having "Friends of IDS 110," a collection of faculty with interest in the area who can give advice on the direction of the course, but his hidden agenda was to corral lecturers. We found two courageous souls, Dr. Ray Neff and Prof. Merrill Shanks, to take over the course for the second year.

Knowing that others would be teaching the course, I tried to electronically record anything that a future lecturer might want. The course organization reduces the lecturer's task as well, with the TAs handling the labs, demos, and final projects, and multiple choice tests reducing the tedious grading chores sometimes associated with large classes.

Some potential lecturers might be attracted by the course organization, some by the large class size that accrues benefits for their department, but I recommend it because it is the most enjoyable class a teacher could want. Highly motivated students conquer computing in just 15 weeks, and they are anxious to learn what you know. I had several say this was the best class they had taken at Berkeley. It is the type of class that makes you glad that you decided to become a teacher.

Selecting the Head TA

Denise Kiser was the first head TA, and my best advice is to try to clone her. She cares about the course and makes sure every facet reaches the standard we would expect at Berkeley. She is a former high school teacher, so perhaps you want a person with plenty of teaching experience to manage the course.

Selecting TAs

I think the first characteristic you are looking for is good teaching. After that you want good "team players," people willing to pitch in and do the dirty work needed to make the class work. I think the next important set of characteristics are previous computing experience, a sense of humor, and intelligence. Once the course has been offered a few times there is no reason to take computer novices—after all, they could always take the class—and I think without experience it will be difficult for them to answer the wide variety of questions that can occur in programming. A sense of humor helps when you are running a lab with 40 students and 40 computers, and brain power can overcome obstacles as they occur.

The first step in selecting TAs is getting applicants. Since our course is not associated with a single department, we do not have a captive group of graduate students and so we have to beat the bushes. We tried fliers, ads in the personal section of the campus newspaper, electronic mail, and letters to graduate advisors and department heads, and I think we found good TAs using all those means. If I had the money and manpower, I would have tried to put a flier in every graduate student mailbox.

In our first interview cycle we received about 5 times as many applicants as positions. I then had to decide whether I would take the time from my busy schedule to interview the candidates or just make the decision based on the applications. Fortunately, I decided to interview. It is much easier to make good decisions once you have interviewed the likely candidates, especially since

some good people write poor resumes and vice versa. We set up 20 minute interviews with 5 minute breaks to discuss each applicant. This last time the interview team was Prof. White, the head TA, and myself. Figure 12 shows examples of the questions we used. There are three things you must do during the interview:

- (1) Let them know that this job involves a lot of work, including making example projects;
- (2) Find out if they have other opportunities and if they are more likely to take a position within their department than this one; and
- (3) Tell them that there are many more applicants than positions, that it is a highly qualified group, that the decision will be difficult, and that you are telling this to every applicant.

-
- What is your teaching experience?
What do you like and NOT like about teaching?
 - What is your programming experience?
Which language/machine do you prefer? Why?
 - What degree are you working on?
How long before you expect to graduate?
 - Explain your PhD area/topic.
 - What was your undergraduate degree?
What other majors do you think you could relate to?
 - Do you have suggestions for possible projects in your area?
 - Why do you want this position?
 - What are your other job possibilities besides this one?
If offered both, which would you prefer?
 - Do you have questions about the course?

Figure 12. *Example Questions to ask Applicants.*

I went through the applicants and graded them A to D first to see whom to interview. Interview the best qualified first in order to get calibrated: A's on Tuesday and B's on Wednesday. The interview is what we used to make the final decisions.

In making your decision you should keep a few things in mind:

- Try to find potential multi-year TAs so that you never end up with a complete turnover and all new TAs;
- Try to balance the majors covered by the TAs (Figure 13 is a table of the popular liberal arts majors on the Berkeley campus);

Majors acceptable to IDS 110	Grad+Undergrad	% Total
L&S:All Biology Related Fields	1142	12%
Natural Resources	1130	12%
Environmental Design	965	10%
L&S:English	799	9%
L&S:Political Science	631	7%
L&S:Psychology	474	5%
L&S:Economics	438	5%
L&S:History	426	5%
L&S:All Foreign Languages	396	4%
Public Health	395	4%
L&S:Double	386	4%
L&S:Social Sciences	382	4%
Education	365	4%
L&S:Bio Chem (Not IDS 110?)	283	3%
L&S:Pol Econ - Indstrl Soc (pseudo Econ)	229	2%
L&S:Sociology	226	2%
Social Welfare	225	2%
L&S:Anthro	211	2%
L&S:Comp Literature	152	2%
Library Inf/Sys	141	2%
TOTAL	9396	100%

Figure 13. 20 Popular majors at U.C. Berkeley.

This includes the number of graduates and undergraduates in each major, with the right column being the percentage of the total number of eligible students. This does not include students from math, engineering, or the hard sciences. It also does not include the 10,000 lower division L&S students with undeclared majors. Business administration majors are excluded because they are required to take a traditional introductory programming course. These results are based on enrollment figures for Fall 1984.

To help make good decisions, I would collect everything you can before the interview: résumé, transcript, grade point average, GRE scores, letters of recommendation, teaching evaluations, example programs, and so on.

Once you have decided whom you want, you should give the candidates a few days to decide. The primary goal of graduate students is to graduate, so prospective TAs must decide if this offer is best for their careers. TAs are demanding, requiring up to twenty hours per week in an area that can be outside of their immediate field of interest. Although courses like ours contribute to the development of TAs, their principal reason for deciding to teach is that it provides financial support. Since our course is not associated with a pool of students who can start teaching

this course at the last moment, the job offer must make it clear that acceptance is a commitment that can not be compromised. Yet each year we have lost a person who agreed to be a TA, in one case causing a lab section to be cancelled. Thus it may be wise to designate a few applicants as first alternates.

Selecting Student Aides













Student aides should be students who have recently completed the course and would like to help. Students selected to present their projects in the final lecture are likely candidates, but it is every TA's responsibility to identify and encourage two or three of his or her students to volunteer next semester. I think you are looking for outgoing students who did very well in the labs, and TAs should find people they would like to work with.

Selecting Students

The first step is to let students know there is a computer course for students who are not enthralled with computers. One way is with posters and advertisements such as Figure 14. The only strong recommendation for selecting from the students who apply is to keep out "ringers": students who have already mastered computing and are just taking this course to improve their grade point average. I imagine that language classes have the same problem when a foreign student decides to take an introductory course in his native tongue. Such students can undermine the TAs' leadership by acting as if the TAs aren't as good as they are, and can really disrupt the labs. (The one annoying ringer we had in the first course breezed through the applications and then did rather poorly in programming. This gladdened all our hearts.) We tell students they cannot get credit for IDS 110 if they have already taken a Berkeley computing course, and this seems to discourage ringers.

Since the course is oriented to liberal arts majors, we actively discourage engineering and science students from taking the course. Conversely, we give first priority to students in the same major as the TAs. Within a major, we give highest enrollment priority to juniors because we hope to have follow-on computing courses in the major, and juniors will be around for at least another year after the course. (L&S students at Berkeley do not declare their major until their junior year.) Grad students have next priority, so that they can get into the class, and then seniors. Sophomores come next because they normally haven't decided their major, and finally the lowly freshmen.

A pleasant surprise is that every semester we have had one disabled student (in a wheelchair) take the course. Fortunately the lab has wheelchair access. These students were successful in the course and completed all the labs. Special arrangements were not necessary. They just asked fellow students to operate the printers since they are raised above the Macs in Tolman. In fact, things worked out so well that two of the disabled students bought Macs for their own use.

Introduction to Computers	
	IDS 110/110L
	The computer course for liberal arts majors. Learn graphics, word processing, data bases, spread sheets, programming in Pascal & Basic in a non-competitive environment.
	Apply these programs to <u>your</u> field!
	
	
	
	
	
	Lectures: MWF 3-4 PSL1 3 units (IDS110) 88032
	Labs: IDS 110L 1 unit (see schedule) Enroll in both IDS 110 & IDS 110L
	Pre-requisite: No previous university computer courses
	For more info: 642 8790 230 Bechtel

Lab Sections: 1535 Tolman

- 1 Architecture TT12-2 88035
- 2 Biological Sci. MW8-10 88038
- 3 Econ/Business MW12-2 88041
- 4 English/Educ TT4-6 88044
- 5 History M4-6F12-2 88047
- 6 Poli.Sci. M10-12F8-10 88050
- 7 Social Sci TT8-10 88053
- 8 Classics WF10-12 88056
- 9 Multi-major TT10-12 88059
- 10 Multi-major TT2-4 88062
- 11 Faculty/Staff WF4-6

Figure 14. Advertisement for IDS 110

Since computerphobic students will not even read the the section of the course catalog on computing, a new course for them must be creative to find its audience. This advertisement appeared in the Daily Cal in October 1985.

PART III. ESPRIT DE CORPS

In any large project where people are working hard, it helps to have people feel good about what they are doing. The lab instructor and lecturer will enjoy visiting labs and helping students with their problems—it is very rewarding to help them with hands-on learning—plus the students feel good about meeting the instructors. Meeting the instructor reduces barriers and helps students ask questions in class. Undoubtedly the most important time for this is during the final projects when students want to show what they have done and could use a pat on the back for all their hard work.

Since the TAs are going to do a great deal of work, I would also recommend at least one party to celebrate the course. It is a good idea to have some real fun in addition to my version of fun. We had a few parties, including one where we invited the "Friends of IDS 110" to meet the TAs and to see student projects.

Chapter 6. Conclusions

Rolling Your Own Course

"Freeware"

Videotapes

Let Us Know What Happens

Impact of the Course

The Future

This course bridges a very large educational gap, starting with computerphobes and ending up with computer fanatics. The combination of the challenge of what we are trying to accomplish, the feeling of teaching something that is unquestionably important, the insight of guest lectures, and the reduction of grading chores create a rewarding teaching experience for the TAs and instructors. It is tremendously satisfying to see students create imaginative and challenging applications just three months after first using a computer. Comments like those in Figure 15 make glad you became a teacher. On the other hand, the course doesn't work for everyone, and my Pollyanna-ish character can't help but feel disappointment. My advice for instructors (and myself) is to remember that we purposely excluded science and math students (who likely have an affinity for computing) and thus to find joy in any successes and be thrilled that it works for most students.

-
- Before:** Computers are only for the technical and trained.
After: Computers are simple, easy to use.
- Before:** I thought they were more complex than this.
After: Now I think that they are a necessary part of a civilized existence.
- Before:** I was intimidated by the thought of using computers.
After: Now, I'm every comfortable with them.
- Before:** I was really scared!
After: Now, I think they are really great tools. Rarely does my opinion about anything change so fast (15 weeks) and so much (180°).
- Before:**
After: I feel much much more comfortable about computers—not just the Mac.
- Before:** Blah—Yuk—Eeek! = Paranoia!
After: I feel comfortable using a computer although I haven't quite mastered programming yet.

Figure 15. *Quotes from Student Before and After IDS 110.
 (Taken from Appendix II.)*

Rolling Your Own Course

If you, dear reader, are tempted to bring computing to the liberal arts on your campus, then let me pass on a few pieces of advice. The first is that the biggest problem is not getting the computers; the biggest problem is finding them a home. Dick White spent months canvassing the Berkeley campus before getting the sympathetic ear of Dean Gifford in the School of Education. Perhaps instead of trying our laboratory model, you can try scattering machines about the campus and then have the TAs give extensive office hours. It will take some work to make this successful. If you want to try this model, then you need to be concerned about hardware and software security and software distribution. I always thought a library would be an ideal place for such an arrangement, as it already has a security system and a distribution system. Since Apple (finally) has a network, you could isolate the noisy printers to preserve the tranquil library setting. Thus you might start with your quest for space in your library.

My next advice is to get a satisfactory budget before you agree to create a new computing course. You will want to negotiate to get the best discount you can for whatever computers and any software that you buy in bulk, but you need to be able to buy one copy of new hardware or software at your local computer store to see if you can use it in your course. It takes so long and so much work to write letters to manufacturers to beg them to loan you a copy that you probably won't even try. And if you don't update your labs, you are sure to be teaching out of date material in this fast moving field. Since this course is certainly going to involve many segments of your campus, you must hope for an enlightened administration that will fund such an effort off the top before the budget is dispersed to the departments. Without such enlightenment you will have to beg funds while trying to satisfy every fiefdom, and fiefdoms have are distinguished by conflicting sets of needs, demands, and opinions.

"Freeware"

If you are still tempted to bring computing to liberal artists, then we would like to do everything we can to help. The first suggestion is to read this report. The second suggestion would be to attend the Liberal Arts and Computing at California (LACCAL) Workshop given at Berkeley on November 16, 1985. It is too late for this advice, but we did have about 200 people from 100 colleges spend a day at Berkeley and perhaps someone attended from your campus and can help. We also distributed lab manuals and software to attendees. If you are interested in receiving a set of floppies and manuals, please send mail to

INTERDISCIPLINARY STUDIES
COLLEGE OF ENGINEERING
230 Bechtel Engineering Center
University of California
Berkeley, California 94720
Attention: LACCAL Material

or call (415) 642-8790 and ask to get the material. IDS will ask for a small amount to cover the cost reproduction of the material. This includes one floppy for the lab software, two floppies for the lab manual, one floppy for the exams, one paper copy of the lab manual, and copies of lecture notes.

Videotapes

My original plan for helping you prepare lectures is my cryptic lecture notes being distributed by IDS, but we recently were given a generous offer from the Televised Instruction Program in the College of Engineering. They are willing to make copies of the videotapes of the 40 IDS 110 lectures and the LACCAL workshop available to educational institutions at their costs. (Industrial institutions pay a considerably heftier sum.) The LACCAL tapes will give you an overview, and you might find the six demonstrations segments useful in class. I think the lecture tapes are primarily useful for instructors to watch my lectures using a VCR to help make sense of my lecture notes so as to give even better lectures. There are also a few tapes that you might want to show students in the classroom. The guest lectures and the graphics lecture are candidates, and I would especially recommend the tape of the student presentations at the end of the course. We now use edited versions of the student presentations to inspire the current crop of students to create their own imaginative projects. If you are interested in getting tapes, please write to

TELEVISED INSTRUCTION PROGRAM
COLLEGE OF ENGINEERING
312 McLaughlin
University of California
Berkeley, California 94720
Attention: Special Programs Coordinator (LACCAL)

or call (415) 642-5776 and ask for the order form for the IDS 110/LACCAL tapes. It is more than a small amount to cover costs of making tapes, but you can order tapes individually to lessen the initial expense. (Keeping things in perspective, Jane Fonda charges more for her tapes than we do.) I think the total set will be about the cost of a Mac, and perhaps that is a good investment for a campus.

Let Us Know What Happens

If you decide to try such a course, please write me to let me know what happens as I would enjoy getting letters from such courageous souls, especially telling me what worked and what didn't. My address is

Prof. David Patterson
Computer Science Division / EECS
573 Evans Hall
University of California
Berkeley, California 94720.

Impact of the Course

It is pleasing to report that several students have become hooked by computers and are now making a living using them. I come from a long line of storytellers, so I can't help but relate a few anecdotes. Last Spring a Public Health graduate student became so excited about computers that he bought his own Mac. His project was a programmed quiz for people who work with hazardous material to see if they know which gas mask they should wear for each type of environmental danger. During the class he interviewed for a job at the Atlanta Center for Disease Control and he brought his Mac to demonstrate his project. The interviewers became excited and offered him a job and the last I heard they plan to use his program to train workers.

I was a bit surprised to find that many liberal arts students have an entrepreneurial bent. Several students didn't want their programs included in a demo floppy because they plan to sell it, although I am not aware of any commercial success stories.

The course has also had an impact on the instructors. Last year's TA from Psychology, Zoe Kersteen, is now doing research on why people have difficulty learning about computers. Neel Smith, the TA from the Classics Department, has led an effort to bring computing into the classics, taking advantage of computers to examine classic works in the original Greek. This summer he and Joshua Kosman, another IDS TA, wrote a 5000 line C program that essentially models the Greek language so as to find all forms of Greek words. This program is the first to include accents—and every Greek word has an accent—and they can now search the on-line Greek manuscripts using the basic form (e.g., "to see") versus literal search for one instance (e.g., "we saw"). They believe that this information will allow them to go much farther in natural language understanding than is possible with languages like English. Classics seems to be a good match to computers, for a professor who took the course last Spring has started using computers in the main first-year Classics graduate course. Denise Kiser, the head TA and a former high school teacher, used the lab and course material to teach a three week short course for high school teachers on using computers in high school. Her course received rave reviews from the high school teachers.

The course also changed me. Personal computers were generally thought of as the bastard child of computer science, and this course has allowed me to form my own opinion. I now use Macs routinely in my preparation of documents and lectures. The experience of creating a course from scratch and making sure there was a sound educational reason for every facet has made me rethink everything I teach. This course has piqued my interest in the applications of computers versus the building of better computers, so I wouldn't be surprised if my research changes direction in the next few years.

The Future

I expect new exciting applications will arrive on the scene every few years, and we will want to include them in the course. One area obviously missing is communication, as we need a network to show the joys of electronic bulletin boards, electronic mail, and file servers. Beyond communication, I expect the marriage of computers and compact disks to open new horizons. The massive amount of inexpensive storage for information and graphical images combined with a personal interactive computer seems ripe for exciting applications. Finally, the 1984 Macintosh is not going to be the best computer for liberal arts students in 1990, so we should expect to change the hardware base periodically to keep the course in line with the modern computing.

Since computing ideas are filtering from graduate school to undergraduate school to high school and to grade school, we should expect the course content to change as high schools and colleges do a better job of teaching computing. The theme of this course will always be the application of computing to liberal arts, and I believe there will always be new ideas in computing that are not easy to learn yet are important for liberal arts students. The course should always be many years behind the research in the computing field and many years ahead of what is being taught to most high school students. As high schools start routinely teaching applications and Pascal programming, the course might move into artificial intelligence by programming in languages such as Lisp, Prolog, or Smalltalk. When high schools start teaching about artificial intelligence and programming in Lisp, the course could move into the computerization of knowledge in their fields by building expert systems using program generators not yet invented.

The future is not static nor mundane: it will continue to provide exciting opportunities for those who learn computing as a means of solving important problems in their chosen field.

Acknowledgements

The Teaching Assistants bear the brunt of the work of teaching this course, and I would like to thank all of them for their dedication: Mark Bedau, Geri Bunker, Noel Cary, Henry Chen, Maurilia Flores, Zoe Kersteen, Denise Kiser, Joshua Kosman, Drew Kravin, Andy Lazurus, Eric Lum, Stephen Manweiler, Anne Marchant, Rand Miyashiro, Neel Smith, Sterling Speirn, and Jennifer Sutter. I hope this course has been as rewarding for them as it has for me. Special thanks go to Mark Bedau, Denise Kiser, Joshua Kosman and Neel Smith for working during school breaks revising the labs. Kathy Schermerhorn-Cousens and Judy Ackerman of IDS deserve credit for their patient support of this course. Discussions with Mark Tuttle, Mike Clancy, and Doug Cooper, who teach introductory programming courses at Berkeley, were useful in shaping my ideas on the course and the labs. I am also indebted to Denise Kiser, Alan Portis, Neel Smith, Sterling Speirn, and Noel Cary for giving me feedback on drafts of this report, and to Carlo Séquin for suggesting that I write this report.

The course could not have been offered if we didn't have computers, and Prof. Richard White deserves credit for his efforts to create the Tolman Microcomputer Facility and his help in this shaping the ideas used in this course. This was not a one man effort, and I would like to thank the Office of Computing Affairs, under the leadership of Dr. Raymond Neff, Joe Yeaton, Barbara Morgan, Peter Rauch and others for creating and maintaining that facility.

The course itself needed the support of the administration, and I would like to thank Dean Bernard Gifford, Dean Karl Pister, Provost Doris Calloway, Vice-Chancellor Roderick Park, and Chancellor Michael Heyman for approving and financing this unusual course. The various chairmen of my department encouraged me and stood by my decision take leave of my normal teaching responsibilities to teach this course, so I would like to thank Domenico Ferrari, Donald Pederson, and Carlo Séquin.

Prof. Séquin deserves special credit for telling Alan Portis, Dean of IDS, that I was tempted to try this course. Dean Portis then gently twisted the arms of the administration, Dick White and myself to make it happen. This course couldn't have occurred without Dean Portis to navigate our ideas through the dangerous bureaucratic waters of Berkeley.

Special thanks goes to Denise Kiser, who has been the Head TA since the very beginning. She has proved to be the steadying influence through all the problems and has provided valuable ideas to make it better. I think it was her idea to create the lab manual, and she rallied the TAs to work hard and improve it. I shudder to think what would have happened if she hadn't been involved.

My final acknowledgement goes to Dr. Barry Boehm. I am sure he is unaware of it, but it was his comment—that he was sad that social scientists at UCLA could not take the overcrowded data base courses that they surely need for their research—that led me to wonder whether this was also a problem at Berkeley, and whether there wasn't something we could do.



Appendix I. History of IDS 110

Dave Patterson's involvement with IDS 110 began at a weekend retreat of the UCLA Computer Science Department in 1983. Like C.S. departments all over the country, UCLA was facing skyrocketing student demand with limited resources. As the UCLA faculty explained to those invited to the retreat, the department decided to focus its resources on students who wanted to major in C.S. An alumnus said that while he understood the decision, he was saddened to learn that people in other disciplines would not have the opportunity to learn how to use computers to solve their problems. Patterson thought if it was true at UCLA, it might also be true at Berkeley.

This started him thinking about how such a course might be created at Berkeley. The first step was computers, but he believed it might not be long until many students had access to microcomputers. The second step would be finding graduate students who could act as Teaching Assistants. Since unemployed Computer Science graduate students were hard to find, he thought you might try finding TAs outside the CS department. This idea evolved to having each TA be associated with students from his department, allowing the TA to tailor some of the labs to the student's field. Patterson also thought that it might be easier to teach about computing if students were first exposed to important applications on computers such as word processing, spread sheets, and data bases. The inflexibility of these applications might inspire students to make their own programs. Since every class he had taught ended with a team project, he thought a project in which groups of students solved problems in their field of interest would be a great way to finish the class. In discussions with colleagues in other fields, it became clear that the lack of computer expertise was not just an undergraduate problem, for many graduate students and faculty were inexperienced as well.

As these ideas began to crystallize, Patterson confided his thoughts to his good friend Carlo Séquin, who happened to be chairman of the C.S. Division. He had first made Séquin promise not to tell anyone else, since Patterson was not sure he could handle such a course. (He had never even taught a programming course.) In less than a week after being sworn to secrecy, Séquin received a phone call from Alan Portis, Dean of Interdisciplinary Studies. Portis had noticed that calculators came on the market that understood the Basic programming language, and wondered if a computing course could be taught to non-science students using those calculators. Séquin decided to tell Portis about Patterson's ideas.

Portis called a startled Patterson to tell him that he had heard Patterson was interested in teaching an introductory computer course using microcomputers. He then arranged a meeting with Dick White, a colleague of Patterson in EECS. White had spent the last year or two of his spare time creating an Instructional Technology Center on campus that had as one objective setting up microcomputer-outfitted classrooms so that courses could be offered that used microcomputers. White also tried out some microcomputer lab exercises in Spring 1984 in a course labeled EMST 20, offered in the School of Education. Patterson and White were surprised to discover each

other's plans, so Portis proposed they join forces to offer a class along the lines Patterson had discussed using the labs that White was building in cooperation with the Office of Computing Affairs.

The first step was to find space for the labs and money to purchase the microcomputers. Bernard Gifford, Dean of Education, agreed to provide 1645 Tolman to house 60 microcomputers as well as being a co-sponsor of IDS 110 with Karl Pister, Dean of the College of Engineering. The Office of Computing Affairs, under the leadership of Joe Yeaton, Barbara Morgan, Peter Rauch and others, agreed to purchase the 40 MacIntosh and 20 IBM PC's as well as to hire the staff that run the Microcomputer Lab. (Additional micros are housed in 211 Wheeler.)

The next step was to raise the funds for IDS 110. Doris Calloway, Provost of the Professional Schools and Colleges, had long been a believer in interdisciplinary courses and became an active supporter of the course. Provost Calloway and Chancellor Heyman provided the funds to hire the TAs to develop and teach the course in 1983-84.

With the space and funds, the next step was to find TA's. In Spring 1984 Patterson and White put up a small number of posters advertising for TA's for the course and found more than 70 applicants for the 12 positions. After interviewing 30 strong applicants, they hired excellent graduate students from 12 areas. Patterson and White concluded that strength of the applicants from many areas was another testimonial to the high caliber of departments on this campus.

In Fall of 1984 IDS 110 went through a two-stage "debugging" process. The first version was taught to the 12 TAs. After the TAs found the mistakes in the lectures and the labs, "new and improved" versions of the lectures and labs are given a few weeks later to a group of 28 undergraduates who volunteered to be guinea pigs. The ideas and labs were revised once again during the Christmas break, and in Spring of 1985 the first full version of the course was given to 250 students and 25 faculty.

Appendix II. Results From Student Surveys

This is the summary of three surveys taken in 4th, 7th, and 15th week of the Spring 1985 semester.

Statistics on students in class

1. Year: Freshman Sophomore Junior Senior Grad Staff
 1% **4%** **31%** **43%** **18%** **2%**

2. Sex: Male **39%** Female **61%**

3. Age: < 18 18-21 22-25 26-30 31-40 >40
 0 **50%** **30%** **10%** **6%** **4%**
 (Note: only 2 % staff)

4. L&S "Quantitative Reasoning" Requirement:

5% You are an L&S undergraduate who is **supposed to** and has **not fulfilled** this requirement.

95% You are supposed to and you **have already fulfilled** this requirement.

5. Experience with computers prior to IDS 110 (all that apply):

31% Never used a computer.

48% Used a program like MELVYL.

22% Used word processing program.

10% Used statistical analysis programs.

6% Used "on-line" data collection programs.

2% Used computer graphics/CAD programs.

1% Developed software before IDS 110. (**1 person**)

7% Took a high school computing course

6% Took a college computing course.

5% Took a computing course elsewhere.

6. Microcomputer ownership (asked at the end of class)

5% You now own a microcomputer.

14% Your family owns a microcomputer.

13% You decided to purchase a microcomputer after this class.

74% You would like to purchase a microcomputer after this class, but you really can't afford it.

Time spent in class, each segment of the class

First 4 weeks

7. Hours per week (not including 3 hrs lecture, 4 hrs lab)

<=2	3-4	5-6	7-8	9-10	11-15	>=16
29%	37%	24%	6%	2%	1%	0

(Note: Class average is 3.9 hrs/wk. 91% spend \leq 6 hrs/wk.)

Second 3 weeks

8a. 32% taking IDS 110 Pass/Not Pass (Or S/U). Hours/week for them:

<=2	3-4	5-6	7-8	9-10	11-15	>=16
38%	38%	17%	6%	0	0	1%

8b. 68% taking IDS 110 for a grade. Hours/week for them:

<=2	3-4	5-6	7-8	9-10	11-15	>=16
18%	46%	29%	5%	3%	0	0

(Total(3&4): 24% 43% 25% 5% 2% 0% 1%)
(Note: 92% spend \leq 6 hrs/wk.)

Next 5 weeks (programming)

9. For the time after the second quiz and up to the beginning of projects, what was the average number of hours per week have you spent on IDS 110? (Do not include the 3 hours of lecture or the 4 hours of lab.)

<=4	5-8	9-12	13-16	17-20	21-25	>=26
51%	37%	7%	2%	2%	1%	0

(Avg. = 4.9 hrs/wk. 95% \leq 12 hrs/wk or 88% \leq 8 hrs/wk or 12% \geq 9 hrs/wk)
(Note: we changed the hours/week in this survey.)

Last 3 weeks (project)

10. For the time last 3 weeks (the project period), what was the average number of hours per week have you spent on IDS 110? Do not include the 3 hours of lecture or the 4 hours of lab.

<=4	5-8	9-12	13-16	17-20	21-25	>=26
21%	31%	24%	7%	7%	3%	3%

(Avg. = 9.1 hrs/wk. 90% \leq 20 hrs/wk or 48% \geq 9 hrs/wk)

Project

11. Now list all the programs your team used in completing your project (not including the final report):

4% MacPaint.	24% MacWrite.	7% Multiplan.
19% Filevision.	10% DB Master.	70% MacPascal.
11% Some other application or machine.		

12. Number of people on project team:

17% You didn't have a partner.
50% You had one partner.
31% You had more than one partner.

Programming

First, programming on the blackboard.

- 13. 72%** did the first take home assignment before class.
(50%: put down the knife and fork before it picked up the glass.)
- 14. 50%** did the second take home assignment before class.
(66% did (a), 62% did (b), 27% did (c), 24% thought (c) easier)

15. Programming Concepts you feel uncomfortable with or are unclear:

(Question 15 was asked before the second quiz, and before they had programmed on a computer.)

Loop:	9%
Go To:	9%
Procedures/Subroutines:	18%
Call/Return:	11%
Assignments:	20%
Comments:	10%
Variable:	20%
Name vs Value:	23%
If-Then-Else, If-Then:	26%
Repeat-Until, While-do, For:	38%

16. In creating this course we thought it would be a good idea to show you Basic, a very simple language, after you learned Pascal. We also wanted you to get experience programming on the IBM PC. We would now like to know what you thought. Feelings about Basic on IBM PC:

25% You think Basic is an unrepresentative programming experience and it would be better to try Pascal on the PC.

22% You think IBM PC is an unreasonably difficult Basic and it would be better to try Basic on the Mac.

17. Feelings about Basic:

9% You think people can learn it on their own and we should spend the course time on something else.

40% You we spent such little time on Basic we should either drop it or spend more time to cover it in detail. (As an aside, the course has a limited time budget, so spending more time means cutting something else.)

68% You think we should simplify the labs a bit but you are glad you tried Basic on the IBM PC.

18. Programming Quiz

We believe that the programming quiz is important to see how much you know before we start the projects. Since that was the first isolated programming experience, the exam was more difficult than we expected.

55% You think we should gradually build up to the quiz by requiring more programming with fewer hints in the prior labs.

35% You think the TAs (and other students) should never tell you exactly what is wrong but just give hints, preventing you from relying too heavily on them while learning programming.

19. Questions on Pascal Programming

2% You knew Pascal before this course. (**2 people**)
(These 2 are not counted below.)

You believe you can (asked at end of course):

- 72%** read and understand Pascal programs.
- 76%** modify small (1 page) Pascal programs.
- 57%** modify medium (3 page) Pascal programs.
- 29%** modify large (8 page) Pascal programs.
- 79%** write small Pascal programs from scratch.
- 48%** write medium Pascal programs from scratch.
- 22%** write large Pascal programs from scratch.

Miscellaneous**20. Tolman Microcomputer Lab Use**

- 41%** You plan to use Tolman Microcomputer Facility next year at the current \$1/hour rate.
- 68%** You would use Tolman Microcomputer Facility next year if it was free.

21. Mark any and all statements that cover your feelings about the IBM PC:

- 91%** Glad you got a chance to try the IBM PC, whether or not you liked it.
- 10%** Extremely frustrated with the IBM PC and never want to see it again.
- 52%** Difficult to use but able to get work done.
- 32%** Some things preferred on the IBM PC over the MAC.
- 13%** The IBM PC is almost as easy to use as the MAC.
- 19%** Would be equally happy to use either the MAC or the IBM PC.
- 9%** Prefer the IBM PC over the MAC.

22. Videotapes

- 46%** You never looked at an IDS 110 videotape
(not including those shown during the lecture times in 1 PSL.)
- 42%** You looked at videotapes to make up an occasional missed lectures.
- 9%** You relied on videotapes to see lectures because you frequently missed lectures.
- 7%** You used videotapes to review lectures you had seen in PSL
(for example, reviewing before exams).

23. 54% You consider yourself a born-again computeroid. The IDS 110 computeroid's motto:

"Before IDS 110 I didn't want to be in the same room with computers (or the people that used them), and now I can't imagine doing my papers or my research without a regular computing fix."

Quotes from students before and after IDS110

24. What was your impression of computers before this course, and what is it now? (This was asked on the last day of classes, and the following are sample before and after answers. While the course didn't work for everyone, it did work for these below.)

Before: Computers are only for the technical and trained.

After: Computers are simple, easy to use.

Before: I thought they were more complex than this.

After: Now I think that they are a necessary part of a civilized existence.

Before: I was intimidated by the thought of using computers.

After: Now, I'm every comfortable with them.

Before: I was really scared!

After: Now, I think they are really great tools. Rarely does my opinion about anything change so fast (15 weeks) and so much (180°).

Before:

After: I feel much much more comfortable about computers—not just the Mac.

Before: Blah—Yuk—Eeek! = Paranoia!

After: I feel comfortable using a computer although I haven't quite mastered programming yet.

Before: I didn't like them before.

After: Now, I think that they are the best things in existence. (Really!)

Before: Computers are complicated—Pascal was as easy to learn as Greek—All computer computers and computer courses are over my head!

After: Computers are wonderful and easy to learn—Pascal is logical and takes only practice to master—Computers are not big, scary monsters but useful tools which make file much easier—even for an English major!

Before: I was fairly scared by computers—I thought you needed a billion match courses to understand them, which we English majors don't have.

After: Now I love computers, feel comfortable with them, and plan to continue with my computer education. P.S.: This class helped me to get at a computer company!!

Before:

After: I am more comfortable with them. Doing the project was even fun (but lots of work), and was the part of the course which helped me realize the wide range of uses for computers (even in history!)

Before: I stayed away from them because I didn't know how they worked at all. They were intimidating because it seemed like they knew more than I did. I felt like the Age of Computer Technology was growing by leaps and bounds and I would be left behind.

After: Now I feel much more comfortable with them, and I feel as if they are there to help me, not to scare me. They are only as useful as the person wants it to be. I look forward to taking more advantage of them in the future.

Before: I was intimidated by computers and thought they were imposing.

After: I feel much better about computers—by this I mean that I understand their uses better than I did, and I feel particularly relieved that I understand how to program now.

Before: I was intimidated,

After: but now I'm not.

Before: I was intimidated by them.

After: Now I have a healthy attitude, am no longer scared by them. Very interested in application in my field. I think programming is fun.

Before: I was intimidated by computers before this class,

After: but not so much now. I think programming is kind of fun ... when you get it working!

- Before:** Was somewhat intimidated by them since I had never used one.
After: Now I want to get my own!
- Before:** Intimidating—
After: Now I own a Mac.
- Before:**
After: I am happy to have had the experience and feel less intimidated by computers. I also see how complex certain applications are and appreciate a good program and the value of good software.
- Before:**
After: Now I'm no longer intimidated by computers themselves. However, programming still makes me nervous! I really enjoyed learning about MacPaint, MacWrite, and Filevision, but I'm not sure I understand Pascal as well as I should.
- Before:** I was intimidated by computers because of the many hours friends of mine have spent in the basement of Evans working on a program. *[Ed. CS courses are taught on computers in Evans.]*
After: Depending on the type of computer, they aren't so bad. They can be useful, helpful, and even fun.
- Before:** Computers were something important in this day and age, and I never thought I'd get the opportunity to really use one and learn from it. I thought I would have to take CS 8 or one of the time consuming computer courses for CS majors.
After: IDS 110 was a great introduction course to computers—and my impression of computers has changed. I now have an idea how useful computers can be. I now also know why people "program" and why so much TIME is spent doing it.
- Before:** A little scared.
After: It seems that the other CS courses on this campus try to make something hard that is not incredibly hard.
- Before:** Apprehensive and frightened;
After: More confident, but not a master.
- Before:** Scary
After: Friendly
- Before:** I didn't want to have anything to do with the, because they scared me and I thought they were beyond my comprehension.
After: This class has proved to me that that assumption was entirely wrong.
- Before:** Scary. Impossible to learn.
After: Practical, especially IBM PC. Even I can learn how to use them to my advantage.
- Before:** I was scared by them.
After: Now I think they are easy—its just a matter of learning commands, etc.
- Before:** I was scared bleep-less of computers.
After: Now I'm not scared of computers in general and the Mac in particular.
- Before:** Scary.
After: Let me at 'em!
- Before:** Severe apprehension.
After: Now, while I still have much more to learn and really should strengthen my programming skills, at least I feel very comfortable with computers, especially the Mac.
- Before:** I feared them.
After: I think they are interesting, useful but can be intimidating.
- Before:** I was afraid of them
After: I appreciate them as valuable tools which are limited in their effectiveness by the intelligence of their programmers and users.

- Before: I was pretty afraid of them—I thought for sure that if I touched one that screw something up or everything would blow up.
After: Now, I feel pretty comfortable in front of them.
- Before: Hesitant to go near them. Fear. Angry that they took away a person's individuality.
After: I realize that computers serve many functions. They can be helpful. However, programming is still very frustrating for me, but also very challenging!
- Before: I felt unsure because I thought they were beyond me due to my poor math and logic skills.
After: Now I know I'll never be a hacker, with practice I may become an adequate programmer.
- Before: I wasn't nervous about computers, but I wasn't really aware what they could do.
After: I didn't know programming could be so much fun!
- Before: Didn't know much;
After: Feel pretty competent.
- Before: Unlimited potential
After: Unlimited potential and well on my way.
- Before: Reluctant fascination, a sort of necessary but scary evil.
After: Demystified. I know just how far I want to go, and not go in programming.
- Before: They really excited me—they seemed mysterious.
After: I want to pursue a career with some type of programming involved.
- Before: Computers were distant things,
After: Now they are simple tools I can use to my advantage.
- Before:
After: I like them. I will in all probability buy a Mac within the next 4 years. The application potential is enormous. Incredible time efficient tool.
- Before:
After: I need one but they are too expensive.
- Before: I was fascinated by them before;
After: Now I want to buy one. Bad news.
- Before:
After: Bought one!
- Before: They seemed terribly mysterious (not necessarily threatening).
After: I have found them to be quite accessible. Field in general seems to be a male bastion, though.
- Before: I was sort of indifferent before,
After: but now I like working on computers.
- Before: I didn't think computers would be very useful.
After: Now its helping me a lot, especially in writing papers.
- Before:
After: I see that they can be helpful in lots of situations where I wouldn't have thought of using them. I'm much more in total awe of super programmers and engineers (especially thanks to guest lectures).
- Before: I felt they were very difficult to understand and use.
After: The course was taught very well and computers seem "BASIC."
- Before: Tricky, difficult to figure out and use.
After: very handle-able (thanks to Mac), but the IBM PC still scares me

- Before:** Incomprehensible black boxes.
After: Possibly comprehensible multicolored machines.
- Before:** Didn't want to have anything to do with it. Thought it was very hard to learn.
After: I still think computers and programming is hard overall, but not as hard as before this class.
- Before:** I was ignorant of any of any of the terminology, etc.
After: Now, at least I'm familiar enough to practice and try to master my skills.
- Before:** I thought they were boring and required massive memorization of computer jargon to even use it.
After: Now, Mac shows hope for non-computer jocks.
- Before:** I thought they were boring, ugly things.
After: Now, I think they can be managed.
- Before:** I felt you had to be a weirdo to understand or get into writing programs.
After: Now I realize its not as hard as people make it sound.
- Before:** You had to be a brain to use one,
After: I now feel that anybody can use one.
- Before:** I was in awe, I saw people who would ramble on about them and leave me in the dirt.
After: Now I'm better able to cope with the terms and ideas behind computers.
- Before:** It was hard to understand, requires intelligent people to work with them ...
After: Easy, normal people can program.
- Before:** Unknown enemy before;
After: fun now!
- Before:** Could be useful—but, how the heck do they work?
After: Oh, now I get it; just think football!
[Ed. We said a computer is like a football player, just doing what the coach tell him to do. So blame the coach, not the player.]
- Before:** [Before Computer says] "What are you doing!"
After: [Now Computer says] "Hi! May I help you?"

Appendix III. Letters to Recruit a Guest Lecturer

February 12, 1985

Dr. Alan Kay

Dear Alan:

I am writing to invite you to give a talk before an unusual audience at Berkeley. I am teaching a brand new course, "Introduction to Computers," aimed at introducing computers to humanities students. We have about 250 juniors, seniors, and graduate students in majors such as Philosophy, Music, Sociology, English, History, Classics, Psychology, and so on. We also have about 30 members of the U.C. Berkeley humanities faculty who are taking the course. I have found this to be a very interesting audience. We are educating people who have traditionally been neglected in introductory computer courses. (Recently, Communications of the ACM ran an article that showed that from 60% to 80% of students in such course are male, but our percentages are just the opposite.)

The room is well equipped for giving a presentation. We meet in a large Physics lecture hall that includes a rotating stage, a 35 mm slide projector, an transparency overhead projector, a G.E. lightvalve projection system, and TV cameras to tape the lecture or to show close-ups on small items. We can also project video tapes on a 25 foot screen.

Although I think you could give a variety of interesting talks to this class, I would really like to hear your thoughts about the future of computers. I was planning on trying such lecture myself, but I decided it made a lot more sense for someone who has tried to figure it out before with some success. Besides, I have never heard you talk and I thought it would be fun.

Our class meets MWF from 3:10-4:00. Although we can be flexible about the schedule, it would be convenient if you could give a lecture on Wednesday May 1. If you are interested in giving a talk, please send me a title and an abstract if you have one.

You can contact me at

Computer Science Division
573 Evans Hall
University of California
Berkeley, California 94720

you can send me electronic mail as Patterson@Berkeley or ucbox!patterson, or call me at (415) 642-6587/642-1024.

In case it would be useful, let me give you a little more background about the class. Each lab section has students largely of a single major (e.g. Classics) with a TA who is a graduate student in the same are. They use a MacIntosh computer for 4 hours of lab per week, starting with applications such as drawing, text editing/word processing, spread sheets, and databases. They are next introduced to programming in Pascal. After learning Pascal, they are exposed to Basic. The final step is that they are required to use computers in a project that is related to their major. For

example, some of the classics students are going to explore some of the Greek texts to find where phrases occur and to look at variations in versions of the manuscripts. They spend two of the weeks in the labs using the IBM PC (Dbase II and Basic). By the time of your presentation, they would have completed all the applications, all the programming labs, and be towards the end of their projects.

We are video taping this inaugural version of the course to make it easier for future instructors to teach the course. (Next year a Liberal Arts professor will teach the course.) We would like to tape your lecture and keep it for future versions of the course. There may also be interest at other universities in getting copies of our material so that they could duplicate this course. Would you object to either use of your lecture? I would also like to know whether you would like me to invite C.S. students or not.

By the way, there will be a small honorarium.

Sincerely,

David A. Patterson
Professor

April 8, 1985

Dr. Alan Kay

Dear Alan:

Thank you very much for agreeing to give a talk about the future of computers (or whatever) in my "Introduction to Computers" class. We are expecting you to visit on Wednesday May 1. We will arrange parking permit at the East Gate, but this is a hunting permit rather than a reserved space so you might want to allow for some time to find a parking spot. You have been to Evans before, but if you forget you can ask the parking attendant to point it out. I suggest you aim to get to my office (527 Evans) about 2pm.

If you have the time, I would like to invite you to drink beer after the class with me and interested students. If you planned to be at Berkeley from 2 to 5 or 5:30, that should be enough to enough time. I have included a map with the directions on how to get here.

In case you have nothing else to do in your busy schedule, it turns out that the same day that you are talking we are having a luncheon to celebrate the good job by the TAs and to show off to the administration. We will invite deans and provosts and have several MACs to show the projects that the students have been developing. As I mentioned, there is some tie-in with their majors. The luncheon is from 12 to 1, and you are certainly welcome to come in case you would like to talk to the TAs and see the projects. Let me know if you are interested, and I will wait for you in my office before heading over. This is strickly an offer in case you would like a free lunch, so don't worry if you can't make it.

By the way, we just had Doug Engelbart come speak and it was a rousing success. Doug brought 3 or 4 papers about what he was talking about, and I had to make 2 batches of copies for the students to satisfy their interests. Since this material was strickly optional (I won't ask any exam questions on the papers), this seems to indicate genuine curiosity and interest. This worked so well that if there were a few articles that would supplement your talk, I would be happy if you sent me something to copy for the class. Alternatively, you could bring papers with you and I could have copies made for students to pick up later. For your interest, the other guest speakers are Donn Parker on computer crime, Mark Stefik on expert systems, and John Ousterhout on programming lessons. We also presented a tape with Bill Atkinson talking about MacPaint and MacVision (recorded at the Berkeley Mac Users Group) and a tape with computer generated movies from SIGGRAPH 83.

In terms of your talk, I would say the most important aspect is to leave time—say 10 minutes—for questions, since these people have quite a different viewpoint and I think their questions are part of what makes the lectures enjoyable. I would also like to know what equipment you want to give your talk. We have every piece of equipment I have ever seen available, but I need to tell the staff in advance to set it up.

The semester will be almost finished when you arrive, with the students final projects due the next day. They should understand programming, plus have experience with the following programs: MacPaint, MacWrite, FileVision (a visual data base), and MacPascal on the Mac plus

DBase II and Basic on the PC. I certainly hope that by this time they have their own opinions about the role of computers in their future.

If you need to send me papers or let me know about the luncheon you can contact me at

Computer Science Division
573 Evans Hall
University of California
Berkeley, California 94720

you can send me electronic mail as Patterson@Berkeley or ucbvax!patterson, or call me at (415) 642-6587/642-1024.

We are video-taping this inaugural version of the course to make it easier for future instructors to teach the course. (Next year a liberal arts professor will teach the course.) We would like to tape your lecture and keep it for future versions of the course. There is may also be interest at other univerisities in getting copies of our material so that they could duplicate this course. Would you object to either use of your lecture? I would also like to know whether you would like me to invite C.S. students or not.

So if I don't hear from you, I will assume

- (a) you will be at my office on May 1 about 2pm and will stay to 5 or so,
- (b) you don't want me to invite CS students,
- (c) it is OK to tape your lecture, and
- (d) you only need an overhead projector and blackboard to give your talk.

There is still a small honorarium, and thanks once again for your help.

Sincerely,

David A. Patterson
Professor

encl.

