Workbook for CMPE 185
Technical Writing
for Computer Engineers
and Computer Scientists

Kevin Karplus
Dan Scripture

April 3, 1996

Board of Studies in Computer Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

This document contains course notes and exercises for a course in technical writing. The course is intended for third-year computer engineering majors, and emphasizes technical documentation directed to engineers, engineering managers, technical writers, and other specialized audiences. Exercises include job applications and résumés, memos, electronic correspondence, algorithm description, in-program documentation, naive-user documentation, survey articles, proposal writing, document specification, progress reports, formal technical reports, and an oral presentation.

Students are expected to have reading knowledge of PASCAL or C, and to be able to follow explanations of recursive algorithms.

Keywords: technical writing, computer engineering, writing for engineers

Copyright

Acknowledgements

Some of the assignments in this workbook were originally created by Renny Christopher, and have been used here with minor modifications. We thank Renny for giving us permission to use these assignments. Tracy Larrabee has taught using this workbook several times—her feedback and editing have been invaluable.

Contents

1. Syllabus

1.1 Administrative details

Location and time Porter Acad 144, MWF 3:30–4:40

Instructor and TA

Kevin Karplus	315B Appl. Sci.	M 1:00–2:00 + appt.	459-4250	karplus@cse
Jenny Almy	Trailer 12	M 1–2:30 Th 2–3:30		almy@cse

The office hours may change during the quarter, due to conflicting schedules—it is a good idea to verify the hours before coming.

Tutors We will have one or two writing tutors this quarter (depending on enrollment).

You will be expected to see a tutor about every two weeks (four or five times during the quarter). Some students may be requested to work with the tutors more frequently. Past experience has been that the best writers were also the most diligent about seeing the tutors (or that the students who used the tutors most ended up writing best—cause and effect are a little hard to separate here).

1.2 Required Texts

Strunk and White, *The Elements of Style* [SJW79]. Strunk and White is the best short book on writing style. You should read it from cover to cover at the beginning and the end of this quarter, and re-read it every five years thereafter. The examples are subtly humorous, but you might not appreciate the humor on the first reading.

Huckin and Olsen, *Technical Writing and Professional Communication for Nonnative Speakers of English* [HO91]. We used the first edition of this text (which was called *English for Science and Technology: a Handbook for Nonnative Speakers*) with good results for two years, and the new edition for a few more. The new edition seems to be even better—it is well-organized, has good examples, both positive and negative, has a fairly good chapter on the use of visuals and a good chapter on oral presentations, and is available in paperback. Chapter 1 is a good explanation of why technical writing is a required course for computer engineers—reading it may help you understand the importance of the entire course.

Also, Chapters 29 through 38 are a handbook designed primarily for advanced non-native speakers of English, a necessity for this class. Parts of this handbook are useful for native speakers too; for instance Chapter 32 deals with modal verbs (*would*, *could*, *should*, and so forth), which many native speakers have difficulty using correctly in writing. The appendixes include a good short summary of English punctuation of use to everyone.

The text's major weakness for this class is that it is not aimed at computer engineers, but at engineers in general, and especially mechanical engineers, judging from the examples and illustrative material in the text.

We will assign some chapters to read from Huckin and Olsen, but you'll find that browsing through the unassigned chapters will yield useful information. The book is well-written and easy reading—unlike so many of the writing textbooks on the market.

Karplus and Scripture, *Workbook for CMPE 185 Technical Writing for Computer Engineers.* This workbook, which will be available electronically, is where your assignments actually are, as well as some other material. We are providing them electronically, rather than as a pre-printed workbook, to save paper and money (the pre-printed ones cost more per page than laser-printing them would have!).

These assignments are designed to be appropriate for computer engineering, and have mostly been developed in 1988 and 1989, although some of them go back several years to other courses at other universities. We welcome motivated, specific, written comment on ways you think they might be improved. Don't just tell us there are too many and they're too hard! We know that, and there's not much we can do about it without seriously weakening the class.

1.3 Recommended texts—on reserve in the Science Library

Hornby, Harris, Stewart, *Oxford Student's Dictionary of American English* [HRHS86]. Particularly recommended for non-native speakers, this may be the only dictionary that reports such important properties as whether a noun is an uncountable noun (like *information* or *bread*) or a countable noun (like *bits* or *computers*).

Diana Hacker, *A Writer's Reference* [Hac89]. A dull, boring, but very straightforward undergraduate reference grammar. Easy to use, clear examples, no mystification. Good for clearing up some of the fine points of punctuation and grammar.

Scott Brookie, *UNIX for Luddites* [Bro86]. You should already know most of the material in this manual, but it is a good example of documentation for naive users.

1.4 Also on reserve

Marie-Claire van Leunen, *A Handbook for Scholars* [van78]. A well-written book that is a real pleasure to read, despite the unpromising title.

Edmund Tufte, *The Visual Display of Quantitative Information* [Tuf83]. This is a superb book, but too expensive to require. The publisher has a strange pricing policy (no discount to bookstores), so this book is not often available in bookstores—you'll have to order it directly from the publisher if you want a copy. We will not spend as much time on the subject of graphical design as it deserves. We've been told that Computer Literacy in San Jose, and Stacey's in Palo Alto occasionally carry it. There is a companion volume *Envisioning Information*, which has much of the same material, but is aimed at a somewhat more general audience.

1.5 Other interesting books, not on reserve

The following books have been put on reserve in the past, but demand for them has not been high enough to justify limiting their circulation in this way.

Wilson Follett, *Modern American Usage* [Fol80]. This book explains words that are often misused. Unfortunately, it only covers words in general usage, not the jargon used by engineers and computer scientists.

Donald Knuth, *TeX, the Program* [Knu86]. You are not expected to read much of this book, as it is a complete source listing for the TeX document compiler. It is included as an example of what good in-program documentation looks like.

Knuth, Larrabee, and Roberts, *Mathematical Writing* [KLR88]. This report is based on a course [CS 209] of the same name given at Stanford University during Autumn quarter, 1987. We have deleted the mathematical writing exercise from the course this year (it is now given in CE/CIS 200), but those of you planning to go on to graduate school may want to learn something about mathematical writing anyway.

1.6 Homework Assignments

In previous years, the instructors distributed the workbook as `.dvi` and PostScript files, which some people had difficulty reading. In 1995, Kevin tried setting up World-Wide Web files, so that access would be easier, but the `latex2html` translator had difficulty with a number of the constructions in the LaTeX source code. He provided a table of contents for the assignments, and provided pointers to the `.dvi` and Postscript files that Mosaic or Netscape could follow. In Fall 1995, Carl Roth found a workaround for a bug in the PERL implementation, and Tracy Larrabee managed to get the entire workbook through `latex2html`. Let's hope it works again this quarter! Learning to use the World-Wide Web is a valuable skill for a computer engineer or scientist in any case.

To read the files on-line, use Mosaic (or Netscape or one of the other World-Wide Web browsers), and open the Uniform Resource Locator (URL) `http://www.cse.ucsc.edu/~karplus/`. This is the *home page* for Kevin Karplus, and contains pointers to several other documents, including the assignments for this course. If you have not used Netscape before, almost any of the CATS consultants in any of the computer labs can help you, because browsing the World-Wide Web has become a major form of recreation for the consultants.

Although it is possible to print directly from many World-Wide Web browsers, I'll also provide PostScript files generated directly from the LaTeX originals. These will generally produce a better looking printed copy that will take up less paper. There will be hypertext links (pointers) from the WWW pages to the PostScript files.

Please read each of the assignments carefully—don't rely on a vague memory of in-class discussion. Common problems that have come up with the assignments in the past are discussed in the assignment writeups, but still about half of the problems we see in turned in work are things that we specifically warn about.

1.7 Evaluation

CE185 has no mid-term or final exams. You will be judged about 80% on the papers you write, 10% on the oral presentation near the end of the quarter, and 10% on in-class work, participation in discussions (both in class and electronic) peer editing, and so forth. The final paper is about a quarter of the total weight for the papers (20% overall).

We will have no final exam, and Kevin has to be in St. Louis on the day the exam is scheduled. Everything **must** be turned in by the last day of class, Friday June 7.

1.8 Collaboration

Each paper you turn in must have the names of the authors prominently displayed at the beginning. Anyone caught using a term paper service or copying from books, journals, or fellow students will be punished as severely as the University allows. Flunking the course is an absolute minimum.

On some assignments (like the final project), we will encourage group authorship, and on others we will insist on single authors. If you are not sure which category an assignment falls into, please ask.

We encourage you to have someone else read your drafts, point out errors and unclear passages, and make suggestions, but not do re-writing for you. We will frequently use class time to exchange drafts of papers and discuss them in small groups.

We also encourage you to use the tutors who are assigned to the course. In the past, predictably, only the best students have made substantial use of the tutors. The tutors are students who have taken the course previously, and who know what we want you to learn, and how to help you do it.

This is a difficult course, but anyone who uses the resources we provide can pass it, as well as learn something worthwhile. A lot of you hate to write and think that you are not very good at it. We don't guarantee that you will learn to like to write, but we can guarantee that if you do what we ask, and work hard at it, that you will learn to write competently, and perhaps a good deal better.

The colleges also provide writing tutors, and we encourage you to seek their aid as well.

Anyone whose help you use (including the instructors, tutors, classmates, spouse, ...) should be acknowledged in the turned-in assignment. Formal reports should have an acknowledgement section, but other document styles usually need a separate cover memo to the instructors for acknowledgements—you should regard this cover memo as a standard part of anything you turn in, even if it is not specifically requested for an assignment. Of course, any books or journals you use as sources should be properly cited, and we intend to teach you how to do this, so that you do not plagiarize (copy without citing) unintentionally.

Claiming someone else's work as your own is the biggest academic sin.

If you are not certain about how much help is permitted, how much is encouraged, and how much will be considered cheating, please talk with us. You may be pleasantly surprised to find out that we allow more assistance than you thought.

1.9 Special guest lecturers

We are trying to arrange to have some guest lectures this quarter. As a courtesy to our guest lecturers, please make an extra effort to be on time for their lectures. We are hoping to schedule the following guest lectures:

Scott Brookie	Writing for neophytes	Wed, Apr 24
Catherine Soehner	Using the library	Fri, May 3
	What a tech writer needs from an engineer	

1.10 Schedule (due dates)

The following schedule lists the assignments for this quarter with due dates. If the due dates are changed, we will inform you in class. Note that many assignments overlap; this is the normal case for people who write for a living, as engineers do (30–50% of the work day).

You are required to turn in your early drafts, as well as the final copy—the early drafts allow us to see how well you edit and how well you respond to suggestions from your peers.

Save your assignments after we return them to you! We will ask you to turn them all in at the end of the quarter, so that we can write a fair evaluation of your work. Get a large envelope to keep all returned assignments in, so that you can turn in the entire portfolio at the end of the quarter.

assignment	draft(s) due	final due
1. job application and résumé	Apr 5	Apr 8
2. in-class hiring memo		Apr 8
3. describing an algorithm	Apr 15	Apr 17
4. in-program documentation	Apr 22	Apr 24
5. naive user documentation (e-mail manual)	May 1	May 6
6. library puzzle		May 10
7. survey article	May 17	May 20
8. term project proposal memo		May 20
9. term project document specs		May 24
10. term project progress report		May 28
11. term project	May 31	Jun 7
12. oral presentations		May 29, Jun 3, 5, 7

Note: no final exam is scheduled. All work must be completed and turned in by June 7.

1.11 Reading assignments

We have collected all the reading assignments scattered through this reader into the following schedule, and have included chapters to read that are not mentioned explicitly in the individual assignments. Several of the chapters are listed repeatedly—this does not mean you need to read them repeatedly, but that you should read them the first time they appear, and review the relevant parts for the later assignments. Note that this reading is in addition to reading the material in this workbook, which should be read in order, with each section read a week or two before the corresponding assignment is due.

It is probably a good idea to read ahead, since you wil have more time at the beginning of the quarter than the end of the quarter.

assignment	read before
[HO91, Chapters 1, 3, 7, and 10]	Apr 5
[HO91, Chapters 11 and 13]	Apr 8
[HO91, Chapter 27]	Apr 10
[HO91, Chapters 12, 14, 21]	Apr 12
[SJW79]	Apr 15
[HO91, Chapters 8, 9, 22, 23, 29, and 30]	Apr 17
[Bro86] [HO91, Chapters 3, 5, 9, 12, 17, 24, 25, and 26]	Apr 29
[HO91, Chapters 2, 4, 5, 12, 16, 18, 21, and 28]	May 13
[HO91, Chapter 16]	May 17
[HO91, Chapters 11 and 13]	May 24
[HO91, Chapters 15 and 19]	May 28

We have not scheduled the reading of Chapters 6, 20, and 31–38, but you should make time to read them. Non-native speakers are particularly encouraged to read Chapters 29–38 early in the quarter.

2. Intake Survey

2.1 Purpose of the intake survey

This assignment will not be graded or returned to you, but we'll discuss the survey results in the next class. The survey's purpose is two-fold: to give us some statistical information about the composition of the class and to let us know early who will need special help to get through the class.

2.2 Pre-requisite verification

If you do not have the pre-requisites for CMPE 185 (or if the registrar can't figure out that you have the pre-requisites), you have to get a permission code from the instructor in order to register. For our benefit, we'd also like to know exactly how everyone meets the requirements. Please answer these questions:

- How did you fulfill the Subject A (the high-school writing requirement) requirement? (test in high school, transfer credit from elsewhere, test at UCSC) If you haven't fulfilled the Subject A requirement, you may not take this class.
- How have you fulfilled the general-education code C requirement? (Normally fulfilled by taking Writing 1 at UCSC, by transfer credit, or by exam.)
- Do you have reading knowledge of Pascal or C? What verification can you offer? (Any programming course using either language is more than adequate proof.)
- Have you taken CMP 16 (Applied Discrete Math) or CMP 12B (second programming course)? If not, how can you show that you are familiar with induction and recursion?

2.3 Language background

Please write two paragraphs: one about your language background, and one about your writing experience and ability as you understand them. Self-knowledge about your own writing will be a constant theme in this class, and it begins here. Here are some questions to help you focus your paragraphs (don't answer these questions in order—write decently structured paragraphs):

What is your native language? (If Chinese, which dialect?) What other languages do you speak? What languages are spoken in your home and in the community where you grew up? If English is not your native language, how long have you been using English? Where did you learn English? How much of your formal education has been in English?

Do you like to write? What writing instruction have you had before? How much do you write for other courses, for work, or for pleasure? What process do you use when writing long reports? What kinds of writing do you do best? worst? What has been your experience with writing tutors? What kinds of improvement have you seen in your own writing in the past two or three years? If there has been no improvement that you know of, in what ways do you think your skills have gone downhill?

3. Job application letter and résumé

3.1 Goals—audience assessment and letter writing

This assignment has two major goals: learning to assess your audience, and learning to write and format an effective job application letter and résumé. Most of this assignment sheet describes the correct content and style for an effective application letter and résumé.

In preparation for this assignment, read Chapter 3 (Identifying Audiences and Purposes) and Chapter 10 (Résumés and Job Letters) in Huckin and Olsen [HO91]. This assignment is an exercise in assessing your audience, and finding the appropriate means to address it. These chapters are the background for the lecture and discussion related to this assignment, so read them, even if you think you already know how to do this kind of thing.

The point is not just a good letter and résumé, but, much more importantly, how to write to the right audience. The letter and résumé you will write are simply a practical exercise for learning something about the general problem of audience assessment.

Select a standard letter format that seems appropriate to you from the examples in Huckin and Olsen, keeping in mind the criteria and range of possibilities they suggest. The letter must be pleasing to the eye—not cramped and crowded-looking. It must be easy and quick to read.

Format is even more important in a résumé, because it has to present a lot of information in a easy-to-read form, leave the reader feeling good about the writer, and fit on one page. Again, Huckin and Olsen offer a range of examples here, designed for particular situations. Bear in mind that there is no standard formula, but a range of problems and possibilities, depending on the job you are applying for and your own background and experience.

3.2 Audience assessment

3.2.1 Mass-mailed résumés versus tailored job letters

Originally, a résumé was meant to be directed to no particular employer, unlike the job letter. However, as time has gone by, résumés have been designed more and more to persuade a particular employer. The basic principle is as follows: if you are sending your résumé with a job letter in application for a particular job which you know you really want and have a good chance for, then by all means design your résumé to appeal to that particular prospective employer.

There is another use for résumés: sending one with a letter of inquiry (an all-purpose job application letter), when you are writing to many companies in hopes of locating a job possibility. The rate of response in an operation like this is about like that of most direct mail solicitation: about 1%–3%, if you have a properly selected mailing list. Consequently, it is important not to invest very much time in sending out résumés blind, since the rate of return is very low.

The simplest solution to the low return rate (and in part the cause of it) is word-processing. With a computer, it is very simple to maintain a full, well-written résumé, and a nice, well-written letter of inquiry, and whenever you want to send a batch of résumés out, print what you need and send them. This doesn't involve much investment in time, except for the initial writing.

Once the full résumé is written you may do something else with it: make a copy and then edit and modify it to suit an application for a *particular* job. When applying for a particular job, always write a new job application letter, perhaps after consulting copies of old ones for ideas.

3.2.2 Finding an employment possibility

First, find an actual employment advertisement for a job that interests you and for which you are qualified (or will be when you graduate, if you are graduating this year) from the newspaper, a recruitment notice at the Career Planning Center, or other source.

- You must turn in a copy of the ad or notice with your letter.

3.2.3 Doing the background research

Next, find out something about the company. Significant data include the size, product or service offered, future plans, and so forth. Use personal contacts if you know someone who works for the company: otherwise use reference materials in the library. McHenry Library has directories that give information about local employers. Ask at the reference desk. Don't be shy about this! You may show them this assignment sheet if you don't know what to ask.

You may also call the Personnel Office of the company you are interested in and ask for information. Sometimes this works, and sometimes it doesn't (the company may be too small to have a Personnel Office), but it is often worth trying. Know what you want to ask before you call—write out questions if that will help you remember when you're nervous. Base your first questions on the job advertisement, and follow up with more general questions. Keep notes of what you found out, and where you found it.

- YOU MUST TURN THESE NOTES IN, TOO.

3.3 Writing process—letter writing

3.3.1 Getting started

Now you are in a position to start a draft of the letter. You might begin by making two short lists that answer the following questions:

- What do I have to offer them?
- What do they have to offer me?

Don't be alarmed if all you have to offer is a moderately good grade-point average from a decent university in the right major, and all they have to offer is a reasonable salary and a short commute—but try to pick a job that offers you more than that: experience in an area of your field that you are interested in. And even a short commute can be transformed into "wanting to work in your own community for a company that you already know and respect."

First, the general résumé: get together the necessary information. This includes dates, names and locations of companies you have worked for, organizations you have belonged to, colleges and universities you have attended, special honors or scholarships for academic achievement or merit, and so forth. You may want to include other categories, but if you do, don't call them "Other"—that makes them sound unimportant. You can call the category "Skills" or the like.

Having this information all together in a general résumé will be handy for you in the future; it is surprising how easy it is to forget or lose information of this kind. For those of you who will be needing security clearances from the government in the future, probably about half of you, it is important to keep a file containing every address you have ever lived at, as well as the particulars of all previous employment.

3.3.2 Doing this assignment

Consequently, for this assignment, you will actually be writing two résumés: the first one a full, general one that has everything in it that normally appears in résumés—education, job history, and so on—and the second one tailored to the specific job you are writing the job letter for.

3.3.3 Doing the working draft

Keeping in mind what you learned in class about writing job letters, try writing a draft of your letter. Be as specific and detailed, in relevant areas, as possible. Use your lists as a rough outline.

If it helps you to get organized, you may turn your lists into a more formal outline. Remember that you needn't tell your whole life story in the letter—the people who read these read many, many of them, and they appreciate people who stick to the point and only write about relevant topics. Unless they relate directly to the job, your hobbies are best left out. "I love sports ..." is not of interest to most employers you will be applying to.

Remember that in a job letter you are asking for an interview, not for a job. Normally no one will hire you sight unseen, on the basis of a letter and résumé.

When you've completed this working draft, set it aside for a few hours and do something else—leaving it overnight is best, if possible. The next day, look it over. You will probably find some things you want to change; perhaps you may have thought of things you want to add. Make these changes and additions right on your working draft. When typed, your letter should fit comfortably on one page. This is your *first draft*.

3.3.4 Experimenting with formatting

Use the writing of the general résumé as a chance to experiment with formatting: we will not be concerned with grading your general résumé so far as formatting goes. We are mostly concerned that it be a complete record of any information you might need to write any other résumé. Do organize it in reverse chronological order, the standard format for any résumé, since otherwise it will be difficult for you to use it to construct the résumé to go with your job letter. Educational history goes into reverse chronological order, as well as job history. If you are working on a word processor, print out a version of this résumé to look at.

3.3.5 Tailoring your résumé

This next step is the core of the résumé assignment. In this assignment, what matters is tailoring the information you have available for the particular job you are applying for. Refer to Chapter 10 in Huckin and Olsen [HO91] for further guidance in tailoring.

When you have a working draft of the tailored version, either print out a copy, if you're using a word processor, or else type it to see what it looks like, and modify it if necessary. Formatting is very important in this version. A well-done résumé should get you a job when it's upside-down. That means it should look sharp, as well as containing the relevant information. This version is the first draft of your résumé.

Do not use many fonts or particularly unusual ones. A well-tailored résumé is like a well-tailored business suit, it is subtly elegant, not garish.

3.3.6 Preparing for peer editing

Read *Testing and Revising* in the text [HO91, Chapter 7]. Bring your first draft of your letter, your tailored résumé, and your general résumé to class, so your classmates may read them.

BE SURE TO BRING THE JOB ADVERTISEMENT AND THE INFORMATION YOU COLLECTED.

3.4 Things to keep in mind for editing

Your letter, along with your résumé, is meant to persuade your prospective employer to grant you an interview. These are the main points your letter should conform to:

1. It should be neat.
2. It should contain specific information about you.
3. It should show that you know something about the company and the job.
4. It should sound confident, but not arrogant.

The first three generally give no problems if you have done your background research and are applying for a job suited to your education and experience. Point 4 is about tone. How do you set a confident tone in your writing? Briefly, give information about yourself in a brisk, straightforward way, without a lot of qualification: that is, do say things like, "I believe my education and experience suit me for the position," **not** "I feel I probably meet the qualifications for the position," and **not**, "I am the one and only candidate for the job."

3.4.1 Understanding *confidence* in its cultural context

There is another issue involved in writing a *confident* job letter, one of importance to students who have learned to write letters in other languages in their culture of origin. Displaying confidence in writing is largely a traditional American virtue, originally mostly a male virtue. It is linked with other virtues and values, such as ambition, mobility, and individuality. Other cultures and societies value different virtues in this context: explicitly mentioned sincerity and respect in many Asian cultures, for instance.

There is also a contrast between cultures in how these valued virtues are conveyed in writing. In many cultures, it is expected that one will say in so many words, for instance, "I am a sincere person and very hard working," because if you don't say it, your prospective employer will wonder if you know what you are getting into, and also if you were brought up properly. American culture, on the other hand, is very indirect and circumspect in a situation such as a job letter: one does not say right out something like "I am a very confident person." Rather, one is supposed to write something that will, through the tone of the writing, encourage the reader to conclude that you are confident.

3.4.2 Including a statement of an objective or goal

This is a purely optional item. If you use it, it serves as the first line of the résumé (after your name and address). State your future aspirations that might be seen as a reasonable progression from the position you are applying for. Keep this short—one sentence at most. Be specific and accurate; this is why it matters to know something about the job you are applying for and what it can lead to.

3.4.3 Including other categories

Hobbies, sports, and other extraneous activities: be very careful of this category—it is often irrelevant, and therefore a waste of the reader's time. When is a recreational pastime relevant? Suppose you are applying for a job at a firm in Colorado, and you are a dedicated mountain climber and hiker. Then it is relevant, because it will mean to your prospective employer that you will be happy in the area and likely to stay with the company. Don't, however, make it sound like your interest in climbing is the only reason that you are interested in the job.

3.5 The final draft

Rewrite your letter, making any changes you see as necessary, and incorporating any suggestions your classmates made which you feel work. Don't be afraid to make major changes. Editing is a vital part of writing. Editing may be major reconstructive surgery, or minor cosmetic changes. Make your letter the best it can be by making as many changes as you think it needs.

The final draft must be typed perfectly, just as you would prepare an actual letter to send for a real job application. You are selling yourself to this company, and this letter is your product packaging. Check your spelling carefully! (*Ad* is not the same as *add*, *greater than* is not the same as *greater then*, ...) Typos or any other sloppiness are a sign that you have not proofread carefully, giving the impression that you are careless. No one wants careless employees.

3.5.1 Deciding about length

One page is appropriate for your tailored résumé, unless you have already had a substantial career (10 or 20 years in the industry), and even then, try to keep it as short as possible.

3.5.2 Neatness counts

THERE SHOULD NOT BE A SINGLE TYPO IN YOUR LETTER OR YOUR RÉSUMÉ. It should be neat, have nice margins, and be printed on clean white or off-white paper (pink, blue, orange and so forth do draw attention—generally negative attention). We will accept résumés printed on a dot matrix printer; however be aware that employers strongly prefer to see letter-quality printing in a résumé, because it's so much easier to read.

There is no need to type the word *résumé* at the top—it's obvious from the format what it is—just start with your name.

3.5.3 Hints

Does the résumé list languages spoken besides English? With the current emphasis on global markets, knowledge of other languages can be a significant asset.

Does the application letter mention which issue of the newspaper or journal contained the advertisement being answered? Mis-understandings about what position you're applying for can cost you a job interview.

Professionals should probably **not** include typing speed on their résumés, to avoid being stereotyped as typists. Typing speed should only be included in résumés for jobs in which keyboard skills are important.

Are addresses in standard format for the United States (or for the country mentioned in the address)? In this country the standard format is

Personal Name
Company name
mail-stop
number street
city, ST zip code
USA

It is customary to put two or three spaces before the zip code, to make it easier for mail sorters to find, and to use the official two-letter Post Office abbreviations for state names.

Be consistent in your level of formality. *Thank you* is a neutral level, *Thank you very much* is more formal, *Thanks* is informal. If you mix them up, you get *Thanks very much*, which sounds insulting.

Watch the word choices! Change *utilize* to *use* and *due to* to *because*. Try to reduce the number of "noise" words in your writing—phrases like *the fact that* are immediately suspect. Most UNIX systems have a program called `diction` that checks for some commonly mis-used phrases. Similar programs exist for most personal computers.

One trap for both native and non-native speakers is the tendency for some verb-particle pairs to be separate when used as verbs, but combined when used as nouns. For example, you can *lay out* a *layout* or *set up* a *setup*. Try not to mix up the nouns and verbs (the mixup would be embarrassing).

Watch out for the suffix *-type* used as a separate word. You can't have an *IBM type PC*, but you can have an *IBM-type PC*. Even if you get the hyphenation right, the suffix *-type* is grossly overused—try to replace it (*IBM-compatible PC*). The word *type* is also over-used—you can often remove it without loss. For example, you can replace *what type of skills* by *what skills*.

Watch out for noun clusters! You can often make a an ugly string of nouns much more readable by grouping words together with hyphens. Huckin and Olsen mention this on page 659, but do not give a good heuristic for determining when to insert the hyphens. The simplest rule is to fully hyphenate any noun phrase that is used like an adjective as a modifier inside another noun phrase. For example, *computers with eight bits* are *8-bit computers*, *part-time workers* work *part time*, and *just-in-time delivery* is not a moment too soon. If you have more than one level of nesting, there is no way to make the noun phrase clear without extra information. For example, is *just-in-time inventory control* talking about technique for managing inventory so that it arrives at the right time or a control system that was finally delivered?

Watch the commas and semicolons! When you make a list of three or more items, separate them with commas, including a comma before the last conjunction. (Note: the British style for punctuation does not have a comma before the conjunction, but the American style is more consistent and allows conjunctions within list items, and so we prefer it.) If your list items include commas, you may have to switch to semicolons as list separators.

Here's a tiny bit of typographic lore that will help out those of you who are obsessed with details. There are four different symbols that are commonly typed with the `-` key of a typewriter.[1]

1. Most computer people think first of the *minus-sign*, used for indicating subtraction or negation (for example, $x - y$ or -1). This is conventionally typed with a single `-`.
2. The second most common symbol is the *hyphen*, used to join compound words together, and to indicate that a word continues across a line break. If you use a word processor, you should never type a hyphen to fix a bad line break, as subsequent editing or reformatting may move that hyphen to the middle of a line.
3. Somewhat longer than a hyphen is an *en-dash* (–), used to indicate ranges (for example, 1988–89 or pages 1–100). En-dashes are usually typed with one `-`, but most systems that let you change fonts support a separate en-dash. For example, in TeX [Knu84], you use `--` and in most Macintosh fonts you can use option-`-`.
4. The longest dash is the *em-dash*, which is used as punctuation between clauses of a sentence, to indicate sudden changes of thought. These dashes are usually typed as `--` with no surrounding spaces, but in TeX you use `---` and in Macintosh fonts you can use shift-option-`-`. Be careful not to use too many dashes—they make it look like you can't keep to one subject for very long.

See Section B.5.1 for more information about hyphens and dashes.

Even capitalization can make a difference. If you are applying for a job at Apple Computer, make sure you know the difference between a Macintosh (the computer) and a MacIntosh (a Scotsman from a particular family).

3.6 What to turn in

Turn in

- the announcement of job you are applying for,
- the notes you took in researching the job,
- the final draft of job application letter,
- the final draft of tailored résumé,
- a readable draft of general résumé,
- the first draft of job application letter, and
- the first draft of tailored résumé.

Please keep all assignments we return to you, so that you can turn them in together at the end of the quarter. We need your entire quarter's work in front of us when we write narrative evaluations.

[1] This section looks fine in the PostScript output, but terrible in HTML, since HTML does not support the different types of dashes.

4. Hiring Recommendation Memo

4.1 Goals—reading for content, writing memos

The purpose of this assignment is for you to put yourself on the other side of the job application process. Instead of acting as job applicant, you will pretend that you are the person who requested the new employee.

You have two goals. The first is to assess honestly the applicant's presentation of her or his qualifications for the position she or he is applying for. The second is to present your conclusions in a persuasive memo, either recommending that the applicant be granted an interview, or that the applicant be turned down.

Bear in mind that your memo is not a hiring recommendation but an interview recommendation. If you recommend an interview, the memo should be a guide to the interviewer, who may not be you, about what the candidate might be asked in the initial interview. The initial interview is designed to confirm relevant knowledge and experience for the position. You, the Senior Engineer, are the best judge of what those things are. In short, the reasons you give for interviewing the candidate are the things the candidate will be asked to talk about.

4.2 Audience assessment—Personnel Director

This memo will be addressed to the Personnel Director of the company (us) from the Senior Engineer (you). Write it as a memo with proper formatting [HO91, Chapters 12 and 13]. You will need the first four items of the normal heading, as well as the optional `Enclosure:`, since you will be attaching your memo to the application and résumé.

You must use your imagination for this assignment, putting yourself into the position of a Senior Engineer who has just received an application from someone you don't know. Therefore, even if the résumé you are reviewing is from your lifelong best friend, don't refer to her or him by first name! For this assignment, you don't know each other.

4.3 Writing process—persuasive writing

4.3.1 How to go about persuading the personnel director

First, decide whether to grant an interview for the position. Try to be as realistic as possible in your evaluation; base your decision on a careful study of the job advertisement, on how closely the applicant's qualifications match, and on how well the job letter and résumé are written. Don't worry if you want to recommend that the applicant not be granted an interview: you will turn this memo directly in to us, and he or she need never see it. (Of course, most students do a good enough job of selecting job openings that they are well-qualified for the job—don't be unreasonably picky.)

Start by saying whether you are recommending that this applicant be interviewed for the position or not—the most important information should always appear at the beginning of a memo, and of most technical documents. The applicant's name should appear immediately, both in the *Subject* line and in the first sentence of the body.

If you decide the person deserves an interview, persuade your personnel director to call her/him in. The tone can be very formal: "We should grant an interview to Joseph P. Undergraduate for the position of technician in underground wiring." Or you can be very casual: "We've got a live one for digging ditches! Check out Joe Undergrad." You'll probably be most effective if you match your style to the Personnel Director's tastes. If you are not sure, it is better to be a bit too formal.

Remember, you will be attaching your memo to the applicant's letter and résumé and sending the packet to the Personnel Director. The first thing she will read is your memo. She will know nothing about the applicant, other than what you tell her. Therefore, be specific and detailed. Don't say something like "Her summer at Lockheed qualifies her . . . ." Say rather, "She designed voltmeters at Lockheed, which qualifies her to work on . . . ."

Also, make it clear what position the applicant is applying for—the company may have more than one position open. Present your applicant's strongest qualifications first—remember, you are selling this person to the director. If you have reservations ("This person's qualifications are excellent, but her résumé was sloppy, so she may be a careless worker . . . ") put them last. Emphasize the positive.

If you decide to recommend that the person be turned down, state all the negative things about the application first. Then, if you are unsure, and want to say some positive things, stick them at the end. Remember, the first impression of the person reading your memo will be based on what you say first. It's very startling to read a memo that begins with positive statements about an applicant, and ends with a recommendation not to interview.

If you want to soften the negative tone, you can start with something like "Although Ima Genius has excellent grades, she has no experience or training in computer engineering . . . ." The word *although* warns the reader that what follows is not evidence for the case you are making.

Remember that you are only recommending this person for an interview, not for the job itself. Don't say "We should hire Joe Wong as . . . ."

4.3.2 Memo format versus letter format

Memos are usually formatted differently from letters—mainly because doing so is traditional, but also to keep people from accidentally mailing out internal memos.

One difference is that memos usually do not indent anything—everything starts at the left margin. However, many businesses have adopted a "block" style for their letters, which also has no indenting.

Memos usually have four or five fields at the beginning:

To: The name and mail-stop of the recipient.

From: The name and mail-stop of the sender.

Date: The date and, often, the time the memo was sent.

Subject: A concise summary of the memo (for example, *Interview John Doe as a tech writer*). Making a subject line both concise and specific is an art—practice it!

Don't assume that the reader has read the subject line. Repeat anything important in the first sentence or two of the body of the memo.

Re: *Re* is Latin for *back*, and is used to refer to a previous document or file. Some people mistakenly use *Re:* for *Subject:*—it may be used in addition to a subject line, but should not replace it. For example,

Subject: Interview Jane Smith for project manager
Re: Job opening 89-103, application 89-1139

The way the UNIX mail program builds *Subject:* lines for replies contributes to the misuse of *Re:*.

Cc: A list of additional recipients. The letters stand for *carbon copy*, even though almost everyone uses photocopies. On a letter the list of names goes at the end, but memos put everything at the beginning.

Enclosure: A list of attached documents. You should have this filed for your memo, as you will be attaching a résumé and job application letter.

You must include *From*, *To*, *Date*, and *Subject* fields, but *Re*, *Cc*, and *Enclosure* fields are only included when they are needed.

No salutation or other greeting is needed in a memo—get to business right away. Similarly, no closing is needed—say what you need to and stop.

In many businesses, it is common to initial memos next to your name in the *From* field, rather than signing at the bottom. Either place is OK, but you must initial or sign a memo, to show that it is the final draft, and not an earlier one that needed to be changed. Dr. Karplus prefers signing at the bottom, to indicate that the memo has ended, and that there are no more pages, but Mr. Scripture prefers the more traditional initials by the *From* field.

4.3.3 Writing in class

You will write this memo in class and turn it in immediately. Plan what you want to write, make yourself an outline if you like, then write the memo as neatly as possible. You may make a second draft if you have time.

The only preparation you should do ahead of time is to read this assignment and the appropriate chapters of the book, and have your own job letter and résumé ready for others to read. Do not try to write this memo before class!

Remember that all assignments that we return to you should be kept and turned back in at the end of the quarter. We use our comments on the entire quarter's work as a basis for the narrative evaluation.

5. Electronic mail and newsgroups

5.1 New forms of communication—new writing styles

Electronic mail (e-mail) is a new form of writing that has appeared in the last twenty years. E-mail has become the preferred form for communications for many of us who would previously have used written memorandums or the telephone. A new form of informal publication, newsgroups, is closely related. We will be using e-mail and the newsgroup `ucsc.class.cmpe185` extensively this quarter.

Because it offers easier response and quicker interchanges than traditional hardcopy mail, e-mail tends to be less formal and more spontaneous. Items that are not important enough to justify writing a letter can be quickly sent with e-mail, and quick answers are also possible.

Electronic mail has the advantage over the telephone of not requiring both participants to be available at the same time. Also, with e-mail it is possible to edit one's replies and present them in a coherent fashion, not getting talked into things simply because one can't marshal one's words fast enough.

Because electronic mail is a new medium, there is no consensus on the correct writing style for it. Some people view it as essentially identical to hardcopy mail—only appropriate for fairly formal, well-thought out communications. Others see it as equivalent to chatting in the hall—a quick way to communicate something informally, without the formality imposed by more traditional writing. Still others go even farther, being more blunt than they would be in person, on the phone, or in more formal writing.

Because you will be spending much of your careers reading and responding to e-mail, you should spend some time thinking about which ways of using it are appropriate and which are inappropriate.

5.2 Flaming

One common occurrence in e-mail and newsgroup discussions is *flaming*—verbal attacks on a person who makes an argument you disagree with. It is very easy to get angry at someone who makes a statement that is stupid or hateful, and that anger often causes people to write rude replies. These attacks are generally counterproductive—they antagonize the person they are directed at and often irritate others who are participating in the same discussion.

When you read a statement you disagree with strongly, feel free to write a reply, but don't send it immediately. Give yourself a day to cool down, then re-read your reply. You might find that it is rude and unpleasant to read. Re-write it, removing snide comments and *ad hominem* attacks, adding coherent arguments that bolster your side of the matter. Try to concentrate on what the person *wrote*, and not on the person. Also decide whether the reply needs to be posted so that everyone can see it, or whether it should be sent privately to the recipient.

Another useful thing to do when editing an e-mail or news posting is to try to cut it in half. Long rambling messages generally get skipped or read very hastily, while messages that are short and to the point get remembered. Don't forget your paragraph breaks—you can get away with much longer messages if they look well-organized.

Don't go so far in trimming your writing that your message is incomprehensible out of context. When replying to someone else's message, remember that the person you are mailing to may not have a copy of the original message. Don't quote long messages either—just pull out the salient parts that need to be quoted to establish context.

One of the joys of e-mail and newsgroup discussions is that you don't have to respond immediately—you can take the time to think up the arguments and examples that make your case stronger. A discussion that would last minutes face-to-face may take days electronically, but the issues are likely to have been examined more thoroughly, and there is a better chance of persuading someone to a rational conclusion.

5.3 Humor

Humor, particularly irony, is extremely difficult to convey in writing. In fact, humorous writing is one of the most difficult kinds of writing—good humor writers are even rarer than good stand-up comedians.

If you make jokes on a newsgroup, or in widely distributed e-mail, nearly always someone reading your joke takes it seriously, and starts trying to correct you. There are several ways to avoid this problem:

- Never make jokes. This works, but can be boring. Often a humorous way of presenting an argument is more memorable, and has more lasting effect than a carefully reasoned one.
- Make all your jokes innocuous. Try not to insult anyone when you make a joke—if no one is offended, then it might not matter much if someone interprets your joke as a serious statement.
- Be careful that you don't joke about anything serious. (Anybody remember Reagan's gaffe, when he jokingly threatened to start bombing Moscow on a microphone that he didn't realize was live?)
- If you do make a joke, direct it at what has been said, not the person who said it. This means that clever puns and wordplay are better than simple abuse of another writer. Wit is also more difficult than insults and therefore funnier. Directing an insult at yourself is not sufficient—in many cases you will be inadvertently insulting others as well.
- Mark all your humor explicitly. One common punctuation device that has been adopted by many e-mail users is the *smiley face*. You make it with three keystrokes "`:-)`" and put it at the end of any sentence or phrase that you do not want people to take seriously. Some people have gone overboard, and proposed dozens of different variants on the smiley face for different purposes—so far, none of the others has become as widespread. The smiley face may seem stupid (and it is never used in normal writing), but it has become such a standard convention for e-mail and newsgroups that many readers assume that any statement not marked with a smiley is intended seriously.

5.4 Assignment—newsgroup discussion

We have set up an electronic conference in the form of a UNIX newsgroup local to the campus (`ucsc.class.cmpe185`). The newsgroup can be read with `nn ucsc.class.cmpe185` (or `xrn` on an X-windows display). Please read the manual pages (`man nn`) for more information. Other newsreaders (`rn`, `trn`, and so on) also exist, and you may read the news with whichever you prefer.

We got the idea for the quarter-long conference from some instructors at Michigan Tech [CS90]. They thought it worked well enough to be worth a 23-page journal article.

Good e-mail and newsgroup style requires practice, preferably in an environment where it won't matter much if you make mistakes. A major purpose of this conference is to provide that environment. We will not be grading your writing in the newsgroup! Feel free to be yourself, and don't worry too much about what we'll think of your newsgroup writing.

Another purpose of the newsgroup conference is to have the discussions that are difficult to have face-to-face in a large classroom. As teachers, we are well aware that there are a lot of things not mentioned in class. Even when we stress that we don't mind students having different opinions, and would enjoy having a good discussion, there are still many good reasons why students don't argue.

Using one's authority as a teacher to get students to disagree is a challenge that verges on the paradoxical. We will be reading the newsgroup and participating in the discussions, but as colleagues, not as teachers. Unless remarks are directed specifically at us (like requests for clarifications of assignments), we'll try to keep quiet and listen, so that the newsgroup discussion is truly a student discussion. If the discussions get really interesting, we may not be able to refrain from tossing in our opinions as well.

In a newsgroup discussion, you get a chance to see what several people have said, think carefully about their arguments, and then present your own ideas without fear of interruption. This means that even those students who rarely get a chance to speak in class will be able to participate fully.

Anything we talk about in class is fair game for the conference. If you disagree with something we say, or don't get a chance to make a coherent argument in class, bring it up at the conference! This is also a good forum for complaints about the readings, the assignments, the tutors, the instructors, or anything else related to the class. Cooper and Selfe compare this sort of communication to the whispered remarks between audience members at conference presentations and lectures [CS90, p. 848]. A better comparison might be to the conversations that occur in the coffee breaks between the presentations—the criticisms may be just as harsh, but are often more politely phrased, and more fully supported.

It is perfectly acceptable to use the newsgroup for requests (like asking for someone to read and comment on a draft, or trying to find out who borrowed one of the reserve materials), but personal messages should be sent by e-mail, not to the whole class.

The topic of discussion will vary from time to time, and we may inject new topics when it looks like the class hasn't had anything to talk about for awhile. For example, we never seem to have enough time to talk about practical ethics—what to do when a supervisor asks you to do something sleazy or illegal, how to avoid situations where you'll be asked to compromise your principles, and so forth. We may, for example, quote one of the ethics problems presented in *IEEE Spectrum*, or one that has come up for a former student, and ask your opinions about how to handle it.

Everyone in the class is expected to participate in the conference on a regular basis. Even if you have just been listening in, and don't have much to say, you should post a line or two to let us know you are there. This is not an onerous task—we're looking for short (one- or two-paragraph) messages that make it clear that you have been reading and thinking about the topics under discussion. The goal is to learn to argue persuasively in the electronic mail and newsgroup format.

E-mail does not require the same standards of polish as a final draft of a paper, but it is still a good idea to re-read your replies and run them through a spelling checker before sending them—people give little weight to arguments that are badly written or mis-spelled.

6. Algorithm Description

6.1 Goals—multiple audiences, graphics, sophisticated audiences

This assignment practices

- writing to a technically sophisticated audience,
- writing to two different audiences at the same time,
- describing a recursive algorithm, and
- proper use of displays and figures.

6.2 Audience assessment—writing for multiple audiences

The Dinky Database Dantai[1] has had complaints from their customers about the slowness of the zip code sorting operation. You have been hired as a consultant to diagnose the problem. They can't afford your consulting fees for the time it would take you to code and debug the routines yourself, so you have to prepare a report to their programmers explaining the fixes to be done. You cannot talk with the programmers who wrote the code, because they are all working for Bit Bucket Enterprises now. The new DDD programmers are mainly entry-level programmers (a euphemism for high school and junior college students working in their spare time).

On examining the code, you find that

- zip codes are stored alphabetically as variable-length character strings,
- all sorting is internal (all zip codes read into memory before sorting),
- the sorting is done by a variation of bubble sort (an order n^2 algorithm),
- comparisons are done by passing the pointers to the two strings to a function, and
- exchanges are done by swapping two pointers in an array, not by copying entire strings.

Because Canadian and other foreign addresses use non-numeric mail routing codes, and even US codes vary in length, you decide that the data representation is appropriate, and that alphabetical order is the correct ordering for zip codes (00002-4455 is very close to 00002, not to 24455). The comparison and exchange operations seem to be correctly written, with only minor gains in efficiency possible from re-implementing them more carefully.

After checking with DDD, you find that customers have not complained about the number of entries they can sort, only about how long it takes. This means that internal sorting is a reasonable choice, and the extra complexity and speed penalty for external sorting is not justifiable.

You cannot find any reason for using bubble sort, and decide to recommend a faster sorting technique (quicksort, heapsort, merge sort, shell sort, radix sort, ...). You have to explain briefly why you are recommending a change of algorithms, then give a detailed description that the programmers can understand. You do not have to give a full theoretical analysis of the algorithm, but a few sentences to explain why you chose the algorithm you did would be helpful.

Your report will be read by two audiences: the managers of DDD and the programmers. The managers have no formal training in computer science, but want to know that you've solved their problem. The programmers know a little computer science, but are not familiar with sorting algorithms. The programmers may know as much as you do, but because they can't write nice reports, they get sweatshop wages while you earn consultant rates.

---

[1] *Dantai* is the Japanese word for a private corporation.

6.3 Writing process—algorithm description

Your first task is to choose an algorithm. You may already know one that is good enough. If not, Knuth's *Volume 3—Sorting and Searching* [Knu73] should provide the information you need. Of course, you don't want to read all of Volume 3 before making up your mind. If you don't have a copy of Knuth, almost any sophomore or junior data structures book will describe a few of the more important sorting algorithms. You may want to check the serial published by the ACM containing nothing but algorithms for various problems [ACM].

Choose your sorting algorithm to be fast and to require minimal changes to the data structures in the existing program. You know that they use bubble sort, and so the information is stored in an array, with comparison and exchange operations available. If your algorithm requires some other data structure, you will have to describe how to build that data structure as well. For example, merge sort may be somewhat easier to describe than quicksort, and less subject to "off-by-one" errors, but the simplest implementation requires linked lists, which would increase the memory requirements. If you do describe merge sort, be sure to include the appropriate information about error-checking and pointer manipulation: in the past, many students have used `x->next->next` in their pseudo-code without warnings about odd list lengths and checking for null pointers.

When describing an algorithm, you have to pay careful attention to the level of abstraction. You do not want to describe everything in gory detail—that's the programmer's job. You also don't want to treat the whole problem as a single black box either. If you knew the programmers were highly skilled, it might be enough to tell them "Use quicksort!", but for this crew you'll need more detail than that.

The main parts of an algorithm description are

- purpose (what is the result of running the algorithm),
- data structure (what is manipulated by the algorithm),
- technique (what steps does the algorithm perform),
- justification (proof of correctness, often reduced to hand-waving),
- analysis (speed, space, cost, ...).

Different audiences and different purposes demand attention to different aspects of the algorithm description. When presenting a new algorithm for a well-known problem to the research community, the emphasis is usually on technique, justification, and analysis. In that situation, you're trying to show how much better you've done than previous researchers. When presenting a new data structure (say a better representation of a priority queue), the data structure and analysis might be stressed.

For this report, you will be presenting a well-known algorithm as the solution to a specific problem. Here data structure and technique are most important. You can point to the literature for a proof of correctness and the details of the analysis.

Some of the tools used for proving correctness are also useful for explaining the technique, so don't ignore them entirely. For example, the termination condition for a loop needs to be known both for the justification and for the coding. Loop invariants (statements that describe a property that is unchanged as the loop is repeated) are essential for correctness proofs, and handy for generating the loop in the first place. David Gries gives a detailed presentation on using loop invariants to help write programs [Gri83].

6.4 Figures and displays

A manager reading your report will look at the executive summary and at the figures, to see what the report is about and whether it is worth passing on to the programmers who'll read the complete report. It should be possible to get a fair idea of the content of the report just from looking at the figures and reading the captions. Look at the articles in *Scientific American* for an excellent example of how figure captions can be used to convey most of the substance of a technical article.

You have several opportunities for figures in this report—you can use them for the pseudocode for the algorithm, for showing the change in the data structure as the sorting progresses, for displaying the relative speed of different algorithms as a function of the number of items to sort, and so forth.

There are two ways to insert non-textual material into a report: *figures* and *displays*.

A figure can appear anywhere in the report, and usually appears on the same page as, or slightly after, the first reference to it. Figures are numbered and have a caption explaining what the figure is illustrating. Each figure must be referred to somewhere in the text, either as part of a sentence, like this: "Figure 37 illustrates the relationship between ..."; or as a parenthetical remark, like this: "(See Figure 38.)". Remember to capitalize the word "figure" when it is used as part of the name for a particular figure.

A display differs from a figure in that it is inserted in a fixed place, and is not allowed to "float" to a different place in the text. Displays are used for math formulas, very small program fragments, and quotations that are too long just to put in quotes in the middle of a paragraph. Some empty space is usually inserted before and after each display to set it off from the surrounding text, and displays are often centered on the page. Mathematical formulæ are usually inserted as displays, and are grammatically part of the sentence before them. For example, I can define the golden ratio as the positive solution to the quadratic equation

$$x^2 = x + 1 \ ,$$

and continue writing the same sentence.

6.4.1 Graphics

Pictures are often the clearest way to explain data structures, particularly when pointers are required. Whenever a picture is used, it should be a numbered, captioned figure. Figures should be (nearly) comprehensible without the accompanying text. Make sure the caption explains what the figure means. Something like "Figure 3. Step 2 of the algorithm." is nearly useless. A better caption might be "Figure 3. Swapping out-of-order data elements pointed to by i and j."

The pictures you are likely to draw are block diagrams, symbolic representation of pointer structures, maps of contiguous sections of memory, and graphs showing how something (time, cost, current, voltage, ...) varies when changing various parameters. The rest of this section will talk only about numerical graphs, though you probably will need to generate other sorts of graphics, even for this assignment.

Proper treatment of graphics is really beyond the scope of this class. A fairly good introduction is included in Chapters 8 and 9 of Huckin and Olsen [HO91, 137–184]. If you want a comprehensive treatment of presenting numerical (mainly statistical) data in a professional form, the best book is Tufte's *The Visual Display of Quantitative Information* [Tuf83] on reserve in the Science Library. Other books can be found under subject headings like "Engineering Graphics," "Graphic Methods," and "Computer Graphics." For examples of how not to present numerical information, see almost any mass-market magazine or newspaper. *USA Today* is particularly good at distorting tiny data sets into fancy pictures that hide the data. Another particularly good collection of bad examples can be found in a book by *Time Magazine* illustrator Nigel Holmes [Hol84], although they are presented as if they were the best way to illustrate data.

Tufte summarizes his theory of data graphics as follows [Tuf83, 105]:

> Five principles in the theory of data graphics produce substantial changes in graphical design. The principles apply to many graphics and yield a series of design options through cycles of graphical revision and editing.
>
> - Above all else, show the data.
> - Maximize the data-ink ratio.
> - Erase non-data ink.
> - Erase redundant data ink.

- Revise and edit.

Tufte also has some strong comments about "chartjunk", the cluttering of graphs with non-informative decorations [Tuf83, 121]:

> Chartjunk does not achieve the goals of its propagators. The overwhelming fact of data graphics is that they stand or fall on their content, gracefully displayed. Graphics do not become attractive and interesting through the addition of ornamental hatching and false perspective to a few bars. Chartjunk can turn bores into disasters, but it can never rescue a thin data set. The best designs (for example, Minard on Napolean in Russia, Marey's graphical train schedule, the cancer maps, the *Times* weather history of New York, the chronicle of the annual adventures of the Japanese beetle, the new view of the galaxies) are *intriguing and curiosity-provoking*, drawing the viewer into the wonder of the data, sometimes by narrative power, sometimes by immense detail, and sometimes by elegant presentation of simple but interesting data. But no information, no sense of discovery, no wonder, no substance is generated by chartjunk.
>
> Forgo chartjunk, including
> Moiré vibration,
> the grid, and the duck.

One problem that has appeared repeatedly is that students have given us plots of n^2 and $n \lg n$ for n ranging from 1 to 10, to show the advantages of one of the $O(n \lg n)$ algorithms over bubble sort. There are two flaws: internal sorting of 10 elements is so fast that no one cares which algorithm is used, and for such short lists some of the $O(n^2)$ algorithms are faster than the $O(n \lg n)$ ones. If the plots showed from 100 to 10,000 records being sorted, the graph would be both more meaningful to the reader and more honest.

Also, the graphs are often misleading, in that they are labeled with seconds or some other inappropriate unit. If you had done some real speed tests, then seconds would be the right unit, but if you have just done a theoretical analysis, then it is better to be more explicit about what you are plotting—in most analyses the number would be the number of comparisons or the number of exchanges done. If you want to provide a good graph, you might look up the formula for the expected number of comparisons actually done by each algorithm. The best place to find accurate formulas for the standard sorting algorithms is Knuth's *Sorting and Searching* [Knu73].

Here are some quick hints for creating usable graphs:

- Make sure that each graph has a single, clear purpose, and use the caption to point out what the reader is supposed to notice.
- Label both coordinate axes.
- Do not print a grid, but do put tick marks on the axes.
- Don't clutter up the graph with cute pictures.
- Make sure the ranges of the axes are appropriate.

6.4.2 Pseudo-code

Most of you have seen flowcharts for describing techniques. Although touted as a cure-all to programming woes in the 1960's, they have essentially disappeared from professional programming. They encouraged unstructured "spaghetti" code, and didn't help much in understanding the resulting mess. The small boxes in flow charts give far too low an information density, using lots of paper to say very little.

Instead of flowcharts, most algorithms are now described using pseudo-code. The pseudo-code syntax can be based on any structured language (Algol, Pascal, C, Ada, ...), but should be readable by someone familiar only with a different language. Tests and statements are not written in full detail, instead the intent of the statement or test is given. For example, pseudo-code for finding the minimum of an array might look like

```
min← ∞
for each element of array
        if element<min then min←element
```

Pseudo-code should be properly indented and either displayed like the example above, or put into a figure with a figure number and caption. Figures are generally easier for magazine and book typesetters to handle, and are essential for big chunks of pseudo-code, but displays are easier to read for small pieces of code.

If you do put the pseudo-code in a figure, try to arrange the layout so that the figure comes after the explanation has started. It is unfair to the reader to dump a big chunk of unexplained code in front of him or her. It can be very frustrating to slog through the code only to find an explanation after you turn the page.

6.5 Explaining recursion

Many students choose to describe a recursive algorithm, such as quicksort, for this assignment. Here are some hints on how to describe recursive algorithms so that they are comprehensible to someone who doesn't already know the algorithm, and who may not be entirely comfortable with recursion.

- First describe the outside view of the recursive function when viewed as a "black box" whose internal operations are not visible. For example, when describing a recursive procedure that sorts an array, say something like

  > The procedure `sort(arr,low,high)` sorts the elements of the array `arr[low]` through `arr[high]` into increasing order.

- Describe how the problem is subdivided and recombined in fairly general terms. For example,

  > Sorting is accomplished by dividing the sequence to be sorted into two smaller sequences, sorting each sequence independently, then merging the two sorted subsequences.

  Note that the above description could apply equally well to quicksort or merge sort. The differences between the two algorithms come in how the partitioning and merging are done. Merge sort uses a trivial partition and a more complex merging operation, while quicksort uses a fairly complex partitioning operation and a trivial merge.

- Describe the boundary conditions that cause the recursion to stop.

  > A sequence with only a single element is already sorted, so no partitioning and merging are necessary. Fast variants of divide-and-conquer sorting algorithms often use insertion sorting when the sequence has fewer than five or six elements, as the overhead for recursion is quite high on conventional machines.

- Describe the details of the algorithm after giving the overview. Only after you've described the divide-and-conquer sorting strategy should you start giving the details that distinguish between merge sort and quicksort, and only after that should you start distinguishing between the different variants of quicksort. Here is the place to describe the importance of good pivot selection in quicksort, and to describe which of the many partitioning strategies you recommend.

- Do **not** attempt to walk through an execution of the code without explaining the structure. It is tempting to describe what happens in chronological order, but you'll lose the readers very quickly if they have to keep a six-deep recursion stack in their heads. Iterative algorithms are often best explained in chronological order, but recursive algorithms are not. If you are explaining quicksort, you may find it helpful to illustrate the partitioning step (which is iterative, not recursive) with pictures that show what operations happen, and in what order.

6.6 Titles, title pages, and executive summaries

When you prepare a formal report, it is worth taking a little extra care to make sure that the report will reach the intended audience, and that they will decide to read it. There are several things you can do to make it more likely that your report ends up in the right place:

- Use an informative title. If your report is titled *Quicksort* or *A report on certain problems within a database company*, no one will have any idea who to send the report to, where to file it, or whether it is worth reading. A title that a secretary or an executive can understand is much more valuable: *Speeding-up DDD's zip-code sorting by using quicksort.*
- Use a title page. Put the title and your name on a separate page from the body of the report. Not only does this look more professional than cramming everything into the body of the report, but it makes it easier for a busy executive to sort the reports he or she has to read. You can put your address and phone number on the title page also, so that anyone who wants to hire you again as a contractor can find you easily.
- Put an executive summary on a separate page from the body of the report. If the summary is short enough, it can go on the title page. The summary should tell the executives all they need to know about the report: what the problem is, what the solution is, and how much work will be needed to implement the solution, in just a few paragraphs. Remember that the executive may be handling dozens of reports for different problems in different products, and so a concise summary of the problem is as important as the solution.

6.7 Things to keep in mind for peer editing

6.7.1 About the algorithm

- This is a writing class, not an analysis of algorithms class. We don't really care which of the many decent sorting algorithms you choose. If you know something about sorting, there are a few obvious choices—quicksort, heapsort, and radix sort. Your explanation of the algorithm is far more important than which algorithm you choose. If you're clever, you'll choose an algorithm that is easy to explain, rather than the one that has the best asymptotic performance.
- When you read your partner's draft, check for bugs in the algorithm. Is the termination condition clearly and correctly specified? Does s/he explain how to be sure the program will not bomb when sorting zero entries? How are null strings handled? Could you take the description and write a procedure from it? Would the resulting procedure work?
- Particularly watch out for abuse of "big-O" notation. Many of you have just learned about asymptotic analysis, and tend to get carried away with its notations. The notation $f(n) = O(g(n))$ is badly designed—it is **not** an equality. Saying that $f(n)$ is $O(g(n))$ asserts a property of the function f, but does not give you a right-hand side that you can substitute for $f(n)$. The property is simply that, for sufficiently large n, $f(n)/g(n)$ remains bounded by some constant. Because the constants are all unspecified, it makes no sense to talk about $O(1.5n)$ or $O(n+5)$—both are just $O(n)$.

  Frequently, students misinterpret a statement that an algorithm takes $O(n^2)$ steps to mean that it takes exactly n^2. You can't claim that switching from an $O(n^2)$ algorithm to an $O(n \lg n)$ algorithm will give you a 100-fold speedup for $n > 1000$, because you don't know what the multiplicative constants are. The only way you can make such strong claims is to do a more detailed analysis of the algorithms.

  If you look in Volume 3 of Knuth [Knu73, 107–110], you'll find that bubble-sort requires at most $\frac{1}{2}(n^2 - n)$ comparisons and $\frac{1}{2}(n^2 - n)$ exchanges to sort n items, and that on the average it takes $\frac{1}{4}(n^2 - n)$ exchanges and $\frac{1}{2}(n^2 - n \ln n - O(n))$ comparisons. All of these numbers are $O(n^2)$, but they do differ significantly. Two popular algorithms, heapsort and quicksort, are both $O(n \log n)$ on the average, but heapsort is guaranteed to have fewer than $n\lfloor \log_2 n \rfloor$ comparisons [Knu73, 149], while quicksort could take as many comparisons as bubble sort. On random data, quicksort is usually the fastest sorting algorithm, but on non-random data it can be slower than bubble sort!
- You should also check the verbal translations of the big-O order notation. For example, "grows exponentially" has a specific technical meaning. The formula 2^n grows exponentially, but n^2 does not—n^2 grows quadratically.

- When you report the relative speeds of different algorithms, know what it is you are reporting! In most analyses of sorting routines, the number of comparisons or the number of exchanges is counted. Given the description of the problem, which is likely to be more important?
- When you cite further sources for an algorithm, make sure that the source you cite uses the same variant of the algorithm that you do. This is particularly important for algorithms like quicksort, where every author picks a slightly different variant. For quicksort, the main variations are in how the pivot element is chosen and in how the partitioning is done. The average behavior is $O(n \log n)$ for all the variants, but some of the variants are much easier to program or have less probable worst cases.

  If you have trouble understanding one author's version of quicksort, check another's. But, please, don't get two different versions mixed together! Warning: many authors pick a pivot for quicksort that gives the worst possible behavior with an already sorted list. Because zip code lists are likely to be re-sorted after a few new entries are added, you want to be sure that the algorithm you recommend will be fast even if the list is already sorted.

6.7.2 Mechanical details

- Check your partner's use of articles. Particularly look for articles with names. For example, "quicksort" is the name of a sorting algorithm, so is used without an article. There is a substantial difference between implementing quicksort and implementing a quick sort.

  If you are having trouble with definite and indefinite articles, read Chapters 29 and 30 of [HO91]. Particularly watch out for the word *data*; it used to be a plural noun (the plural of *datum*), but has come to be an uncountable one in American usage. Under no circumstances is it a singular, countable noun, so it can never be used with the indefinite article.
- Check your partner's word choice for accuracy. For example, one popular phrase for this assignment is the oxymoron *slow speed*, which should be replaced with a phrase such as *slowness* or *low speed*. Another meaningless phrase that has been popular is *fast time*, which could have been replaced by *short time*, *little time*, *rapidly*, or *quickly*, depending on the context. You could also use more technical phrases such as *a shorter response time* or *better throughput*, but there is no point to being over-specific.
- Watch out for the back-formation *recurse* from *recursion*. The noun *recursion* comes from the verb *recur*. To recurse would be to swear at someone again. Verbs that add *-ation* to from nouns sometimes get corrupted by back-formation. The verbs that form *permutation* and *formation* are *permute* and *form*, not *permutate* and *formate*. Nouns with irregular plurals sometimes get corrupted also—the singular of *vertices* is *vertex*.
- Check your partner's use of type styles carefully. For example, the names of variables and procedures are often done in a fixed-width typewriter font (such as Courier). If such a convention is used, it should be used consistently.
- Another common convention is to italicize important words when they are being defined. (Use underlining if you do not have italics on your typewriter or printer—do not use both underlining and italics.) For example,

  > A *divide-and-conquer* sorting algorithm has three steps: *partitioning* the sequence into two shorter sequences, recursively sorting each subsequence, and *merging* the two sorted subsequences to form a complete sorted sequence.

  Italicizing words tells the reader that you believe the word is important, that the particular use you are making of the word may be unfamiliar to the reader, and that you are clarifying it here. Do not italicize jargon words every time you use them, but *only* when you are defining them. Italics are also used for foreign words that have not yet been Anglicized, and for general emphasis. In some documents, italics are also used for comments on program fragments, to make it clear that the comments are not part of the program itself.
- If you use the same typographic convention with many different meanings, it ceases to be helpful to the reader, so try to pick just a few related meanings for each typographic convention you use.

6.8 The final draft

Your report will be read by executives who understand the problem, but not programming. The format of the report can aid them considerably in finding the information they need. A jargon-free executive summary at the beginning, explaining the problem and its solution, may be all they need to read.

Use good section headings and captions for your figures. Read Sections 9.3 and 9.4 of Huckin and Olsen [HO91, 176–183] for suggestions about formatting a report for maximum readability.

Prepare your report as neatly as you can. A professional consulting job may produce a 10–20 page report from a $2000 study. At more than $100 a page, a consultant can afford to do it right. You do *not* need to spend enormous amounts on type-setting or professional illustrators. A typescript is still acceptable in industry, though laser-printing is becoming the standard, rather than the exception. Figures can be prepared on separate pages, pasted in, and photocopied. Photoreducing hand-drawn figures before pasting them in can hide the wiggly lines somewhat. The software available on the Macintosh and Windows machines makes combining graphics and text easy.

Choose an appropriate font size for your document. Unless you are writing a document that you don't want anyone to read (a warranty, perhaps?), choose at least a 10-point font. Unless you are writing for young children or people with bad eyesight, use at most a 12-point font. Standard elite and pica typewriters fall in the correct range. Using too large a font usually strikes readers as childish, which is not the effect you want in a report that you are charging hundreds of dollars for. If you want to impress someone with the bulk of a report, use 24-pound rag bond paper, a moderately wide font (say Palatino or Bookman instead of Times), slightly wider margins, and slightly more leading (space between lines), not 14-point fonts.

Check your spelling and punctuation carefully. Check even the spaces around your punctuation:

- Have two spaces after each colon, question mark, exclamation point, or sentence-ending period. (This is typewriter advice—when typesetting with variable-width fonts, no extra space is used after these punctuation marks.)
- Have spaces outside, but not inside parentheses and quote marks.
- Have no spaces before any punctuation marks except "(" and open-quote marks.
- Have no spaces around dashes.

Remember to double-space all assignments turned in in this class, so that we can insert our comments easily. Also, remember to keep the returned assignment to turn in at the end of the quarter.

7. In-program Documentation

7.1 Goals—recognizing that programs are documents

In-program documentation is the most common, and most neglected, form of writing for computer programmers. We hope that this exercise teaches you

- what information to put in in-program documentation,
- one way to format the information so that it can be used, and
- the importance of in-program documentation.

7.2 Audience assessment—maintenance programmers

Just as tailors spend more time making alterations than sewing new suits, most programmers spend more of their working time modifying old code than they do writing new code. If a program is difficult to change, it will be clumsily modified, and the resulting mess will be blamed on the original programmer. To protect your reputation, it is essential that you make your programs easy to modify cleanly.

Writing easily modifiable code is still an art, rather than a science, but all the techniques of structuring and modularization taught in programming classes help. Kevin disagrees with Niklaus Wirth, who uses a book title to claim that *Algorithms+Data Structures=Programs* [Wir76]. Certainly, algorithms and data structures are necessary for programs, but interface specifications and maintenance documentation are also essential.

Programs are read by compilers and by people. Far too many programmers stop when they get a program that is readable by a compiler—totally ignoring their other audience. Who are the people who read programs? Mainly other programmers. Most often the programmers are not reading to learn the language or to admire your style, but to make specific changes to the program. They usually read only those parts of the program they think will be relevant, so you must provide cross-references to information not immediately visible.

Many programs rely on external documentation. For student work, the external documentation is often the assignment sheet. External documentation may be easier to write than in-program documentation, but is not nearly as useful to a programmer attempting to modify your code. First, the external documentation often gets separated from the source, so the programmer doesn't have it to read. Second, the source is often modified without changing the external documentation, so the two may no longer agree. It is essential that maintenance documentation be included in the source code itself.

7.3 Assignment—adding comments to Wirth's code

Read the attached description of a solution to the knight's-tour problem [Wir76, 137–142]. Rewrite Wirth's Program 3.3 presented in Section 7.8 (not changing the algorithm or the data structure) to make it comprehensible without the accompanying text. You will have to change the variable names, and you may want to change the procedure names. You can play with the indenting and other aspects of formatting, if you think it makes the program more readable. (We prefer the vertical alignment of `begin` and `end` used in Wirth's book to the alignment of `if` and `end` sometimes taught at UCSC.) It is acceptable to translate the program from PASCAL into C or C++, but the changes made should be minor syntactic ones, not major restructuring of the program.

Your finished product should be an executable, well-documented program. Wirth's name should appear prominently near the beginning (as the original programmer), but you should take credit for the changes and documentation you provided.

7.4 Writing process—in-program documentation

Despite the way this assignment is structured, in-program documentation is not the last thing you do to a working program. It is most needed in the early stages of the programming, as scaffolding to support the incomplete program fragments. Whenever you write a procedure, you should write the explanation of the interface and function before writing the procedure itself. If you aren't crystal-clear about the purpose of the procedure, you won't be able to write it correctly.

In good programs, the documentation is written first, and the program written to match the documentation. For large programs with multiple programmers, it is particularly important to get the interfaces between modules well-defined early in the process. If the interface specification isn't written out completely, isn't distributed to the programmers, or isn't comprehensible, it might as well not exist.

At this point in your academic career, you've probably been told a hundred times that you need to document your programs better. When pressed, your instructors have probably mumbled vaguely about wanting more documentation but not precisely what was lacking. We might not do much better at telling you what is expected, but we'll try.

Of course, the best way to find out what is needed in the way of program documentation is to try modifying someone else's code. Whenever you run into something you can't figure out, or, worse yet, something you think you understand but are wrong about, then you've found a place where the documentation is inadequate.

To forestall such misunderstandings, you have to write far more documentation than you, as the creator of the code, believe you need. Because well-documented programs are so rare, you've probably never seen one and aren't aware what they look like. Perhaps the best example is Knuth's type-setting program TeX [Knu86]. We have put the source code on reserve in the Science Library—take a look at it to see what a well-documented program looks like. Note that the ratio of documentation to code is large and that the documentation concentrates on explaining data structures and techniques, not on repeating what the code says.

7.5 Format for in-program documentation

Everybody has her or his own ideas about the best places to put documentation in source code, and the best ways to format it. The "correct" style depends on the language and on the practices of the company that will maintain the program. We'll try to concentrate on the universal features of source code documentation.

Much program documentation is contained in comments. In most documentation styles, two different types of comments are used: block comments and one-line comments. Block comments are multi-line chunks of text, often containing several paragraphs. One good style for block comments in Pascal is the following:

```
(*
 *      Block comments contain ordinary text written in good English.
 *      Several lines may be needed, and blank lines are left
 *      between paragraphs.
 *
 *      If variables or expressions are needed, they should be typed
 *      exactly as they would appear in the code.  For example,
 *      alpha[char] is 1 for each alphabetic character, 0 for others.
 *
 *      For ease of later editing, each sentence can be started on a new line,
 *      and no "enclosing box" is used.
 *      The vertical line of stars makes the block comment easy to find.
 *      If you feel obligated to enclose the block comments, you can
 *      put a horizontal line of stars as the first and last line,
 *      but don't put a right-hand column of stars, as it discourages
 *      programmers from changing the comment when they change the code.
```

```
*
*      A similar style can be used in C programs (with /* and */ delimiters).
*      Ada programs and assembly language programs use a delimiter
*      ("--" for Ada) at the beginning of each line of a block comment.
*)
```

This style for block comments has several advantages:

- blocks are clearly marked, and so are easy to find,
- sentences and paragraphs are easy to modify as corrections are made to the program,
- comments are not crowded and sloppy,
- not much time and memory are wasted on "noise" that conveys no information.

One-line comments are used to mark interesting places in the program, and often are simple declarative or imperative sentences (for example, "`x[i] is now the largest element.`" or "`Swap leftmost and rightmost incorrect elements.`").

The following sections explain when to use block comments and when to use one-line comments.

7.5.1 Identify your work

Put your name and corporate address in a block comment at the beginning of every file. When you modify someone else's code, do not remove the original names, but add a comment saying that you modified the code, what you did to it, and when. In this way you retain the modification history of the program with the source code. These block comments can be extremely useful when tracking down a new bug to find out what changes have been most recently made to the program.

7.5.2 Use white space freely

Ideally, each procedure and its accompanying documentation should fit on a single page. Use page breaks and blank lines to make the printed appearance of the source code correspond to the logical structure of the program. The emphasis on appearance is not to satisfy some perverse artistic æsthetic, but to make the program more readable.

Within procedures, adding a blank line between sections of code acts like a paragraph break to indicate a change in emphasis. Starting a new page before each procedure acts like a section or chapter break, to indicate a whole new topic. The page-break character (often called a form-feed) is control-L—put it before a procedure's block comment to make the whole thing start on a new page.

Because white space is a powerful tool for grouping text together visually, try to use it in semantically meaningful ways. Don't break code up arbitrarily, just because it has gotten too long—do add white space between sections performing different tasks. If you feel that a section has gotten too long, then look for a way to break into explainable subparts, don't just add a blank line in the middle.

7.5.3 Indent to show block structure

Block-structured languages are commonly printed with varying indentation. The more deeply nested a statement is, the more deeply it is indented.

Some programmers use tabs for indenting, but the resulting programs are often too wide to fit on a screen or piece of paper. Other programmers use only one or two spaces for indenting, making the level of nesting difficult to see. Each level of indentation should be about four spaces deeper than the previous, and indentation levels should be consistent. (If you use vi, try "`:set shiftwidth=4 autoindent`".)

Slightly different styles of indenting are used by different programmers, which can get confusing when reading a program that has been written by a larger team or modified by several different programmers. It is good practice for all members of a programming team to use the same indenting style and for all subsequent modifications to use the same style. Large development projects often have a style guide containing the conventions to be used by the programmers.

One popular style calls for the bodies of loops and then- or else-clauses to be indented 4 spaces deeper than the loop- or if-statement, with the punctuation (**begin** and **end** or { and }) at the same level as the outer text:

```
if (test)
{   body
}
```

A sequence of tests that determine which of several actions to perform should have all the tests indented to the same level, and all the actions one level deeper:

```
if(test1)
{   action1
}
else if(test2)
{   action2
}
else
{   default action
}
```

7.5.4 Name variables carefully

A variable name should tell the reader (not just the programmer) what its function is; calling a variable `rownum` is much better than calling it `i`. Try to avoid using the same variable for multiple purposes.

7.5.5 Use a block comment for each procedure interface

A procedure's block comment should describe the external view of the procedure. As a minimum, the comment should tell which parameters are input, which are output, what global data structures are used, and what side-effects the procedure may have. The function of the procedure should be clearly explained. Any preconditions that must hold before the procedure can be safely executed should also be included. Choose a standard format for all procedures, perhaps something like

```
(*      procedure-name
 *
 *      action:  explain what the procedure does (not how it does it).
 *      inputs:  list each variable that is input and what it means.
 *      outputs: list each parameter that is an output and what it
 *               means.
 *      returns: what is the meaning of the value returned by a
 *               function?
 *      globals: list any global variables or data structures needed by
 *               procedure.
 *      side-effects:  list any changes expected as a result of running the
 *               procedure.
 *)
```

The same variable may occur in globals, inputs, and outputs. A global constant may be listed as a global, but not as an input. A global variable that is used, but not changed, may be listed as both a global and as an input. A global variable that is set should be listed as an output or as a side-effect. For example, if a procedure reads data from a file and puts it into a global array, that array is properly viewed as an output of the procedure. On the other hand, if a procedure finds a path through a maze and keeps some statistics about the search as it goes, the changes to the global statistics are a side-effect.

A side-effect is any action performed by the procedure that is not completely contained in the values of the output variables. Common side-effects are changing other global variables, reading input (thus advancing the position in the input stream), and printing output. You often have to mention a global variable twice—once in the section listing global variables, to indicate that the global variable is used in some way, and once in the side-effects section, to explain how the global variable is changed by the procedure.

In many cases, it is better to say something like "`side-effects: none`" than to leave the reader trying to decide whether a procedure has no side-effects or you simply forgot to document the side-effects. But be careful—programmers have written block comments that claimed a procedure had no inputs, outputs, global variables, or side-effect—in other words that the procedure did nothing at all! In most cases, the procedure printed a message or read something from an input—both of which are classified as side-effects.

7.5.6 Use a block comment inside each procedure to explain method

Unless a procedure uses a completely trivial technique, a sentence or two of explanation at the beginning will make it much easier for the reader to understand. You do not have to cram 30 pages of analysis from a text book into a couple of sentences. If you can't say all that needs to be said in a couple of paragraphs, you should at least mention what technique is being used and give a complete citation to a more complete external description.

The interface description comment is not the correct place for an explanation of the techniques used. You should be able to change techniques (say, from insertion sort to heap sort) without changing the interface description. Use a separate comment for methods.

The methods comment explains what is done, why it is done, and when it is done. The details of how it is done should be left for the code. If you are not sure how much to put in the methods comment, it is probably better to put in too much than to put in too little.

7.5.7 Use a block comment for each data type

When you define a record or array structure, explain the purpose of each field. Give an overall view of the data structure (is it a binary tree? a linked list? a heap?). What special properties of the data structure are you relying on?

7.5.8 Use a block comment for each data structure

You will often use apparently simple data structures (such as arrays) without having to declare a special type. Often, the simpler the data structure, the more ways it could be being used. Elementary data structures may need more commentary than the structured types that have special declarations. For example, a simple array has many different possible meanings. It may store a mapping from one set to another—what are the sets? what does the mapping mean? Or the array may be being used as a hash table—what are the keys? what method is used to resolve collisions?

Identify global variables that are used as constants throughout the program. Although PASCAL only recognizes scalar constants, programmers often use global arrays and sets as constants. Because PASCAL does not support pre-loaded arrays, you should say where the structure is initialized.

7.5.9 Use a one-line comment for each local variable

Each local variable should have a single, distinct purpose. The purpose is hinted at by the name of the variable, and explicitly stated in a comment where the variable is declared. For example,

```
var L: integer; {A[L] is the leftmost unchecked value.}
```

The one-line comment may be just a noun phrase, with an implicit "The variable foo is" in front of it. For example,

```
var L: integer; {the index of the leftmost unchecked value}
```

Watch out for slightly inaccurate statements. If you're in a hurry you might write "`first unchecked value`" when you mean "`the index of the leftmost unchecked value`". Saying "`SP is the stack pointer`" is much less valuable than saying

```
var SP: integer; {STK[SP] is the top element of the stack, and
          STK[SP-1] is the next available empty slot}
```

If you are cramped for space in the comment, you can sometimes leave out the definite articles (for example, `{index of leftmost unchecked value}`). Don't do it if any confusion could result; make the comment multi-line instead.

Be sure your names are not misleading. Using `x` for a variable that indexes the vertical direction in a printout, and `y` for the horizontal variable is a common mistake—people are conditioned to expect `x` to be the horizontal direction. Be particularly careful in checking input and output loops.

7.5.10 Use comments sparingly inside the body of the code itself

Many beginning programmers, when told to document, attempt to be explicit and write useless comments like

```
i := i +1;    {increment i.}
```

Many short comments interfere with reading the code itself. A long block comment at the beginning of the procedure can be far more helpful and is usually easier to write. Reserve one-line comments for such tasks as

- identifying cases in a long if-then-elseif-else statment
- headers at the beginning of a new section of code (after a blank line)
- identifying end statments
- displaying loop invariants
- translating complex expressions back into meaningful pseudo-code.

7.5.11 Use assertions.

Many languages support (or, by using macro packages, can be made to support) `assert` statements. The format varies with the implementation, but usually consists of an expression that is supposed to evaluate to true and an error message to print if the assertion is not true. When an assertion fails, it usually breaks to a debugging program (or causes the program to abort). For example, at the beginning of a procedure, one might test a precondition as follows:

```
assert(L<R, "left and right pointers are crossed");
```

Loop invariants are particularly good things to put in assertions. David Gries recommends using the loop invariants as the foundation for designing new algorithms and accurately implementing existing algorithms [Gri83].

If your language does not support `assert` statements, it is still worthwhile to put the assertions in as comments, since they tell the reader something about what you were expecting when you wrote the code.

7.6 Things to keep in mind for peer editing

Read your partner's code as if you had never seen the algorithm or data structures before. Consider what you would need to know to make a major modification. For example, consider writing a graphical procedure to draw the chess board with the tour, or consider changing the simple backtracking search to an "intelligent" heuristic search. What do you need to know about the data structures? about the algorithm?

Look at the spacing and indentation on the page. Is the code visually divided into natural chunks? Compare with the guidelines in Sections 7.5.2 and 7.5.3. Can you see at a glance where each loop begins and ends?

Check that all block comments are in good English. Telegraphic style, in which you leave out articles and verbs, may be used only in one-line comments.

Check that the variable names are meaningful. For example,

- The name `width` may be used for a size, but not for a horizontal position.
- The word `solution` is a poor name for a Boolean variable, unless the problem is to answer a yes/no question. One better name would be `SolutionFound`, but this is still a bit too generic. A name specific to the problem, such as `TourCompleted`, is better.
- The name `try` is a poor one for the recursive search procedure, as it does not say what is being tried. It might be better for the procedure to be a function called something like `CompleteTour`. After all, the purpose of it is to complete a partially completed tour, or to report that no such tour exists.
- Sometimes names can be chosen to make code read easily, such as choosing to do a range check by a PASCAL set-membership test (`if (x in range)`), but don't let your clever word play interfere with good programming—range checks should be done by a pair of comparisons, not a set membership operation.

  Wirth uses set membership range checks in his code, which he claims is an optimization. In fact, on the CDC 6500, sets with 60 or fewer possible elements could be implemented efficiently as single-word bit vectors, and the machine had hardware for doing fast bit checks. On his machine, set membership checking was faster than pairs of comparisons, but this is not true of most modern machines, which have more modest word sizes and no special hardware for bit testing. When you do this sort of machine-specific optimization, you should explain it in a block comment, so that someone porting the code to a different machine can know what to change.

Check that the global data structures are well-commented. For example, the array for the board is the most important data structure in the knight's tour problem. It is important to document the meaning of the contents, not just of the indices. What does the number in the board array mean? Special values (such as 0 to mean that a square has not yet been visited) should be explicitly described.

Is the output format for the program described? To say that a program "prints the solution" is not particularly helpful. There are many ways to print the solution to a knight's tour—listing the positions visited as (x,y) pairs, drawing a chessboard with numbers to indicate the order for visiting the squares, moving a drawing of a knight around on a display, or using either of the two standard chess notations for describing the movements. The description of the format should be explicit enough that another programmer can write an interface to the program without having to learn all the internal data structures and operations.

Is the input format correctly described? Even Wirth's external documentation is a bit inconsistent, as the starting location and board size are not parameters but compile-time constants!

Is the explanation of the recursive procedure clear? The best way to describe a recursive procedure is with preconditions and postconditions. How much of the tour is already completed before each call to the procedure? What do the different possible return values mean? If global variables are changed, what do they contain before and after the procedure? Re-read Section 6.5 about how to explain recursion.

Termination criteria are also useful—when does the recursion stop? Does the programmer clearly distinguish between the action of one iteration of the recursion (lengthening the tour by one) and the overall effect of the procedure (determining whether there is a tour that starts with the current partial tour)?

Are the side-effects properly described? The main purpose of the recursive procedure is to change the global board array—the returned Boolean value is just an output to flag when the procedure is finished. The documentation should make it clear what the contents of the array are after the procedure returns. Warning: the values are different depending on whether a tour was found or not.

Are the words carefully chosen? Be careful not to copy Wirth too slavishly—he is not a native speaker of English. For example, "covering" is a technical term mis-used by Wirth—a *covering* is a collection of sets whose union contains the object being covered. You should write using your own words, being particularly careful with any words that you have only recently acquired.

Watch out for vague words like *valid* or *good*. When they occur, look for a more descriptive word.

When technical words are used, make sure they are used consistently. Try to have only one meaning for each word, and only one word for each concept. As an example of what can go wrong, one programmer used *square*, *field*, *record*, *record field*, *box*, *location*, *playing field location*, and *board square* interchangeably for two concepts that were not distinguished, the coordinates of an element of the board array and the contents of that element.

Watch out for **off-by-one** errors in both the programs and the comments!

7.7 The final draft

Typos in programs are not just minor inconveniences; they can be fatal errors. Make sure your final program compiles and runs before handing it in. It is easy to insert a comment that isn't properly terminated, and so even if all you edit are the comments, you have to recompile and re-run to make sure you didn't break anything.

Try to keep each procedure and its documentation on *one page*. If you must print a procedure on two pages, make the page break in a *natural division* in the code. Insert page breaks and white space to make the appearance of the printed source code correspond to the logical structure of the program. The unix filter `pr` will automatically add the file name, date, and time to the top of each page. If you're using UNIX, look into using `pr` (or the equivalent `lpr -p` option). Some of the machines on campus have `enscript`, which does an even better job of printing program listings.

For this assignment only, **do not double space** what you hand in. There should be enough white space in a well-formatted program for us to write whatever comments we need. If you end up with dense blocks of code or text, something needs to be changed.

Note: the usual 8.5" × 11" rules apply. If you have 14" wide paper, cut it down to 11". Remember to burst the pages and staple in the upper left corner. The emphasis in this class is on making the program as easy for the reader to understand as possible—if it cascades off a desk and doesn't fit in a file folder, the reader is much less likely to try to read it.

Our old eyes are also tired of straining to read pale grey, fuzzy characters. If the printout you get is not crisp and clear, demand that it be reprinted with a decent ribbon. If you are using your own printer, make sure that you have a spare ribbon available.

7.8 Wirth's description of the knight's tour program

The remainder of this section is copied with the permission of Prentice-Hall from Niklaus Wirth's Algorithms + Data Structures = Programs, *1976, pp. 137–141. Figure and display numbers have been changed to fit the numbering of the current document.*

Backtracking Algorithms

A particularly intriguing programming endeavor is the subject of "general problem solving." The task is to determine algorithms for finding solutions to specific problems not by following a fixed rule of computation, but by trial and error. The common pattern is to decompose the trial-and-error process into partial tasks. Often these tasks are most naturally expressed in recursive terms and consist of the exploration of a finite number of subtasks. We may generally view the entire process as a trial or search process that gradually builds up and scans (prunes) a tree of subtasks. In many problems this search tree grows very rapidly, usually exponentially, depending on a given parameter. The search effort increases accordingly. Frequently, the search tree can be pruned by the use of heuristics only, thereby reducing computation to tolerable bounds.

It is not our aim to discuss general heuristic rules in this text. Rather, the general principle of breaking up such problem-solving tasks into subtasks and the application of recursion is to be the subject of this chapter. We start out by demonstrating the underlying technique by using an example, namely the well-known *knight's tour*.

Given is a $n \times n$ board with n^2 fields. A knight—being allowed to move according to the rules of chess—is placed on the field with initial coordinates x_0, y_0. The problem is to find a covering of the entire board, if there exists one, i.e., to compute a tour of $n^2 - 1$ moves such that every field of the board is visisted exactly once.

The obvious way to reduce the problem of covering n^2 fields is to consider the problem of either performing the next move or finding out that none is possible. Let us therefore define an algorithm trying to perform a next move. A first approach is shown in (7.1).

(7.1)

```
procedure try next move ;
begin initialize selection of moves ;
    repeat select next candidate from list of next moves;
        if acceptable then
        begin record move ;
            if board not full then
            begin try next move ;
                if not succsessful then erase previous recording
            end
        end
    until (move was successful) ∨ (no more candidates)
end
```

If we wish to be more precise in describing this algorithm, we are forced to make some decision on data representation. An obvious step is to represent the board by a matrix, say h. Let us also introduce a type to denote index values:

type *index* = 1 .. *n* ;
var *h*: **array** [*index, index*] **of** *integer*

The decision to represent each field of the board by an integer instead of a Boolean value denoting occupation is because we wish to keep track of the history of successive board occupations. The following convention is an obvious choice:

$h[x, y] = 0$: field $\{x, y\}$ has not been visited
$h[x, y] = i$: field $\{x, y\}$ has been visited in the i^{th} move $(1 \le i \le n^2)$

The next decision concerns the choice of appropriate parameters. They are to determine the starting conditions for the next move and also to report on its success. The former task is adequately solved by specifying the coordinates x, y from which the move is to be made and by specifying the number i of the move (for recording purposes). The latter task requires a Boolean result parameter: $q = true$ denotes success; $q = false$ failure.

Which statements can now be refined on the basis of these decisions? Certainly "board not full" can be expressed as "$i < n^2$." Moreover, if we introduce two local variables u and v to stand for the coordinates of possible move destination determined according to the jump pattern of knights, then the predicate "acceptable" can be expressed as the logical combination of the conditions that the new field lies on the board, i.e., $1 \le u \le n$ and $1 \le v \le n$, and that it has not been visited previously, i.e., $h[u, v] = 0$. The operation of recording the legal move is expressed by the assignment $h[u, v] := i$, and the cancellation of this recording as $h[u, v] := 0$. If a local variable $q1$ is introduced and used as the result parameter in the recursive call of this algorithm, then $q1$ may be substituted for "move was successful." Thereby we arrive at the formulation shown in (7.2).

		3		2	
	4				1
$y \rightarrow$			N		
	5				8
		6		7	
			$\uparrow$		
			x		

Figure 7.1: The eight possible moves of a knight.

(7.2)

```
procedure try(i: integer; x, y: index; var q: boolean);
var u,v: integer; q1: boolean;
begin initialize selection of moves ;
   repeat let u,v be the coordinates of the next move defined by the rules of chess
      if (1 ≤ u ≤ n) ∧ (1 ≤ v ≤ n) ∧ (h[u, v] = 0)then
      begin h[u, v] := i;
         if i < sqr(n) then
         begin try(i + 1, u, v, q1);
            if ¬q1 thenh[u, v] := 0
         end else q1 := true
      end
   until q1 ∨ (no more candidates)
   q := q1
end
```

Just one more refinement step will lead us to a program expressed fully in terms of our basic programming notation. We should note that so far the program was developed completely independently of the laws governing the jumps of the knight. This delaying of considerations of particularities of the problem was quite deliberate. But now it is time to take them into account.

Given a starting coordinate pair $\{x, y\}$, there are eight potential candidates for coordinates $\{u, v\}$ of the destination. They are numbered 1 to 8 in Figure 7.1.

A simple method of obtaining u, v from x, y is by addition of the coordinate differences stored in either an array of difference pairs or in two arrays of single differences. Let these arrays be denoted by a and b, appropriately initialized. Then an index k may be used to number the "next candidate." The details are shown in Figure 7.2. The recursive procedure is initiated by a call with the coordinates x_0, y_0 of that field as parameters, from which the tour is to start. This field must be given the value 1; all others are to be marked free.

$$H[x_0, y_0] := 1; \qquad \text{try}(2, x_0, y_0, q)$$

One further detail must not be overlooked: A variable $h[u, v]$ does exist only if both u and v lie within the array bounds $1 \ldots n$. Consequently, the expression in (7.2), substituted for "acceptable" in (7.1), is valid only if its first two constituent terms are true. A proper reformulation is shown in Figure 7.2 in which, furthermore, the double relation $1 \leq u \leq n$ is replaced by the expression u **in** $[1, 2, \ldots, n]$, which for sufficiently small n is possibly more efficient (see Section 1.10.3 [Wir76]). Table 7.1 indicates solutions obtained with initial positions $\{1, 1\}$, $\{3, 3\}$ for $n = 5$ and $\{1, 1\}$ for $n = 6$.

It is possible to replace the result parameter q and the local variable $q1$ by a global variable, thereby simplifying the program somewhat.

```
program knightstour(output);
const n=5; nsq=25;
type index = 1..n;
var i,j: index;
    q: boolean;
    s: set of index;
    a,b: array [1..8] of integer;
    h: array [index, index] of integer;

procedure try(i: integer; x,y: index; var q: boolean);
    var k,u,v: integer; q1: boolean;
begin k := 0;
    repeat k := k+1; q1 := false;
       u := x+a[k]; v := y+b[k];
       if (u in s) and (v in s) then
       if h[u,v] = 0 then
       begin h[u,v] := i;
           if i<nsq then
               begin try(i+1,u,v,q1);
                  if not q1 then h[u,v] := 0
               end else q1 := true
       end
    until q1 or (k=8);
    q := q1
end {try};

begin s := [1,2,3,4,5];
    a[1] := 2; b[1] := 1;
    a[2] := 1; b[2] := 2;
    a[3] := -1; b[3] := 2;
    a[4] := -2; b[4] := 1;
    a[5] := -2; b[5] := -1;
    a[6] := -1; b[6] := -2;
    a[7] := 1; b[7] := -2;
    a[8] := 2; b[8] := -1;
    for i := 1 to n do
        for j := 1 to n do h[i,j] := 0;
    h[1,1] := 1; try(2,1,1,q);
    if q then
        for i := 1 to n do
        begin for j := 1 to n do write(h[i,j]:5);
           writeln
        end
    else writeln('NO SOLUTION')
end.
```

Figure 7.2: Knight's tour

1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

23	10	15	4	25
16	5	24	9	14
11	22	1	18	3
6	17	20	13	8
21	12	7	2	19

1	16	7	26	11	14
34	25	12	15	6	27
17	2	33	8	13	10
32	35	24	21	28	5
23	18	3	30	9	20
36	31	22	19	4	29

Table 7.1: Three Knight's Tours

8. Naive-user documentation

8.1 Goals—better paragraphs and writing for non-technical audiences

This assignment has two purposes—to improve your paragraph structure and to make you appreciate how difficult writing for an ignorant audience is.

Huckin and Olsen [HO91] have an excellent chapter on paragraphing, Chapter 22, and another on parallelism, Chapter 23. Section 12.1 has good general ideas about engaging more-or-less ignorant audiences. This is also a good time to read Chapter 3 (Identifying Audiences and Purposes). Of course, Chapter 17 (Instructions, Procedures, and Computer Documentation) is worth reading for this assignment.

Notice how the paragraph you have just read is parallel in structure to the introductory one that precedes it. This is an example of an important kind of parallelism, and is discussed in Huckin and Olsen [HO91, Section 23.3].

Also notice that Section 8.1 of this workbook has rather short paragraphs. Short paragraphs are one of six important formatting conventions for making reading easier [HO91, 177–179]. Read Sections 9.3 and 9.4 and Chapter 21 of Huckin and Olsen for more suggestions on increasing readability [HO91].

Finally, take a good look at Scott Brookie's *Unix for Luddites* [Bro86]. This manual is a classic for introducing a complex system to an audience who is not only ignorant, but hostile. Brookie recognizes, and attempts to defuse, this hostility—his first sentence is:

> Luddites are people who believe that new kinds of technology will make their lives worse.

Huckin and Olsen discuss introducing a problem with two directly conflicting terms in their Chapter 5. Brookie's sentence is a rather subtle version of this. After you have read Huckin and Olsen's Chapter 5, see if you can figure out how Brookie's first sentence, and the two introductory paragraphs of his manual, work to make effective contact with the naive user.

8.2 Audience assessment—non-technical audiences

You have been hired as a training consultant by the Boring Business Company. Boring Business has recently purchased a UNIX computer system, to be used primarily for document preparation. Someone suggested setting up an electronic mail system for phone messages, meeting announcements, and similar short memos. You have to write a training manual explaining how to use the UNIX system. For this assignment, you will write a chapter of the manual on reading and sending mail.

Most of the readers will be typists and receptionists, but some executives at Boring Business will glance through the manual so that they can pretend to understand what their secretaries do. You can assume that all your readers know how to use a keyboard, but at least some of the readers will not have looked at any previous chapter of the manual.

You do not need to explain all the details of the mail program, only as much as a secretary needs. Remember that secretaries have different needs from students: they frequently mail to the same group of users, they need to forward mail that should be handled by someone else, and they need to mail memos that have already been typed into the system.

Make sure the reader won't get caught accidentally invoking some feature s/he doesn't understand. You can use any existing documentation as a reference, but be sure to cite it properly. Remember that you were hired to write this manual because Boring Business was not happy with the documentation already available.

8.3 Writing process—paragraph structure

Paragraph structure is an aspect of writing that used to be taught in grade school and high school, but seems to have disappeared from the American school curriculum. Students who learned English as a second language may never have gotten past sentence structure. Fortunately, proper paragraph structure is not complicated and can be learned quickly.

For programmers, there is a useful analogy between program structure and document structure. A sentence is analogous to a program statement, and a paragraph to a procedure. Like a procedure, a paragraph should have a single well-defined purpose. Just as a procedure begins with a block comment explaining its function, a paragraph begins with a topic sentence that gives the main subject of the paragraph.

Although some writing texts mention other possible paragraph structures, for technical writing the topic sentence should always be the first sentence. Check the first sentence of each paragraph to see if it says what the paragraph is about. If it doesn't, try to decide whether the sentence should be fixed to match the paragraph, or the paragraph fixed to match the sentence.

The two most common paragraphing faults are *rambling paragraphs* and *artificial breaks.* A rambling paragraph has several topics, often not closely related. It can be fixed either by breaking the paragraph into smaller ones or by removing extraneous material. Artificial breaks occur when novice writers try to divide paragraphs that are too long. The lack of topic sentences defeats the entire purpose of having paragraphs.

The proper length for a paragraph varies, depending on the content and the importance of the information. Three to five sentences (about 30 to 60 words) is a common length. The topic sentence is supported by a few, well-chosen, additional sentences. If your paragraph runs on and on, your readers may glance at the first sentence or two, then skip to the next paragraph. Never bury the key points in the middle or end of a long paragraph.

Short paragraphs make emphatic statements. Use them sparingly.

The continuity between paragraphs is as important as the internal structure of the paragraphs. Sudden changes of subject will make readers stumble, but choosing a suitable ordering for your topics will often let you flow smoothly from one to the next. When no ordering seems to work, try breaking the text into separate sections, each on a separate subject.

Within a section, paragraph breaks can be made less abrupt by anticipating the next subject at the end of one paragraph, or by referring to the previous subject in the topic sentence of the new paragraph. Notice how abrupt this paragraph would have been if I'd left off "Within a section".

Paragraphing is especially important in naive-user documentation, because documentation is consulted, not read like a novel. Related information *must* be brought together into one paragraph or into contiguous paragraphs, so that users looking for one piece of information will see the essential related material. Using headlines for sections (as in this workbook) helps guide the user to the right place.

8.4 Things to keep in mind for editing and partner work

Students usually write for a single audience, their teacher. Because the teacher usually knows more than the students, the purpose of the writing is usually to impress, and not to inform. The teacher's purpose is to find out whether you learned anything, and, sometimes, whether you can articulate your thoughts clearly. Rarely is the teacher interested in the ostensible content of the paper.

In this assignment, you are writing for a different audience, one presumed to know less than you. Your readers will not know all the jargon you use almost unconsciously, so you'll have to be extra careful about which words you use. You'll also have to be careful about the order in which you present new ideas. New computer users do not want to have to read documentation twice—most don't even want to read it once.

Not everyone will choose the same features of the mail program to cover in an introductory document. Consider carefully the intended purposes of the electronic mail system before deciding whether to describe particular features. You should make your decisions based on how important it is for the users to be able to use the features, not on how easily you can describe them.

Make sure that you do not assume that the UCSC mail setup is identical to the mail setup at Boring Business. For example, very few sites set the `askcc` option—even at UCSC, the CE and CIS department machines don't set it. If you want to make some assumptions about how mail is set up, put them in a cover letter, so a technician or consultant can check that the installation matches what you are documenting.

A one-page quick summary can be a useful adjunct to the more detailed tutorial you have written. It should be usable as a reference both by those who have read the entire chapter, and by those who think they know what they're doing, and don't have time to wade through the introductory stuff.

Be careful not to assume too much knowledge on the part of the user. (Will a novice UNIX user know how to use `more`?)

Watch out for sentence fragments! Fragments? Noun phrases without verbs. Ex-President Bush, famous for fragments. Very hard, figuring out what fragments are supposed to be saying, especially when they are as long as this one and contain verbs in subsidiary clauses.

Avoid Latin abbreviations! The most often abused Latin abbreviation is *etc.*, and abbreviation for *et cetera* which means *and other things.* Do **not** say "and etc.", and do not use *etc.* for people. If you must use a Latin abbreviation for *and other people*, use *et al.* (for *et alia*), but the use of Latin in formal writing is a relic of the era when all literate people were expected to know Latin, and all writing was done with pens. With word processors, there is little reason not to use the equivalent English expressions.

Other Latin abbreviations to avoid are *i.e.* and *e.g.* If you must use them, punctuate them properly (i.e., like this). I prefer to use the simple English phrases (for example, I use *in other words* for *i.e.*).

Watch the jargon! Using *login* as a synonym for *login name* is a UCSC idiom that will not be understood in other contexts. In standard usage, *login* as a noun refers to the process of logging in, not the person who does it. The verb is *log in*—like many verb-particle pairs, the verb is two words, but the corresponding noun is one. Table 8.1 gives a table of other jargon words adapted from a list posted to the newsgroup rec.humor.funny.

Another sloppy usage that seems to be common at UCSC (and perhaps more generally) is the use of *substitute y with x* when *substitute x for y* is intended.

Watch out for words beginning with *any.* In my dictionary, only *anybody*, *anyhow*, *anymore*, *anyone*, *anyplace*, *anything*, *anyway*, and *anywhere* begin with *any.* Any time you find yourself using a different word beginning with *any*, break it up. Sometimes, words beginning with *some* are spurious compounds also—a good spelling checker can be a big help.

Although English allows almost any noun to be used informally as a verb (a process known, appropriately, as *verbing*), formal writing does not allow this process. For example, *access* is a noun not a verb, you should *get access* or *obtain access*—but this abuse is widespread enough that within a decade or two it will probably become acceptable. Another abused noun is *reference*, which is formed from the perfectly good verb *refer*.

It is perfectly acceptable to use *we* in a formal document, if you mean *you and I* or *we, the multiple authors.* The singular pronoun *I* is much more problematic, and should only be used when the human being peeks out from behind the formal facade: *the first time I used UNIX mail, I managed to send blank messages to all the faculty in the department, because I didn't understand the difference between reply with* `R` *and reply with* `r`*.*

Watch out for split infinitives. For some reason, this assignment always brings out a rash of them (for example, *to simply send* and *to just send*). Many times, the offending word could be dropped from the middle of the infinitive without loss.

Some amusing dangling modifiers have occurred as well: *After typing in the subject, the cursor will ... .*

word	computer usage	normal usage
code	software instruction	cryptic message
boot	load operating system	footwear
virus	makes computer sick	makes you sick
memory	data storage	retained ideas
news	Usenet	NBC/CNN/C-Span
mail	electronic letters	bills/junk mail
FIDO	subnet	dog
pen	pointing device	writing with ink thing
SLIP	serial line interface protocol	a fall/undergarment
TIP	open line for comm.	money for waiters/waitresses
mouse	pointing device	rodent
screen	terminal face	metal mesh
spool	place to save mail or output	thing that holds thread
thread	code structure method	stuff on spools
OOPS	C++ or Ada	a booboo
ports	serial, parallel, ...	place where ships dock
floppy	removeable disk	limp
hard drive	fixed disk	difficult trip
Windows	GUI nightmare	cleaning nightmare
root	UNIX system administrator	bottom part of plant
Smalltalk	programming language	chit chat

Table 8.1: Some common jargon, modified from a list posted to the newsgroup rec.humor.funny.

8.5 The final draft

As always, the final draft should incorporate any improvements that resulted from your partner's editing. You will probably have to add more material at the beginning of the chapter and change the order in which you cover the topics. If you use a word-processor to rearrange your first draft, be careful to redo the transitions so the seams don't show.

The final draft should be on $8\frac{1}{2}$" $\times$ 11" paper, stapled together or bound on the left edge. Paper clips and slip-on plastic binders are not suitable, as pages tend to slip out of them and scatter. We generally prefer papers with one staple in the upper left corner—they are easy to handle and to photocopy.

Make sure you remove all spelling and punctuation errors. Good typists will not trust documentation that contains errors in their area of expertise.

Check your use of articles and subject/verb agreement. For example, *mail* is not a countable noun, and so it can not be used with plural verbs or indefinite articles. If you need to count items of mail, you can use the word *messages*.

Use a standard form for citing your sources of information. There is no widely accepted form for citing on-line documentation, but the UNIX man pages have been printed as a book, which can be cited.

Check that all your examples are 100% accurate. Try them out to be sure. It is very frustrating to read a manual that says "you will now see ... on your screen," and to have something else entirely appear. Make your examples explicit, rather than using the Backus-Naur Form notation that computer scientists often adopt for describing syntax in a generic way. People are much better at reasoning by analogy than computers, and much poorer at substituting values for variables. Give them examples they can modify, rather than rules for which they have to fill in the blanks.

When you give examples of what people are to type, or what they should see on the screen, use displays. That is, put the material on a line by itself, even though it is part of the surrounding sentence or paragraph. Punctuate the sentence normally—it is incorrect to add a colon before a display, unless the display is a list that would require a colon before it anyway. For example, this sentence has

one line displayed,

but is otherwise normally punctuated.

Since manual chapters don't usually have acknowledgements, prepare a separate short memo to the instructors saying who helped you with the editing. Turn in both the first and final drafts. This memo should also explain why you chose to include or omit certain material from the manual.

As for all assignments this quarter, keep the graded assignment when we return it to you, so that you can turn it back in at the end of the quarter. We need your entire quarter's work in front of us when we write the final narrative evaluations.

9. Library Puzzle

This assignment is an exercise to get you more familiar with the resources in the library, particularly with the computer indexes available on MELVYL and with the hardcopy indices of most value to computer people. The puzzle was devised by Kevin Karplus, in imitation of puzzles created by Alan Ritch, a librarian at UCSC, who writes and edits the *Mind of Melvyl* newsletter (affectionately known as *MOM*), which frequently contains such puzzles.

The puzzle will stretch your MELVYL search skills far beyond the level of competence attained by most students or faculty. We hope that the assignment will make it easier for you to do any necessary library research for your final paper, and the library search skills exercised should also be applicable to other courses and research projects.

Some of the questions below are simple, straightforward exercises of the obvious MELVYL commands, others are puzzle questions requiring ingenuity and perseverance to find the requested information. These library puzzles often appeal to crossword puzzle fanatics.

For each question below, give one or two sentences as an answer—not just the jone word or number the question asked for. Show the search command that found the answer you got, and, perhaps, some commands that you thought should work and didn't. When you get a negative result, show partial solutions (e.g., *UCSC doesn't have it, but someone else does, and we have something else by the same author*).

All the questions can be answered without visiting the library, since MELVYL is accessible from any terminal on campus. However, you might want to visit McHenry library or the science library to see some of the books and journals that you find.

9.1 Catalog database (`CAT` and `TEN`)

1. Does UCSC have the latest edition of *The Joy of TEX*?
2. What package does *The Joy of TEX* describe?
3. Give the UCSC call number of a book on "computer gamesmanship".
4. Find books or conferences from the last ten years about random graphs. Use the `MAIL` command to mail the results to yourself. Give the search strategy and the total number you found.
5. How many computer files are catalogued by the UCSC branch of the library? How many of them are more than ten years old? (Note: *computer file* is a type of format that can be searched for in the `FORM` index—you may have to use `HELP FORM` to get the correct abbreviation for the format name.
6. Try to find more than 100 books (or things cataloged as books), all published in the last ten years and all in the UCSC library, that are relevant to CMP 215. Remember that you can find course names and catalog copy on the InfoSlug gopher server. Hint: use `TEN` and `f SU`. The tricky part is finding a relevant subject—start with a book or conference that you believe is relevant, display subjects for it, then look for other books with the same subject. A few iterations should converge. You can try keywords from the catalog description, but that may take a little longer to converge.

   I don't want the long list of books, but the search strategy and the number of books you found.

9.2 Magazine database (`MAGS`)

What percentage of the Ph.D.'s granted in computer science in 1989–90 were to women? Find articles that attempt to explain why there are so few women, and articles explaining what is being done to improve the situation. [Note: this question can be answered entirely from MELVYL, using the stored abstracts.

9.3 World-wide web

1. Using Lycos (`http://www.lycos.com`), Alta Vista (`http://www.altavista.digital.com`), or other Web search engine find the 1994 version of the surveys that the data about women with PhDs in computer science is based on. Note: this question will be much easier if you can get the real name of the survey from the previous question.
2. Find the issue date and city for the Richard Nixon postage stamp. While you're at it, find a quote "you won't have Dick Nixon to lick around anymore."

9.4 Computer article database (`COMP`)

1. Find an article on fast matrix multiplication.
2. How many articles can you find on FPGAs? You should get a list of at least 300.

9.5 You figure out what indices to use

The following questions may require hardcopy indices in addition to the computer indices. Please outline your search strategy, and tell us what false leads you followed, as well as how you finally found the solution.

1. How many searches were made from UCSC in the last month for which statistics were available, counting both dedicated terminals and internet access? Note: can get the data from MELVYL, but getting totals of local usage may require some processing of the data. Explaining how to get the numbers will suffice, with extra credit for actually producing them.
2. Find the best article you can on asymptotically fast matrix multiplication. Give the reference and the asymptotic running time.
3. In an early 1980s conference, Ramachandran wrote a paper on driving long lines in a VLSI layout. Find the conference paper and the journal paper that updated it.
4. Find the earliest paper you can that mentions the main ideas in field-programmable gate arrays. Note: there is a pre-1984 paper, but I don't believe that the library has it, nor that normal library search techniques will uncover it. I am curious to find out how far back the ideas can be traced without inside knowledge of the early publication.

   What is the earliest article about FPGAs that you can find in the Design Automation Library CD-ROM?
5. Find a recent paper from Berkeley that cites Bryant's results about multipliers having exponentially large binary decision diagrams. (Hint: first find the full citation for Bryant's paper.)
6. Find out how many books on C or C++ the Stanford Bookstore carries. This search can be done without leaving your workstation. For extra credit, separate the C books from the C++ ones (I believe that this cannot be done using only the library search programs, but should be fairly easy if you know how to use your computer effectively).

10. Survey article

10.1 Goals—short surveys, library practice

This assignment has several goals:

- To require students to learn to use the reference resources available to them. These resources will be discussed in detail in a guest lecture by a reference librarian.
- To teach you how to write a short survey article (five pages is sufficient).
- To get you started on your final project. This survey article may form a "background" section of your final report, or it may be a broad look at a topic that you will focus on more closely in your final project.

A short survey article is a relatively simple form of writing, although finding and selecting the right technical content can be a demanding task. The emphasis in this assignment is on collection, integration, and accurate presentation of technical information.

Good technical writing stands firmly on these two legs: effective rhetorical strategies for meeting the needs of your audience, and absolutely accurate presentation and explanation of technical information.

10.2 Choosing your topic

In choosing your topic for this paper, pick some technical issue or problem with which you are already to some degree familiar, and would like to learn more about. Picking some topic about which you are largely ignorant will make it impossible for you to do a good job with this assignment. If you are thinking of documenting a design project you have already completed for your final project, use this assignment to learn some more about some aspect of your project.

Hint: this survey article may well serve as the first draft of a previous-work section in your final report. Read Section 11 for information about selecting a final project.

We will also discuss appropriate topics in class.

10.3 Textbook resources

The resources in the main text for writing this kind of paper are found in several areas, since, in general, this type of writing is most commonly a component of other kinds of writing. A coherent and inclusive survey of the available information is part of certain kinds of feasibility reports, for example, and a summary of previous research is part of every journal article presenting new research.

Chapter 18 of Huckin and Olsen (Theses and Journal Articles) is relevant to this assignment in part, as are parts of Chapter 14 (Feasibility Reports). Read both of these chapters in preparation for this assignment. Chapters 2 and 21 are also helpful for the general principles of information selection and presentation, Chapter 5 presents an approach for introducing a topic, and Chapter 12 talks about the basic structure of a report.

10.4 Audience Assessment—fellow students

As indicated above, audience assessment is not the main stress in this assignment. Your audience is other people like you—people who knew as much (or as little) as you, **before** you did this assignment. The more similar the audience is to you, the easier it is to write for—just think about what you would want to know.

The main principle to keep in mind here is always to assume that your audience can be counted on to know rather less than you think they ought to or wish they did.

Warning: in many cultures it is polite to pretend that your reader is highly intelligent, and you flatter them by giving them material that is difficult to understand. This is not true of American technical writing. Here, you flatter people by taking the time to explain something in such a way that they can understand it. Watch out for unconscious cultural effects on your writing.

10.5 Writing process—finding, organizing, writing

The process of writing this assignment can be broken down into three steps: the library search, organizing the information, and writing the paper.

10.5.1 Library search

The library search itself has three steps:

1. *The topic statement.* Write a short, quick topic statement that explains what you want to research, and what you would like to know about it. Be as specific as you can, but don't labor over punctuation, sentence structure, and paragraph structure. It is meant to get you started, and will be completely re-done later, or perhaps even thrown out. Mostly you are trying to generate the key words you will need for on-line data base searches and for index use. A couple of paragraphs should be sufficient.

   Be careful about the breadth of your topic. If you choose too narrow a topic, you'll have difficulty finding information. If you choose too broad a topic, you'll never be able to organize the overwhelming material well enough to say anything. You should probably start with a slightly broader topic, and focus on some narrower part once you have found some material.

2. *The library.* Take your topic statement and head for the library, and use your key words (or key terms) in the manner described by the reference librarian in the guest lecture. Be sure to take good notes on this lecture—it is a key part of this assignment.

   Don't be afraid of reference librarians if you get completely lost in the library. They have the jobs they have because they like working with students, and really do want to help you learn how to use the resources the library offers. They have more time available in the morning, however, than they do late in the afternoon, when most students seem to descend on them in desperation, so it's wiser to get started early in the day.

   If you want to use an on-line index, but don't want to pay the search charges, try the *Current Contents*, *INSPEC*, or *Computer Articles* databases on MELVYL. The MELVYL catalog can be reached from almost any machine on the Internet (it's address is `melvyl.ucop.edu`), but you need a password to get access to the specialized databases. The password is available from the reference desks at either the Science Library or McHenry Library.

   Try to find three to five articles relevant to your topic—don't pick the first three you find, however. Find the most relevant and useful ones. Be sure you can read and understand the articles you choose!

3. *Photocopying.* Photocopy the articles you intend to use, or the parts you intend to use, if the article is excessively long. Be sure, however, to photocopy completely (including the notes and references cited) the article that you found most helpful and relevant to your interest. This article will be used later as a model for appropriate citation of sources.

   Whenever you photocopy an article, make sure you have written on the copy a complete citation for the article, so that you can attribute quotes or paraphrases. It is very frustrating to have a perfect quote, and not be able to use it because you have forgotten where you found it. Many journals and conference proceedings make this easier, by providing adequate citation information in the headers and footers of each page.

10.5.2 Organizing the information

1. *Deciding what to look for.* First, make some preliminary decisions about different aspects of your topic. You will need to decide what is most important to put in a survey that is only five pages long. Make a short list of these subtopics: five to seven is likely to be plenty. You will need to have read through all your articles in order to make these decisions. You may change some of these subtopics later, as you go on. Some of them, at least, will be the same or similar to the key words and terms you used in your library search.
2. *Locating the needed information.* How you do the next step is pretty much up to you. You need to go through each article and find the information relevant to your subtopics. Here follow some practical suggestions for doing this. Taking notes from each article is the most traditional way, and was the way everyone did library research before the invention and general proliferation of photocopy machines. Copious note taking is still a very good way to proceed if the material is very new to you, since it helps you learn it.

   Since you have photocopies, however, you may, rather than taking extensive notes, simply mark up the article to locate all the information you intend to use from it. There are less and more complex ways of doing this, ranging from penciled marginal marks to different-colored highlighting for each subtopic. Work something out for yourself.
3. *Ordering your information.* When you have located all the information relevant to your subtopics, you are ready for the next steps.

   First, note any major contradictory factual or theoretical claims. You will need to address any area like this very carefully in your paper, comparing the different claims carefully and faithfully. You don't have to try to decide who is right; you're not an expert. What you need to do is get across to your reader what the different prevailing opinions are.

   Second, find the clearest explanations for each subtopic, by comparing similar parts for each subtopic from each article. You will be basing your survey on these explanations.

   Third, experiment (on paper) with different serial orderings of the information you have found, using simple outlines. When you have found one that seems sensible to you, you are ready to begin to write. This outline will often be visible in the final paper as section headings.

10.5.3 Writing the paper

1. *The initial draft.* Begin writing the main body of your paper. Don't worry about writing the introduction yet. Just start with the first point of your rough sequential outline. You are writing a summary of other people's work, fundamentally, so you will be paraphrasing and citing heavily. Don't lose track of where you have borrowed what from. Get the important authors' names into your text: "Jane Doe shows that ... ." Work your way through this initial draft. The main writing job is drawing the connections between the different subtopics, and summarizing and citing **accurately**!
2. *The section headings.* Read through your draft and split it up into sections by topic. Try to come up with a good, short header for each section. Add these section headers to the draft—they are extremely valuable to the reader.
3. *Writing the introduction.* Now go back and look at your initial topic statement. This is probably now somewhat outdated, but it may provide a basis for your introduction, which you have not yet written. Introductions are fundamentally problem statements. Consult Huckin and Olsen on this issue, in the chapters mentioned above, especially [HO91, Chapter 5]. Write a good, clear, brief introduction to your survey article. Since you have already written the body of the article, and you know what you are going to say in it, the introduction should be fairly easy to do.

   Disclaimer: one of the authors (Kevin) always leaves the introduction as the last thing to write and still has trouble figuring out what to say. It helps a lot if the document has some clear purpose—for example, showing that RISC machines will outperform CISC machines for the next five years, rather than simply comparing the performance of the two design styles. It is very difficult to write an introduction for a paper that rambles from topic to topic.

4. *Revising the initial draft.* Now rewrite your initial draft, reorganizing and rewriting to make the writing smoother and the explanations and transitions as clear as possible. Where you have particularly abrupt changes of topic, you should try rearranging the text or adding a missing section header before you try smoothing over the gap with a transition sentence.

   Your section headers should stand out a bit from the rest of the text. Use bold face if you have it, or underline. If the section headers aren't numbered, you may want to center them. Also, don't put a section header at the bottom of the page—if you don't have room for a paragraph after the section header, start a new page.

   Cut or add as necessary to hit five pages, plus or minus a half page. (double line-spaced pages, standard 10-pitch pica type—12-point on a word processor). You can fudge the length with funny size margins, but we prefer a standard 1-1/2" on the left and 1" on the right (for a text width of six inches). If you are more comfortable with character or word counts than with page limits, this translates to about 9000 characters or 1500 words.
5. *Citing your sources.* Now examine how your most useful article cited its sources. Figure out the system, and do your citations of your sources that way. The reason we have you do this, rather than give you a list of rules and forms, is that every journal does this differently, although there are broad resemblances in citation style within disciplines and between related disciplines. Another reason is that if you have to figure it out for yourself, you will learn how to do it. Bring this draft to class for peer editing.

   Unfortunately, many of the articles you will be reading are from trade magazines rather than refereed journals, and may not have adequate citations. In that case, look at one of the *IEEE Transactions*, the *Journal of the ACM*, or some other major research journal in the field. Citations should be complete enough that an inter-library loan librarian can find and photocopy the article.

   Don't abbreviate journal names, use as much of the authors' names as you can find, and give all the relevant numbers (volume, issue, date, page numbers, ...). For books, give the publisher and date of publication, and for conferences give the dates and location of the conference. Nowadays, citations are usually sorted in alphabetical order by the author's last name, as this order makes finding particular citations easier after you've forgotten where in the text it was cited. You can look in the References section of this workbook (page 74) for examples of one citation style.

10.6 What to bring for peer editing

1. Your draft.
2. Your most useful article, which is your model for citation of sources.
3. A short written list of any questions that have arisen for you while working on this paper.

10.7 Final draft

The final draft should incorporate everything that came out of peer editing, from editorial improvements to more careful citation. You will probably end up reworking your introductory section some, too. If you do any substantial rearranging of your draft on the word processor, be sure to redo the transitions so the seams don't show.

Make sure that your final draft is free of spelling and punctuation errors. Nothing destroys people's confidence in writing more than trivial errors. Your word processor should have some kind of spelling program, and may have a style program that will find sentence fragments and commonly mis-used words. Check to be sure that every item on the list of references at the end is cited somewhere in the text—it is childish to pad your list of references with papers that you didn't directly use. If you have a paper that you feel must be included, but you haven't quoted anything from it, you can add a sentence in your final section, "Further information can be found in ... ."

11. Final project proposal memo

11.1 Goals—proposal writing, choosing the final project

The purpose of this memo is for you and your partner to propose your final project to us. We would prefer to have people work in groups of two for the final project, as it reduces the number of reports that we have to read, and gives you practice in working with another author on a document. Individual projects are quite acceptable, but groups larger than two usually are not.

Your proposal memo will allow us to respond and help you focus and organize your project. We will provide guidance in class about possible topics, and will try provide a list of possible projects around the department that need doing at the moment. The final project can be a description of a design project you have done, a detailed proposal for a senior thesis, user or maintenance documentation for existing software or hardware, or any other technical writing except in-program documentation. (Warning: each year someone tries to write a game specification, and each year the technical content and writing quality have been low—we will question closely any attempts at game specification.)

Ways to generate ideas are included in Huckin and Olsen [HO91, Chapter 2]. Their Section 16.2 offers guidelines for writing a short informal proposal, as well as a sample informal proposal. Their Section 16.3 discusses editing your proposal. Read these sections before attempting this assignment.

Because this assignment is the beginning of a sequence of assignments, including a document specification, a progress report, a formal technical report, and an oral report, you should read those assignments as well as this one. (That is, read Chapters 12–15 of this workbook.)

11.2 Audience assessment—the instructors

Your audience for this assignment is your instructors, and your purpose is to get our advice and approval for your final project. In order for us to give good advice, we need a clear idea of what you propose to do, and how you propose to do it.

Huckin and Olsen's guidelines cover a fuller proposal than you need write for this assignment. For our purposes, if you cover the three areas they suggest for the introduction to an informal proposal, that will be sufficient. These three areas are

1. the problem,
2. the objectives of the proposed work, and
3. the significance of the work.

To cover these areas adequately, you will have to do some preliminary research—you can't just name an abstract topic, like compiler design, and expect approval. Your survey article should have provided you with enough background to narrow the focus adequately—broad enough that you can find information to provide, but narrow enough that you can say something meaningful about the topic.

11.3 Writing process—informal proposals

Even though this memo should be relatively brief and to the point—a page or two at most—and even though it is an informal proposal, these facts do not mean it is casual. Even a memo like this requires planning and a simple outline.

Make notes or an outline of all the things that your supervisor (namely us) would want to know. You should indicate whether you are writing a formal proposal, reporting on an investigation, writing documentation, or doing a library research project. Next, include the subject and scope of your report, using the breakdown provided by Huckin and Olsen.

For joint projects, as most of yours will be, discuss how you propose to divide the work, being specific about exactly what each of you is going to do first, and giving us some idea of the division of labor for the rest of the project.

Bear in mind that you are aiming at an upcoming assignment, the document specification, where you will be making a completely specific division of labor for the whole project. If you are successful in this division of the work, you will not have to change it later, and each of you will do your fair share of the work.

Also keep in mind that most of the areas discussed in your proposal will reappear in your progress report, which is the assignment following the outline. It is normal procedure for your supervisor or sponsor to compare your first major progress report quite closely with your original proposal. We do not mean that nothing can be changed, but rather that you will need to tell us in the progress report what changes you have made in your goals. Reporting changes is discussed in more detail in the progress report assignment (Section 13.3 of this workbook).

12. Document specifications

12.1 Goals—three purposes for document specifications

A document specification contains several parts: a description of the audience(s) for the document, a detailed outline giving the structure and contents of the document, and a work plan showing who is responsible for each part of the document and what the deadlines are for completing each task. For large documents, there may other managerial information, such as number of pages allocated for each section, graphics budgets, printing costs, and so forth. We do not expect you to produce this level of detail in your document specification.

In the workplace, formal document specifications serve three important functions: economy of effort, work planning, and writing organization.

12.1.1 Economy of effort

First, document specifications are used to help you reduce the amount you have to write. When you are requested to write a major report, quite often you will be told something like, "Do a draft and then come see me with it." After talking with your boss, you will have to revise the draft completely so that it will have in it what the boss really wants, since he/she hadn't thought about the project carefully until reading your draft. You can often save yourself (and your boss) much work if you write a detailed document specification instead. A document specification is much easier to create, change, revise, and add to than a draft is.

12.1.2 Work planning

The second major function of a document specification is work planning. This can mean either budgeting your own time, or, in large formal reports, distributing the work among many people. Multi-author writing is probably the most common form of workplace writing. The more you know about the specification, the more likely you are to get only your fair share of the writing and no more.

For the purposes of the class, the document specification will serve to distribute the work between you and your partner. You should bear three things in mind in dividing the work: even work load, respective areas of expertise, and getting things done in the order that you need them. These points sound obvious, but achieving a fair distribution of the work is not always easy. It involves keeping to a schedule, for one thing, and staying in close touch with your partner, for another.

12.1.3 Writing Organization

The third major function is the organization of the report itself. This function is discussed in Section 12.3.2 below.

12.2 Audience assessment

As with the proposal, we are the supervisory audience for this document specification. However, we are **not** the audience for your final report. Don't get confused on this point. If you are writing user documentation for software meant to be used by first-year students in a chemistry lab, **they** are your audience, not us. For more on this issue, refer to the final report assignment (Chapter 14).

The document specification you are writing is a working document, meant to allow us to help you with your project. The fuller and more complete your document specification is, the more we can help you.

A tool that is useful for doing the audience assessment for your final report is a *user matrix*. Each row of a user matrix corresponds to one group of readers for the final document, and each column corresponds to one topic in your report (perhaps a section heading). Each entry in the matrix indicates how important that topic will be to that group of readers. For example, in hardware documentation the parts list may be essential to Manufacturing, but useless to Marketing.

You can use any quantification scheme you want for the entries of the matrix, but there is no need to get fancy. In most cases, a single bit (needs to read this/doesn't need to read this) is sufficient.

Before writing your document specification, try to make a list of the different readers you expect for your final project. At the very least, you should include your classmates, as they will be the audience for your oral report. If your project is the final project for some other class, you should include the instructor for that class and students who are planning to take the class.

Once you have the main topics of your report identified, you can use the user matrix to help you order them. Try to put the most important information first—the things that everyone needs to read should be near the beginning, and the things that only a few readers need later or in separate appendices. If possible, try to arrange things so that most readers can read a contiguous chunk of the report, not having to skip irrelevant pieces.

The user matrix is an important part of the document specification, so please include it in what you turn in.

12.3 Writing process—document specifications

12.3.1 Outlining is organizing

Some of the material that follows is also found in the final project assignment. The material will be repeated because it is about organizing technical reports. Here the stress will be on the detailed outline in a document specification as a management tool for writing technical reports.

Outlining is organizing. This is a point sometimes obscured in presentations that get bogged down in what type of number to use for the third innermost nested subhead. We assume here that you can figure out how to do that kind of thing in a clear and consistent way, and concentrate instead on the underlying principle involved—organization of technical reports, and organizing the process of producing them. Organization, of the writing process and of the report itself, is the basis for two of the three major functions of document specifications.

In preparation for this assignment, read Chapters 12 and 15 in Huckin and Olsen, where a somewhat similar scheme for organization is discussed. Their scheme differs mostly in detail, not in principle. The process and the product are the same. If Huckin and Olsen's scheme makes more sense to you, then use that one.

12.3.2 The basic tripartite structure of a formal report

The outline that follows is the core of the one presented in the final project assignment (Chapter 14). It displays the basic tripartite structure of most technical reporting. This outline is very general, and very abstract, and is designed to show you the internal structure of virtually all technical reporting, not provide you with headings for your own document specification. These headings used here show internal structure, but they have no technical content.

Your headings must contain the primary content of your report. Generic headings, like the ones used here, will not be acceptable.

1. **Front matter** Each of the sections of the front matter should include a brief version of your problem statement and your recommendation, if appropriate. (See parts 2.2 and 3, below). Most documents use only two or three sections in the front matter—it depends on what you are writing.

   1.1 *Executive summary.* In the workplace, your recommendation is the most important part of an executive summary, and should immediately follow a brief statement of the project or problem.

- **1.2** *Abstract.* It is unusual to have both an abstract and an executive summary, because both serve essentially the same function in different types of reports.
- **1.3** *Preface.* A preface usually contains a description of the target audience and intent of the document, and a list of acknowledgements for contributions made by people other than the authors.
- **1.4** *Introduction.* An introduction is used to explain the background or significance of the main part of the paper.

2. **Technical Discussion**
   - **2.1** *Problem Statement.*
     - **2.1.1** What was studied, designed, built, or investigated
     - **2.1.2** What the purpose of the study, design, or investigation was
     - **2.1.3** What you found out
   - **2.2** Full explanations for Heading 2.1, above, that is
     - **2.2.1** How you did it, including, as appropriate, tests, procedures, methods, materials, processes, equipment, programs (if they're short, otherwise leave them for the Appendices)
     - **2.2.2** What happened, including, as appropriate, test results, data collected, full description of design(s), procedures modified, processes completed
     - **2.2.3** Your interpretation of your work
3. **Recommendations** (especially important in the workplace: finding out what to do is why they have you do the report)
   - **3.1** Description of solution to problem or proposal for change
   - **3.2** Benefits to the company
   - **3.3** Rejected alternatives

   For research reports, this section is often titled "Future Work", giving recommendations for new ideas to pursue. A design document may end with suggestions for improvements to make, a marketing plan, or whatever else needs to be done with the design. User's documentation will often end with suggestions for further reading for those interested in more details.

12.3.3 Organizing the work and the writing

The sequence described in Section 12.3.2 is typical of the sequence of topics in a technical report. It is **not**, however, the sequence in which the report is written or the work is done. In terms of organizing work and writing, this is the crucial point: the work, and the writing, begins in the **middle**!

Everything begins in the middle because, if you look at part 2.2, you will see that this section deals with things that engineers actually *do*. Here you are working with ideas for your project based on your proposal, which, once you have done the work and know how it really came out, you use as a basis for writing part 2.1, the Problem Statement.

The work in part 2.2, and the writing that goes with it, are the first things that you and your partner need to divide up. Divide both work and writing. For this paper, you may very well be documenting an already completed project, in which case you will be dividing just the writing, but be sure to go over each step of the project together, so you both agree on what you think was done. What documentation actually exists almost certainly amounts to notes for part 2.2 of the Technical Discussion, so work with them accordingly.

Thus, the first writing that gets done covers the areas in part 2, Technical Discussion. Your interpretation of your work (part 2.2.3), leads to the next section normally written: Recommendations (part 3). Once you have this figured out, you can go back and write the Executive Summary (part 1), because you now know what the work you did was designed to find out or produce, namely, the answer, the solution, the usable design, which provides the basis for your recommendation.

Since this is what you are being paid to figure out, it goes right up front in the Executive Summary, where your employer can see it!

12.3.4 Which projects do not use the *Tripartite Structure*?

The outline in Section 12.3.2 is the basic tripartite structure of formal reports. This structure is especially appropriate for documentation of your own research projects, what we are calling here design documents. However, if you are doing **user** documentation, your format is likely to be more like the formats used in the naive-user documentation assignment (Chapter 8). If you're doing user documentation, review Chapter 8, re-read your notes from the guest lecture, and take a good close look at *UNIX for Luddites* again.

12.4 What to turn in

The document specification should be written as a memo to us. Leave plenty of white space, so we have room to make suggestions and amendments.

You and your partner should turn in to us as full and as explicit a document specification as you possibly can—the Technical Discussion section should be particularly detailed. It should also identify, item by item, who is doing and writing what.

By now, your project should have a title. Use it on your document specification.

Turn in your user matrix. We'd like to see who you are writing for, and what you think they need to know.

Turn in your project proposal again with the document specification. We want to see that your document specification matches your proposal—if it doesn't, you should have an explanation of the discrepancies, perhaps even a modified proposal. We cannot be expected to remember the details of all the projects being done!

13. Progress Report

13.1 Goals—writing progress reports, making sure there is progress

This assignment is an opportunity for you to tell us, in writing, how you are doing on your final project. Put your progress report into the form of a memo.

The example in Figure 13.1 shows the format, but not, we hope, the content of a progress report. The body of the memo was written by William Cohen of China Lake, California, as an example of the many cliches and circumlocutions that appear in technical reports. Translations of the phrases appear in parentheses.

13.2 Audience assessment—instructors and supervisors

Your audience for this assignment is your instructors, and your purpose is to tell us what progress you've made on the final project. In industry, you will often need to file progress reports with your supervisor. The goal of such progress reports is either

- to convince your audience that you are making progress, that it's the proper progress, and that you will finish on time, or
- to explain problems and to request assistance or guidance.

> **To:** Kevin Karplus and Sharman Murphy, Instructors
> **From:** A panicky student
> **Date:** 26 Feb 1993
> **Subject:** Progress on the report that hasn't been started
> During the current reporting period, considerable progress has been made in the preliminary work directed toward the establishment of the initial activities. (*We are getting ready to start, but we haven't done anything yet.*) The background information has been surveyed and the functional structure of the component parts of the cognizant organization has been clarified. (*We looked at the problem and decided that George would do it.*) Considerable difficulty has been encountered in the selection of optimum materials and experimental methods, but this problem is being attacked vigorously, and we expect that the development phase will proceed at a satisfactory rate. (*George is looking through the handbook.*) In order to prevent unnecessary duplication of previous efforts in the same field, it was necessary to establish a survey team which has conducted a rather extensive tour through various facilities in the immediate vicinity of manufacturers. (*George and Harry had a nice time in New York.*) The Steering Committee held its regular meeting and considered rather important policy matters pertaining to the overall organizational levels of the line and staff responsibilities that devolve on the personnel associated with the specific assignments resulting from the broad functional specifications. (*George and Harry were complaining about their jobs at the corner bar again.*) It is believed that the rate of progress will continue to accelerate as necessary personnel are recruited to fill vacant billets. (*We'll get some work done as soon as we find someone who knows something.*)

Figure 13.1: Example of good memo form and bad style in a progress report (by William Cohen).

13.3 Writing process—short progress reports

Even a brief, one-page memo requires planning and perhaps a simple outline. Make notes or an outline of all the things that your supervisor (namely us) would want to know. You should include a brief description of the general outline of the report, whether you are writing a proposal, reporting an investigation, writing documentation, or doing a library research project. Next, include the subject and scope of your report, including the major areas your table of contents will cover, and then tell us how you're doing on it. Talk about any research you've done, how the actual writing is going, and any problems you've run into. Don't be afraid to mention problems–there may be something we can help you with. Include any information you find relevant.

For joint projects discuss how the division of the work is working out, if you think changes should be made, and if so, what.

Be sure to tell us, right at the beginning, if you have changed the topic or the scope of your report in any substantial way since we approved your proposal for a final report. These are the kind of changes that we need to know.

13.4 Professional Ethics

13.4.1 Honesty is the basis of professional ethics

Progress reports call for complete honesty. If you have good reason to believe that a project will not be completed on time, say so, and explain what additional resources would be needed to be able to complete it. Your manager may want you to write glowing progress reports, then when the project falls apart, he can point at the reports and say that so far as he knew everything was on track, and that the whole mess is the fault of lying subordinates. Don't get stuck with the blame!

Even when routine progress reports are not required, you may want to write one when a project is going badly, to inform management of a problem. There is always some risk in being the bearer of bad news, as the managers will start looking for someone to blame long before they start looking for a solution.

Sometimes you will be asked, with more or less subtlety, to aid in a deception—to make things look better either to higher management or to a customer. Resist such requests! A firm "I can't do that, it would be dishonest" is often all it takes to avoid some very messy situations. As an honest, competent engineer, you are far more employable than the dishonest managers who would ask you to compromise your principles—if they insist, you can walk away from the job, writing a letter to higher management explaining why you cannot work for the company any more.

It is much easier to resist pressure if you have enough money in the bank to live for six months without a job. Build up that cushion before you buy a new car, a new stereo, or a new house. Even if you never have to quit for ethical reasons, layoffs are common in the computer and electronics industry. Usually you can get re-hired somewhere else fairly soon, but having some money in the bank allows you to pick your job a bit more carefully.

Huckin and Olsen also have some mention of ethical issues for engineers [HO91, Chapter 2].

The next section includes the 1979 text of the IEEE Code of Ethics for electrical engineers. The Code was revised recently, mainly in attempts to strengthen the wording about engineers' responsibility to society (see Section 13.4.3). Other than some rather obvious self-serving provisions, the 1979 code is still a pretty good one. We will discuss aspects of the code in class—please read it carefully to identify parts that you disagree with, that are open to misinterpretation, or that emphasize one aspect of integrity at the expense of another.

13.4.2 1979 IEEE Code of Ethics

Preamble

Engineers, scientists, and technologists affect the quality of life for all people in our complex technological society. In the pursuit of their profession, therefore, it is vital that the IEEE members conduct their work in an ethical manner, so that they merit the confidence of colleagues, employers, clients, and the public. This IEEE Code of Ethics represents such a standard of professional conduct for IEEE members in the discharge of their responsibilities to employers, to clients, to the community, and to their colleagues in this Institute and other professional societies.

Article I

Members shall maintain high standards of diligence, creativity, and productivity and shall

1. Accept responsibility for their actions;
2. Be honest and realistic in stating claims or estimates from available data;
3. Undertake technological tasks and accept responsibility only if qualified by training or experience, or after full disclosure to their employers or clients of pertinent qualifications;
4. Maintain their professional skills at the level of the state of the art, and recognize the importance of current events in their work;
5. Advance the integrity and prestige of the profession by practicing in a dignified manner and for adequate compensation.

Article II

Members shall, in their work,

1. Treat fairly all colleagues and co-workers, regardless of race, religion, sex, age, or national origin;
2. Report, publish, and disseminate freely information to others, subject to legal and proprietary restraints;
3. Encourage colleagues and co-workers to act in accord with this Code and support them when they do so;
4. Seek, accept, and offer honest criticism of work, and properly credit the contributions of others;
5. Support and participate in the activities of their professional societies;
6. Assist colleagues and co-workers in their professional development.

Article III

Members shall, in their relations with employers and clients,

1. Act as faithful agents or trustees for their employers or clients in professional and business matters, provided such actions conform with other parts of this Code;
2. Keep information on the business affairs or technical processes of an employer or client in confidence while employed, and later, until such information is properly released, provided such actions conform with other parts of the Code;
3. Inform their employers, clients, professional societies, public agencies, or private agencies of which they are members or to which they may make presentations, of any circumstance that could lead to a conflict of interest;
4. Neither give nor accept, directly or indirectly, any gift, payment, or service of more than nominal value to or from those having business relationships with their employers or clients;
5. Assist and advise their employers or clients in anticipating the possible consequences, direct and indirect, immediate or remote, of the projects, work, or plans of which they have knowledge.

Article IV

Members shall, in fulfilling their responsibilities to the community,

1. Protect the safety, health, and welfare of the public, and speak out against abuses in these areas affecting the public interest;
2. Contribute professional advice, as appropriate, to civic, charitable, or other nonprofit organizations;
3. Seek to extend public knowledge and appreciation to the profession and its achievements.

13.4.3 1990 IEEE Code of Ethics

The Board of Directors of the IEEE approved a new code of ethics at their August 1990 meeting [IEE90], to replace the rather wordy 1979 one. Note how the writing style has changed, and how much terser and crisper the new code is.

> We, the members of IEEE, in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession, its members and the communities we serve, do hereby commit ourselves to conduct of the highest ethical and professional conduct and agree
>
> 1. to accept responsibility in making engineering decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;
> 2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;
> 3. to be honest and realistic in stating claims or estimates based on available data;
> 4. to reject bribery in all its forms;
> 5. to improve the understanding of technology, its appropriate application, and potential consequences;
> 6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
> 7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;
> 8. to treat fairly all persons regardless of such factors as race, religion, gender, disability, age, or national origin;
> 9. to avoid injuring others, their property, reputation, or employment by false or malicious action;
> 10. to assist colleagues and co-workers in their professional development and to support them in following this code of ethics.

13.5 Assignment—final draft only

The progress report is short enough that we do not expect you to take several days preparing it. Because the progress report has significant time value (we want to know the current progress–not last week's), we do not ask that you bring in a draft for comments before turning in the final memo.

Please use standard memo format [HO91, Chapter 13].

Don't forget to tell us the title of your project, preferably in the "Subject" field at the beginning of the memo—we can't remember exactly what everyone is doing!

As always, retain the returned memo to turn back in at the end of the quarter.

14. Final project

14.1 Goals—formal report writing

The main goal of this assignment is to learn how to write a formal report in several defined steps, each of which is a separate sub-assignment. You will have done, in sequence, the *Proposal Memo*, the *Document Specifications*, and the *Progress Report*. This chapter will discuss the final form of the *Final Project*. Some of what is covered here was also discussed in the document specification assignment, although largely from the perspective of organizing the writing process. Here we will be discussing the sequential organization of the report itself, and the reasons for it. The relevant sections of Huckin and Olsen are Chapters 12 and 15 and Sections 9.3 and 9.4 [HO91].

Your final project can be a design description (for a design you did yourself), a library research paper, a formal proposal for a future design project, or user documentation.

14.2 Audience assessment—choose your own

You must define your own audience for this project.

If you do a design description, your audience might be the management of your company, or one of your professors, or students who will be doing similar design projects in future years. Writing for future students is probably the easiest, as you can have a better idea what things they will be interested in.

For a library research report, your audience might be the readers of a magazine in which you intend to publish the survey article. (If you pick this sort of audience, specify the magazine or journal you're aiming at.)

If you are doing user documentation for any of the things suggested on our project list, think carefully about the real audience, which differs considerably from project to project—some are for beginning students, some for advanced students and professors. Remember that the audience is not Kevin or Sharman, or the professor you have contacted about the needed documentation.

Keep in mind that for the oral report based on this assignment, your audience will be the class.

14.3 Writing process

14.3.1 Writing the Report

We will go over the elements of a formal report, and the reasons for them, in class. The tripartite structure discussed below is especially appropriate for documentation of your own research projects, what we have been calling design documents in class. If you have any questions as you are working, refer to this chapter or to Huckin and Olsen, Chapters 12 and 15 [HO91], for the structure of formal reports.

Some moderately useful stuff for library papers can be found in Chapter 16 of Huckin and Olsen, but you should also consult one of the other texts on reserve for this sort of paper. McGuire and Putzell [MP88, Chapter 12] and Trzyna and Batschelet [TB87, Chapters 3 and 4] may be useful. Also, your experience with the survey article is relevant. Look over Chapter 10 again, as well as your notes from the guest lecture by the reference librarian.

If you are doing user documentation, as we mentioned in the document specification assignment, your format is going to be more like the ones used for naive-user documentation, so review Chapter 8, your notes from Scott Brookie's lecture, and look at *UNIX for Luddites* again. Also, the section entitled "Manual Writing" in Trzyna and Batschelet [TB87, Chapter 12, pages 281–297] has a good deal of useful general advice and some helpful examples.

14.3.2 The tripartite structure of a formal report

As discussed in the documentation specification assignment, what follows is intended to be general, to provide an idea of the basic components of a formal report. It is not an outline for you to fill in, especially since some of the headings name functions, rather than being titles for sections. You will have to choose which things you need to include. You are unlikely to need all four items in Section 1, for instance, and you may or may not want to include a letter of transmittal to the contact person for your project. See pages 220–223 of Huckin and Olsen for an explanation of letters of transmittal [HO91, Section 11.3]. You might have to include a quite substantial appendix or supplementary section, depending on the project.

Front Matter Letter of transmittal, title page, contents, list of figures, list of tables, and acknowledgments.

1 Executive summary, abstract, preface, and introduction.
See Section 12.3.2.

2 Technical Discussion
See Section 12.3.2.

3 Recommendations
See Section 12.3.2.

Appendices and Attachments bibliography or list of cited works, tables, documents, data, source code, maps, letters, testimonials, brochures, resumes, index—depending on what you are writing. Usually the list of cited works (labeled *References*) comes first, and the index comes last. The others come in whatever order is appropriate, and are called *Appendices*.

Versions and modifications of this outline may be used for your final reports. Modify this outline to suit your project—**don't** force your paper to fit the outline. You may find that the main body of your report divides naturally into four or five parts instead of three; that's fine, but ten parts would not be fine. Be wary of subdividing into too many parts at one level, as that indicates that you haven't properly grouped related subjects together.

Most important, give each of your sections meaningful headers, not the generic headers we used in the outline.

The outline above is a guide, not a rule. Compare it with the schemes in Huckin and Olsen, Chapters 12 and 15. There are similarities and differences, but the underlying ideas are consistent.

14.3.3 Explanation of *Tripartite Structure*

The outline in Section 14.3.2 is the basic tripartite structure of formal reports. The three parts of the tripartite structure are the beginning, the middle, and the end, not surprisingly. As we discussed in the outline assignment, starting down in the bowels of the report, the *Technical Discussion* has three parts: 2.1, 2.2, and 2.3. But the *Technical Discussion* itself is the middle, or the body, of a larger structure, where the beginning is part 1, and the end is part 3.. This whole piece, parts 1–3, is the main body of your report and forms the "middle" between the Front Matter and the Attachments for the entire report.

Besides being organized in nested "middles", between beginnings and ends, you will notice that the document progresses serially from the simple to the complex, and repeats itself in increasingly complex ways at least once, and more likely twice. For example, the simplest text is going to be in the executive summary, and this may be as far as some management readers will get. The most technical material may very well be a long list of design specifications in an appendix, and may not even be recognizable for what it is by some readers.

The purpose of this feature, the organized and repetitious progress through ever more complex explanations of the same thing, is that technical reports have many audiences with many purposes. The management reader is concerned with allocating resources, both money and personnel, and planning for the organization as a whole. The executive summary tells her/him what projects have become feasible, what problems have been solved, what new products are possible. This is the kind of information on which management decisions are based.

The production engineer who will implement the new design, however, may very well skip most of the beginning material, go to the heart of the technical discussion, read it quickly so she/he will understand what's going on, and then move on to the detailed specification of the new design. On the other hand, another engineer with a similar design problem may consult only the technical discussion, looking for ideas for solving her/his somewhat different problem.

Overall then, organization of a technical report in this fashion serves the needs of all the audiences who will consult the report.

14.3.4 A note on page numbering

Every page after the title page must be numbered, and your table of contents must list the page numbers for at least the major sections of the report. There are many popular styles for page numbering, and you may choose whichever you like, as long as you use it consistently.

Many computer scientists prefer a page numbering that starts on page 1 (page 0 is the title page), and continues sequentially until the end of the report. This style is easy to generate with word and document processors, and is easy for the reader to understand. When printing on both sides of the paper, the odd-numbered page is always the right-hand page, and the even-numbered page the left-hand page.

The more traditional scheme uses lower-case Roman numerals for the front matter, and starts page 1 at the beginning of the body, numbering sequentially from there. It is a holdover from when books were typeset by hand, and the front matter was often not written when the main body was being typeset. Some traditionalists still insist on this scheme, even though the technical motivation for it has almost disappeared.

One compromise scheme sometimes found in frequently modified documents (such as computer manuals) is to number separately within each chapter, so that Chapter 1 starts on page 1–1, Chapter 2 on page 2–1, and so forth. In traditional formats, appendices are often numbered this way, with Appendix A starting on page A–1. Separate numbering for each chapter allows updating a manual by replacing one chapter, rather than replacing the entire manual.

14.4 Oral Reports

You may still be working on your draft when you give your oral report. This is an advantage; you can incorporate any feedback you get from the class in your final draft. Wise students will do their oral presentations as early as possible. In industry, oral presentations (sometimes more than one) are often given before the final draft of a report is submitted.

14.5 What to turn in

The due dates for the assignments are given in the syllabus.

Draft 1 The first draft of your project is due in class. We will work on them extensively in class that day. This session will be concerned primarily with organization of material, adequate explanation, and inclusiveness.

Draft 2 The second draft of your project is due in class. This time we will be concerned with formatting, and a host of editorial questions involving clarity of presentation and explanations. The nitty-gritty stuff.

Final The final project is due, no later than 5:00 p.m., in Kevin's office. You will turn in the following:

- the final version of the report (20–30 pages, a bit shorter for one-author reports), produced as nicely as you can,
- **both** earlier drafts, with comments by your partners,
- **all previous assignments** (so we can look at them when writing narrative evaluations).

15. Oral Reports

15.1 Goals—clear oral presentation

Oral reports are often a form of advertising—either internally, within a company, where people from different departments see projects displayed by their creators, or externally, where a company presents something to people from outside the company in an attempt to sell it. Oral presentations are often an opportunity for people who read your reports—senior engineers, managers, sales staff—to see who you are and how you work. This means they're as interested in you, and how you handle yourself, as in the material you present. Therefore, present a problem and a solution with just enough technical detail to make your point. What you want to do is let these people know how competent you are, and what sorts of work you do. You will be showing them your thinking style, your working patterns, and your problem-solving ability. With this in mind, try to put on the most professional, persuasive, confident display of your project that you can.

15.2 Meeting the expectations of the audience

Your oral report is a presentation of the material covered in your final written report. You will have seven minutes to present your report. You may use less time, but you may not exceed the limit. In industry, oral reports are often given before consultants, who charge extravagantly for their time; therefore, you must learn to budget the time allotted you and not exceed it. You will know in advance precisely when, day and time, to the minute, you will be expected to give your report. Be ready to go as soon as you stand up—don't waste time with preparatory goofing around.

Rehearse your talk ahead of time. Practice turning transparencies, writing on the blackboard, or using whatever visual aids you choose.

15.3 Presenting your report

Read Chapter 19 (Oral Presentations) of Huckin and Olsen. They give some good basic advice on planning and presenting an oral report.

15.3.1 Using notes, transparencies, charts, and displays

Oral reports are not the best nor the easiest way to present technical information, because the information is difficult to assimilate, often full of statistics, numbers, equations, and so forth, which are hard to follow when given verbally. Therefore, using transparencies, charts, tables, graphs, diagrams, and illustrations can help a great deal. Don't hesitate to use them.

Overhead projector transparencies are commonest in computer science and computer engineering for giving presentations to audiences with fewer than 100 people. There will be an overhead projector available in the classroom for your use.

Here are some tips on using visual aids and notes:

- When using transparencies, don't leave a blank screen, don't flip back and forth between transparencies, and don't cover up substantial parts of your transparency with opaque objects.
- You can point to important things either on the screen (by standing to one side and pointing with your arm or a long stick), or on the transparency (by standing beside the projector and using a thin opaque object, such as a pencil). In either case, be careful that you do not block the image on the screen.

- You may hand-write your transparencies, or photocopy a black-and-white original onto plastic. In either case, you should not have more than 10 lines of text on a transparency. You should use 24-point fonts (1/3" high or larger if you are writing by hand) for standard text, and 18-point fonts for "fine-print". Anything smaller will not be readable in the back of the room. A handy guide is that the unenlarged transparency should be readable from ten feet away. If you are using 35mm slides, the equivalent rule of thumb is that the slide should be readable at arm's length. Use the most readable fonts you can find, not fancy, decorative fonts.
- Many speakers like to write on their transparencies as they talk, highlighting the points dynamically. If you plan to write on a transparency produced by photocopying, wipe it off with a damp rag first. This wipes off any oily residue from the photocopying process, and eliminates the static that can cause transparencies to be difficult to separate.
- The usual rule of thumb for transparencies is that you should talk about each one for one to two minutes. That means that you should have from five to ten transparencies for a 10-minute talk, not counting the title-page transparency (if you have one).
- You may use notes to talk from, but don't stand up, bury your face in your notes, and read them verbatim. Write your notes in outline form, with just the major points you want to cover listed. If you're afraid of forgetting major topics, have something on your transparencies for each one, and use them as your mnemonic device. Using your transparencies as the notes for your talk reduces your reliance on hand-held notes and increases the eye contact you have with your audience. It looks much more professional to know your material well enough not to need a sheaf of notes.
- It often helps to have a transparency near the beginning that amounts to an outline of your presentation. This will not only help you stay on track, but it will help your audience follow what you are doing and see where you are going.
- This assignment is a speech, not an exercise in reading aloud. You should be somewhat spontaneous in your delivery—the audience will be much more with you if you are. If you're shy about looking out into the audience, pick one person to look at. It's important not to speak to the blackboard, or the door, or the ceiling. If you look out into the audience, people will feel as if you are really speaking to them, and they will be much more receptive to what you have to say.
- You aren't expected to present a lot of data. If someone wants all the data, he or she can read your written report. You are to give an idea, an overview, of what your project is about in your oral presentation. Be prepared to answer questions regarding specific data either with a pre-prepared transparency, or with an offer of a specific time and place when you will provide the data.
- If your report involves interpreting some complex set of data that your audience needs to see, have it ready on a handout that you may distribute, rather than writing it on the board, or expecting your audience to read it from a displayed table. Graphs are much easier to comprehend at a glance than tables, so you can usually present them on transparencies without hardcopy.

15.3.2 Practicing Your Presentation

Practice presenting your report at least once before presenting it in class. Time your talk so you know exactly how much material you can fit in, and how to pace the material. Don't just read your notes to yourself—stand up and give the talk the way you will to the class. You will find it very difficult to speak clearly at your normal silent reading speed.

You will probably find, if you are like most people, that you have too much material. If possible, practice presenting your report in the room where you are going to present it formally. Learn to fill the room with your voice, as described in lecture. Concentrating on the sound of your voice will also help you not to be nervous when you are presenting the report. Nervousness may make you speak faster or slower than in your rehearsal. Be prepared with a little extra material, in case you speak too fast.

If English is not your native language, and you find speaking without a prepared text difficult, or if you are overwhelmingly nervous about speaking publicly, it is all right to write your presentation out verbatim and memorize it. However, you must treat memorizing a report in the same way you would treat memorizing a play script. This means that you must memorize it with normal pauses, emphasis, and intonation, and take special care not to speak faster than the normal speech rate. If humanly possible, don't do your report by memorizing a speech—it is not the best way. Huckin and Olsen have some advice and exercises for helping non-native speakers with pronunciation and intonation [HO91, Chapter 38]. Confidence in your understanding of your material, and taking your time will make up for a lot of awkward English and hyper-nervousness. So will substantial practice presenting your report.

Get enough sleep the night before. I have seen someone present a paper at a professional conference after running on adrenalin for a few days, then pass out and fall off the platform when he was asked a question.

Above all, remember that in an oral presentation, you must make each major point in several ways. The old saying about this is, "First you tell 'em what you're going to tell 'em, then you tell 'em, and then you tell 'em what you told 'em." Of course, using exactly the same words each time does not help comprehension—what you are trying to do is to find the explanation that works for each member of the audience, and different people in the audience will understand different explanations.

15.4 Evaluation of the presentation—form and content

15.4.1 Form

You will be evaluated on the basis of what may be generally called "professional behavior". This means being prepared, being dressed appropriately for your audience, speaking clearly and loudly enough, making eye contact with your audience and maintaining it, handling questions with grace, using equipment and displays skillfully, and generally making your audience comfortable.

Note that being dressed appropriately for a presentation to fellow students or engineers is quite different from being dressed appropriately for a presentation to venture capitalists or the Board of Directors for a large company. You may choose what style of presentation you wish to give in class, but let us know, so that we may judge it by the appropriate standards.

15.4.2 Content

Content means having a well-organized presentation, presenting enough information to make your point but not overwhelming your audience with irrelevant detail, and being accurate about what you say. Many speakers, in their rush to get as much information as possible into their talk, lose their audiences almost immediately, and end up conveying no useful information. Pace your material to the rate at which your audience can understand it.

Some people have been taught that it helps to "soften up" your audience with a joke. Humor is valuable when it is relevant, that is, if the point you are trying to make can be illustrated with a short joke, go ahead and use it. Be careful not to waste any time on irrelevant jokes—you're not auditioning for a job as a stand-up comic, you are trying to convey as much information as possible in a short time.

15.5 Scheduling the talks

The oral presentations will be given during the last two weeks of class. You will sign up in advance for your day and time.

We will have ten speakers a day, seven minutes each. This means that two-person projects will have fourteen minutes, and we expect the time to be divided roughly equally between speakers. We may need to use the final examination period.

Because class periods are only 70 minutes long, the first person will have to start on time, and we'll have to keep a strict schedule. There will be no room for running overtime, and no time for setting up between talks.

Please come every day, even when you are not speaking—on those days you are the audience! Don't let your classmates down. We will be especially attentive about attendance during oral presentations.

The best thing is to do your presentation as early as possible, rather than as late as possible, because you will have the benefit of having done the presentation when you make final revisions in you final project. Also, being human, we tend to judge the presentations against the ones we have heard previously, so we are going to expect more of the later presentations.

15.6 Note on oral presentations at SIGGRAPH

The following article is a column by Jim Blinn from *IEEE Computer Graphics and Applications* containing suggestions about presenting technical material to a large audience [Bli88], reprinted with the permission of the publisher.

Things I Hope Not to See or Hear at SIGGRAPH

No, I'm not going to talk about flying logos or glass balls. I am going to talk about that special form of performance art known as Giving a Technical Presentation. These ideas apply to speakers in panels and tutorials as well. I realize that the direct audience for this subject is somewhat small, but others of you might be able to use this information in your own talks elsewhere. Also, you should expect this from presentations you hear at SIGGRAPH. SIGGRAPH sends out a lot of stuff about how to prepare visuals, etc. There is no good excuse for not reading it, although from what I see, not many do. The following ideas are just my own personal biases. I will phrase many of them as things not to say or do, because let's face it, it's a lot easier to complain.

Talks read verbatim

A technical talk is just one facet of a multimedia event built on your work. An adventure story appears different in the film version and the book version. Likewise, different things are more appropriate for the spoken version of your paper than for the printed version. A much more conversational style is best for the talk. Tell a story about what got you interested in the problem in the first place. Briefly relate some routes that you tried that turned out to be dead ends. But please don't read your paper verbatim. We are people out here in the audience; we're all your friends; just talk to us. The only exception to this rule is if you are not a native English speaker. If you are not fluent in English, it is probably best to have your words already prepared.

Illegible slides

The most important part of your talk is the visuals; this is SIGGRAPH after all. I am sometimes amazed at how many illegible slides are shown, most especially by representatives of organizations (who shall remain nameless) that sermonize about high-quality imaging. Here are some things that have disturbed me most about slides I have seen.

Microtext

Many of you are involved in the microcircuit revolution and tend to think this also applies to the text on your slides. It doesn't . My personal rule is to put no more than six lines of text on any one slide. And while you're at it, use the biggest font you can that will fit on the slide. Six lines of teeny-weeny text with gigantic borders is still not readable. But, you may ask, what if I have more than six lines? Well, just use more than one slide. See? Simple. A good check for readability of slides is to hold them at arm's length and see if they are still readable[1]. (That is what I do, and my arms are probably longer than yours.) Believe me, that is how small they look from the back of the room. In fact, I make all my slides on my animation system, which has only video resolution. This may seem to be a disadvantage, but it's not. It forces me to keep the slides simple enough to be legible from a long distance. One effect of this restriction concerns equations. You simply can't have a complex equation on a slide. Even if you shrink its many terms down so they will fit, it will look like grey noise from the back of the room. Recast your equations into simpler chunks and give each chunk its own name. Make one master slide with the basic equation in terms of these names. Then make a separate slide to define each chunk. Don't put more than one equation on a slide unless it is fantastically necessary. Use separate slides for each equation to focus attention while you are talking and give you more room for each one.

Yellow lines on a white background

Another design issue concerns colors and contrast. Your best bet is to use some dark background (like a dark blue) with very light color text (like white or yellow) on it. In any event the main idea is to keep high contrast between the text and the background. Even then, I have seen some terrible slides that use black letters on a white background. Even though the letters were big, the slides were illegible because the lines were too thin. Light areas seem to expand visually, so dark lines tend to get eaten up by a white background. If you must use light backgrounds, use a much thicker line width for the dark lines to compensate for this phenomenon. If you want to emphasize some items on the slide, make them in a lighter color than the rest (not just in a different color).

The entire text of the talk echoed on slides

The audience is not going to want to read a lot of text while simultaneously trying to pay attention to what you are saying. Text on slides should consist of just section headings. If you have a section of your talk that you don't have any obvious graphics for, don't feel compelled to put the text you are reading on a slide just to have something there. The days of silent movies are over. If you must have something, try showing a picture of a pretty waterfall. And remember, folks, no overhead transparencies allowed. There is a reason for this: They look terrible no matter what you do.

"I'm sorry these slides are so dark."

I don't think I have ever seen a slide at SIGGRAPH that was overexposed When you film your efforts, make several exposures and pick the brightest one. In general err on the side of overexposure; make the exposures longer than you think will be necessary. But for heaven's sake if, despite my sage advice, your slides don't come out bright enough, don't make a big production out of apologizing for them. It doesn't make them any more readable, and may just call attention to problems that might not be as noticeable as you thought. Just show them and get on with the talk. Remember that your view of the slides from where you speak

[1] This suggestion is for *35mm slides* not $8\frac{1}{2}'' \times 11''$ transparencies—stand back 8–10 feet from transparencies to check readability.

is not the best one. The slides will look a lot brighter to the audience than they do to you. Likewise, don't spend a lot of time fiddling with the focus (which requires shouting at the AV people in the back of the room). In the first place, your slides should be big and bold enough that a little bit of out-of-focus shouldn't bother them. Remember, from the back of the room the screen looks like a postage stamp. Problems with focus that appear bad to you, with your nose three feet from the screen, won't show up to the audience. Talking about and taking time with these issues detracts from your presentation.

The tops of the speakers' heads

No, I'm not saying this because I'm tall. I mean that speakers should look straight out at the audience instead of burying their noses in their notes. I know it looks like a black hole out there, what with the dim house lights and the spotlight on you. You can't really see the audience, but there are people there. If you look down all the time, all that people will see is the top of your head. This is so important that I'll say it again. Look up at the audience; it looks a lot better for the TV cameras. Furthermore, don't turn around to admire your face on the big TV screen. It just won't work. All people will see is the back of your head. Likewise, don't turn around and look at your slides all the time (except maybe for a brief glance to make sure you are on the one you expect). People really are traditionally more used to seeing the front of people's heads than any other side.

"I'm almost out of time so I'll just run through the rest of the slides real fast."

You are hereby warned: You have only about 15 minutes to do your brain dump (for a talk in the technical session). The time you have is well known to you in advance. You must use it wisely. About all you can expect to do in this amount of time is give an overview of your paper and inspire those in the audience to read the paper itself for details. Plan to spend most of your time talking about your new ideas. I have seen talks where the speaker spends 13 minutes giving a review of the field and a justification for why the specific problem is interesting. Then—what do you know—there's no time left for the meat of the talk. I think you can safely assume that most everyone in the audience thinks computer graphics is a good idea and that, in fact, the specific problem you are addressing is worth solving. You can probably do fine with about two minutes introduction before getting to the good stuff. Don't go into enormous detail in derivations of the math; just give the basic assumptions and the results. This simplification process goes hand in hand with the simplification of your equation slides. The gist of the math should be describable without going into a lot of fine details that people will best get out of the paper. If you have a videotape, time it and make sure it doesn't eat up the whole time for the talk. Speaking from experience, I know it is very embarrassing for a session chairman (whose main duty is time police) to have to interrupt a nifty tape because there's no time left.

"Uh, I guess that's all I have to say."

Probably the most important parts of your talk are the first and last sentences. Have these all figured out before you go up to the podium. Try to have something snappy to end with rather than just drizzling off. You also must give the audience a signal for when to applaud. Usually a simple "Thank you" will suffice.

Remember

Look up, bright slides, big letters. Uh, I guess that's all I have to say. Thank you.

References

[ACM] *Collected Algorithms from ACM.*

[Bli88] Jim Blinn. Things I hope not to see or hear at SIGGRAPH. *IEEE Computer Graphics and Applications*, pages 69–70, May 1988.

[Bro86] Scott Brookie. UNIX for Luddites. Technical report, Division of Social Sciences, Merrill College, University of California, Santa Cruz, 1986.

[CS90] Marilyn M. Cooper and Cynthia L. Selfe. Computer conferences and learning: Authority, resistance, and internally persuasive discourse. *College English*, 52(8):847–869, December 1990.

[Fol80] Wilson Follett. *Modern American Usage—a Guide.* Avenel Books, New York, 1980.

[Gri83] David Gries. *The Science of Programming.* Springer-Verlag, New York, 1983.

[Hac89] Diana Hacker. *A Writer's Reference.* St. Martin's Press, New York, 1989.

[HO91] Thomas N. Huckin and Leslie A. Olsen. *Technical Writing and Professional Communication for Nonnative Speakers of English.* McGraw-Hill Book Company, New York, 1991.

[Hol84] Nigel Holmes. *Designer's Guide to Creating Charts and Diagrams.* Watson-Guptill Publications, New York, 1984. Nigel Holmes is an illustrator for Time Magazine, and this book presents his style of almost information-less charts and diagrams.

[HRHS86] A. S. Hornby, Christina A. Ruse, Dolores Harris, and William A. Stewart, editors. *Oxford Student's Dictionary of American English.* Oxford University Press, Oxford, 1986.

[IEE90] IEEE. IEEE code of ethics. *The Institute*, 14(9):2, October 1990.

[KLR88] Donald E. Knuth, Tracy Larrabee, and P. M. Roberts. Mathematical writing. Technical Report STAN-CS-88-1193, Stanford University. Department of Computer Science, January 1988. This report is based on a course [CS 209] of the same name given at Stanford University during Autumn quarter, 1987.

[Knu73] Donald E. Knuth. *The Art of Computer Programming, Volume 3—Sorting and Searching.* Addison-Wesley Publishing Company, Reading, MA, 1973.

[Knu84] Donald E. Knuth. *The TEXbook.* Addison-Wesley Publishing Company, Reading, MA, 1984.

[Knu86] Donald E. Knuth. *TEX: the Program.* Addison-Wesley Publishing Company, Reading, MA, 1986.

[MP88] Peter J. McGuire and Sara M. Putzell. *A Guide to Technical Writing.* Harcourt Brace Jovanovich, San Diego, CA, 1988.

[O'H86] Frank O'Hare. *The Modern Writer's Handbook.* Macmillan, New York, 1986.

[SJW79] William Strunk Jr. and E. B. White. *The Elements of Style.* Macmillan, New York, third edition, 1979.

[TB87] Thomas N. Trzyna and Margaret W. Batschelet. *Writing for the Technical Professions.* Wadsworth Publishing Company, Belmont, CA, 1987.

[Tuf83] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, 1983.

[van78] Mary-Claire van Leunen. *A Handbook for Scholars.* Random House, New York, 1978.

[Wir76] Niklaus Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

Appendix A. Generic Evaluation Sheet

A minus (−) in the parentheses means that this particular item has not been done properly or well. The entries in the brackets (for example, [sp]) are approximately what appear in the margins of your paper to explain a correction or a needed change. (Your instructors vary somewhat in their notations.) These marginal notations are the key to this sheet. Explanations for the problems indicated may be found by looking up the references to Huckin and Olsen [HO91], the grammar handbook by Hacker [Hac89], or in the appropriate section of Appendix B of this workbook.

A.1 Accurate presentation of technical information

A.2 Overall document design and organization

A.2.1 Discourse structure

1. Well-ordered sections?[HO91, Chapters 4,5,12,21] ( )
2. Clear and informative section headers?[HO91, Section 9.3] ( )
3. Parallelism in paragraphs[HO91, Chapter 23] ( )
4. Parallelism between paragraphs[HO91, Chapter 23] ( )
5. Parallelism in lists[HO91, Chapter 23] ( )

A.2.2 Formatting

1. Appropriate use of Figures and Tables?[HO91, Chapter 9] ( )
2. Lists used correctly?[HO91, 656–658] ( )
3. Proper white space around displays? ( )

A.3 Grammar, punctuation, and diction

A.3.1 Sentence grammar

1. Articles used properly?[HO91, Chapters 29–30] ( )[art]
2. Countable and uncountable nouns used properly?[HO91, Chapter 29] ( )[+/−count]
3. Subject/verb agreement?[Hac89, G1][HO91, Section 31.3] ( )[s/v]
4. Verb tense, mood, and voice correct?[Hac89, Ge-c,d,e; E4-a,b,c][HO91, Chapters 31–32] ( )[tense]
5. Plural use correct? ( )[plural]
6. Appropriate uses of prepositions?[Hac89, R1-f,W1][HO91, Chap. 37] ( )[prep]
7. Pronouns (agreement, reference, consistency of address)[Hac89, G3] ( )[pro]
8. Passive used sparingly and correctly?[Hac89, G2-e][HO91, Section 25.2] ( )[pass]

A.3.2 Punctuation[Hac89, P][HO91, Appendix A]

1. Sentence Sense
   Fragments?[Hac89, G5] ( )[frag]
   Dangling and misplaced modifiers?[Hac89, E3] ( )[mod]
   Parallel structure?[Hac89, E1] ( )[||]
2. Commas used for clarity in long sentences?[Hac89, P1] ( )[clarity]
3. Run-on sentences
   Fused sentences?[Hac89, G6] ( )[fused]
   Comma splices?[Hac89, G6] ( )[c/s]
4. Commas[Hac89, P1,2]
   with non-restrictive (non-essential) clauses and modifiers?[Hac89, P1-e] ( )[−rest]
   with non-essential appositives?[Hac89, P1-e] ( )[app]
   no commas with restrictive (essential) clauses and modifiers?[Hac89, P2-e] ( )[+rest]
   before conjunctions in lists?[Hac89, P1-c] ( )[list]
   no commas between subjects and predicates?[Hac89, P2-b,c] ( )[s/p]

A.4 Diction

A.4.1 Word use[Hac89, W][HO91, Chapter 27]

1. Trite and faddish words and phrases avoided?[Hac89, W4-e] ( )[trite]
2. Wordiness?[Hac89, W2] ( )[wordy]
3. Technical jargon explained at a consistent level appropriate to the intended audience?[Hac89, W3-a,b] ( )[jargon]
4. Acronyms explained? ( )[explain]
5. Word use accurate and clear? [Hac89, W1][HO91, Chapter 27][any dictionary] ( )[w/c]
6. Noun phrases and clusters clear and unambiguous?[HO91, Chapters 24–25, Appendix A.2] ( )[cluster]
7. Who/which/that usage?[Hac89, G1-h, p.96] ( )
8. Who/whom usage?[Hac89, G3-d] ( )

A.4.2 Tone

1. Level of formality[Hac89, W3]
   too colloquial?[Hac89, W3-c] ( )[colloq]
   too personal?[Hac89, W3-d] ( )[pers]
   too formal?[Hac89, W3-d] ( )[formal]
   condescending or arrogant? ( )[arrogant]
   sexist?[Hac89, W3-e][HO91, Section 27.1] ( )[sexist]
2. Inflated diction or gobbledegook?[Hac89, W3-a,b] ( )[inflated]
3. Choppy?[Hac89, E6] ( )[choppy]
4. Underexplained? ( )[say more]
5. Overexplained? ( )[say less]

A.5 Spelling

1. Capitalization used where appropriate?[Hac89, M1] ( )[CAP]
2. Hyphens used properly?[Hac89, M6] ( )[hyphen]
3. Apostrophe used in possessives correctly[Hac89, P5] ( )[poss]
4. General spelling?[Hac89, M5][spell programs or dictionary] ( )[sp]

Appendix B. Grammar and format notes (read this appendix first)

This appendix is a mish-mash of things we want to say to students, but that we have not found a natural home for elsewhere in the workbook. Most of the comments are a result of things we have seen done poorly by previous years' students. We recommend reading this appendix at the beginning of the quarter.

We have divided things roughly into 6 sections: content, discourse structure, format, sentence structure, punctuation, and word choice. This order is almost the same as the order you use in writing the paper. Content should be the first thing you consider; formatting should be part of the final polishing. No single section should be regarded as more important than the others, although we may choose to concentrate on different aspects of your writing in different assignments.

B.1 Content errors

In most technical writing, the content is the most important aspect of the document. Be sure your information is accurate and complete.

What we look for in the content will vary with each assignment. We try to anticipate what things people will leave out, and what extraneous material will be included, and warn you about them ahead of time, but writers are ingenious at finding ways to fool us, and errors that we did not expect crop up all the time. The new content errors will be discussed in class after each assignment.

B.2 Discourse structure errors (see A.2.1)

This section covers stylistic problems that are above the sentence level. There are four areas that commonly cause problems: sections, paragraphs, pronouns, and tone.

B.2.1 Sections and section headers

Technical writing is often consulted, rather than read from beginning to end. You must use clear informative headers for your sections, so the reader can find the desired information quickly. If the sections are too long, the reader won't have enough headers to find info quickly. If the sections are too short, the headers won't stand out enough from the body of the text. Don't look for solid rules about how big a section should be—it depends on both the material and the audience.

B.2.2 Paragraphs

Your paragraphs should have a single topic, explained in a topic sentence. Your readers will expect sentences within a paragraph to be more closely related than sentences in different paragraphs, so you shouldn't just chop your writing up into arbitrary pieces.

Within a paragraph, the transitions between sentences generally relate the end of one sentence to the beginning of the next. Repeating a sentence subject in adjacent sentences often makes your writing sound choppy and unpolished.

Between paragraphs, the transitions are a little more distant, but the first sentences usually has some reference to the topic of the preceding paragraph. The next paragraph of this explanation doesn't follow that guideline—the paragraphs correspond to different parts of the evaluation sheet, and so I have split this section into subsections.

B.2.3 Pronouns

When you use a pronoun (such as, *it*, *this*, *these*, *those*), be sure it has a clear antecedent. Pronouns without antecedents are often grammatically correct, but stylistically awful (for example, this sentence could have been *It is often the case that pronouns without antecedents are grammatically correct, ...*). Pronouns at the beginnings of sentences are particularly prone to this sort of vagueness.

Students often use pronouns when they would do better to repeat part of the noun phrase. For example, I'd rather see a noun phrase like *this message*, than the simple pronoun *this*.

The correct use of *this* is often tricky. The referent has to be crystal-clear to the reader before you can use *this* (or *those*). If another noun phrase intrudes between the referent and the noun phrase you are attaching the article to, you probably can't use *this*. Many writers use *this* as a pronoun, particularly in constructions like "This is due to the fact that ... ." Although grammatically correct, this usage is poor style. You'll have much clearer sentences if you use *this* only as an adjective, not as a pronoun. Any sentence that contains "this is" or "it is" can probably be reconstructed to read more clearly.

Some people err in the other direction, and use incredible contortions to avoid pronouns, particularly *I* or *we*. You'd do better to say, *I believe the method will work well*, than to say *It is believed that ... .* Don't say *we* unless there are multiple authors, or you are addressing the reader and mean *you and me together*. You should only use you for addressing the reader, not some imaginary third-person. The idiomatic usage *you can*, meaning that *someone can*, should not be used unless you really mean that the reader can.

Cleaning up the pronouns is something to do in a late draft, but make sure you do it before the final draft.

B.2.4 Tone

Technical writing is usually fairly formal. Slang is avoided, people are referred to by their full names (or by their titles and last names), and standard grammar is used. Americans are often too informal in their writing—remember, you don't know exactly who will read what you write.

Part of the formality is expressed by avoiding contractions. Expand *don't* to *do not*, *can't* to *can not*, *won't* to *will not*, and so forth. If you look at this sentence and at the previous paragraph, you'll see that I've used contractions freely—this use of contractions is a deliberate attempt to reduce the formality of the document and make it somewhat friendlier. Formality is not an end in itself—try to judge the right level for the audience and material to be presented.

Trying to avoid the problem of being too slangy, some people try writing in a stiff, formal style. Unfortunately, the result sounds more like a Victorian gothic novelist than a modern engineer. Your aim should be to write crisply and clearly, not stiffly. If you concentrate on presenting the right material to the right audience, the tone should come almost automatically.

Another aspect of tone, *confidence*, is discussed in Section 3.4.1.

B.3 Format errors (see A.2.2)

We want all assignments turned in on standard 8 $\frac{1}{2}$" by 11" paper, with reasonable margins on all four sides. The paper should be typed (double-spaced) on standard typing paper, and should be free of typing errors ("typos") and spelling errors. Watch out for doubled words (the same word twice in a row), particularly from the end of one line to the beginning of the next. For in-class assignments, we can't expect typing, but we still expect neatness. Try to have some clean paper without ragged edges for each class.

Each assignment will have some required standard format (letter, memo, manual, or some such). Most of the standard formats require titles, and many need section headers as well. When you use them, make sure that the section headers are formatted properly, with a blank line before and after each header.

Choosing good titles and section headers is an art—they should contain as much information in as few words as possible. We don't want to see any *generic* titles this quarter—any one who turns in a final project titled *Final Project* has clearly not learned much in this class.

When you receive help from a person, you should acknowledge that help, usually in a separate Acknowledgements section, sometimes in a cover memo. When you obtain information from a book, journal, or other printed source, it should be properly cited and included in a References section. In computer science and computer engineering, footnotes are used sparingly, and only for comments that need to be said but are not part of the main flow of the discussion. Do not use footnotes for citations; use a reference section at the end instead. The preferred citation format if none is specified is to put the author's name and the date of publication in square brackets, but some journals specify other, older forms. See [van78] for a more detailed discussion of citations.

Using someone else's work without giving proper credit is plagiarism—the most serious of academic sins. Collaboration and cooperation are fine—this is not a competitive class, but you must always give credit where it is due.

Remember to put you name and the date on every assignment. Most formats have a standard place for the author's name, a title, and the date. For this class, we always want to see your first draft with your partner's comments.

B.4 Sentence structure errors (see A.3.1)

This section covers both stylistic faults and grammatical faults. Correcting stylistic faults will improve your writing; correcting grammatical faults is essential.

B.4.1 Faults in style

The most common stylistic sentence structure faults are run-on sentences. A run-on sentence strings several independent sentences together, often with grammatically correct transitions. Modern style in technical writing favors fairly short straight-forward sentences, with occasional longer ones for variety. Any extremely long sentences should be broken into smaller pieces. But be careful not to go to extremes—a choppy style with all short sentences is as bad having run-on sentences.

Novice writers tend to over-use the passive voice in formal writing. Check each passive sentence to see if it can be made active. Sometimes, however, passive sentences are the best way to say things. For example, the active sentence *The student's advisor must approve the choice of courses*, is subtly incorrect. It asserts that the advisor has no choice—s/he must approve the choice, no matter how bad it is. The passive sentence *The student's choice of courses must be approved by the advisor*, is less emphatic about the obligation on the advisor. Perhaps even better is *The student must have the choice of courses approved by his or her advisor*, which avoids the passive and leaves the obligation clearly on the student, and not the advisor.

Passive sentences can also be used to maintain the normal topical flow of a sentence (from old information to new information). Note how the flow encouraged me to make the previous sentence passive, and how I avoided the passive in this sentence. See [HO91, Chapter 25] for more information on using information ordering in constructing sentences.

Starting each sentence in a paragraph the same way sounds childish. Starting each sentence the same way can sometimes be used effectively in poetry or songs. But starting each sentence the same in technical writing is not a good idea. (If this paragraph does not make it clear how bad the repeated sentence opening sounds, I don't know what will.) Don't vary the sentence starts randomly; try to make smooth transitions from the end of the previous sentence instead.

Repeated starts are not the only bad sentence beginnings. Watch out particularly for *There are*, *There is*, *It is*, *This*, *These*, and other contentless beginnings.

Watch out for elided words. For example, the sentence structure *If A, then B* is often spoken or written as *If A, B*. The elision of *then* is grammatically acceptable, but can lead to confusion, particularly if *A* or *B* is a list. Compare *If a, b, or c, d* and *If a, b, c, or d* with *If a, b, or c, then d* and *If a, then b, c, or d*.

B.4.2 Faults in grammar

The grammatical faults are more serious than the stylistic ones. You can often get away with bad style, but your readers will be seriously confused or offended by bad grammar. For example, always use complete sentences, not fragments. Fragments are acceptable in some literary styles, but not in technical writing. Section 8.4 has some examples of fragments, illustrating how difficult they can be to read.

If you start a sentence with a modifying phrase, be sure it is supposed to modify the subject. A dangling modifier can be unintentionally humorous. For example, *Never having passed an exam, the teacher failed the student*, says that the teacher flunked the exams, not the student. Similarly, *After typing a return, the cursor moves to the next line.* asserts that the cursor types.

These dangling modifiers seem to occur most often in the assignment about naive-user documentation, where they are the most dangerous. The less familiarity one has with a subject, the harder it is to figure out what the dangling modifiers are really modifying.

Even if the subject of your sentence is correct, you can confuse your reader if it doesn't agree with the verb that follows. Number agreement is particularly important. Remember that a subject like *each of the elements* is singular, but subjects like *all of the elements* are plural. Semantic agreement is also important. The most common semantic problems come from leaving out or mis-placing the actor of an action verb. For example, *The data decided me to ...* , is wrong—though you could say *The data convinced me to ...* .

The subject is used as the topic in English. Having a separate topic phrase (*As for the procedure, it sorts numbers.*) sounds like a bad translation of good Japanese. Rearrange the sentence to make the object of the topic preposition the subject or object of the sentence (*The procedure sorts numbers.*).

Many non-native speakers have trouble with English tenses. They do not match the tenses of Asian languages in any simple way. If you have trouble with English tenses, see an English-as-a-second-language tutor. Huckin and Olsen's explanation of English tenses is also valuable [HO91, Chapter 31]. Occasionally a native speaker will use one of the more complex tenses (particularly the subjunctive mood) incorrectly. Any standard grammar text should explain the formally correct usage sufficiently for most native speakers.

B.5 Punctuation errors (see A.3.2)

B.5.1 Hyphens and dashes

Learn the difference between hyphens (-) and the two types of dashes (– and —). When typing, use one hyphen (`-`) for a hyphen or en-dash, two (`--`) for an em-dash. Spaces should not be used around hyphens or dashes of either sort. See also Section 3.5.3 for details on hyphens and dashes.

A hyphen is used to join compound words together (for example, mother-in-law), particularly when a complex noun phrase is being used as a modifier. (Be warned: the complex-noun-phrase-modifier style of writing is hard to read.) Hyphens are sometimes used to attach prefixes and suffixes to words, particularly when the word is likely to be misread without the hyphen (for example, *pseudo-operation*, *re-enter*, and *pre-enroll*).

Some students detach prefixes and try to use them as adjectives. The prefixes *sub-*, *pre-*, *pseudo-*, and *proto-* must be part of another word, either as a single word (*subroutine*) or attached with hyphens (*Proto-Indo-European*).

Hyphens are also used at the end of a line to indicate that a word continues on the next line. Unlike many other languages, English has strong rules about where a word can be hyphenated. You can only break between syllables, and you should have at least three letters before and after the hyphen. If you are not certain, it's far better not to hyphenate than to hyphenate incorrectly. Most word processors that hyphenate do an awful job of it—don't trust them!

En-dashes and em-dashes are different lengths and have different uses. The shorter en-dash is used to indicate a range of values or times (for example, June 30–July 31); the longer em-dash is used to indicate a long pause and a discontinuity in the flow of ideas. If you have too many em-dashes, your readers will think that you couldn't keep your attention on the subject, or that you are hopelessly badly organized.

B.5.2 Quotation marks

Quotation marks ("quotes") are used to enclose a directly quoted statement from another source, or, sometimes, to set off a slang word or deliberately mis-used word. The second usage probably derives from the first, attributing the word to an outside source. Don't use quotation marks for emphasis—use italics or underlines instead. Single-quotes are used for quotations inside quotations. Some fonts have separate left and right quotes ("like this" and 'this'); if yours does, use them. Brackets [ ] are for comments from the quoting author inside a quoted passage. One popular bracketed comment is [*sic*], which is used to indicate that the error in a quotation was in the original, and was not added in transcription.

We disagree with many punctuation experts on one point—they insist on putting commas inside quotes. This is correct when quoting human conversation or human-to-human writing, but when quoting any communication with a computer, retain the original punctuation inside the quote marks. For example, you type "mail", not "mail." Exact punctuation is often critical in computer communications—resist the attempts of those who know no better to "correct" your usage!

B.5.3 Parentheses and brackets

Parentheses should only be used for additional material that can be omitted without changing the meaning. They are used for short examples, or for comments from the author. Longer digressions usually belong in footnotes, or, better yet, in some other section of the writing. Parentheses bind tightly to the words inside them—there should be no space after a left-parenthesis or before a right-parenthesis (like this). There should be a space before left-parenthesis, and either space or punctuation after the right-parenthesis. If you put an entire sentence in parentheses, the sentence ending punctuation should be inside the parentheses and the the whole parenthetical structure should not be inside another sentence. (That is, you can punctuate such a remark this way.) I, along with many people who compose at the keyboard, over-use parentheses. You can often replace them with commas, or remove the parenthetical comment.

Don't use brackets [ ] in place of parentheses ( ). They serve different functions in English. Although some authors use parentheses for citations, brackets are preferred in most computer science publications. Another use for brackets is to interpolate editorial comments into the middle of a quotation (vulgarly known as butting-in).

B.5.4 Commas and related marks

Most problems in English punctuation come in the use of commas, which have fairly complex rules. Traditional grammar books and secretary's handbooks (such as Hacker [Hac89]) cover the rules in some detail. One mis-use of commas that is not always well covered is the comma-splice. A splice happens when two independent clauses get joined with only a comma for glue. The problem can be fixed in several ways: by adding a conjunction (usually *but* or *and*), by splitting into two separate sentences, or by replacing the comma with a longer pause (usually a semi-colon or dash).

Semi-colons and colons have specific uses. They are not all-purpose substitutes for commas. Huckin and Olsen's explanation seems inadequate, but O'hare's is not bad [O'H86, 142–147].

Watch out for punctuation around Latin abbreviations! The periods that are part of the abbreviation are not used for sentence and clause punctuation (except that an abbreviation at the end of a declarative sentence does not need a double period). The abbreviations *i.e.* and *e.g.* should almost always have a comma after them (i.e., like this). You are probably better often avoiding Latin abbreviations.

Put two spaces after a sentence-ending period. Readers rely on that cue to break your text into sentences, and your text can look quite dense and forbidding if you omit the extra space.

B.6 Word choice errors (see A.4)

This section deals with grammatical and stylistic problems with diction. Many American writers suffer from puffy, pompous writing, particularly when they are trying to write formally. Cut out the words that carry no meaning, watch out for fashionable words and phrases, and, always, know what your words mean! All jargon and acronyms should be explained at the level your audience requires. Generally, more explanation is needed than most authors are aware of. If you're not certain, it's generally better to explain too much than to explain too little.

B.6.1 Commonly misused words and phrases

Certain words and phrases are so commonly mis-used that it is worth looking for them specifically. For example, the *-ize* suffix can be added to almost any noun or adjective in English to make it a verb, but the result is almost always hideous. (See [SJW79, 50–51].) *Utilize*, *prioritize*, and *finalize* are prohibited—you can use *use*, *rank*, and *finish* instead.

The *which/that* distinction is disappearing from spoken English (in America), but is still important in writing. Strunk and White [SJW79, 59] have an explanation of the correct usage.

Some words can't be intensified. You can't have something that is "more unique" or "most unique," because uniqueness is a property that is either present or absent, nothing in between. Similarly, nothing can be "especially essential"— to say *A is essential to B* means that *A is the essence of B*, that is, that *A* is what makes *B* what it is. Either *A* is essential, or it is not, but *A* can be more or less important to *B*.

Another word that is often mis-used is *optimal*. When you say that something is optimal, you mean that it is the best—not just good. It is impossible for something to be more optimal, just as it is impossible for something to be more perfect—both should be expressed as *more nearly optimal* or *more nearly perfect*.

The expression *locally optimal* is often used in describing solutions to combinatorial optimization problems, but is meaningless without more information. Technically, a solution is *locally optimal* if no other solution in some neighborhood is better. Although neighborhoods are usually well-defined in Euclidean space, there are many different neighborhood definitions possible in the search spaces of combinatorial optimization problems. Something can be locally optimal with respect to one set of neighborhoods, but not with respect to a different set.

Speakers of Californian English are fond of adding adverbs to their sentences to slow them down. Most of you are aware that *totally* is not used as a general intensifier in formal English, but only with the meaning *in total*. Unfortunately, far too many students use *basically* and *actually* as substitutes. These also have exact meanings—look at the corresponding prepositional phrases, which have not been as badly corrupted: *as a basis* and *in actuality*. If these phrases don't fit, you are probably mis-using the adverbs—try omitting them.

Non-native speakers occasionally use the wrong preposition. Many preposition usages are idiomatic, and cannot be predicted from the meaning. The only thing to do is to learn verbs and prepositions together. Reading well-written English is the best way to learn the correct combinations. Huckin and Olsen's chapter on informal conversational expressions is also useful [HO91, Chapter 37].

The words *first*, *second*, *third*, ... , *last* are already adverbs. The constructions *firstly*, *secondly*, and so forth are not words.

The word *thus* is not a shorter form of *therefore*—it means *in this manner*, not *because of these things*. Because *thus* is already an adverb, there is no word *thusly*.

One problem that many of us share involves the word *only*. Because *only* can modify several different things in a single sentence, it is important to put it in the right place. Unfortunately, most of use tend to move it forward in the sentence so that it modifies the top-level verb. Notice the difference between *I'm only working on my homework for an hour*, *I'm working on my only homework for an hour*, and *I'm working on my homework for only an hour*.

Another strange usage I've seen, but do not understand the origins of, is "A leaf node is when ..." for "A leaf node is a node that ...".

Recursion comes from the verb *recur*. There is no verb *recurse*.

B.6.2 Sentence grammar affects word choice (see A.3.1)

Non-native speakers, particularly Asians, have difficulty using English articles and using singular and plural nouns correctly. Correct usage depends on the noun and on the thing it refers to (its referent). Chapters 29 and 30 of Huckin and Olsen [HO91] have some excellent heuristics for choosing articles correctly.

The most useful heuristic is determining whether a noun phrase has a unique referent, which is occasionally tricky. One case is obvious—when you have already explicitly mentioned the referent. For example, I can now talk about this case, but I can't say "a case" or "one case" to refer to it. If I haven't explicitly mentioned the referent before, but it is clearly unique (for example, the sun and the moon), I still have to use *the*, not *a*. I can't say, "A first proof was found by ... ," because by identifying the unique first proof, I have ensured a unique referent. I can say "a possible first attempt", implying that it is not only one possible one. A plural noun can have a unique referent if the set it refers to is unique (for example, *the stars* or *the moons of Jupiter*).

You may have noticed that I left *that* out of the discussion on articles and demonstrative adjectives. It is mainly for pointing to particular objects, usually in contrast to *this*. For example, "I want you to use this article, not that one." You can probably do fine technical writing for years without using *that* as an article.