COMPUTER SCIENCE 111

Ken Steiglitz
Jeff Blum
Rich Feit
Jonathan Thompson

CS-TR-346-91

September 1991

ABSTRACT

This is the lab manual for COS 111, an introductory course in computer science for liberal arts students, as it was taught for the first time in the fall semester of 1991. It includes a general introduction, topic outline, a set of nine laboratory exercises for the NeXT, and a collection of CheatSheets that provide supporting material for the labs. The lab topics are:

(1) Introduction,
(2) Communications [news, mail, ftp, telnet],
(3) Graphics [PostScript, lpr, Edit, Yap, TopDraw, etc.],
(4) Presentation [WriteNow, troff, etc.]
(5) Sound [SoundEditor],
(6) Programming in C,
(7) Selection sort,
(8) More sound, using C, and
(9) Information processing [grep, wc, sort].

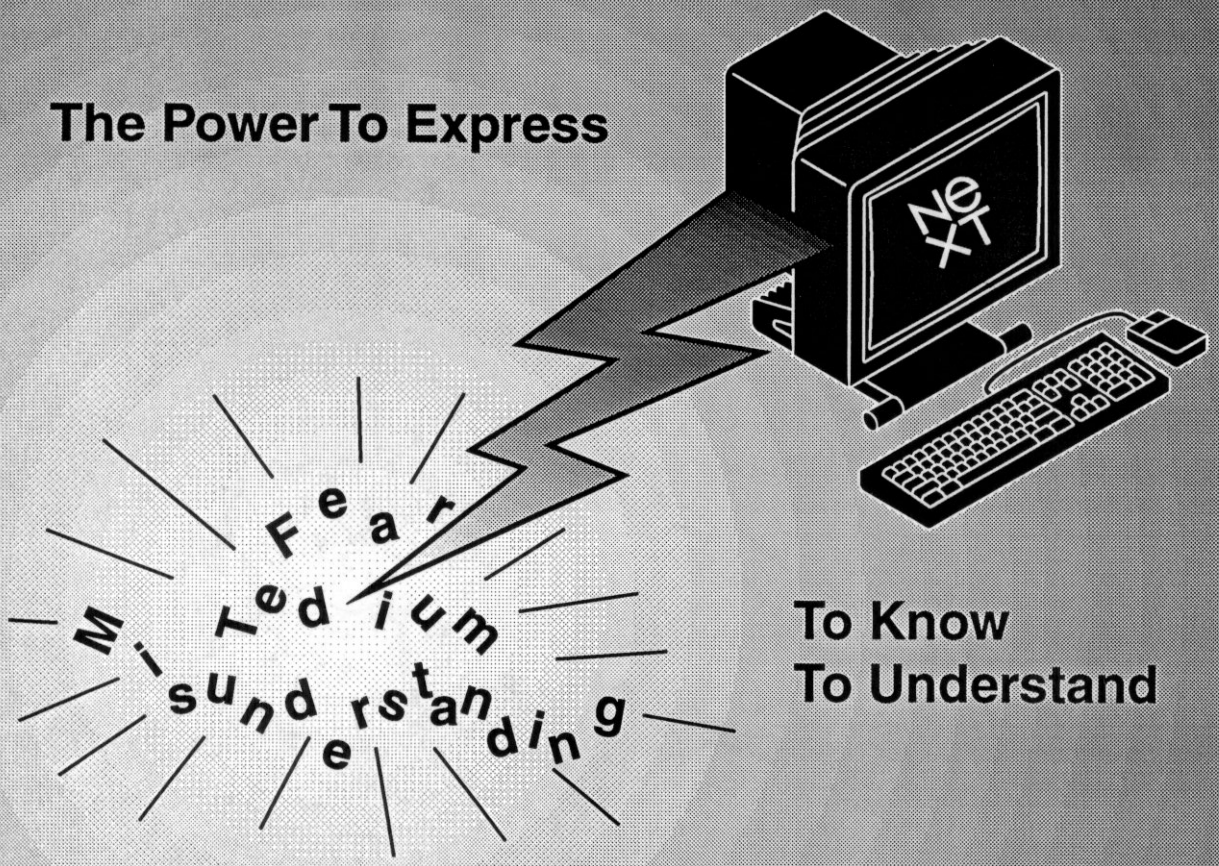
The lectures themselves included an introduction to complexity theory [big-oh notation, analysis of sorting, etc.] and computability theory [Turing machines, the halting problem, etc.].

COS 111

Fall 1991, Princeton University

Version 1.10 Beta

The Power To Express



To Know
To Understand

Technical Report CS-TR-346-91, Department of Computer Science, Princeton University, Sept. 1991

Interpersonal Computing

Authors:

Ken Steiglitz
<ken@cs.princeton.edu>

Jeff Blum '94
<jeffblum@phoenix.princeton.edu>

Rich Feit '94
<richfeit@phoenix.princeton.edu>

Jonathan Thompson '93
<jonthomp@phoenix.princeton.edu>

INTRODUCTION

(by Ken Steiglitz)

1. General Comments

This course is intended for non-science, non-engineering students with little or no computer experience or knowledge. The ideal student has not used computers at all, or, better yet, has a vague fear of them. You're going to have a lot of stuff thrown at you, but I hope that by the end of the course you're at home with computers and the ideas that go along with them. If I do my job right, you'll become an enthusiastic computer user for the rest of your life.

The class material will combine two interleaving threads[1]. The first of these, the **Practical Thread**, will directly support the labs, where you will get your own hands on computers. The lab assignments and their cheatsheets were developed in the summer of 1991 by three Princeton undergraduates: Jeff Blum, Rich Feit, and Jon Thompson. Their goal is to expose you to a wide a variety of useful applications in a stimulating, friendly computing environment. They largely determine the overall shape of the course. You'll be using the NeXT machine, which has good graphics, sound, lots of interesting programs, but at the same time allows you to get close to what's really happening when you want to. Each week I'll cover topics that will help with the labs.

We've taken care to make the labs do-able in the allotted time of two or two-and-a-half hours. They have a basic part, which everyone should do, plus plenty of extra material to challenge the budding hacker. The general spirit of the labs is non-threatening and free; we'll have more to say about the ethos of computer users later.

The other thread of the course, what I'll call the **Theoretical Thread**, is about how computers came to be, why they're built the way they are, and what they are and are not capable of doing. I hope to talk about what they'll be like in the future, but of course that's risky business. I include in this thread software as well as hardware. As we will see, both aspects of computers are dominated by the idea of hierarchical structure. Layer upon layer, structure within structure, we'll keep coming back to these ideas.

The theoretical thread includes some of what computer scientists call 'computability theory' and 'complexity theory'. This is the theory of what can be computed and what cannot, and how long it takes to compute things when they can. Although this is strictly speaking a 'mathematical' theory, we will be able to get very far using just common sense.

---

[1] *thread*:
/thred/ n. [USENET, GEnie, CI$] Common abbreviation of 'topic thread', a more or less continuous chain of postings on a single topic.
— *The Jargon File*, (copyright) Eric S. Raymond, version 2.8.2, March 25, 1991.

Throughout the course we'll touch on ways that computers affect people and their institutions. I encourage you to explore this area as the course progresses, and hope some of you will find good topics for midterm papers here. Of course computers have affected almost every area of human enterprise: music, graphic arts, literature, economics, medicine, law, and so on. I particularly want to raise the important issues of privacy and civil rights; this will be good discussion material for the precepts.

2. Schedule

Week	Tuesday		guest	Lab #	Lab
1	Sept.	17		-	
2		24		1	introduction
3	Oct.	1	Phillips	2	communication
4		8	Sorensen	3	graphics
5		15	Sedgewick	4	text presentation
6		22		-	MIDTERM PAPER DUE
		29			..... break .....
7	Nov.	5	Lansky	5	sound
8		12	Hanrahan	6	C programming
9		19	Dobkin	7	selection sort
10		26			..... Thanksgiving .....
11	Dec.	3	Munro	8	ein
12		10		9	information processing
		16	last meeting (Monday)		

3. Mechanics

The class meets twice a week for lectures, and has two sections of precepts and two sections of labs. Each week you should attend both lectures, one lab (if there is one that week), and one precept. The lectures meet twice a week, Tuesday and Thursday mornings, 10:30 - 11:50, in the small auditorium, Computer Science Building. The first week we'll schedule the one-hour precept sections so that everyone can attend one a week. We'll also schedule 2.5 hour lab sections, which meet in the NeXT cluster, Room 001, Computer Science Building.

Grades Grades will have three components: (1) each lab assignment will have work to complete and hand in (mostly electronically); (2) there will be a paper due at midterm; and (3) a term paper due at the end of the term. Notice that the grade is based completely on products that you produce.

Lectures The lectures themselves will cover the practical and theoretical threads we discussed. I'll start with the material you need to get the most out of the coming labs. I plan to interleave the topics to keep people awake, and also to give you an appreciation of how wide the subject of computer science is. (Actually, 'lectures' is an overly formal term, but seems to be institutionally entrenched; I prefer to think of them as classes.) I've also lined up seven guests who use

computers in interesting ways:

Dick Phillips	multimedia presentation
Vibeke Sorensen	visual art
Robert Sedgewick	producing a book
Paul Lansky	computer music
Pat Hanrahan	new input/output devices
David Dobkin	programs and life
Ian Munro	the Oxford English Dictionary project

They're leaders in their fields, doing exciting things. Dick Phillips is from Los Alamos National Labs. Vibeke Sorensen is Artist and Director at the Computer Animation Laboratory, California Institute of the Arts, Valencia, CA, and also a Visiting Lecturer in the Computer Science Department. Ian Munro is professor of computer science at the University of Waterloo. Paul Lansky is professor of music here, the others homegrown professors of computer science.

Precepts The precepts will be devoted first to topics that spill over from the busy lecture schedule, and then to discussion of any other topics that seem like they need more attention — questions you may have about lecture material or readings, problems you may have with the labs, issues you want to raise from the news, from science fiction, the business world, and so on. Following the lab schedule, the first few precepts in the second half of the term will concentrate on programming. We'll also use some precept time to discuss your choice of topics for the midterm and term papers.

Labs In the past, introductory computer courses were really just *programming* courses, and the labs were traditional programming assignments: "Write a program to do this, write a program to do that." I think this approach is outdated. For one thing, there is a lot more about computer science that I want to teach besides programming. For another, the boundary between what is programming and what is not has become blurred. Two out of the nine labs are devoted to programming *explicitly*, but programming is sneaked in throughout the term, in forms you may not recognize. Today, programming is more a way of thinking than the act of writing code.

As I've mentioned, the labs are the backbone of the course. Rich Feit, Jeff Blum, and Jon Thompson worked hard to make these labs clear, efficiently instructive, and fun. If you find bugs, or have suggestions for improvements on the labs, please send email to them and me (flames → /dev/null[2]). There will

[2] *flame*: v. 1. To post an email message intended to insult and provoke. ...

/dev/null: /dev-nuhl/ [from the UNIX null device, used as a data sink] n. A notional 'black hole' in any information space being discussed, used or referred to. A controversial posting, for example, might end "Kudos to rasputin@kremlin.org, flames to /dev/null". See {bit bucket}, {null device}.

— *The Jargon File*, (copyright) Eric S. Raymond, version 2.8.2, March 25, 1991.

always be a TA available to help you during scheduled labs. If you get stuck at other times, see the TA during office hours, see me during office hours, send email to anybody, ask your neighbor. Lab assignments will be due at prespecified times, and you'll learn how to submit them electronically both to save paper, and to turn in sound and video.

Midterm paper The midterm paper will give you a chance to show off your skills at document preparation, and to research some area that interests you. I ask that the presentation use the tools you'll learn to work with on the NeXT machine, insofar as they are appropriate to the presentation of the material. Here's a chance to be creative. Pick an adventurous, weird, or controversial topic. Make the presentation as snappy as you can. Incorporate graphics of different types, and even, if you want to read ahead, sound.

Term paper The most obvious thing to do for a term paper is keep working on your midterm paper. In fact, you may want to consider your midterm paper as a proposal for your term paper. On the other hand, you will have been exposed to more applications and you'll know a lot more when you come to choose your term paper project. In particular, you won't get to making sound or programming till after midterm break. All I ask is that the paper represent a substantial investigation of some aspect of computers that interests you. You can take one of the labs and keep running with it, you can find a computer project of your own, or you can do a paper that doesn't directly involve running programs. A study of some theory would fall into the last category, for example. Some suggestions for midterm or term papers are given in the next section. They are representative of topics I would choose, and they are only meant to stimulate your thinking.

4. Paper topics, samples

1. Read the book called "The psychology of everyday things," by Donald A. Norman, New York : Basic Books, 1988. (Find the call number using the Online Catalog.) This book makes you look at the devices you deal with every day in a new way. Apply some of the techniques of criticism and analysis to the computer programs you will be using. For example, use three different editors and compare how well they help you avoid errors (or encourage you to make them). Explain why I hate *vi*. Explain what is beautiful about UNIX and what is terrible. Compare troff and TeX from the perspective of this book.
2. People tend to communicate quite differently in email than in person, or on the telephone. Do some experiments to see whether you observe this. Compare with the *talk* program. Do a literature search and see what you can find out about the influence of medium on style of communication.
3. People with an interest in games should have no problem finding interesting paper topics. Chess has been used as a testing ground for computers since the early days, and was mentioned in Turing's paper on computers and thought (Handout [Tu2]). There are regular computer chess tournaments, and frequent matches between people and computers. There are also an

amazing variety of games available for free, for money, of all degrees of complexity. You might be interested in studying the sociology of gamesters, the impact of games on people's psychology and behavior, or the economics of the computer game industry, for example.

4. Research the early history of electronic music, back to the time when composers painted on sound tracks by hand. You might concentrate on tracing the effects of technological advances and the time delays for those effects.
5. Research and report on the use of computers in some particular field that interests you: plastic surgery, consumer audio electronics, medical diagnosis, aids to the handicapped, epidemiology, banking, gambling, baseball ... You get the idea — anything at all.

5. Authenticity

You wouldn't want to take a French course without hearing natives speak, and you wouldn't want to take a poetry course where the poems had been edited for simplicity. So I've tried hard to stick with the real stuff in this course. In that sense the course is like Capretz's *French in Action*, an immersion course. I'll be passing out readings to supplement the lectures; I've tried to include especially significant papers, often of great historical interest.

For example, one of the handouts is Turing's 1936 paper on computable numbers [Tu1] — probably the most famous paper in all of computer science. You'll read at least the beginning of it, and we'll try to get at the essence of what Turing was thinking about. This is a major intellectual achievement, coming to grips with the problem of what computation really is — back when there weren't even any computers! The paper will seem opaque, abstract, and irrelevant. But its ideas have influenced basic thinking about computers more than any other single paper. Every educated person should know what a Turing Machine is and why it's important.

When we talk about printing, we'll show you PostScript. That's the language printers understand. Other programs generate PostScript from your hints, but PostScript is the real thing.

When we do some programming, we'll use the language called C. That's what most computer scientists consider the real language, the language that God speaks instead of Hebrew. Likewise, the operating system underlying the NeXT Machine is UNIX, the operating system of choice among connoisseurs. Whenever possible we'll go to the source and read the masters.

The NeXT machine itself is a departure from the usual sheltered environment of the users' world. It is Steve Jobs' vision of where computers are going. (Steve Jobs is founder of Apple Computer, Inc. and NeXT, Inc.) Some hackers consider it clunky, without enough horsepower, but it is ideal for exploring shared tools in a computer world with images and sound.

6. Why Compute?

I want to provide a little background and perspective by discussing the reasons people use computers — you might want to think about where your personal goals and interests fit in. I'll try to do it roughly in increasing order of difficulty and challenge.

At the simplest level, computers are used to keep records, like bank statements, airline reservations, library withdrawals, and so on. This is the most easily observed use of computers, and no one can avoid bumping into computers used as record keepers all day long, especially in the business world. Because these applications are usually simple, this level has been the first to be exploited on a large scale.

It's interesting to observe that computer applications at this *simplest* level of record-keeping, together with modern communication, gives us electronic mail and networked news and bulletin boards, the computer applications that are so popular right now, and the ones that have led to the development of rich subcultures of users. Each news group ties together hundreds or thousands of people in a way not possible a few years ago, and the rate of exchanging information and opinion appears to be accelerating.

For example, I regularly read *rec.radio.shortwave* (see the *news* CheatSheet). From that group I learned I can get shortwave schedules from an anonymous *ftp* site (see the *ftp* CheatSheet) in Finland, and satellite photos of the US every hour from one in Illinois. I can also find out which radios are the best deals for the money, which stores are having sales, when there's a geomagnetic storm approaching, how to protect my radio's input stage from lightning, and on and on. Not to mention the discussion of what happened to Radio Moscow during the attempted coup, or what service of the BBC Gorbachev listened to in captivity.

At the next level of computer application (the ordering of levels is a bit arbitrary) comes signal processing. I have to admit to a bias in this direction, this being my first love and the subject of my dissertation. Here come the applications where computers are used to transmit, code, decode, combine, mutilate, and enhance sound and pictures. If someday robots are equipped with eyes and ears, signal processing will be necessary to deal with the sense data. (Maybe some day tastes? smells? feelings?) Signals require *tremendous* amounts of data to represent them. This will be brought home to you when you start digitizing sound and run out of room on your disk. We'll talk about how the information is stored and sheer volume. We'll also have guests who deal with applications of signal processing in the arts, a graphic artist (Vibeke Sorensen) and a composer of music (Paul Lansky).

The next level, somewhat less visible to the general public than the first two, is what is usually called 'scientific computing'. At this level, computers do arithmetic, as in the first level, but they do more complicated things that are useful to scientists. For example, scientists use computers to solve the differential equations that describe fluid motion so they can predict the weather, or design airplanes, or understand how speech sounds are produced in the vocal tract. Or

they analyze vibrating systems to predict the effect of earthquakes on buildings, or the noise levels inside your car, or the way a trombone works. In contrast with the record-keeping applications, these tasks usually push computers to their limits, and are challenging computer manufacturers to constantly produce faster and bigger machines. Today these applications are also one of the main reasons manufacturers and researchers are struggling to build systems where many computers can operate on the same problem simultaneously (parallel computation). Also in contrast with record-keeping, many scientific computing problems remain unsolved because computers are not fast or big enough. In some areas, like fluid dynamics and astronomy, it is unclear if some of the important computational problems will *ever* be solved!

Some problems that push the limits of computers are a little different from the scientific applications just mentioned because they don't deal with arithmetic in the usual way. They use 'discrete mathematics', and deal with objects like schedules, arrangements, and graphs. This field is called *combinatorial optimization*, and I'll talk it about it more later in the term.

The next level of computation has been the subject of a great deal of discussion, exploitation in the media, and general ballyhoo, but is really in its very early stages. This is the use of computers to 'think'. (When you get to the lab where you learn how to play sounds, don't forget to listen to the files called **Hal02.snd, Hal03.snd** etc. in our **/LocalLibrary/Sounds**. Do you know where Hal comes from and how he got his name?) Alan Turing formulates and discusses the question of whether computers can think or not in Handout [Tu2]. The paper is famous as the one in which the 'Turing Test' is introduced, and I think you'll enjoy reading it. It's not only brilliant; it's quirky. This last level of computer application brings us to the edge of today's thinking about what computers are capable of. Nobody really knows where we are headed or how far we'll go.

7. Topic Outline by Week

Week	Lab	Practical	Theoretical	Handouts
1	-	command-line vs. graphical interfaces, windows, UNIX directory structure, simple commands (*ls*, *cp*, *mv*, etc.), source/object distinction, *man*	binary numbers, storage volume, trees	[Tu2]
2	Introduction	networks, mail systems, *ftp*, *telnet*	concept of an algorithm, selection sort, bubble sort	[MT, Kl]
3	Communications	bitmapped vs. object-oriented representations	merge sort, *n*log*n* analysis, big-oh notation, Towers of Hanoi	[LP]
4	Graphics	sharing and combining tools. UNIX pipes, *history*, *more*, *tail*, *wc*, *grep*; NeXT tools.	analog/digital, noise, gates, hardware	[Ba, vN]
5	Presentation	arrays, stacks, implementation of recursion	sampling, quantization	[Ho]
6	-	digital sound, digital pictures, file formats	big-oh notation, survey of operations-research type problems with applications	[Ka]
7	Sound	C-syntax, libraries, variables, *for – loops*	idea of problem reduction, P, NP-completeness	[Kn1]
8	C Programming	more C, conditionals, types	languages, assemblers and compilers	[Di, Kn2]
9	Selection Sort	more C, functions, recursive functions	tree searching, game playing	[Th]
10	-	profiling, programming structure, discipline, software development, viruses	sound sampling, aliasing, frequency concepts, filtering	[As]
11	ein	indexing, searching, hashing	sound synthesis techniques, speech	[FM]
12	Information Processing	sound files, formats	Turing Machines, computability	[Tu1]

8. Handouts

[As] I. Asimov, "Robbie," reprinted with an introduction in the collection of Asimov stories *I, robot*, Fawcett Publications, 1950. This was originally published as "Strange Playfellow," *Super Science Stories*, 1940.

[Ba] C. Babbage, "On the mathematical powers of the calculating engine," unpublished manuscript (Dec. 1837) in the Buxton Collection, Museum of the History of Science, Oxford University. Reprinted with introductory material in *The origins of digital computers*, B. Randell (ed.), Springer-

Verlag, Berlin, 1982 (third edition).

Also included is the report of the committee that was formed to evaluate the feasibility of constructing the machine, a later report by Babbage's son, Major-General H. P. Babbage, with the decimals of multiples of π printed (with printing errors caused by weak springs), and a report of the successful construction of a difference engine in 1991 (3 tons!).

[Di] E. W. Dijkstra, "Go To statement considered harmful," *Comm. ACM*, vol. 11, no. 3, pp. 147-148, March 1968.

This short note triggered a revolution in programming style, one that was perhaps waiting to happen.

[FM] G. C. Fox and P. C. Messina, "Advanced computer architectures," *Scientific American*, vol. 257, no. 4, pp. 67-74, Oct. 1987.

[Ho] J. E. Hopcroft, "Turing machines," *Scientific American*, p. 86, May 1984.

The concept introduced in [Tu1], revisited with the perspective of a leading computer scientist, 44 years later.

[Ka] R. Karp, "Combinatorics, complexity, and randomness," *Comm. ACM*, vol. 29, no. 2, pp. 98-111, Feb. 1986.

[Kl] D. V. Klein, " 'Foiling the cracker': a survey of, and improvements to, password security," unnumbered technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1991.

You may decide to change your password after reading this one.

[Kn1] D. E. Knuth, "Von Neumann's first computer program," *Computing Surveys*, vol. 2, no. 4, pp. 247-260, Dec. 1970.

[Kn2] D. E. Knuth, "Computer programming as an art," the 1974 Turing Award Lecture, reprinted in *ACM Turing award lectures: the first twenty years (1966-1985)*, pp. 33-46, ACM Press, New York, NY, 1987.

Actually, handouts [Th] and [Ka] are also Turing Award lectures and are in this volume. It is a good source of general and reflective articles by pioneers.

[LP] H. R. Lewis and C. H. Papadimitriou, "The efficiency of algorithms," *Scientific American*, p. 96, Jan. 1978.

[MT] R. Morris and K. Thompson, "Password security: a case history," *Comm. ACM*, vol. 22, no. 11, pp. 594-597, Nov. 1979.

Compare the timing estimates in this paper with those in [Kl], a scant

twelve years later!

[Th] K. Thompson, "Reflections on trusting trust," *Comm. ACM*, vol. 27, no. 8, pp. 761-763, Aug. 1984.

[Tu1] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proc. London Math. Soc.*, ser. 2, vol. 42 (1936-7), pp. 230-265; corrections, *ibid*, vol. 43 (1937), pp. 544-546. Reprinted in *The undecidable*, M. Davis (ed.), Raven Press, Hewlett, NY, 1965.

A highly technical paper — read as much as you can or want to.

[Tu2] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433-460, Oct. 1950. Reprinted in *Computers and thought*, E. A. Feigenbaum and J. Feldman (eds.), Krieger Publishing, Malabar, FL, 1981.

[vN] J. von Neumann, "Computing machines in general," First of Five lectures delivered at the University of Illinois, Dec. 1949. This edited version of a manuscript in *Theory of self-reproducing automata*, J. von Neumann, (A. W. Burks, ed.), Univ. of Illinois Press, Urbana, IL, 1966.

Thanks

Many colleagues and friends contributed ideas to this project. The list is too long to put here, but we especially want to thank Profs. Appel, Dobkin, Hanrahan, Lansky, Sedgewick, the CIT staff, and Dean Eva Gossman.

TA Overview

Hahaha. So, you got roped into TAing COS111, eh? Well, hopefully this section will help you understand what's going on a little better.

First off, we would like to emphasize one thing about COS111. It is intended not only to give students useful knowledge about computers, but interest them as well. When the two goals interfere with one another, we feel that the latter should take precedence. If students get very turned off by computers, this is far more damaging than if they don't know how to use a certain UNIX command. This may be difficult to keep in mind during the course, but it is nonetheless important.

The structure of the labs should be pretty obvious from reading through them. There are several points worth mentioning here, however.

Keep It Electronic!

The introduction mentions immersion as an important part of learning. To continue in this vein, try never to do something manually for the class that can be done electronically. Strive to place documentation online and use e-mail as much as possible. Commenting on each lab, for example, could easily be done via e-mail. While correcting, simply store all comments, etc. in a file that you can mail to the students when you are finished. Be creative. You may want to form an incentive such as building a list of people who use NeXT mail. These people can be mailed documents, etc. along with their grading, while the others will receive text only. Of course, students should not be strongly penalized for not using NeXTmail, but I think you understand the sort of idea we are driving at—practice what you preach.

End-of-Term Procedure

We have tried to arrange the disk quota for the cs111 account so that none of the student's stuff will have to be thrown away. We thought it might be nice for you to `tar` and `compress` all their files at the end of the course and give them back, along with your comments if you did them electronically. We leave it up to you how to do this. If all else fails, you can just have each student bring in a floppy disk and copy their work onto it.

Formatting Of The Materials

The labs and CheatSheets for this course are all done according to a fairly rigorous style and formatting. Whenever we agreed or had no opinion about a style question, we stuck with the NeXT Style Guide from June, 1990. For formatting, we pretty much forged our own methods. There is a file called **indents.wn** in the **Labs-general** directory that contains the indentation and bulleted list formats along with the headers and footers containing page numbers, etc. If you Copy&Paste rulers from this document, you will find it easy to standardize formats. Good deal. The line on the bottom of **indents.wn** is a small line that we used between paragraphs and minor headings. Normal (2-return) spaces were used only between major sections and between the heading and the first line. Good luck! I am sure that if you look hard enough you will find discrepancies in our adherence to these standards. Oh well. We admit we are not perfect.

The Handin Procedure

When students turn in their homework electronically, it will show up in the directory **~cs111/submit/userid/<assignment#>**. The only difficult part of this procedure is the handling of late assignments, explained below.

If the dummy file **~cs111/submit/userid/<assignment#>** is readable by the students, then they will be able to submit into their normal submission directory, **~cs111/submit/userid/<assignment#>**. If this file is not readable, then the student will be told that the files will be submitted late. If the student tells the program to go ahead, the files (`tar`red and `compress`ed) will go into a directory called **~cs111/submit/userid/<assignment#>_late**.

You must alter the **~cs111/submit_time/accept_<assignment#>** files manually at the proper times or figure out a way to do it automatically. Either `chmod` them so they are not readable by the students, or remove them entirely.

When you are ready to grade the assignment, you must un`tar` and `uncompress` the submitted files. We have written a shell script called `unveil` that will do this automatically. If the student has multiple submissions, it will un`tar` and `uncompress` them in order, giving you the last version of each submitted file. The late files will be placed in the late directory, and the on-time files will be placed in the normal directory. The syntax for `unveil` is as follows:

```
/u/cs111/bin/unveil <assignment-number> <userid>
```

This will take care of that student's files. If you want to `unveil` just one specific file, the syntax is:

```
/u/cs111/bin/unveil <filename>
```

This should be all that you need to know about submitting and the handing in. If not, then you can trace through the scripts and C code to figure it out, just as we did. :-)

Lab Philosophy

If you have one phrase echoing through your head during these labs, it should be this:

"It can't hurt!"

You can't hurt anyone by pressing that button. The computer won't blow up if you type that command. And it certainly can't hurt to *submit incomplete work.*

The philosophy of these labs is much the same as that of the lectures. We're going to throw a lot of material at you, and you'll come across everything from the blatantly obvious to the near-incomprehensible. Your task is to absorb everything you can.

The advantage to this method is the kind of computer experience you will have under your belt at the end of the course. This is the Real Stuff. Although there's a limit to how deep we can take you in the time allotted, your breadth of experience will most likely dwarf that of the average sophomore computer science major. UNIX, PostScript, C, graphics, sound... each of these topics will mean something to you. We'll give you the authentic material, not the Cliff's Notes.

The most important thing you'll learn (by experience only), is that computers are the most harmless of all creatures. They *only* do *exactly* what you tell them to do, and the worst thing you'll experience at their hands is frustration.

On to specifics...

In order to give you an idea of what we feel you can accomplish in two and a half hours, and in order to keep the more difficult topics optional, we divided each lab into three main sections: **Procedure**, **Above and Beyond**, and **For Further Exploration**.

We tried to make the **Procedure** of each lab short enough that everyone would get through it. If you hit the two-hour mark, though, and you're only half-way through the lab, it's possible that we miscalculated. Even if you're the only one in the whole class who isn't done, it's much, much better to be thorough with whatever you can get to. The absolute worst thing you could do is hurry to get through this section. If you don't absorb everything as best as you can, you may as well have skipped the entire lab.

Hopefully, everyone will have a chance to try something from the **Above and Beyond** section of each lab. Some of the activities in this section are more basic (but perhaps less educational) than those found in the main **Procedure**; others are more advanced — and usually more fun. Do *anything* here, in any order. If you finish only a small part of a section, *hand in what you have!* Improvise! If you find something enjoyable that fits in with the lab, hand in results from that, too. Turn it in even if it *doesn't* really fit in!

Finally, the **For Further Exploration** section is meant for those who find the lab especially interesting. It's mainly for your own enrichment or enjoyment. You might wind up in the library, or you might just find yourself doodling around with an intriguing program.

CheatSheets

CheatSheets were divorced from the lab Procedure for two reasons:

- To teach you concepts and skills which you can apply outside of any specific lab. This is to give you a taste of the real world... where you won't be working within the framework of a "lab."
- To keep the lab structure clear — it's easy to lose the structure of a lab amidst an involved tutorial for an application.

Before each lab, go down the list of required CheatSheets and read each one. Unfortunately, if you don't read these beforehand, you will probably feel lost and frustrated at some point in the lab. Don't try to memorize everything, but make sure to pick out the important concepts.

Grades

For better or for worse, everyone will receive a grade based on the work they do. So follow this basic tenet: **HAND IN ANYTHING YOU DO.** Anything. A valiant attempt that fails is as good (gradewise) as its successful counterpart.

You'll learn in the first lab how to submit things electronically. From then on, if you generate a funky picture, or a startling sound, or weird results from a program, **hand it in.** It can't hurt.

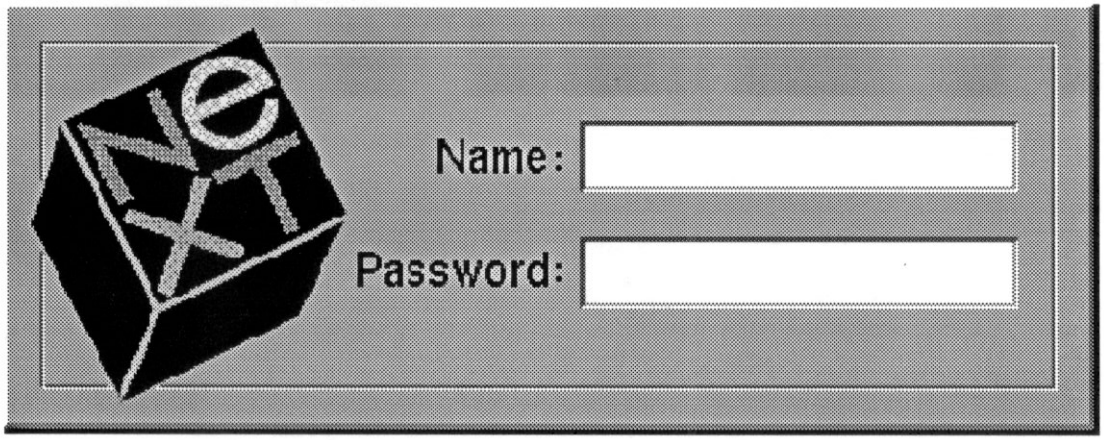
That's all. Good luck, smile `:-)` , take a deep breath, and go to it.

Welcome to the NeXT

When you first enter room 001 of the Computer Science building, you will be greeted by 20 black computers. The NeXT computer is known for its excellent graphical user interface, as well as its special treatment of sound. All of the machines are connected to an international network called, conveniently enough, the Internet, so all can be used to reach throughout the world.

How to Login

The first thing you will see on the screen of your NeXT computer is this box:



There should be a blinking cursor in the **Name** field; go ahead and type your userid – the name the friendly people over at 87 Prospect gave you – and press the <Return> key. This will move the cursor into the **Password** field; here you should type the secret password that you chose when you signed up for your account. Don't worry about hiding the screen from everyone because the computer won't display what you type in this field. Type the <Return> key again, and if all went well, the login box will disappear and you will hear some noises coming from the computer's disk drive. If you mistyped either your userid or password, the box will shake and reset itself, forcing you to type everything over. (You should mess up once anyway; it looks great!)

The NeXT Screen

The monitor is the physical device which houses the computer screen – the display that you read. There are quite a few parts to the screen on the NeXT. Some of the more useful items include the File Viewer, the Dock, and the Menu. The first time you start up the NeXT computer, your computer screen will look awfully bare, but as time goes on, it can get rather complex:

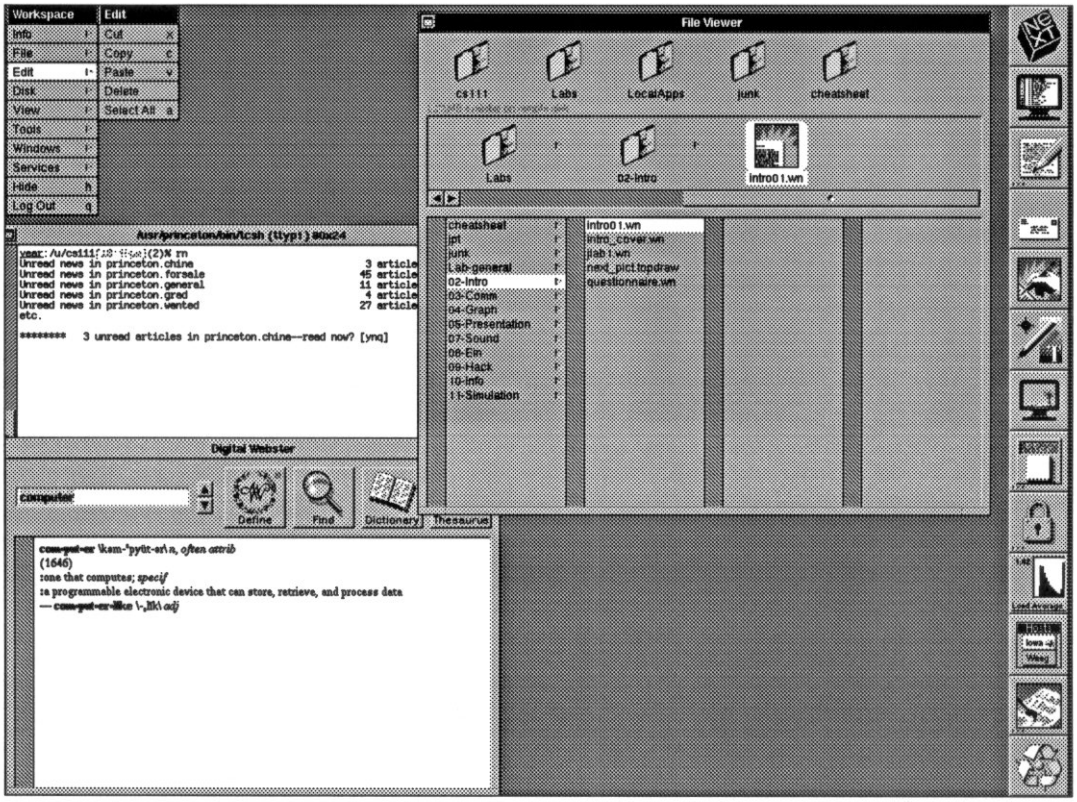
This is the file viewer: the top part is called the shelf, the middle part is the icon path, and the bottom part is the browser

This is –> the Menu area

An open Terminal –> window running net news

The Digital Webster –> finding a definition

<– This is the area known as the Dock



When you start up your computer for the first time, you should recognize only the File Viewer and the Menu, as well as the top and bottom items on the Dock from your own screen. These items will be explained in a short while, but first...

Controlling your environment

There are two input devices that you will use to control the NeXT computer. The keyboard, which is much like a typewriter, is covered in its own CheatSheet. The other input method is a special tool that goes along very well with the type of graphical display used by the NeXT, a pointing device. Several different types of pointing devices exist, but the most common kind is called a mouse: . This small mechanism moves an onscreen arrow (the cursor) as you move the mouse across the table. There are a few fundamental mouse operations that you should know:

- **click** quickly push and release the left button on the mouse once.
  common use: selecting objects.
- **double-click** push the left mouse button twice in rapid succession.
  common use: opening an application or folder, selecting a single word.
- **click-drag** push the left mouse button down, hold it, and move the mouse.
  common usage: moving objects, selecting text.
- **drag** same as click-drag.

- **shift-click** hold down the shift key and click the left mouse button.
  common usage: selecting multiple objects.
- **control-click** hold down the control key and click the left mouse button.
  common usage: <none> (it's uncommon).
- **Alternate-click** hold down the Alternate key, and click the left mouse button.
  common usage: <none> (it's also uncommon).

Note: these terms can be intermixed, as in **shift-click-drag**, which means hold down the shift key, click and hold the button, and move the mouse.

The mouse is often used for selecting text. To do this, just drag over the text you want to select. It will become highlighted, like this: `this is highlighted text`. The selected text is now ready for manipulation (i.e. font change, size change, deletion...)

The File System - Your Everyday Hierarchy

> *Information in a UNIX system is stored in files, which are much like ordinary office files. Each file has a name, contents, a place to keep it, and some administrative information such as who owns it and how big it is. A file might contain a letter, or a list of names and addresses, or the source statements of a program, or data to be used by a program, or even programs in their executable form and other non-textual material.*
>
> – Kernighan and Pike, The UNIX Programming Environment

Just as important as knowing what a file is is knowing where to find it. If UNIX systems stored all files in the same location, there would be chaos. Imagine trying to find one particular file from a list of one hundred, or one thousand, or the much more likely one hundred thousand files that exist on just this university's computer alone.

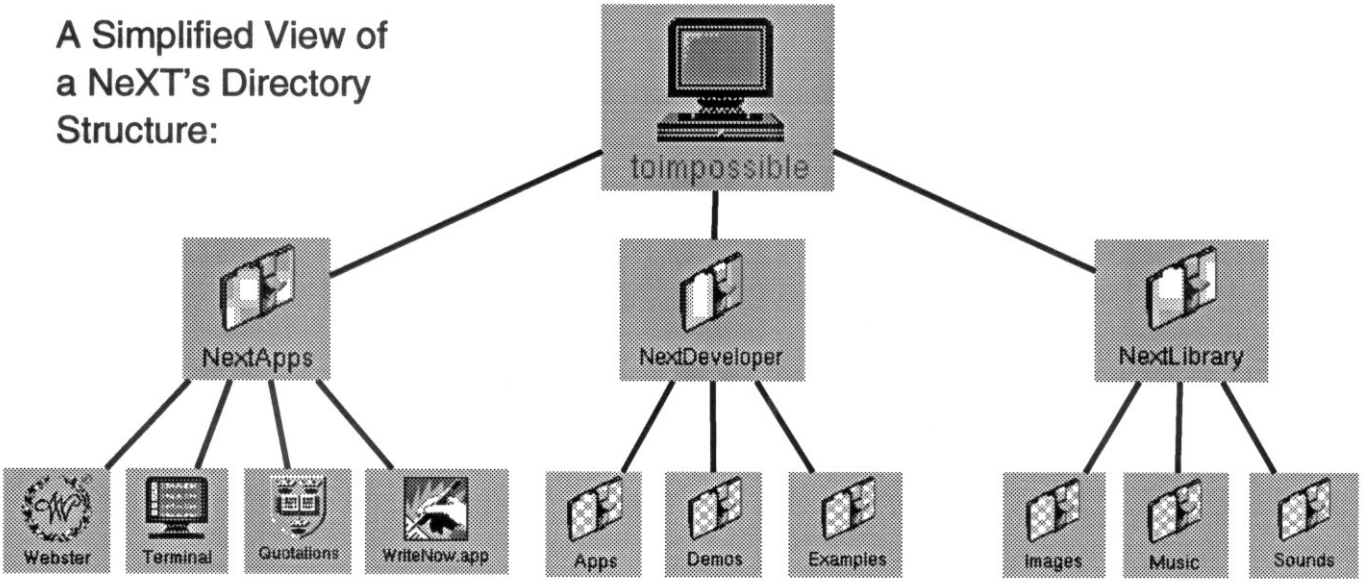
To avoid this mass confusion, computer files, like office files, are grouped together by subject. The office files are placed in *folders*; the computer files are placed in *directories*. This folder/directory analogy is so consistent that the NeXT shows a picture of a folder for a directory's icon.

When you set up your account on the university's computer, you were given your own directory (folder) which is named after your userid. Since all of the files stored inside your directory should be your own, it is logical to have them all grouped together. Furthermore, directories can contain other directories. Every single student's directory is inside another directory called **u** which, on a NeXT, is inside even another directory called **/** (slash), or **root**, which is the biggest directory there is.

From this very highest level, every directory as well as every file can be accessed by following the proper 'path'. In fact, to describe the location of a file, most people will just describe it in relation to the root. So if I want to tell you where the word processor WriteNow is, I would tell you to look in **/NextApps**. The / symbol means start at the top of the directory structure. **NextApps** is the name of one of the directories underneath this top level, and WriteNow is inside that directory. To get

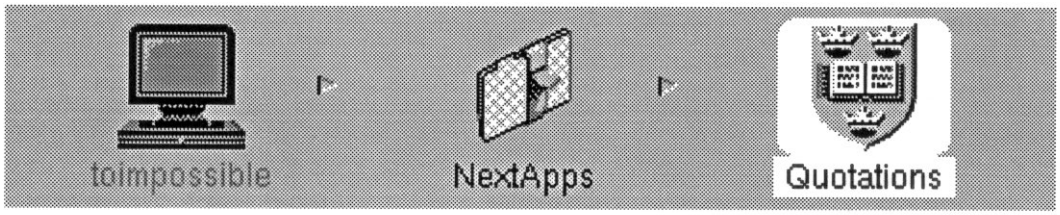
to this point from anywhere in the system, you just have to go up to the **root** and then back down, following the given path:

A Simplified View of a NeXT's Directory Structure:



If you follow the diagram and start at the **root** (on NeXT machines, the icon for the root directory shows a NeXT computer with the computer's name underneath it) and go left to **NextApps**, then go to the right, you will find **WriteNow.app**!

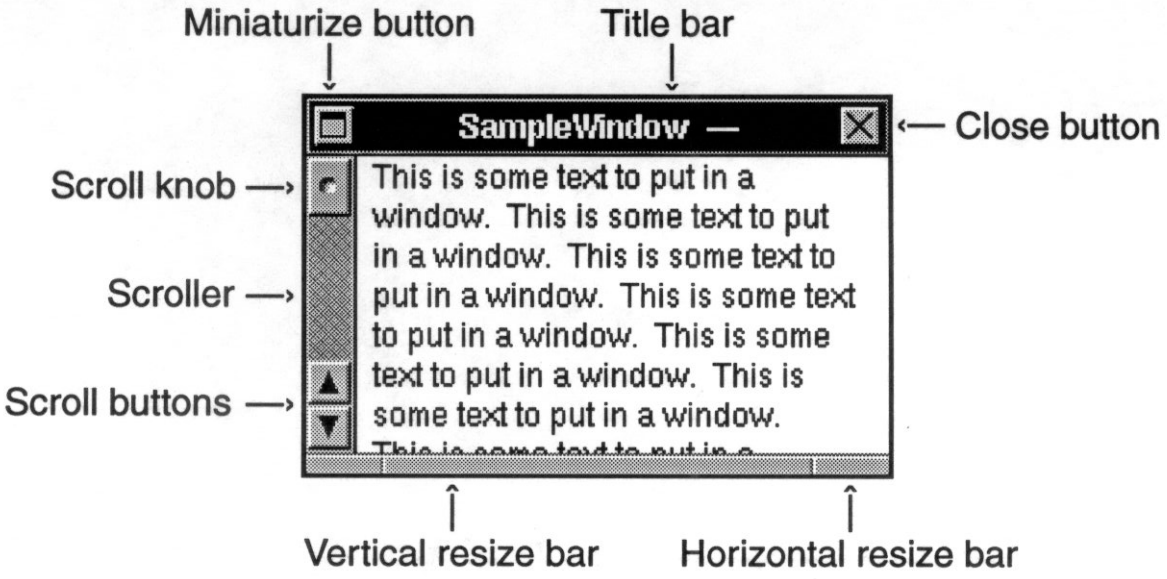
By the way, this type of tree structure is found in almost all computers, not just the NeXT. The difference is in the way the computer interacts with you, showing pictures instead of text. (This is where the term GUI comes from: Graphical User Interface). Instead of just text controlling the machine, icons are used to make the system easier to use. For example, if you were moving in and out of directories and applications in a text-based system, you might get lost trying to keep track of which directory you were in. Fortunately, the NeXT has a built in display to instantly tell you where you're located, and what file, if any, is selected. This information is presented in the middle section of the File Viewer. If, for example, you wanted to look up a quotation in the Oxford Book of Quotations, your path would look like this:



The information presented is pretty straightforward: you are on the computer toimpossible, inside **NextApps**, with the application **Quotations** selected. In order to go back up the tree, just click any item in the path and it will take you back up to that level. So if you wanted to return to the **root** directory, you would in this case click the toimpossible icon. This is one of the major advantages of a GUI—relaying information such as this in an easy-to-understand format and allowing intuitive access to all parts of the computer system.

Manipulating Windows

Just about every application that you run on a NeXT will have at least one window associated with it. Each separate window inside an application has its own features, controls, etc., and each can be moved around and placed where you find it to be most convenient.



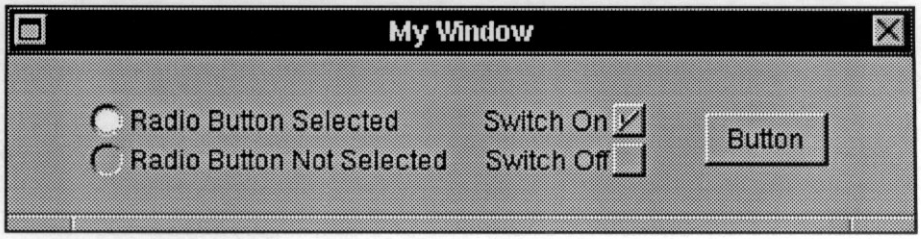
Things to do with a window:

- Move the window around on the screen by dragging the title bar at the top. When the window is where you want it, release the mouse button.
- Resize the window by dragging either the horizontal or vertical resize bar at the bottom of the window.
- If the window can be scrolled, it will have a scroller along the left side (or at the bottom); you can click-drag the scroll knob of the scroller to view different parts of the window contents, or you can click the scroll buttons to scroll a little bit at a time.
- Clicking the miniaturize button in the upper-left corner of the window turns it into a "miniwindow" or icon, which should appear along the bottom of your screen. You can get the full window back by double-clicking the icon.
- Clicking the close button in the upper-right corner of the window will destroy the window and close the document in it. If the X inside the close button is broken, like this:  , the document in the window contains unsaved changes, and you'll be asked whether you want to save the document before closing.

Buttons, Radio Buttons, and Switches

Sometimes applications will have windows with various controls that will allow you to toggle various settings in the program. These usually take the form of a button, a radio button, or a switch. To operate one of these, move the mouse cursor over to the item and click it. If you click a button, the function that is printed on the button

will execute. If you click a radio button, the white dot will move from one well to the other. Note that only one well in a group of radio buttons can have the white dot in it. If you click an empty box, a check will appear in the box, signalling that it is now on. If the switch was on when you clicked it, it will turn off.



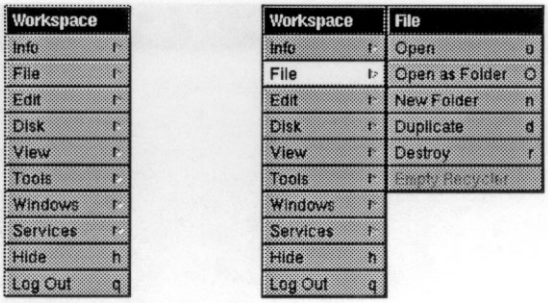
The Application Dock

At some point, you will come across an application that is so useful that you wish you didn't have to move all the way to **/This/Is/Quite/Far/Down/The/Tree.app** each time you wanted to run it. The solution is to move the application over to the Application Dock (or just Dock). This area, on the right of your screen, can hold up to 13 icons, two of which are preset: the Workspace icon (upper right corner) and the recycler (lower right corner).

To insert a new item, find its real location, then drag its icon over to the right of the screen to the desired location. The icon should lock into place. Whenever you want to run this particular application, just double-click its icon in the Dock. To remove an unwanted item, drag it out of the Dock area and it should disappear. (Make sure the application isn't running when you try to do this, or it will just fly back into place). You can easily tell if an application in your Dock is running or not by looking in the lower-left corner of its icon. If three dots are present, then the program is not running; if they are missing, then it is.

The Menu

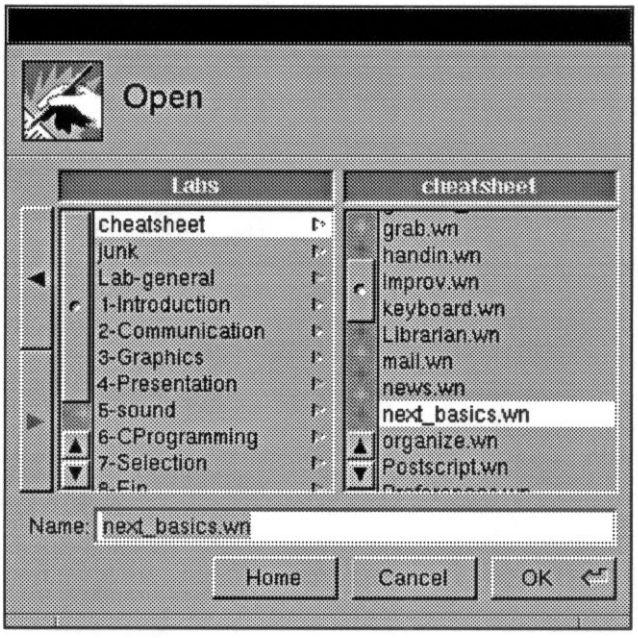
The menu houses all of the possible things you could ever want to do in a particular application. (Well, a lot of things anyway...) It enables you to start a new document, save a sound you just made, print out a picture you drew, or quit that boring game that only *looked* like fun. Every program has a menu which can, at the absolute least, be used to quit. More likely though, it will have many options, some of which are revealed by looking at sub-menus (designated by an arrow to the right of the menu option). To get to these sub-menus, just click the main menu item that you would like to look at (for example, **File**) and it will pop up. As soon as you click another choice in the main menu, the sub-menu will disappear.



Each menu (including any sub-menu) is its own little window, so each can be moved anywhere on the screen. A snazzy little feature is that a sub-menu can be 'torn off', meaning you can grab it as a separate window and move it away from the primary menu. This way, it won't close when you move on to something else in the main menu.

Opening and Saving Files

There are two ways you can open a file that you created with a particular application. The first way is to move in the File Viewer to the directory that contains the file you made, and double-click it. The NeXT will automatically start up the application that created the file and load it. The other way to open your file is to start up the particular application, move to the submenu that will allow you to open files in this application, usually either the **File**, **Document** or **Window** menu (which one depends on each individual application) and choose **Open...**. A window will appear that looks similar to this:



You can move up and down the directory tree by using the arrows at the far left of the window, and scroll through each individual directory by moving the scroll knob. As soon as you see the file that you want, either double-click the name in the browser, or click the name then the **OK** button in the lower-right corner. The application will open your document.

When you want to save a file, there are also several options. If you created a new file and want to save it for the first time, choose the **Save As...** option in one of the submenus (again, most likely either **File**, **Document** or **Window**). A window similar to the Open window will pop up. Move to the directory where you want your file to be located and type a descriptive filename. If the filename that you pick already exists, another window will open up, asking you if you want to **Replace** the other file or **Cancel** the operation. If you replace it, the old file will be wiped away; if you cancel, you can enter a different filename.

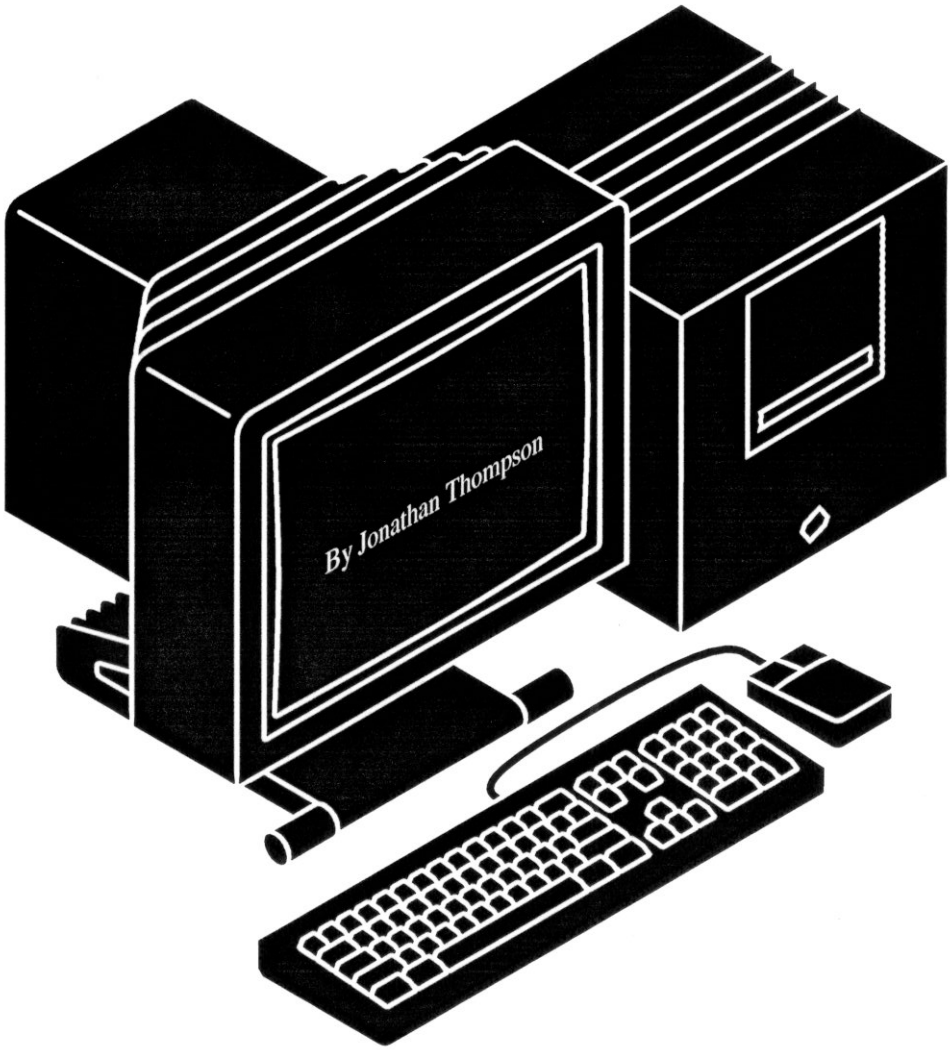
To save a file that already has a name, open up the appropriate menu and choose **Save**. This will automatically update your file. Keep in mind that just because you already have a filename, you might not want to **Save** and overwrite your copy on disk. If you made any changes to your file that you aren't sure you will like, you can still select the **Save as...** option and save under a new name. This way, you can keep the last several versions of a file you have been working on and if something doesn't work out, you can always go back to a previous version.

Ack! Get Me Out of Here! (or How to Logout)

To log out:

- Save everything you are working on (once you quit, it's gone forever unless you saved it!).
- Click the File Viewer and go to the File Viewer's menu.
- Click **Logout**.
- When asked if you really want to log out, click the **OK** button.

Lab #1 - Introduction to the NeXT Computer



BiCapitalization: adj. The act said to have been performed on trademarks such as NeXT, {NeWS}, VisiCalc, FrameMaker, TK!solver, WriteNow and others which have been raised above the hoi polloi of common coinage by nonstandard capitalization. Too many {marketroid} types think this sort of thing is really cute, even the 2,317th time they do it.

–Hacker's Jargon File

Welcome to your test drive of the NeXT computer. Just as you would take a new car to its limit, we want you to take the NeXT to the limit. There are a few things that you have to get used to first, such as moving around and understanding all of the parts on the screen. But once that is under control, we want you to explore and discover what extras are included in this model. Have fun!

Purpose

- Learn some basics about the NeXT computer.
- Learn how to navigate around the NeXT computer.

Materials

For this lab (and all future labs), you will need the following items:

- An account on the NeXT computers
- This lab notebook

References

Required CheatSheets

- Welcome to the NeXT
- Printing
- Terminal
- handin

Helpful Cheatsheets

- Keyboard
- WriteNow
- UNIX

Procedure

- Log in to your account.
- Explore the directory structure, moving up and down through various directories.
- Find the **WriteNow** application (in **/NextApps**).
- Place **WriteNow** onto the application dock.
- Move to the **cs111** home directory by going to the Workspace Menu and choosing **Finder** from the **Tools** sub-menu. Once the **Finder** window opens, click and hold the pop-up list that says **Find File**. This will open up several other choices; move the mouse down to **Find User** and release the mouse button. Type "cs111" and press <return>.
- From the **cs111** directory, move down into the directory **Labs**, then to the **1-Intro** directory. To describe this location in the future, we will just say "go to **~cs111/Labs/1-Introduction**."

- Double-click **questionnaire.wn** (if a notice comes up saying the file is locked, click the **Open Copy** button).
- Using the mouse, move to each question and answer it with the appropriate information.
- Click **Document** from WriteNow's menu. This will open up a sub-menu. From here, click **Save As...** and a new window will pop up which has a button in the lower right corner labeled **Home**. Click the button. This tells the computer to move to your own directory for saving files. Now type "question" and it will save your personalized questionnaire to a file named **question.wn** in your account. (The **.wn** ending, which is placed there automatically, lets the computer know your file is a WriteNow document.) *As a general note, it is good practice to save your files as often as possible, just in case the computer should crash.*
- Go back to the WriteNow menu and click **Print...** to print out your document. If you ever need to turn in a hardcopy of your work, this is the way to do it. For more information, read the Printing CheatSheet.
- Choose **Quit** from the WriteNow menu.
- Open up a **Terminal** window (it is located in **/NextApps**).
- Type `~cs111/bin/addpath` . This command will run the program called `addpath` located in the **bin** directory of cs111. The addpath program will modify your path variable to tell the computer to look in **~cs111/bin** every time that you type a command. Therefore you will not have to type the directory name explicitly when you want to run a program located in **~cs111/bin**. For more information about paths, see the UNIX cheatsheet.
- Type `handin 1 questionnaire.wn`. (If you didn't type the `addpath` command from the last step, you will have to type `~cs111/bin/handin` whenever you want to run the `handin` program.) The program just automatically forwarded the file **questionnaire.wn** that you just created to the cs111 TA's. The number 1 specifies that you are turning assignments in for the first lab. Every time that you are supposed to turn in your work electronically (which is most of the time), this is the way to do it. For more information, read the handin CheatSheet.
- Log out.

What should be handed in

Your questionnaire should be turned in using the `handin` program.

Above and Beyond

If you get finished with all of the other parts of this lab and you still have some time, you might want to explore the NeXT computer some more. There are quite a few interesting applications available.

- Pretty Pictures: from the directory **/LocalLibrary/Images/Scene_screens**, scroll down the list until you find **calvin.tiff**. Double-click it and a background should

show up on your screen. Look at the other files until you find a picture that you like.

- Snazzy Sounds: move to **/LocalLibrary/Sounds** and find **Sack_of_wet_mice.snd** towards the bottom of the list. Double-click it and you should be greeted with an interesting statement. (If you can't hear anything, or if the volume is too high, read the Preferences CheatSheet to learn how to control the volume.)
- Groovy Games: To try your hand at some of these wonderful time-killers, go to **/LocalApps** and double-click at least one of the following: **BlastApp** (similar to Choplifter), **BoinkOut** (like Break Out), **MazeWar** (great with several people, meaningless with one), **Reversi** (like Othello), and of course, **Tetris**.

For Further Exploration

- The CIT document entitled "Getting Started on a NeXT Computer"
- In **/NextDeveloper/Demos** there is an application that will show you around the NeXT. **Guided Tour.app** is really fun to watch, but beware, you should have everything saved before you run it because it automatically logs you out when it starts.

Lab #1—TA Sheet

August 23, 1991 - Jonathan Thompson

This lab will either be the easiest or the hardest to run, depending on how quickly the students pick up the fundamental concepts presented. The lab is purposely short for two reasons: to allow those students who have no idea what is going on to get the hang of things, and to give the other students who finish the lab early an opportunity to go exploring.

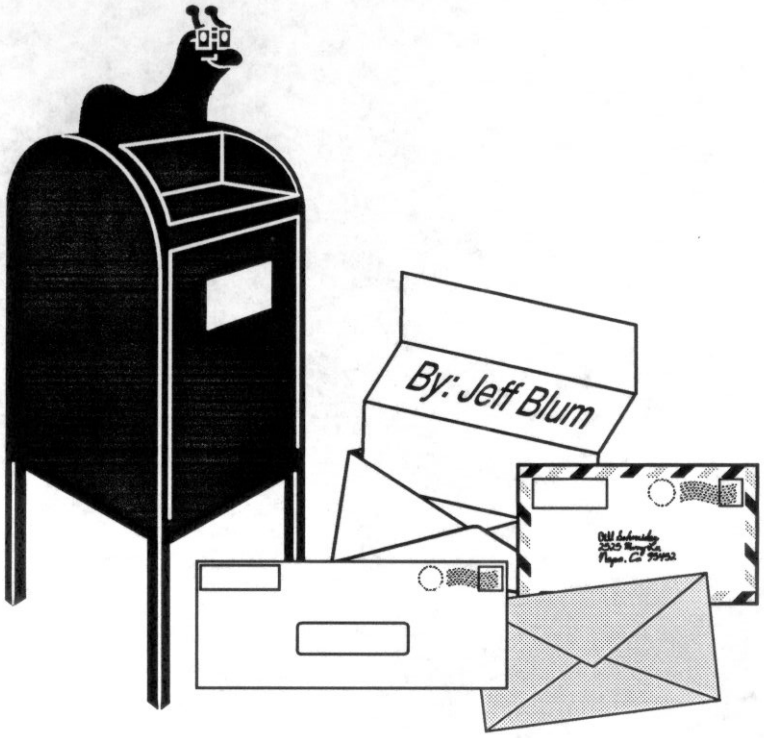
It is imperative that the students read Welcome to the NeXT in the beginning of the lab manual. This covers all of the fundamentals from understanding how a Unix system is set up to learning how to operate a mouse.

In the **Above and Beyond** section, the students play sounds out of the **/LocalLibrary** directory. You will want to go around to each machine and make sure that the volume is not muted and at an audible level. Fortunately, this is possible without having to log into the machine:

> Go to each NeXT and type the <Enter> key on the numeric keypad. If the volume is set to a good level, you will hear the error beep. If you don't hear anything, or the sound is too soft, press the 'up-volume' key for a couple of seconds and then hit the <Enter> key again. If you still don't hear anything, try typing the command key and the 'down-volume' key which toggles the mute function on and off, and hit the <Enter> key yet again. Keep playing with these two commands until you get the error beep to sound.

Make sure that the permissions are set so that the students will be able to have access to the **questionaire.wn** file in **~cs111/Labs/1-Intro**, but not the ability to write to it.

Lab #2 Communications



snail-mail: n. Paper mail, as opposed to electronic. Sometimes written as the single word 'SnailMail'. One's postal address is, correspondingly, a 'snail address'. Derives from earlier coinage 'USnail' for which there have been parody posters and stamps made. Oppose {email}.

email: /ee'mayl/ 1. n. Electronic mail automatically passed through computer networks and/or via modems common-carrier lines. Contrast {snail-mail}, {paper-net}, {voice-net}. See {network address}. 2. vt. To send email to a person.

–Hacker's Jargon File

Communication is part of being human. Banging away at a computer keyboard all night may seem completely anti-social, but used properly, computers can greatly enhance communication between people, especially over long distances. This lab will cover several different ways of communicating and exploring via computer. When you are finished, you should be better aware of the vast communication potential of the Internet and be able to reach out to people across the globe. Armchair travel takes on a whole new meaning when you can talk to people in Finland, Germany, and many other countries at the drop of a hat.

Purpose

- To become an expert at e-mail.
- To learn to start and read news.

Materials

- An urge to explore the world.

References

Required CheatSheets

- Copy&Paste
- News
- Terminal
- E-mail
- WriteNow
- handin

Helpful CheatSheets

- UNIX
- ftp
- telnet
- talk

Online Help

- mail man page
- **Help** screen for Touch

Procedure

Log in to your account

Prepare a document for submitting your data

- Start WriteNow.
- Choose **Save As...** from the **Document** menu.
- Save this (currently empty) document as **Communication.lab.wn**.

*This document is where you will **Paste** or type all of the data we ask you to hand in. Most labs will be operated in a similar fashion. You will create one or more documents to store your findings and proof of doing the lab, then use the* `handin` *program to submit them before the due date. As you progress through each lab,*

place any comments or revelations into a file as well. You can submit anything you like...not just what we tell you to. Be bold!

Mail

- Open a Terminal window.
- E-mail your favorite quote to everyone in your row of computers.

Note: While you are within `mail`*, you will not be notified of any new mail you receive. Therefore, you must quit and re-start* `mail` *in order to see if new mail has arrived.*

- Compile a list of the quotes you receive, placing yours first. You can easily do this by using Copy&Paste to place them directly into **Communication.lab.wn**, then rearrange them into the proper order using Cut&Paste.
- Save this file again. (So your changes to the document are on disk.)

News

- Start news (use `rn`)
- Read articles from at least 3 newsgroups. Possible choices (merely suggestions):
  - comp.sys.next (all about the NeXT computer)
  - alt.stupidity (strange)
  - alt.folklore.urban (very silly)
  - rec.humor (not very funny for the most part)
  - alt.flame (people yelling at each-other)
  - alt.activism (issues&issues)
- Copy&Paste a few articles into **Communication.lab.wn**.
- Save your file again! (The more often you save, the safer you are.)
- Exit news

Before the deadline given to you, make sure you submit the ***Communication.lab.wn*** *document you created, as well as anything else you would like us to see. Remember that the* ***Above and Beyond*** *section counts too!*

Log out of the NeXT machine.

What should be handed in

- The quotes you collected, with yours first.
- Articles from 3 newsgroups of your choice.

All of this should be contained in ***Communication.lab.wn****. Feel free to put any other comments or findings there as well. If you don't put it in this document, we will probably never see it! Just make sure you submit it on time using the* `handin` *program. Good luck!*

If you get done with all of the above

Do this before the **Above and Beyond** section.

Telnet

- Choose a site to `telnet` to.
- Log onto the site as per the instructions for that site.
- If it is a BBS (bulletin board) set up an account.
- Get and `handin` proof that you were actually on another site. (Once again, Copy&Paste this into **Communication.lab.wn**.)

Above and Beyond

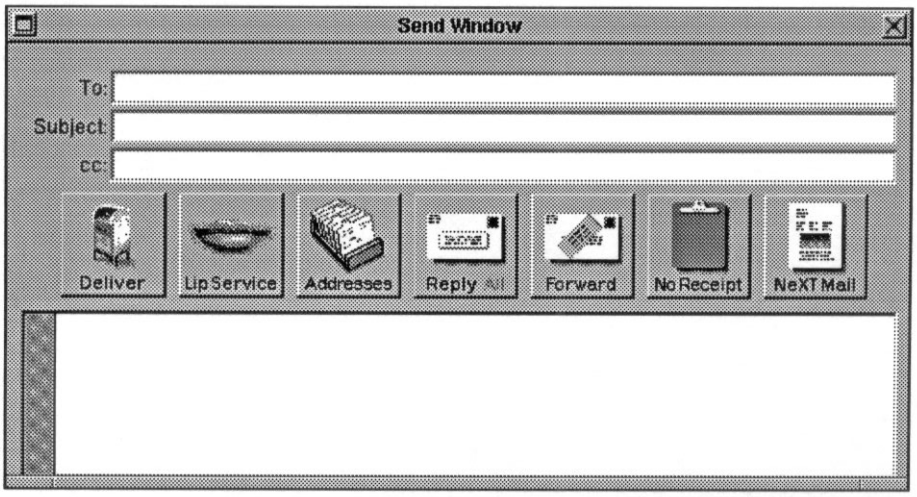
Try anything you have time for, in any order.

NeXTmail

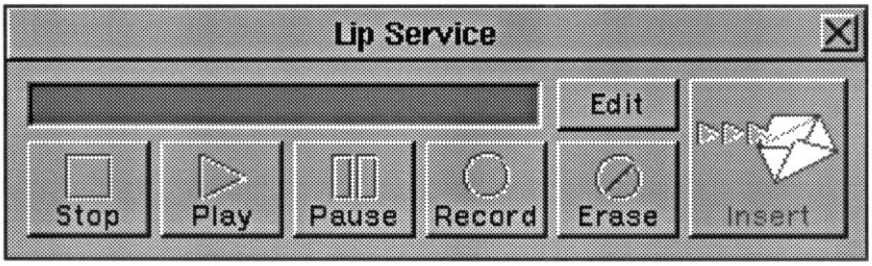
Want to send your voice in mail? There is an application on the NeXT that allows you to send not just text as mail, but your voice as well. To try it out, find a willing partner (victim) to send the voice mail to. Since this will goof up the normal `mail` *program, your partner must know that it is coming and use the same application you do.*

To send NeXT mail (with sound)

- Find the application **Mail.app** in **/NextApps**. Double-click it and choose **Send...** from the **Tools** menu. You will be faced with a screen similar to the following:



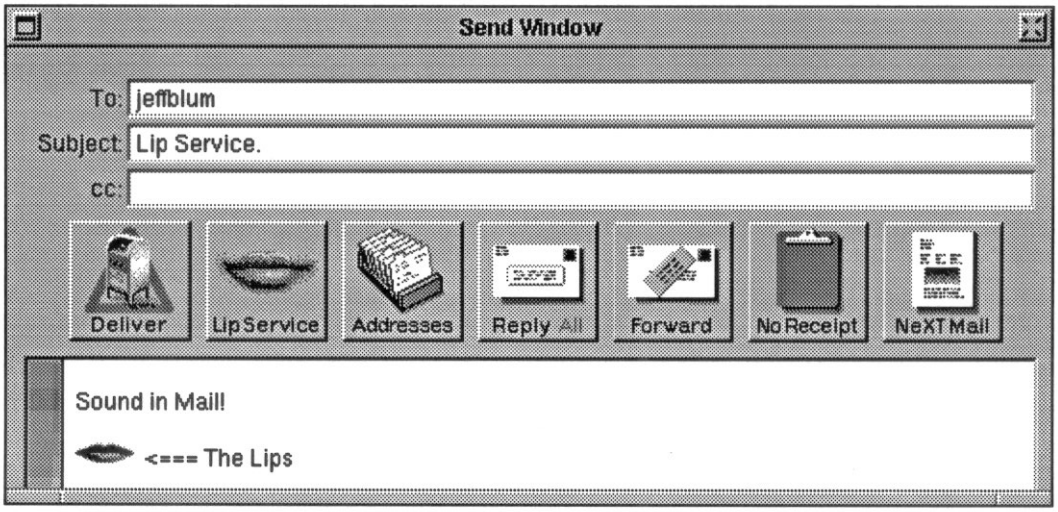
- Enter your partner's e-mail address in the space marked **To**, and the subject of the message in the **Subject** area. You can now type text in the window, Copy&Paste graphics, and of course, add sound. To include your voice, click the **Lip Service** button in the window. Up comes a tape recorder-like window:



- Click **Record** to begin recording your voice (the microphone is on the front of the monitor), **Stop** when you are done speaking, and **Insert** to put the recording into your mail.

If you want to modify the sound before sending it, click **Edit**. *This is a fun option to play with. All I will say is that you can Cut&Paste parts of the sound to mutilate it however you like by selecting the piece of the sound you want to fiddle with, and using the* **Cut** *and/or* **Copy** *options from the* **Edit** *menu. For more information on digital sound, see the Sound cheatsheet.*

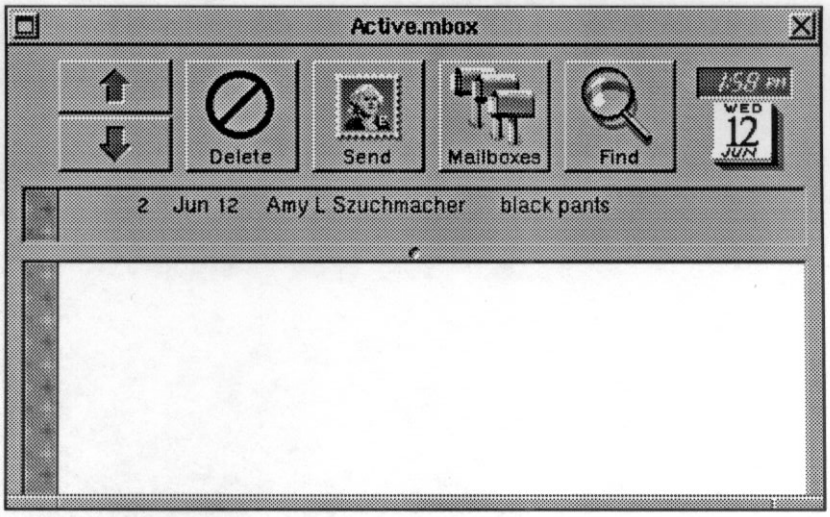
- When the sound is finished, edited or not, and you have clicked **Insert** to put it into your mail, a pair of lips will appear in your letter:



- The voice mail is ready to send, so click the **Send** button. Now it is your partner's turn to receive the mail.

To receive NeXT mail (with sound)

- First, load up **Mail.app** exactly as above, but instead of clicking **Send**, you will need to open up your mailbox. If you do *not* see a window similar to the following on your screen, follow the instructions below:



- Go to the **Tools** menu and choose **Mailboxes...**
- Click **Active.mbox**, then the **Open** button in the lower left-hand corner.
- The above window should soon appear.

- Once you have this window, read the mail you have received by performing these steps:
  - Click the mail header (the Amy L Szuchmaker line) to read the message.
  - Double-click the lips to hear the voice message. *Make sure your sound is turned on, or you will not hear anything! See the Preferences CheatSheet for more information.*

If the Message that was sent to you does not appear in the header area, either the mail is still in transit, or something went wrong. Depending on how much sound and/or graphics were sent, it can take several minutes (possibly even 10 in the worst case) for the mail to arrive. To force **Mail.app** to check if any new mail has arrived, choose **New Mail** from the **Utilities** menu. If it is taking forever, you may want to try a short voiceless message to make sure you are doing everything else correctly. Repeat the steps from the beginning, but do not add any voice to the message. If even this does not arrive, flag down a TA.

Talk

Writing mail to somebody is a great way to communicate, but it lacks the immediacy of a face-to-face conversation. Unfortunately, we do not have the facilities to do video teleconferencing, but you can write messages to another person in real-time using a program called `talk`. See the `talk` CheatSheet to learn to use it.

NewsGrazer

Sure, `rn` is fine for reading news, but just as NeXTmail takes normal mail several steps further, there exists a NeXT application called NewsGrazer that takes `rn` into another realm. It automatically checks for new news periodically; it offers point&click simplicity rather than difficult-to-remember `rn` commands; it allows you to move between articles with great flexibility.

Just so you are warned, NewsGrazer is one of the most controversial newsreaders on the net since it allows RTF (Rich Text Format) postings, which can contain graphics, sound, and different fonts—very similar to what NeXTmail allows. Why is this controversial? It takes much more 'bandwidth' (time and money) to distribute non-text articles than plain text ones. This angers some people since they have to pay long distance phone charges to get their news. Longer RTF postings can take quite a while to transfer, so these folks' telephone charges rise considerably. Nonetheless, if you avoid posting long RTF articles nobody should get too upset. Just one other thing to keep in mind is that the majority of people reading non-NeXT specific newsgroups will not be able to use NewsGrazer since it is a NeXT application. Therefore, chances are that 90% of the people reading your post will never have heard of RTF postings, and will see gibberish on their screens.

Don't let these warnings scare you away from using NewsGrazer. It will do plain-text postings that are indistinguishable from normal posts. And even if you never post anything, NewsGrazer offers so many advantages for reading news that is worth checking out.

NewsGrazer is located in **/LocalApps** and is exceedingly simple to use. When it starts, it will present a hierarchical list of newsgroups. If you want to get to comp.sys.next.misc, for example, you would first click "comp" in the list, then "sys" in the new list that appears, followed by "next" and "misc." A list of articles will appear in the upper right of the window. Click the one you want to read. The article will show up in the large lower box!

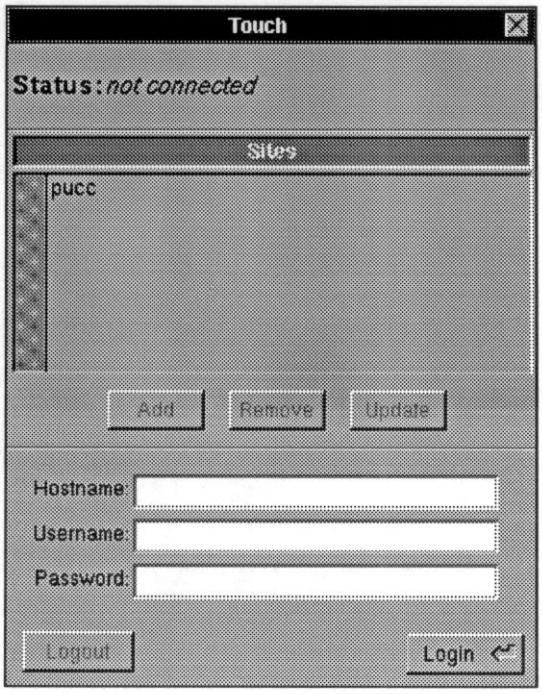
Ftp

You know how to `telnet` to different places, but not how to get data from one machine to another. Let's say that someone tells you about a REALLY neat file that you can use in your next PSY207 paper, but it's in Finland. Hmmm. If the site in Finland is what is known as an "anynomous ftp site" you are in luck. All this means is that you can use the `ftp` command to go to the site, log in anonymously, and grab files. There are only a few commands that you need to know, and all of them are either explained or discussed in the `ftp` CheatSheet. If you are looking for programs for your own computer, ftp sites are a very valuable resource since they often have the newest releases of free software. See the `ftp` CheatSheet for some of the better sites to try.

Touch

Some clever person has created an application to make `ftp` far easier to use than going through the above procedure. It adds a NeXT interface to `ftp` and makes files from an `ftp` site appear to be in another File Viewer in your Workspace. You can drag files from the ftp site to your directories and avoid all the typing, etc. that plagues normal `ftp`. It will also take care of binary files for you, so you do not have to worry about the `binary` command. However, the main advantage Touch offers is the appearance that the ftp site's files are actually on your system. You can run applications, view text files, etc. as if they were in a normal directory. In fact, you can even get to the files in a Terminal window! Just `cd` to **/tmp/touchdir**. The ftp site's files will appear there.

To use this application, double-click **Touch** in **/LocalApps**. It will load and present you with this window:



In the lower part of the window enter the name of the ftp site and the login information you would normally use in `ftp`, then click the **Login** button. The window will disappear and you will be placed in the Workspace. After a few moments (be patient) the site should appear in its own mini-File Viewer. Use it like the normal File Viewer to find the file(s) you want, then drag them into your directory in the normal browser. It will say "copying" as if it were copying a normal file. When this message disappears, it is finished, and you can disconnect from the site. Double-click the **Touch** icon and click **Logout** in the lower left of the window. In a few seconds the window should again look like the above illustration. The second File Viewer will still be there, but the directory called **touchdir** no longer exists since you logged out of the site. For this reason, you may want to click the close box on the second File Viewer to get rid of it. If you do not, it will just hang around being annoying, but do no other harm.

The only thing you must avoid while using Touch is choosing **Update Viewers** from the **View** menu. This will crash the application force you to reboot.

For Further Exploration

The best way to learn about communication with computers is to go off exploring. There is no real manual for scouring the net, just the accumulated wisdom of many people. `ftp` to different places; use `mail` to its fullest; keep up with a few netnews groups. If you stay involved with the Internet, you will eventually accumulate quite a knowledge base. Since it is so large and diverse, few people, if any, ever master the net. Use e-mail to communicate with your TA's and professors. Contact friends at

other Universities. Play games at `telnet` sites. There are many documents on specific aspects of the net, so try running an archie search on a topic you are interested in, then `ftp` to the site and get the data. (Archie is explained in the `telnet` CheatSheet.) Great exercise while getting the information you want!

One other thing to try...There is a chat system you can join that allows you to talk to an entire group of people at once. Subject matter varies widely, depending on who is on at the time and what subject groups exist at the site you visit. At the time of this writing (8-25-91), the program to connect you up can be found in **~dcoster/bin** and is called `irc`. Therefore, to run it, type, `~dcoster/bin/irc`. Unfortunately, it only works on Phoenix, so you will have to `telnet` there before typing this. Go ahead and run the program. Look around. You can try typing `/help` once you are connected, but I make no guarantees that it will always work. This system seems to change quickly, and sometimes doesn't work how you expect it to. Also, you are running it out of someone's personal directory, so it may disappear at any moment. You are on the edge...be adventurous and give it a try sometime. :-)

Lab #2—TA Sheet

8-22-91 Jeff Blum

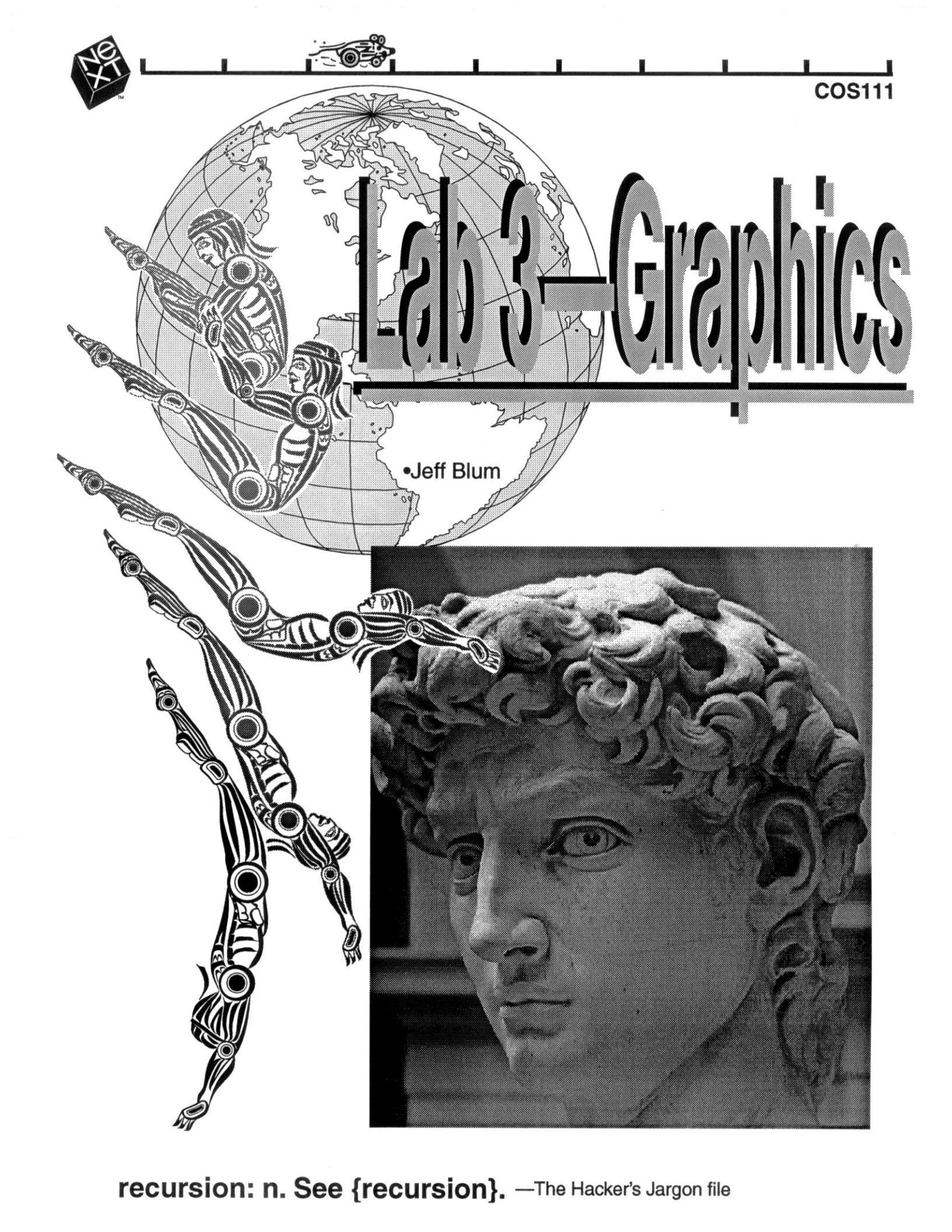
Before this lab begins, you may want to go over Copy&Paste with the entire class since it is nearly required for the lab. A quick demonstration may be in order to achieve maximum clarity. This concept should be fully grasped before the lab to avoid confusion on how to save stuff for submission. Our concept of what the student should do is create a WriteNow document that they can Copy&Paste material into as they progress through the lab. You can mention that this is probably the best method, but do not stifle any other ideas they may have, as long as they look like they will work. Your discretion.

The primary goal of this lab is probably getting the students to use e-mail as much as possible, and learn to feel comfortable with it. Unfortunately, this lab will probably be one of the rougher ones because it is the first lab where they are expected to perform tasks that make use of the NeXT fundamentals. Most students will probably still be a bit shakey on these important concepts, so expect a few problems. We have included only two mandatory tasks in this lab: mail and news. `telnet` was supposed to be required, but we felt that it was stretching the limits for this lab. If even this seems to be too much, you may want to scale it back on the fly. Remember one of the most important tenets of this course:

Computers are our friends!

Sacrifices in the amount of material covered may have to be made to meet this goal. It would not do to have 30 frustrated students hating computers because they were pushed to learn one more application.

I think you get the idea. Good luck!



recursion: n. See {recursion}. —The Hacker's Jargon file

*More and more of the world's computing power is going into processing graphics. The interface you use on the NeXT is a GUI—*Graphical *User Interface. Movies such as Terminator II use computer graphics for many of their special effects. Even the text that you print out on laser printers is graphical since each character of a font is drawn using a graphics package. In this lab you will learn to speak a common graphics language—PostScript. You will also see demonstrations of color graphics, including real-time video on the NeXTdimension computers in the back of the lab.*

Purpose

- To learn the fundamentals of PostScript for several reasons:
  - It is what the NeXT computer uses to display everything on the screen.
  - It is the language used in most laser printers to describe what to print.
  - It is a stack-based language, which is interesting in itself.
- To explore the current state of high-end graphics processing.

References

Required CheatSheets

- PostScript
- UNIX, especially the section on `lpr`.
- Edit
- Yap
- TopDraw

Helpful CheatSheets

- Recursion

Online Help

- man page for `lpr`
- Help within TopDraw—click **Help...** in the main menu.

Procedure

Log in to a NeXT

PostScript

- Start Yap.
- Draw a nifty picture, using PostScript. (See the PostScript CheatSheet for some examples.) From Yap, save it in your directory as **postscript.ps**. *This will be submitted later, after you have printed it out and looked at it.*
- Open a Terminal window.
- Use `lpr` to send the PostScript file you just saved directly to the printer. *This is to show you that this is indeed the language the printer uses. It is going directly there!*
- Start TopDraw and create a simple drawing similar to the one you created using PostScript.

- Save the drawing in one of your directories by choosing **Export...** from the **File** menu. Use the filename **TopDraw.drawing.topdraw**, and make sure the EPS icon (toward the bottom of the panel) is highlighted. If it is not, click it. This will ensure you are saving in a PostScript format. *You will submit this file later, so don't lose track of it!*

- Load the file from within Edit :
  - Start **Edit** by double-clicking it in **/NextApps**.
  - Choose **Open...** from the **File** menu.
  - Find your file, click it so it is selected, then click the **OK** button.

- Realize that hey...this looks extraordinarily similar to (although longer than) the stuff you wrote in the PostScript section...

Congratulations. You can now speak the same language as the printer. Many Macintosh, NeXT and IBM laser printers use this language, so most drawing programs do nothing more than take what you draw and translate it into PostScript. You now know how to cut out the middle man and talk directly to the printer!

Nifty graphics & video

- Meet the TA in the back of the room to see the graphics demos on one of the color NeXTdimensions. Bring popcorn.

Do not forget to turn in all of the material listed below before the given due date!

Log out of the machine.

What should be handed in

- Your PostScript drawing(s). (**postscript.ps** and any others you create)
- Your TopDraw masterpiece (**TopDraw.drawing.topdraw**)
- Anything you do in any other sections of this lab, or anything else you want us to see. We're not picky!

Above and Beyond

Try anything you have time for, in any order.

More PostScript

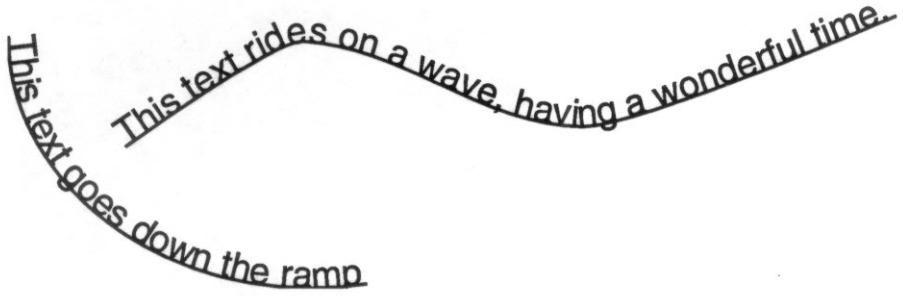
Go for it. Do anything you want in PostScript, but go beyond the simple drawings in the mandatory part of the lab. Try some of the more advanced PostScript operators, especially the ones for looping. If you really want to test your knowledge of PostScript, try creating your own recursive picture. See the Recursion CheatSheet for more information. You may not succeed in making a real recursive drawing in the time you have, but you may still get interesting results. Turn in anything you do, including non-working code. Recursion is difficult, so a valiant attempt to use it will be smiled upon. A good exercise to try is modifying the example code in the Recursion CheatSheet. Tell us what you try to do, then include the modified code and your evaluation of the results it produces. Good luck!

Special Effects in TopDraw

Choose several advanced features of TopDraw and use them to create a drawing that uses more than just basic shapes and fills. Possible choices:

- **Effects** from the **Draw** menu
- **Skew**ing from the **Transform** menu
- **Flip**ping from the **Transform** menu
- Placing text along a path as in the example below. (Just draw your path, double-click it using the pointer tool, and type away!)

These are just some of the more obvious choices. Explore others! Also, don't forget to put your results into a file and use the `handin` program to turn them in!



For Further Exploration

More color graphics

If you want to see some more amazing color graphics, check out the Interactive Computer Graphics Laboratory (ICGL). This lab contains many IRIS computers, which are very fast graphics machines with some incredible demos installed. They are located in E-Quad room B423. You can login with your normal userid and password, but the operating system is different than the NeXT's, so you will have to pester a guru there to show you how to get to the demos. If there is enough interest, perhaps a TA will go over for a few hours and show you some of the better ones. Another neat feature of the IRIS computers is a flight simulator called DOG, which allows you to fly against other humans in an aerial dogfight. Amazing graphics, amazing game. It was banned a while back for using too much computing power, so you can currently only fly a plane solo...no dogfighting. Nonetheless, it is still quite a bit of fun.

Lab #3—TA Sheet

8-23-91 Jeff Blum

This lab has the potential to be quite a bit of fun. Seeing computer novices at work using powerful graphics tools should prove interesting, especially if they have never done *any* computer graphics before.

Expect several people to bog down on the PostScript section of this lab. Since this is probably their first exposure to any sort of programming, the key is for each student to get results quickly. Encourage typing in one of the example programs from the CheatSheet and changing parts of it. Students will probably become more hyped if they see the results of their actions as soon as possible. After everyone seems able to get results on screen, encourage experimentation. Quantity over quality in the Postscript section of this lab may not be entirely bad. The next lab (Presentation) will focus on more aesthetic presentation work, so this lab can be more free and fun.

TopDraw should be a welcome relief to most students after they tire of Postscript. It is quite powerful and easy to experiment with. Once again, let them go wild. If it seems natural, you may want to start them cutting and pasting into other applications to get them used to the idea. This will be important in the next lab, so why not start early?

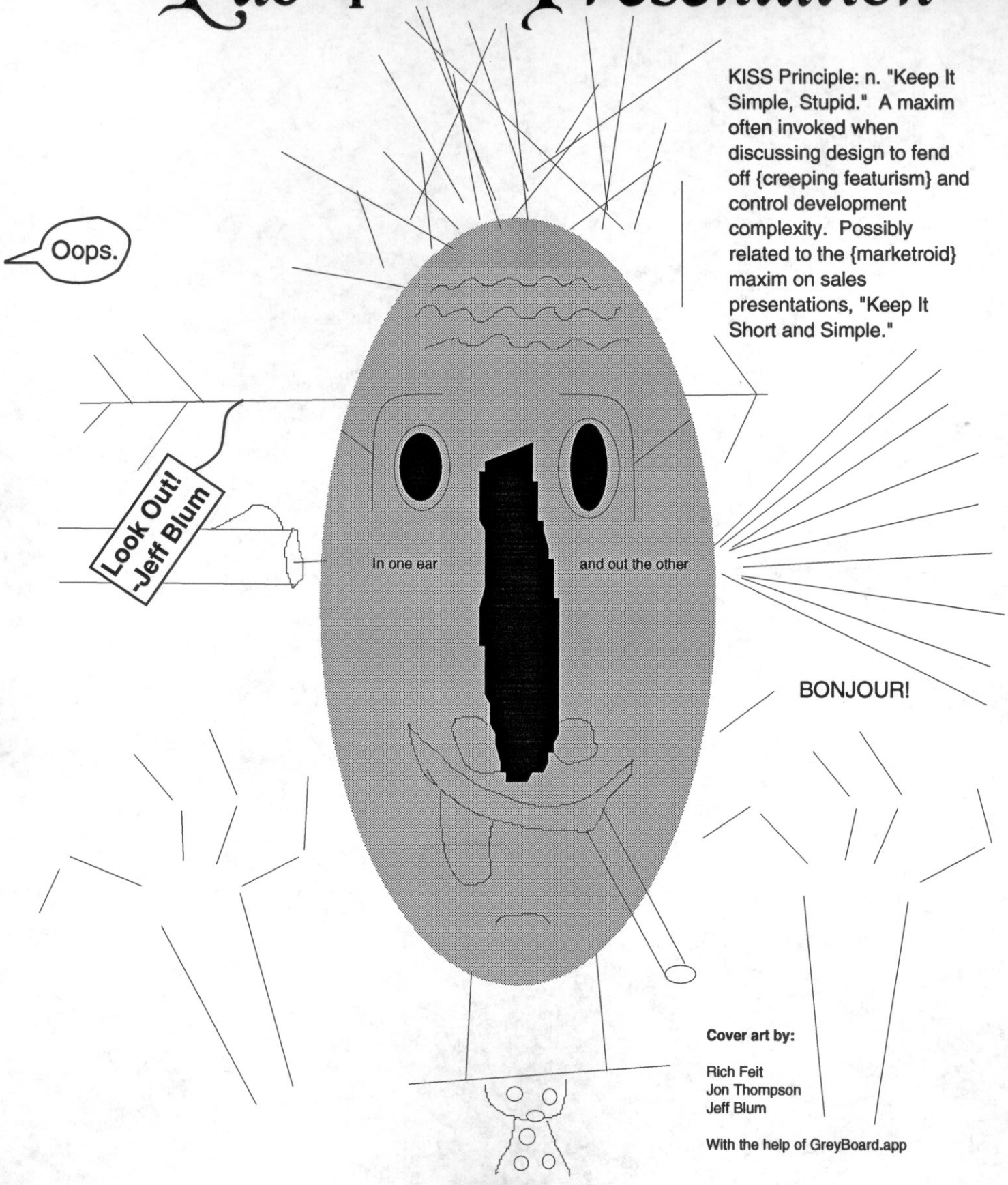
For color graphics on the NeXTdimensions, the best effect we have found is real-time video. Just get a VCR and two RCA cables from Media Services in East Pyne, hook it up to the NeXTdimension board in the back of a color cube, and run **/NextDeveloper/Demos/NeXTtv**. This will allow you to display the video directly off of the VCR— it is *digitized* in real time! For sound, you can either digitize it through the microphone jack on the sound box, or use something like digital ears. Otherwise, you can fudge and hook some (amplified) speakers up directly to the audio output jacks on the back of the VCR. Cheap, but it works. The latter may work out better since it is easier to implement and avoids digitizing delays that will throw the audio out of synch with the video. Good luck, and arrive early to set this one up!

Getting the VCR from Media Services needs advance planning. Ask several *weeks* in advance since you will have to fill out a bunch of forms and work out logistical details with them. Trust me. It is worth the trouble. Of course, if you own a VCR, it would be easy to just drag it along for the labs, but I leave these decisions to you.

The last part of the **Above and Beyond** section is optional on your part. Since at this time any really cool demos for the NeXT are hard to come by, and the IRIS computers have a plethora of nifty ones, you may consider it worthwhile to take a group over to ICGL for an optional get-together. You should probably go over and pre-select the neat demos so you know what you are doing before shoving 20 impatient novice computer folks into a small room with you. Just a thought. :-)

In case you did not know, **Buttonfly** is the key to viewing most IRIS demos. The people that staff ICGL are very helpful, so you can ask them for assistance if you get stuck. BTW—If you have not seen the IRIS demos in action and flown in DOG then you are in for a surprise as well!

Lab 4 -- Presentation



Cover art by:

Rich Feit
Jon Thompson
Jeff Blum

With the help of GreyBoard.app

The computer is a powerful tool that can help you present your ideas with flair. A good computer presentation can appear difficult to produce, but this is often not the case. In this lab you will learn how to use several different applications on the NeXT computer to create a single final product. Since this will combine the best aspects of several powerful programs, the results should be fairly impressive.

Also in this lab you will experiment with a tool that attacks the problem of producing nicely formatted text and equations from a very different angle than a typical NeXT application. This tool is called `troff` *and was widely used in the past to do most text and equation processing on UNIX machines. In fact, many people still use it today—Professor Steiglitz wrote the Introduction to this lab manual entirely in* `troff` *and its associated tools. In addition, UNIX documentation is still done in* `troff`*. So, even though you may scoff at its primitive interface, it is still a very useful utility. Later in this lab you may find that it is far better at creating equations than a modern word processor!*

Purpose

This lab has two major goals:

1. To learn different forms of text processing
   - Text formatting (`troff`)
   - WYSIWYG (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et)
2. To learn about integration between applications.

References

Required CheatSheets

- troff
- Edit
- WriteNow
- CheatSheets for the applications you choose.

Helpful CheatSheets

- UNIX
- handin
- Print

Other

- Manuals for the applications you use. (These can be found in the manual rack near the door of the cluster.)

Procedure

Troff

- Start Edit, choose **New** from the **File** menu, and save the document that appears as **troff.paper**. This is where you will type in your `troff` code.
- Create a fairly simple document using `troff`. Use at least 3 of the following `troff` features:
  - centering
  - title

    - author
    - underline
    - change the font (italic, bold, etc...)
    - change the margins
- Insert an equation. (It need not be too complex.)
- Make sure it says "troff" somewhere near the top of the page.
- Save the `troff` file.
- Print out the document using the commands in the `troff` CheatSheet so you can make sure it worked. If it did not, fix and reprint it until it is correct. Remember to keep saving your changes, or they will not be printed or submitted!

WYSIWYG

- Start WriteNow.
- Create a document as similar as possible to the `troff` document you just finished. (Including the equation...but don't spend all day! hehehe)
- Make sure you put "WriteNow" somewhere near the top of the page.
- Save this file as **WriteNow.paper.wn**. You will submit it later. If you like, you can print it to see how similar it actually is to the `troff` document you created.

Integration

This section will introduce you to various applications on the NeXT and teach you how to use more than one of them to produce a single final product. For example, you can use the Improv application to produce a 3-dimensional bar chart that you can then copy into a WriteNow document. (See the Improv CheatSheet for an example!)

Each application you will be using has its own strong points. Your task is to decide which parts are useful, then use them to their fullest in your final piece. TopDraw, for example, is an excellent drawing program, but you would not want to write a novel with it. WriteNow is good for text, but has almost no internal graphics capabilities. However, you can use both of them to get a document with excellent text and graphics.

- Use *at least* two different applications to produce one final product. We have prepared CheatSheets for the following applications, but feel free to use any that you like:
    - WriteNow
    - Improv
    - TopDraw
    - Diagram!
    - Yap
    - Scene
    - Grab

What should be handed in

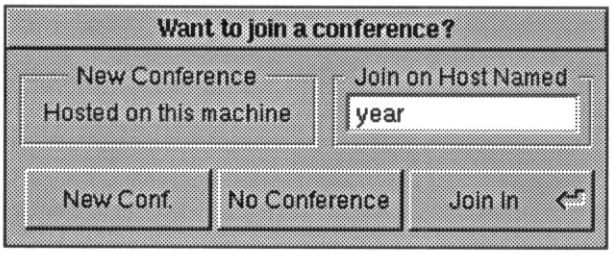
- Your `troff` document, **troff.paper**.
- Your WriteNow document, **WriteNow.paper.wn**.
- The document you created using at least two different applications.

Above and Beyond

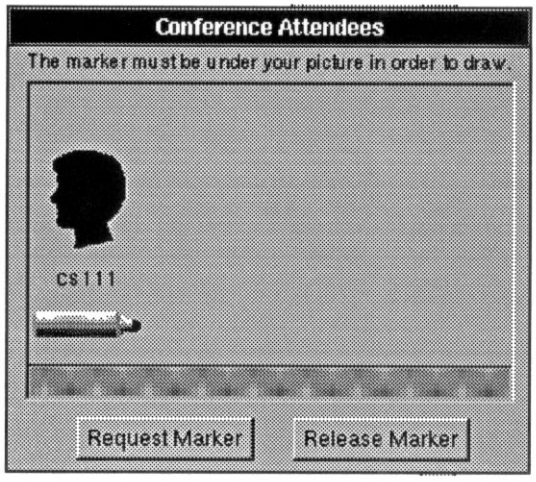
GreyBoard

OK—this paragraph is being added soon before the final labs are printed out. When this section was written, GreyBoard was a stable application—it always worked. When we tried it recently, however, it failed to work at times, giving an error about not being able to connect to a host. Hmmm. If this happens, just try it a few times—it seems to work if you try it enough. If it continually malfunctions, you may wish to skip this section.

This application can be incredibly amusing since it allows more than one person to work on the same drawing at the same time. To use it, you will need a group of people all logged into their NeXT accounts. Look in **/LocalApps** for the application **GreyBoard.app**. One person in the group must start the application and click the **New Conf.** button in the following window:



This will bring up a new "drawing board" to work in. The others in the group can now start the same application and enter the name of the machine that the first person is on in the "Join on Host Named" box. Click **Join In** to join the group. There should be three windows, one large one for the drawing, one with the drawing tools, and one that looks something like this:



As each participant joins in, a new silhouette will appear with their userid under it. The head with the marker beneath it is the person who can currently modify the picture. If you want the marker, click **Request Marker**, then when you want to give it to the next person, click **Release Marker**. Draw to your heart's content. You can save the finished masterpiece, Copy&Paste data from other applications, etc. Beware...these sessions usually turn out to be a bit on the silly side. (e.g. the title page for this lab...)

ShowAndTell

ShowAndTell is the application used to make demonstrations like GuidedTour.app. Using it, you can make your own presentations combining action on the NeXT with simultaneous sound. The application is located in **/NextDeveloper/Demos**, which is a polite way of saying that it isn't perfect. It will crash occasionally, and it does some strange things at times. Nonetheless, it is fun to play around with. After starting the application, click **record** to begin recording a new script. Any sounds you make will be picked up by the microphone in the monitor, and anything you do on the computer (typing, moving the mouse, opening applications, etc.) will be recorded for later replay. Click the **Stop** button to end the recording, and **Play** to see what you have created.

Important Note!!!!!

It is important to realize that this application records *what you are doing* and ***not*** the results of those actions. Therefore, if you are recording a script and double-click an application's icon, when you play back the script, you must have that icon in exactly the same place on the screen. If it is not there, ShowAndTell will double-click in that area anyway, and *strange things will happen*. Guaranteed. The machine may crash, a different application may be started, whatever. On the other hand, this should not stop you from experimenting. If the machine blows up, who cares? Just reboot it. (See Emergency CheatSheet.)

For Further Exploration

This lab has shown you some of the tools and technical details that you can use to create effective presentations. It did not, however, tell you what stylistic techniques you can use, or how to arrange presentations effectively from an artistic standpoint. Reviewing others' presentations before making your own may be a good use of time. If you see an advertisement that you really like, look at it and think about *what* makes it attractive, then try to incorporate those elements in your own work. There are even specific courses available for learning desktop publishing (DTP). Try calling CIT (258-6028) and asking if they have any courses running in the near future. Since you are a student, any course you take from them should be free.

Lab #4—TA Sheet

8-23-91 Jeff Blum

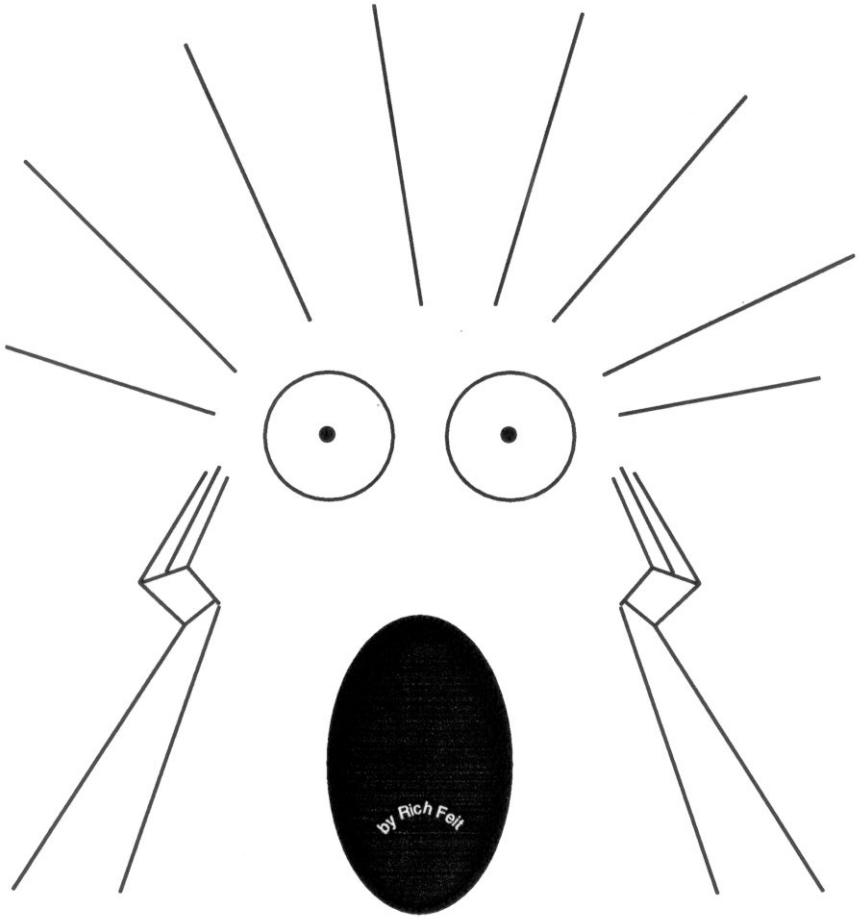
If nothing else, students should find this lab useful to their academic careers. If they take this lab seriously and actually try to walk away with a decent knowledge of the material, they should be able to produce very nice presentations using the NeXT computer. One of the goals in designing this lab was to keep it open-ended. Hence, the students are given their choice of many different applications to exercise their creativity; restrictions are at a minimum. On the other hand, this lack of restrictions may confuse some people. They will possibly need a prod in the right direction—a hint or suggestion at what they should try and do. Improvise.

There are two primary goals in this lab. The first is to learn about a primitive yet powerful text formatter, `troff`. Hopefully by showing that this program (a standard UNIX tool!) can outperform WriteNow when it comes to equations, they will get the point that any application can have its strong points. In the meantime, they will have learned a new skill that few people can claim: the ability to create a good-looking document on a computer without using a word processor. This section will also illustrate the two fundamental ways of working with text: formatting and WYSIWYG.

The second part will expand on the idea that each application has its strong points by showing how to focus those strong points on one final goal. Hopefully, whether they realize it or not, the students will glimpse the underlying structure of UNIX—many small parts that work together to accomplish larger goals. If they truly catch on, they should find themselves with a springboard for creating sophisticated documents which they can use in their other courses throughout college.

Whew. You call this a lab of lofty goals? Well, if nothing else, students should become interested by the realization that they will be able to outperform their computer illiterate counterparts.

Lab 5 - Sound



fortune: You will be surprised by a loud noise.

When a computer records a sound, it merely samples what is going on at the microphone and stores the information in a file — several thousand times per second. Once the information is on the computer, you can edit it just as you edit other computer files. Read on for a taste of the digital recording and editing of sound files...

Purpose

- To learn to record and play sounds from a Terminal window.
- To understand what's in a sound file.
- To learn how to manipulate soundfiles using SoundEditor.

Materials You Need

- **Earphones!**
- A cassette tape, if you want to have a recording of your sounds.

References

Required CheatSheets

- Preferences
- Sound
- UNIX
- SoundEditor

Helpful CheatSheets

- Storage
- rt.app
- SoundWorks

Online Help

- **Help** window (in the **Info** menu) for rt.app.
- **Help** window for SoundWorks.

Procedure

Setup

- Log in to a NeXT.
- Plug in your earphones in the back right of the monitor, under the picture of headphones. You might have to walk around to the back and look for the jack.
- Open up a Terminal window.
- Use the Preferences application to set the volume, and turn the mute on so the sound only comes through the headphones. Also, make sure you are in **UNIX Expert Mode** so you can view hidden files and directories, such as **/tmp**.
- Finally, open up a WriteNow document and save it as **sndstuff.wn**. You'll use this to **Paste** in text to hand in. **Save it every once in a while** (this is always a good practice)

Playing sounds, recording your own

In general, sound files are HUGE. You'll probably run out of space if you try to store too many of them in your home directory—remember that you have a disk quota. *But... every UNIX machine has its own built-in disk drive called* **/tmp** *with lots of space available. In this lab, you'll save everything you do into* **/tmp**, *and copy the important things to your home directory when you're done. (Find out how to use the* `cp` *command in the UNIX CheatSheet.)*

- `cd` to **/tmp**. If you want, you can use the `mkdir` command to make your own subdirectory in which to store your sounds.
- First you need a way to play sounds. On the NeXT, there is a UNIX command called `sndplay`. To test it out, type `sndplay ~cs111/Labs/07-Sound/hello.snd` — you should hear a short intro we prepared for you.
- Now you get to record your own sounds. The easiest way is to use the command `sndrecord`. Try it— type `sndrecord <filename>`, and follow the instructions that appear. If you don't end your filename with ".snd", it will be added for you (every NeXT sound must end in ".snd"). After you're done, the computer will report the amount of information it recorded.
- Play your sound, and don't forget the ".snd" at the end! If you don't like it, remove it with `rm` and give it another try.
- When you get a sound you like, rename it **snd1.snd** and save it into your home directory so you can submit it later. You can use the `mv` (move) command to accomplish this (`mv <old filename> <new filename>`).

Getting information about your sounds

- You can get some interesting information about a sound using the `sndinfo` command. Type `sndinfo <filename>`, and five lines of information will pop up, like so:

```
Filename: boxcars.snd
Size: 22784 bytes, 22784 samples, 2.844 seconds
Format: 8 bit muLaw
SamplingRate: 8012.821 Hz
Channels: 1
```

The Size (number of bytes) can be thought of as the number of characters the computer stored on disk when it recorded your sound. Similarly, the number of samples is the number of times the computer saved information from the microphone.

The Format shows the method the computer used to store the sound information. In this "muLaw" format, the computer stored each sample as an 8-bit number on a logarithmic scale.

The number of Channels tells you whether the sound is stereo (two channels, with different sounds for each ear) or monophonic (one channel, with the same

sound played in each ear). Not surprisingly, stereo files take up twice as much space as mono files do.

Finally, the SamplingRate is the number of times the computer saved information each second. Multiply the sampling rate by the number of seconds and  *you get the total number of samples.*

Think about this for a moment: in the three seconds that the computer recorded information, it stored over twenty-two thousand characters. That's ten pages of text! Furthermore, this is the lowest quality sound on the NeXT.

Recording CD-quality sound requires almost six times that much storage, because the computer records information almost six times as fast. To record one minute of CD-quality stereo sound would require over two hundred pages of information. (ooh, aah!) Check out the Storage CheatSheet for more insight.

- Get information on your **snd1.snd** with `sndinfo`, and use Copy&Paste to transfer the output into **sndstuff.wn**. Also give a rough estimate of how many pages (at about 2200 characters per page) your sound would fill.

Playing with sounds in SoundEditor

- Launch the **SoundEditor** application, and open the sound you just created. (Use **Open...** from the **File** menu.)
- Reverse the sound to hear yourself played backwards.
- Reverse the sound again, back to its original state.
- Your assignment is to hack up this sound until it is unrecognizable. (For starters, though, you might want to get rid of any silence at the beginning or end.) Deleting the whole thing doesn't count. :-) Look for tips in the SoundEditor CheatSheet.
- If you don't like what you have done, you can close the sound without saving it, and open up **snd1.snd** again. When you're done fiddling around, and you have a sound that you like, choose **Save As...** (not **Save)** from the **File** menu, and call your sound **snd2.snd**.
- Close up the editing window.
- That's all you have to do. But wait! Hopefully you have time left... find something fun to do from the **Above and Beyond** section, and after you do everything there :-) , find something in **For Further Exploration**. Enjoy!

What should be handed in

- **sndstuff.wn**, including the output from the `sndinfo` command and possibly short descriptions of your exploits in **Above and Beyond**.
- **snd1.snd** and **snd2.snd**.
- Any sounds you create in the **Above and Beyond** section.

Above and Beyond

Try anything you have time for, in any order.

Recording to and from the tape deck

To use the tape deck, you need to set up shop on the machine called ***page*** *(the first NeXT in the third row).*

...from the computer to a tape

- Press RECORD on the tape deck.
- Press the PLAY button to start recording, and play sounds on the computer as you normally would. The tape deck will record everything.

...from a tape to the computer

- Get to a Terminal window, and `cd` to the **/tmp** directory, so you have lots of space in which to record your sounds.
- Type `sndrecord -d <filename>`. The -d option tells the computer to record from the DSP port (see the Sound CheatSheet), where the computer gets information from the cassette deck.
- Follow the instructions for `sndrecord` as you normally would, except instead of speaking into the microphone, press PLAY (▷) on the tape deck.
- Just a note of caution: sounds recorded from the tape deck are saved in the highest-quality format with stereo. Use `sndinfo` to see how BIG they are. If you don't need your sound to be stereo, type `sndconvert -c 1 <filename>`. The `-c 1` tells the computer to convert the sound to one channel (monophonic). Saving the sound in monophonic format reduces your chances of running out of room on the disk.

/LocalLibrary/Sounds

- Check out this directory! In **/LocalLibrary/Sounds** there are over 300 sounds available for your listening pleasure.
- If you find a sound you like, you can save a copy into **/tmp** or your home directory and play around with it in SoundEditor, rt.app, or SoundWorks.

Mixing Sounds with rt.app

Whether you're creating your own symphonic masterpiece or you just want to add your own voice to your favorite song, you will want to learn how to mix sounds. rt.app is the most powerful and fun mixing application around.

- You can record your own sounds to use, or you can use sounds from **/LocalLibrary/Sounds**. If you choose to record your own, use `sndrecord` and then follow the instructions below.

  When you use `sndrecord` *with the microphone, your sounds are recorded in a format called "muLaw." Since rt.app requires that your files be stored in a*

different ("linear") format, we wrote a short program called `mu2linear` *which will convert them for you.*

  - For any sounds you recorded yourself, type `mu2linear <old filename> <new filename>`. The file you named `<new filename>` will be saved in a format acceptable to rt.app.

- Read the rt.app CheatSheet, and make some wild sounds with `playnote` commands. If something doesn't work, go back to the general procedure in the rt.app CheatSheet, or look in rt.app's **Help** screen.
- Save any interesting sounds with **Write mix to disk...** in the **Document** menu.

Mixing and Altering Sounds with SoundWorks

Although SoundWorks is not as powerful as rt.app, it has a fantastic interface for simply mixing two sounds together and for playing with glissandos (sliding up or down the musical scale). To use this program, however, you must be working on the NeXT machine called ***page****.*

- Read about mixing and the Effects Panel in the SoundWorks CheatSheet.
- Save some mixed sounds, and save some sounds which you changed with the Effects Panel.

Hidden Messages

Recording artists have always had fun putting hidden messages in their work. The Beatles did it when they went on a "Paul is dead" kick. Here's an example from Queen's "Another One Bites the Dust." Make your own judgement on this one.

- In SoundEditor, open up **~cs111/Labs/5-Sound/dust.snd**. Listen to it.
- **Reverse** it, and try to find the message (it's hard to understand).
- For an clearer, eerier example, get a copy of the Beatles' *The White Album* and reverse a chunk of "Revolution Number Nine." Turn me on, dead man.

Setting up a startup tune

Everyone with a UNIX account has a file which runs a set of commands for every new Terminal window. It is called ***.login****. Notice the "." in front... that just means it doesn't show up (for convenience) when you type* `ls` *to list the files in your home directory. To include "." files in a directory listing, type* `ls -a`*.*

- To add a sound to your startup routine, find **.login** in your home directory and open it up in Edit (see the Edit CheatSheet).
- Add in a `sndplay` statement at the end, and save the file. Now, whenever you open a new Terminal window, this sound will play.
- Take it out when it gets annoying.

For Further Exploration

- Check out **Sing** in **/Net/dobro/musr/Apps/SPASM1.06**. It... sings. Hint: find the switch that toggles between **Quiet** and **Sing**, and switch it to **Sing**.
- Record some good-quality music from the tape deck, and use rt.app to create a professional-sounding mix.

Lab #5 — TA Sheet

8-30-91 Rich Feit

Before the Lab

- Reboot each NeXT machine to clear **/tmp**. Soundfiles take up LOTS of space.
- Bring extra earphones.

Things to Watch for

I think digital sound is fun by nature. To get the full benefit, though, the students should understand concepts like samples, sampling rate, and channels. Hopefully the **Getting Information...** section won't be too confusing, but be prepared to answer any questions. You can always point them to the Sound CheatSheet for more information.

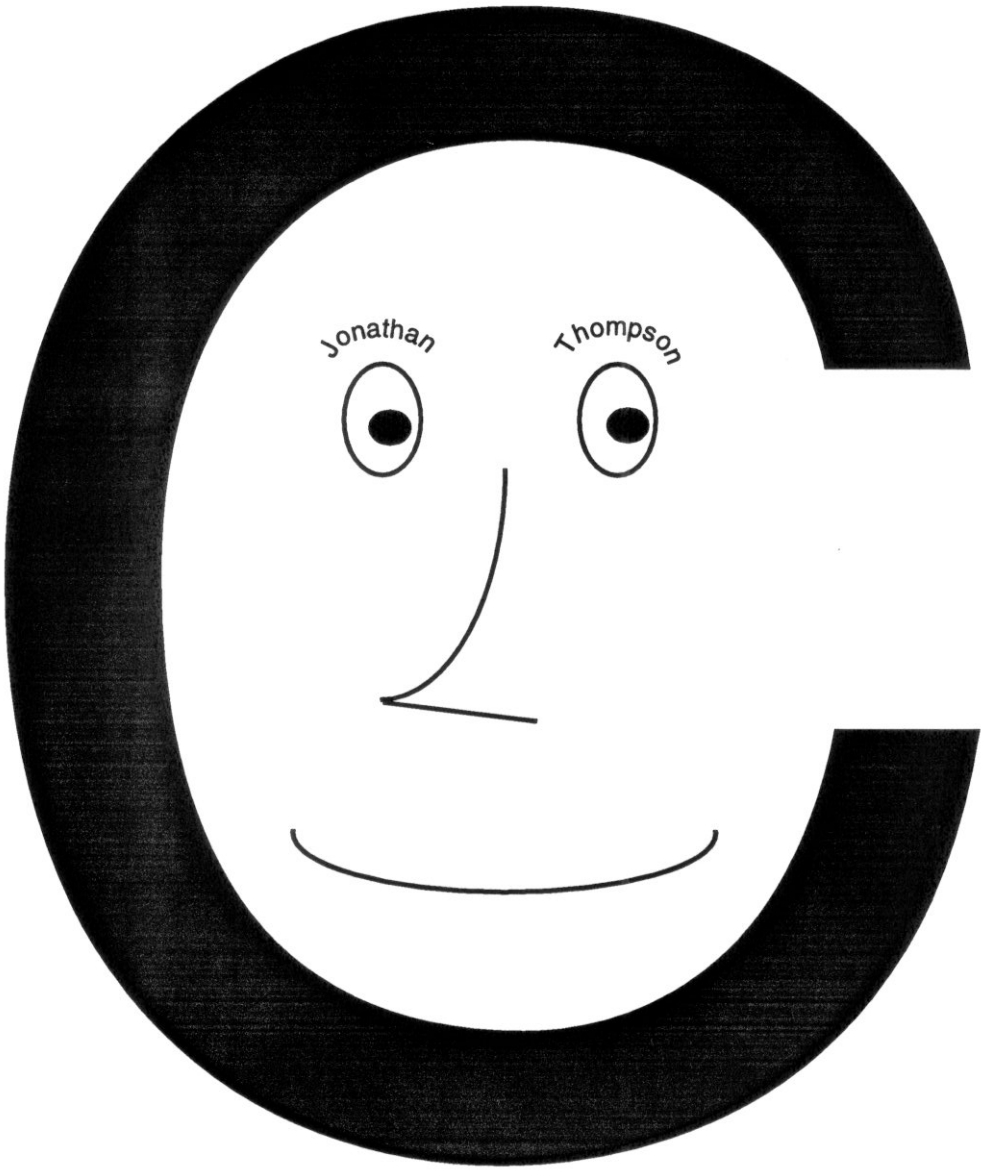
In SoundEditor and SoundWorks, a good analogy for sound editing is text editing. I tried to make that clear, but... you might have to demonstrate a Copy&Paste or two.

After that, the only thing everyone might need help with is rt.app. Familiarize yourself with it, and look for errors in **playnote** commands (such as missing parenthesis). Kent Dickey '92 and Professor Paul Lansky (music dept.) wrote rt, and they could probably help if you find something weird.

I originally wrote this lab for use with SoundWorks, but the licensing fell through and I discovered some really stupid bugs in the application anyway. If a bug-free version ever comes out, SoundWorks might be a good thing to incorporate into the **Procedure**.

Also, Professor Lansky told me about a new version of **edsnd** (**/Net/dobro/musr/Apps/edsndP**) which seems to do much of what SoundWorks was supposed to do. More importantly, it's stable, and it has a **Help** screen. You might want to check it out.

LAB 6 - PROGRAMMING IN ...



C: n. 1. The third letter of the Latin alphabet. 2. ASCII #b1000011. 3. The name of a programming language designed by Dennis Ritchie during the early 1970s and immediately used to re-implement {UNIX}. So called because many features derived from an earlier interpreter named 'B' in commemorations of *its* parent, BCPL; before Bjarne Stroustrup settled the question by designing C++, there was a humorous debate over whether C's successor should be named 'D' or 'P'. C became immensely popular outside Bell Labs after about 1980 and is now the dominant language in systems and microcomputer applications programming. See also {languages of choice}, {indent style}.

C is often described, with a mixture of fondness and disdain varying according to the speaker, as "a language which combines all the elegance and power of assembly language with the readability and maintainability of assembly language".

– Hacker's Jargon File

The time will often arise when there is a specific task that you want the computer to do, but there is no software to do the job. If you are lucky, you can afford to have a programmer do the work for you, but if you are not you will either have to get along without it or figure out how to solve the problem yourself. Since nobody can do without, a little programming experience could be very useful.

Purpose

- To become at the absolute least passingly familiar with C.
- To learn how NeXT programmers make such uniform user interfaces.

References

Sample Programs

- A sample of all three programs will be available in the **~cs111/Labs/6–CProgramming** directory. These will be in executable form only, so you can see how the output looks. After you have turned in your work, the TA's will give you a copy of the source code for the two C programs.

CheatSheets

- C - The Programmer's Language of choice - 1 (It is VERY IMPORTANT to read this!)

Procedure

Preparation

- Open a WriteNow document and save it as **cprog1.wn**; you will paste your Terminal window sessions into this file. Make sure that you save it every once in a while.

Program 1

The first program might seem kind of silly, but it actually requires a fair amount of knowledge such as where to type the program in, what to call it, what the basic structure should be, and how to compile it and run it. The program has one goal: print the words "hello, world" on the screen.

- Open a window in Edit.
- Type the following program in the Edit window:

```
#include <stdio.h>              /*include standard library info*/

main()                           /*define a function named main*/
{
  printf("hello, world\n");   /*call the library function printf.
                                 \n represents a newline*/
}
```

Notice that the comments are short and to the point; this is typical style.

- Choose **Save As...** from the **File** menu and move to your own directory (if you are not already there). Name the program **world.c**.
- Now open up a Terminal window
- Make sure you are in the same directory as your program and type `lcc world.c`.
- After the compiler finishes and you get the prompt back, type `a.out`. Assuming there were no errors, the computer should print the words "hello, world" on the screen. Congratulations! You just successfully compiled and ran your first C program.
- Copy everything that you did in the Terminal window into your WriteNow document and save it.

Program 2

For this next program, instead of giving the code for you to type in, we will only give you some specifications on how to make the program. The program is just slightly more involved than the first; it will print out your name on the screen in a mesmerizing pattern. To program this, you should employ a modular design scheme, meaning create the program one step at time, making sure each part works before going on to the next stage:

- Open up a new Edit window.
- Type in the appropriate `#include` statement (we only need stdio.h) at the top.
- Create the function `main()` which contains a command that will print out your name.
- Save the program to a file **printname1.c** and compile and run it to make sure it works.
- Now, insert a definition for the constant `NUM` equal to 500 (or whatever), which you will use in a `for`-loop.
- Declare an integer variable named ??? (pick a name), which you will use as the counter in the loop.
- Make a `for`-loop from 1 to `NUM` around the `printf` statement.
- Save the updated program as **printname2.c** and see if this version works.
- Now for the final part, just remove the `\n` from the `printf` line and replace it with 2 or 3 spaces.
- Save as **printname3.c**, compile and run.
- Again, copy your Terminal window session into **cprog1.wn** and save it. For brownie points, you can print out more or fewer lines by changing the value of the constant `NUM` and recompiling the program. Feel free to add any embellishments you want such as printing out the value of `NUM`, or adding a final

newline after the program is done with the for-loop. After these additional changes, make sure that you save your updated version.

Program 3

Have you ever noticed how all applications that run on the NeXT have a similar appearance? They all have a menu with certain items such as ***Info***, ***Edit*** *and* ***Quit****; they all have windows with scroll bars, etc. To create such uniformity, most NeXT-specific programs are created using an application called Interface Builder. This application allows people to create very uniform front ends for their own programs, as well as manipulate code in an easy manner. For the final part of this lab, we are going to create our own simple yet exciting NeXT application.*

- Run **InterfaceBuilder** in **/NextApps**.
- From the **File** menu, choose **New Application**.
- After a bit of whirring around, a blank application window will show up on the screen entitled "My Window."
- Reshape your window to an attractive size.
- From the **Palettes** window on the right of the screen, find the object called "Button" and drag it over into your window.
- Enlarge or shrink the button to an attractive size.
- Double-click the word "Button" to change the name to "MySound".
- After modifying the window and button to your liking, go down to the box in the lower left of the screen. Click the **sounds** icon and a window will pop up with many different sound files. Pick a sound file that you like and drag it onto the button in your window.
- Choose **Save As...** from the **File** menu and name your program **button** in your home directory.
- From the **File** menu, choose **Project**. A box will open up on the lower right of the screen, asking you what type of program you are making. We are trying to make an **Application**, which just so happens to be the default setting, so click the **OK** button.
- One last time, go to the **File** menu. Choose **Make** from the list. Interface Builder will now compile your application and make it executable.
- After it is finished compiling, quit or hide Interface Builder, go to your home directory and double-click the program that you just created (Interface Builder named it **button.debug**). A window identical to the one that you were just working on should pop up. Now, click the button and voilà! The sound you chose will play!

You are probably wondering what this third 'program' is doing in a lab about C. Well, the fact is that everything you just did was is C, or rather an expanded version of C. The particular version of C that the NeXT uses in the Interface Builder is called Objective C, and contains as a subset of itself everything that you have learned so far about C. But it also offers a lot more tools that allow you

do some pretty fancy stuff. (Just look at the majority of NeXT applications!) This new type of C, including both Objective C and another called C++, is probably the way the future of programming will go.

What should be handed in

Using `handin`, turn in the following files (working or not): **world.c**, **printname1.c**, **printname2.c**, **printname3.c**, **button.debug**, and **cprog1.wn**. Remember, comments in your program are extremely beneficial; you should probably have at least a few of them in your programs (hint, hint). Also, any embellishment is good, but make sure you comment it. Good luck!

Above and Beyond

It's great having the computer talk to you, but it's even better if you can talk back to it. C has several built in functions that allow you to get input from the keyboard, but they aren't particularly flexible. However, a little is better than none at all!

The function that we will use is called `scanf`. It works a lot like `printf`. Let me show you:

```
#include <stdio.h>                          /* include standard library */

int i;                                       /* variable to hold input */

main()
{
  printf("Enter a number of your choice: ");
  scanf("%d",&i);                                /* %d, same as printf */
  printf("You entered the number %d.\n",i);
}
```

When you run this (simple) program, it prints out the first line, then pauses, waiting for you to enter a number. After you type in any number and press the <return> key, it will print it back out to show you that it got it correctly. Notice that in both `scanf` and `printf`, we had to use a `%d` to indicate that the variable `i` is an integer. Also note that in `scanf`, we put an ampersand (&) in front of the variable name.

Open up the file **printname3.c** that you created in part two of this lab. Modify it so that the user can tell the computer how many times to print out the name. To do this, we need to:

- Take out the `#define` statement and replace it with an integer variable.
- Use a `printf` at the top of `main` to let the user know what we are doing and ask for the number of times to repeat.
- Use a `scanf` to get the number that the user types in.
- Change the `for`-loop so that it loops from 1 to the inputted value.

Save this new file as **printname4.c**, and the corresponding Terminal session as

a&b.wn and `handin` the two files.

For Further Exploration

The Bible of C programming, Kernighan and Ritchie's The C programming Language, is an excellent reference source for the language. Since the book can sometimes be a bit dense, especially for those without too much programming experience, another book such as the The Waite Group's The C Programming Primer can also prove to be very useful.

Lab #6—TA Sheet

August 23, 1991 - Jonathan Thompson

This lab is most likely the first time that any of these students has ever programmed in C. We need to be very careful that we ease into it gently.

It was mentioned in both the lab and in class that it is very important that students read the first C programming cheatsheet and attend both lecture and precept. Certain key ideas were presented that might seem very confusing to the uninitiated. These include, for example, a discussion about what a library or variable is.

The most important thing to remember about this lab is to prevent the students from getting frustrated! If students are worried about completing items on time, assure them that there is no rush. It is a thousand times better for the student to really understand what is going on if it takes him a bit longer to 'get the hang of things' than to rush through to meet the deadline.

If any students come up with a 'bug' that they want help with, try to have them find it for themselves. The best thing you can have them do is run through the thought processes that went into making the program. Sometimes, speaking these out loud will illuminate glaring gaps in their logic process. If the bugs are a bit more insidious, you might want to be a bit more explicit in your help.

The program listing for **printname3.c** and **printname4.c** should be handed out to the students after they have turned in the lab. Note that **world.c** is already in their manuals.

Make sure that permissions are set so that the students will be able to run **hello**, **printname3**, **printname4** and **button**.

Lab 7 - Selection Sort

bogo-sort: n. (var. 'stupid-sort') The archetypical perversely awful algorithm (as opposed to {bubble sort}, which is merely the generic *bad* algorithm). Bogo-sort is equivalent to throwing a deck of cards in the air, picking them up, then testing whether they are in order. If not, repeat. Used as a sort of canonical example of awfulness. Usage: when one is looking at a program and sees a dumb algorithm, one might say "Oh, I see, this program uses bogo-sort." Compare {bogus}, {brute force}.

– Hacker Jargon File

It is often necessary for computer programs to sort large amounts of data. By sorting, we mean placing the data (i.e. numbers, people's last names, etc.) into some type of order. Many different sorting algorithms (strategies for solving a problem) have been created with varying performance characteristics. We are going to build our own sorting program called "selection sort," which runs in time proportional to the number of elements squared. It isn't the fastest, but it is conceptually clearer than a lot of the other sorting algorithms.

Purpose

- To learn how a simple sorting algorithm operates.
- To learn and use arrays and functions in C.

References

Sample Programs

- Sample output from the selection sort program will be available in **~cs111/Labs/7–Selection**. After the lab is handed in, the program listing will be handed out.

CheatSheets

- C - The Programmer's Language of Choice - 2 (It is VERY IMPORTANT to read this cheatsheet before coming to lab.)

Procedure

This sort uses the following idea: we look through an entire array of numbers, find the smallest value and put this in the first location. We then find the second smallest and put this in location two. If we continue this, we will eventually have a completely sorted array. In a little more detail, the algorithm is:

1) Begin at the first element in the array.
2) Search through the entire array looking for the smallest item.
3) Exchange the smallest value found with the value at the first location.
4) Move forward one element in the array, effectively making the array *one element shorter.*
5) Search for the new smallest item.
6) Switch it with the first array element in the *shorter* array (which is the *second* in the full array).
7) Continue in this same manner until the entire array is in order.

A couple of examples might help to clear this up. For the first example, suppose that we have 4 shoeboxes, each with a random card placed face down in it. To implement our algorithm most effectively, we want to keep track of three things: the smallest card found so far, the location of the smallest card found so far, and the

place to put the smallest card.

We pick up the card in the first shoebox and look at it. It is the 8 of hearts. We remember the card as the smallest element that we have found so far and the first box as its location. We put the card face down back into its box and pick up the card in box 2, a queen of hearts. This is greater than the first card that we found, so we don't change anything. We put the card back face down and go on to the third box. There, we find a 2 of hearts. This is smaller than our previous smallest card, so we remember this card and its location instead. Finally, we move to box 4, and pick up a 10 of hearts. This is larger than the 2 of hearts we found, so we place it back into its box face down. We now know the smallest card in all four shoeboxes (2 of hearts) as well as its location (box 3), so we move the card from box 3 to box 1 and the card that was in box 1 into box 3. Since the first box is sorted we move it a little to the side so that it won't distract us.

We continue, following the above procedure by turning over the card in the first box (originally the second). We find a queen of hearts and mark this down as the smallest card found so far and box 2 as its location and move on. In box three, we turn over an 8 of hearts. Since this is smaller than the queen, we change our smallest card to the 8 and the smallest card's location to box 3. After we replace the card facedown in its box, we turn over the card in box 4, a 10 of hearts. This is larger than the 8, so we ignore it. We have come to the end of the boxes, so it is time to switch again. The smallest card is the 8 of hearts in box 3 and we want to change it with the first box in our array, box 2. After exchanging these cards, we continue with our sort, remembering to put the second box off to the side with the first.

Turning over the card in box three, we find (much to our surprise) a queen of hearts. This is so far the smallest card and its location is box 3. Replace the card facedown and pick up the card in box 4, the ten of hearts. This card is the new smallest and its location is box 4. Again, we have finished all of the boxes, so we now exchange the card in the smallest card location with the card in the first box of our small array (box 3). After changing the cards we move box 3 over to the side with the others.

Continuing on, we pick up the card in box 4 and discover that it is ... the queen of hearts! This becomes the smallest card so far and its location is box 4. Since this is the last box, we switch the card in box 4 with the card in the first box of our array, which is also box 4. Obviously this doesn't change anything, so we could have stopped after the last step. (The only reason that I include this step here is because a computer wouldn't think of this as 'obvious'. You need to be specific when you want the computer to do something, and even more specific when you want it to stop.)

Grouping the boxes together again and flipping them over one at a time reveals that the cards are: 2 of hearts, 8 of hearts, 10 of hearts, and queen of hearts, all sorted.

For the second example, we'll use less of an analogy and instead try to show the algorithm a little more like the computer would sort the numbers .

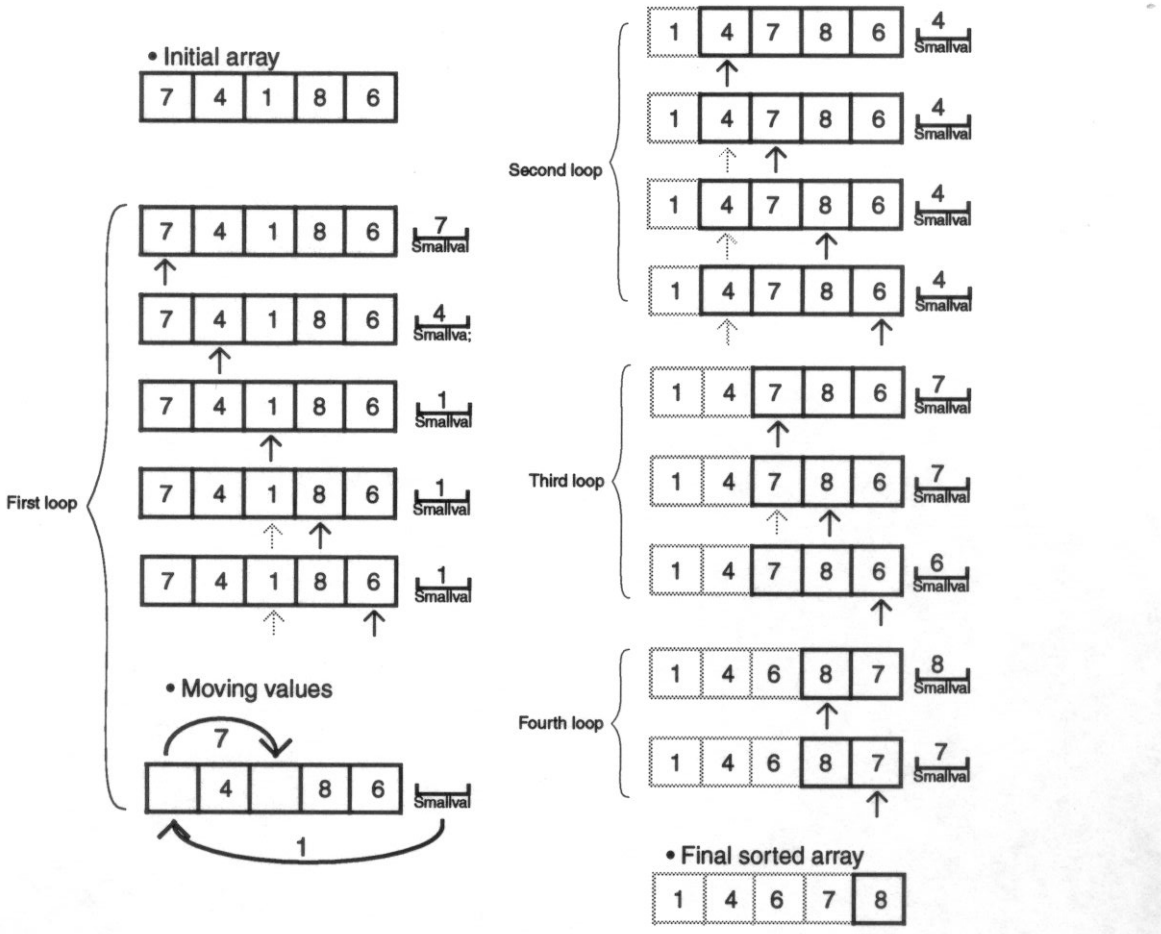
Suppose that we have a 5-element array comprised of the following numbers: 7, 4, 1, 8, 6, and the variables `smallval` (the smallest value found so far), `smalloc`

(the location of the smallest value found so far), and `toput` (the location the smallest value found will be swapped into).

- Start at the first location with the element 7 (this is the smallest value so far, so `smallval` = 7, `smalloc` = 1, and `toput` = 1). Start searching through the array.
- At location two, we find the element 4 (which is smaller than 7) so we change `smallval` to 4 and `smalloc` to 2 (`toput` stays the same).
- At location three we find the number 1 so we change `smallval` to 1 and `smalloc` to 3.
- At location four the number (8) is larger than `smallval` so we continue.
- At location five, the number (6) is also larger than `smallval`, so there is no change.

Now knowing the smallest value and its location, we just need to exchange it with the element in position one:

- Move the value in location `toput` (the number 7) to the location `smalloc` (location 3).
- Place the value of `smallval` (1) into the location `toput`.



We have just completed the first loop (the left-hand side of the diagram) of the sort. We now want to repeat the above process, but without worrying about the

first location because the smallest value is already in place. Therefore, we just increase the value of `toput` to 2 and sort through the shorter array from 2 to 5. And then 3 to 5, and on and on. The easiest way to make the computer do this is to have two loops: one (the inner loop) controls the array that we are searching through and the other (the outer loop) tells the computer what the inner loop's starting value should be. Note that the outer loop happens to control the variable `toput`, too. It is the fact that we are using this two-loop method that makes the program run in time proportional to the number of elements squared.

The finished program will need to do the following:

1) Print out a short description telling the user what the program is going to do.
2) Create the numbers to sort (use a random number generator), putting them in an array of size given in a `#define` statement.
3) Print the numbers out so we know what they are.
4) Sort the numbers in the array.
5) Print the sorted numbers out so we know that it worked.

(It might be a good idea to use one or two functions, especially for steps that get repeated (such as 3 and 5). Just do what you think is best and make sure to put in lots of comments.) The modular method used in the last lab is probably the best way to write this program:

- Create a program that will tell the user what the program will do. Compile and run.
- Next, have the program create the array of random numbers and print it out. Again, check to make sure it works.
- Then, locate the smallest element in the array, move it to the first spot, and print the value out. You should know what to do by now. :-)
- Finally, have it sort the entire array and print out the results.

After you have the entire program finished:

- For the first version of your program, have it sort 10 elements.
- When you have finished writing your program, save it to a file named **select.c**.
- Open a Terminal window and compile and run your program.
- Change your program so that it will sort 50 numbers. Save it, compile it, and run it.
- **Copy** your entire Terminal session, **Paste** it into a WriteNow document and save it as **cprog2.wn**.

What should be handed in

Use the `handin` program to turn in both **select.c** (working or not) and **cprog2.wn**. Don't worry if you didn't get the hang of everything. We realize that this is a difficult assignment, especially since last week was most likely your first experience with C programming.

Above and Beyond

There is a Unix command called `time` that will act as a stopwatch and allow you to see how long the execution of your program took. After you have your working selection sort program, change the size of your array to 100. Compile for this value and then at the prompt type:

```
time a.out
```

This will run your program normally, and at the end give you a brief description of the amount of time it took to run:

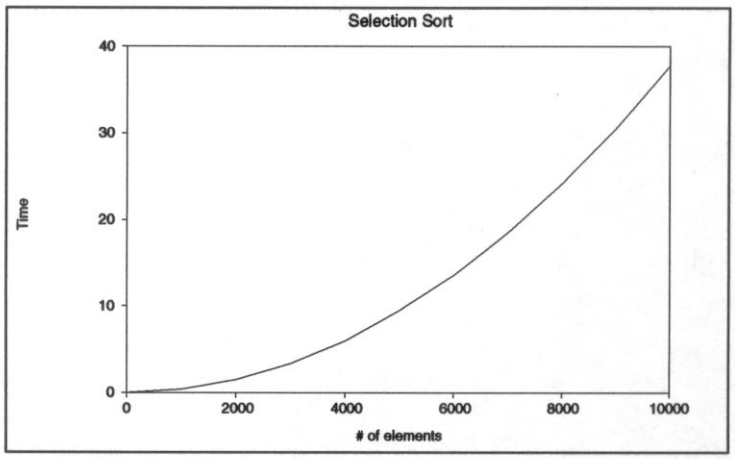
```
0.030u 0.154s 0:00.43 41.8% 0+0k 0+0io 0pf+0w
```

The number that we are interested in is the first one (the one which is underlined) which tells us how long it took your program to run in user time (in this case, .03 seconds). Now, go back to your program and change the number of elements in the array to 500 (this should be easy if you have a `#define` statement at the top of your program!) and recompile and type `time a.out` again. This is great, except for the fact that the program has to print out to the screen which can end up taking a lot of time, especially for large arrays. To fix this, tell the program that you want all of the screen output to go to the great VOID by typing:

```
time a.out > /dev/null
```

Now, you will just get the time that it takes for the program to sort the numbers and you won't have to worry about the numbers flashing by your screen.

Open up an Edit window, type in the size of the array (in this case 500) and then how long it took to run. Go back into your program and increase the size of the array by 500, compile and time the program with output going to **/dev/null**. Again, record the appropriate numbers in the Edit window. Repeat this until you get a decent sized table (about 10 times). Save this table to a file and then go to **/cs111/Apps** and run **nxyplot**. Select Open from the File menu and enter the name of the file you just saved. The program will automatically plot all of the points for you, and you should see something that resembles a quadratic curve!



We can therefore conclude that the selection sort takes time proportional to N^2 to

sort N numbers. From the **File** menu, choose **Save to EPS**, name your file **nxy** (it will be named **nxy.eps** by nxyplot) and turn it in using `handin`.

For Further Exploration

- A book on the C programming language such as the infamous The C Programming Language by Kernighan and Ritchie.
- An excellent book by Princeton's own Robert Sedgewick, Algorithms in C. This book (among other things) offers several good chapters on different sorting techniques.
- An application that can help you to visualize different sorting algorithms (including selection sort!) is **SortingInAction**, located in **~cs111/Labs/7–Selection**. This application will graphically show you what goes on during a selection sort. Try watching it with the **Animate Compares** switch on; you should be able to follow what it is doing!

Lab #7—TA Sheet

August 23, 1991 - Jonathan Thompson

Now we are really getting into some dangerous waters. It might be a lot to expect students who have never programmed before last week to write a selection sort program. I guess we'll see...

The difficulty for this lab is probably going to be due to the lack of understanding how the selection sort algorithm works, or confusion about how functions and arrays work. The former is covered (hopefully adequately :-)) in the lab itself. The latter is something that both the C CheatSheet and the lectures should cover. However, if the need arises, don' t hesitate to explain either of them in your own words.

Debugging should be handled in the same manner as in Lab #6: let the students try to uncover the bugs themselves, perhaps by letting them explain to you what their program is trying to do. Since this program will be quite a bit more complex than the 'hello, world' program of lab 6, you might have a lot of people asking for help.

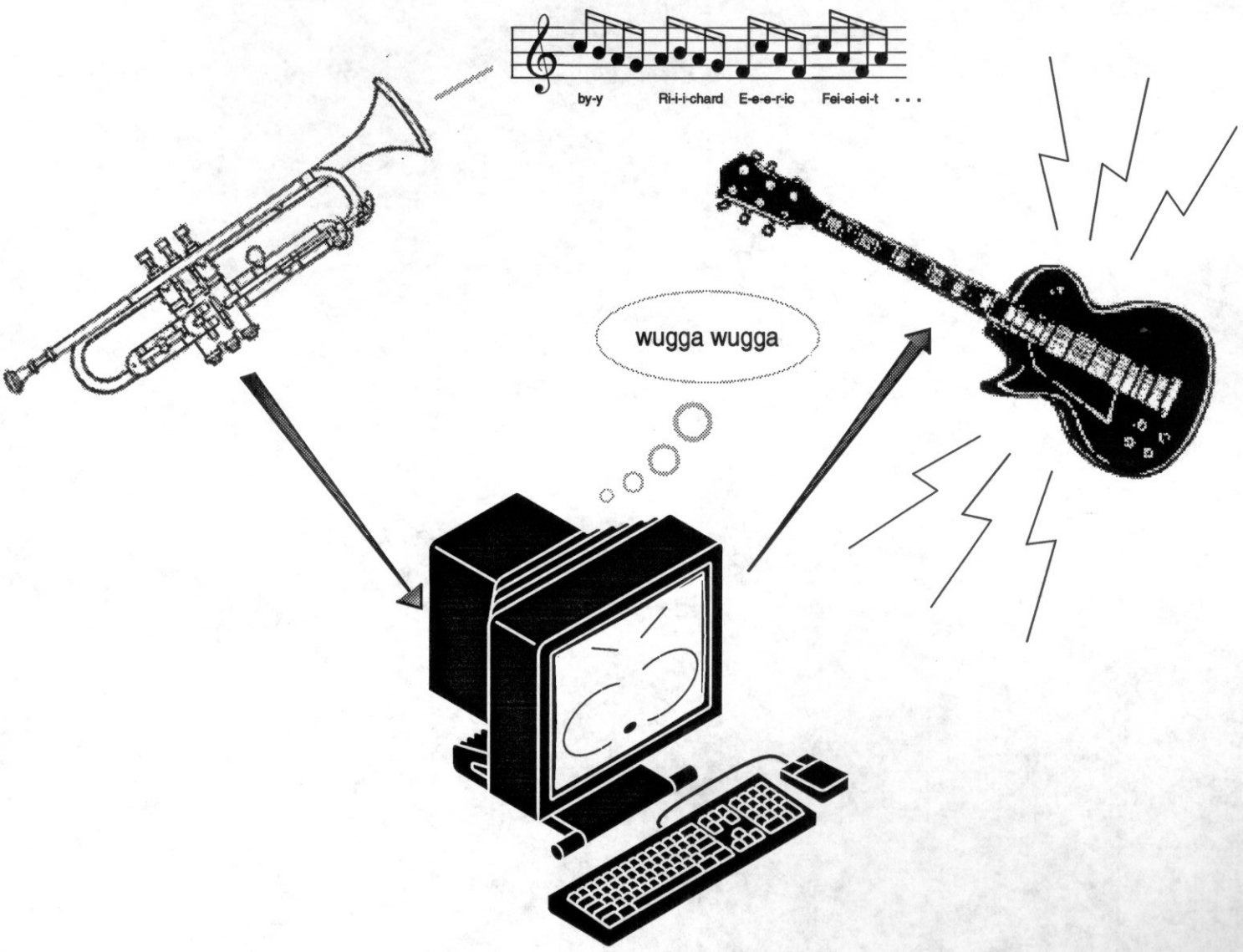
I suppose it is worth mentioning again that we are *not* trying to frustrate people. If the lab seems to be heading in that direction (i.e. everybody is having major difficulties), perhaps a better tactic than throwing in the towel is to go through step by step how you would make this program. It's very possible that they just might not have gotten enough exposure to programming.

As far as preparation goes, you should print out enough copies of the program **select.c** to hand everyone upon completion of their own program (presumably at the end of lab).

Make sure that permissions are set so that the students will be able to run **select** out of the directory **~cs111/Labs/7-Prog2**.

The file **selout** has times from our selection sort program. Open this file from within nxyplot and you will see a quadratic curve.

Lab 8 - ein



wugga wugga: /wuh'g* wuh'g*/ n. Imaginary sound that a computer program makes as it labors with a tedious or difficult task. Compare {cruncha cruncha cruncha}, {grind} (sense #4).

When the computer plays sounds, it sends numbers to the speaker thousands of times per second. Each of these numbers , or samples, *represents a desired position of the speaker cone at the particular time the sample arrives. With ein, you can write a simple (one or two line) C program that will do something to each sample before it gets passed to the speaker. For example, if your program tells the computer to double the value of each sample, the sound will come out twice as loud.*

Purpose

- To analyze soundfiles using ein.
- To "filter" soundfiles using ein.

Materials

- Earphones!
- A cassette tape, if you want to have a recording of your sounds.

References

CheatSheets

- Sound
- Grab

Online Help

- **Info** window for ein — look for some familiar names at the top!

Procedure

Setup

- Log into a NeXT.
- Plug your earphones into the back of the monitor.
- ein is in **/Net/dobro/musr/Apps**. Find it in the File Viewer and drag it into your icon dock.
- Pop open a Terminal window.
- Set the volume to a comfortable level, and make sure the speaker mute is on — you can use the volume keys or the Preferences application.
- `cd` to **/tmp** so you have room to record sounds.
- You need a few soundfiles to work with in ein. Use `sndrecord <filename>`. For the first one, whistle a few notes. If you can't whistle, get someone else to do it for you. For the second one, just speak a few words.
- Finally, open up a WriteNow document and save it as **einstuff.wn**. You'll use this to paste in pictures and text for submission. **Save it every once in a while.**

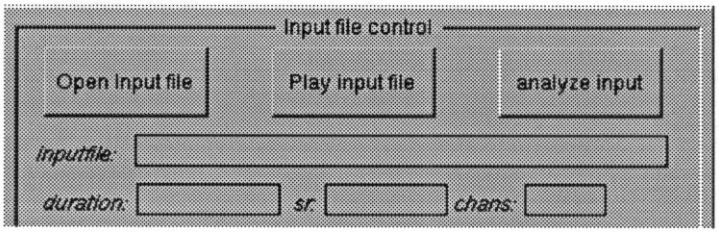
Converting your sound to a different format

When you use `sndrecord` *with the microphone, your sounds are recorded in a format called muLaw. Unfortunately, ein needs its soundfiles to be stored in a "linear" format. Since you probably won't need proficiency in converting sounds, we wrote a short program called* `mu2linear` *(get it? muLaw-to-linear?) which will convert your sounds for you. Use it on both of your test sounds.*

- Type `mu2linear <old filename> <new filename>`. Your sound is now ready for ein.

Analyzing sounds with ein

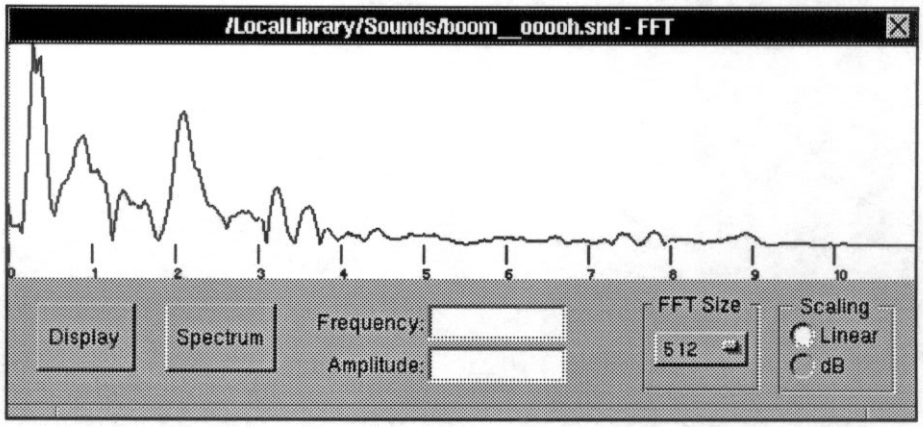
- Double-click the **ein** application to open it up.
- Select **New ein.script** from the **Window** menu.
- For the moment, ignore the editing section and concentrate on this part:



- Everything you need in order to analyze your sounds is here. Click the **Open Input File** button and find your sound (remember that it is in **/tmp**).
- Click the **analyze input** button. Three windows will pop up: SoundView, FFT, and Spectrum.

  The SoundView window should look familiar to you — it's a graph of the sound waveform. It's especially useful in this lab to think of the waveform as a graph of how far the speaker cone is pushed out over time — the top line shows how far the cone is pushed out, and the bottom line shows how far it is pulled in.

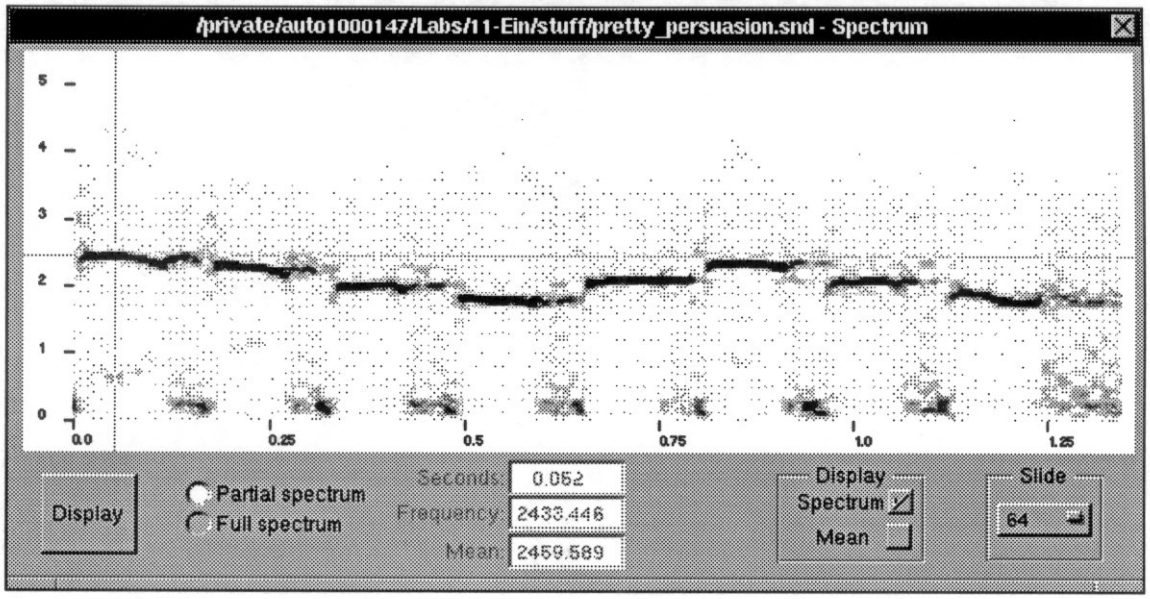
  The FFT tells you how much of each frequency appears in your sound. The frequency is measured across the bottom in thousands of hertz (cycles per second — see your Sound CheatSheet). For example, most of the frequencies in this sound fall below 4000 Hertz:



The higher levels at the left tell you that most of the sound's frequencies fall below 4000 Hertz.

The Spectrum window shows you how much of each frequency can be heard at each instant *in the sound. It's much more revealing than a single FFT of the entire sound. In fact, the Spectrum (technically a* spectrogram*) is really a series of successive FFT's.*

The vertical axis represents the frequency (measured in thousands of Hertz), and the horizontal represents time. The darker it is at a certain height on the graph, the more of that frequency there is. For example, here's some poorly-whistled R.E.M.:



You can follow the tune along the dark lines. Here is this Spectrum's musical counterpart:  .

- To find out the average frequency at any instant in your sound, click inside the Spectrum window. The number labelled Mean tells you the average frequency at the time labelled Seconds. For the example above, at 0.052 seconds the average frequency is 2433 hertz.

  Check out the Sound CheatSheet for more information.

- If your spectrum came out OK, you will want to save a copy to hand in. Go read the (very short & simple) instructions in the Grab CheatSheet for copying a snapshot of a window to the Pasteboard. **Copy** your Spectrum window, and **Paste** it into **einstuff.wn**.

- Now click **Open Input file**, and open and analyze the sound which contains your voice. Try to see how the Spectrum window corresponds to whatever you said in your sample sound. Can you "follow" your voice in the Spectrum window? Copy this Spectrum window into **einstuff.wn**.

- Which of the two windows — SoundView or Spectrum — gives you more useful information? Type your insights into **einstuff.wn**.

Tips for using ein

- Ein usually fills up the whole screen with its windows. When you want to do something in another application, select **Hide** from ein's menu. When you are ready to return to ein, double-click ein's icon.
- When you work with filtered sounds, you can always play the input sound with the **Play input file** button, and the output file with the **play** button at the bottom of the window.
- Also when you are filtering sounds, remember to change the number of seconds in the box marked **dur** to equal the number of seconds in the box marked **duration.**

Filtering sounds with ein

In order to apply the power of C to digital sound, ein adds a few predefined sound variables and one special statement. In fact, ein is nothing more than a C "preprocessor" which passes most of what you type directly on to the C compiler.

The predefined `y` *variable (an integer) corresponds directly to the position of the speaker cone. The greater the value of* `y` *(above 0), the farther out the speaker cone is pushed, and the smaller the value of* `y` *(below 0), the farther in it is pulled.*

This is all that happens when you use ein: *ein looks at each sample of an input sound, one at a time. From each sample, it gets a value of* `y`*, changes it in some way, and saves the new value of* `y` *into an output sound. Hence the idea of "filtering."*

Important: before you filter any input sound, change the number of seconds in the box marked ***dur*** *to equal the number of seconds in the box marked* ***duration****. If you don't, your output sound will probably be cut off at the end. Also, make sure that the radio button marked* ***use input file*** *is selected.*

- Let's start with something that does nothing. Type

```
y = y * 1;
```

  in the editing window. Click the **compile / run** button, and wait for the computer to play the sound through your "filter." Be patient — remember that the computer has to run this script on each of the thousands of samples in your sound...

  OK, so nothing exciting happened. The computer just took each `y` value from the input sound and multiplied it by 1... leaving it in exactly the same condition as it was in before. The point? To understand the idea of "filtering" each sample of the sound.

- Change the script to

```
y = 1;
```

  and click **compile / run**. What was the click you heard? This is still technically

a "filter," but this time the computer ignored the `y` values that it got from the input sound. But what was that click? Put your thoughts on this into **einstuff.wn**.

- Now change the first script to have ein square the `y` value of each sample. For comparison, after you **compile / run** your filter, you can play the original file with the Play input file button, and the filtered output file with the play button at the bottom of the window. In **einstuff.wn**, type in some ideas about what this filter does. Hint: think about squaring a loud (large-valued) sample versus squaring a soft (small-valued) sample. Which of the two would be accentuated? Use **Save As...** to save whatever you came up with into **square.ein**.

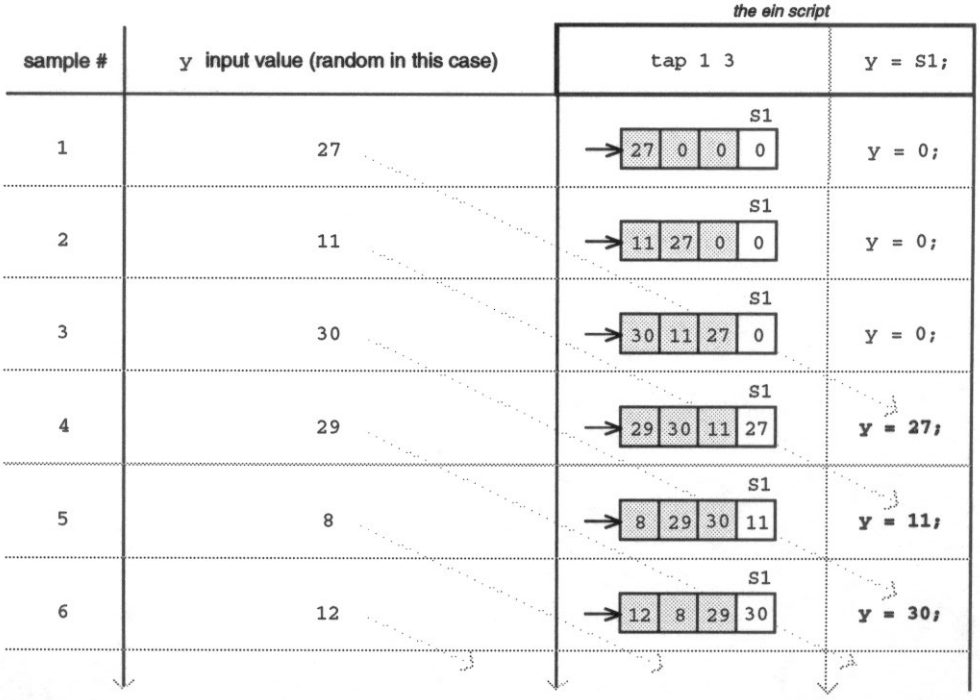
- Ein's only new statement is `tap`, used to create a delay loop in your sound. Since `tap` is best served by an example, open a **new ein.script**, type in this script, and **compile / run** it on your voice sound:

```
tap 1 11000
y = S1;
```

The only thing this filter did to your sound was delay it by 11000 samples (about half a second). Look for silence at the start of the waveform in the SoundView window. When you typed `tap 1 11000`*, ein stuck the value of* `y` *in a buffer called* `S1`*, and told* `S1` *to wait 11000 samples before letting* `y` *out again.*

Here's a diagram of what's going on, using the following filter as an example:

```
tap 1 3
y = S1;
```



- What would happen if, instead of merely setting `y` equal to S1, you would *add* the value of `S1` to the value of `y`? Using the previous `tap` example as your guide, 1) set `y` equal to the *sum* of `y` and `S1`, and 2) shorten the delay to about .1 or .2 seconds, remembering that you are filtering 22050 samples per second. Run this script on your

voice sound, and save the script in your home directory as **feed_forward.ein** with **Save As...** .

You just made a feed-forward filter *— it stored input from a sound and fed it in a tenth of a second later.*

- That's all! Hope you have time for everything in **Above and Beyond**.

What should be handed in

- **einstuff.wn**, containing:
  - copies of the Spectrum windows of your whistling and speaking,
  - your insights about the usefulness of the Spectrum as compared to the SoundView,
  - your explanation of the click heard with the script `y = 1;`
  - details (scripts and short descriptions) of your exploits in the **Above and Beyond** section.
- **square.ein**
- **feed-forward.ein**
- Any sounds you produce in the **Above and Beyond** section.

Above and Beyond

Try anything you have time for, in any order.

Saving sounds you produce

- Before you click **compile / run**, go to the text box labelled ***outputfile*** and type in the name you want the output sound to be saved as, then press <return>. *Don't forget to change the filename for each new sound!*

Stereo

All you need in order to make stereo sounds in ein are two new variables: `left` *and* `right`. *They each correspond to the position of a speaker cone, just as* `y` *did for monophonic sounds. Follow along...*

- Open up a new ein script and open an input sound to work on, just as you did before.
- Be sure you have selected **use input file** in the **Input file control** section
- Check to be certain that the box labelled **dur** contains a number that's as big as the **duration** of the input sound.
- Select **stereo** in the lower left where you have a choice between mono and stereo.
- Type this into the editing section:

```
right = y;
```

- **Compile / Run** your ein filter. Hopefully, you heard the sound played in your right ear. The filter just took the `y` value it got from the input sound, and sent that value to the right speaker.

- Q: What will this do?

```
right = 1 * y;
left = 1 * y;
```

A: not much... it will send the full sound to each of the speakers.

- Change the above filter so the right speaker receives 25% of the sound and the left receives 75% of it.
- Remember variables from C? Type this in as your script:

```
float ramp_up, ramp_down;

ramp_up = t / nsamps;
ramp_down = 1 - ramp_up;

right = ramp_up * y;
left = ramp_down * y;
```

Three new things here:

1) the `float` declaration just tells the computer that `ramp_up` and `ramp_down` are variables which can have decimal places attached to them, such as 11.23.
2) as ein moves along, sample by sample, through your sound, it keeps track of which sample you are on in the variable `t`. So, if the sampling rate is 22050 samples per second, and you have played through two seconds of the sound, `t` will be 44100.
3) `nsamps` holds the total number of samples in the entire sound.

So, `ramp_up` *starts out at 0, and grows to 1, while* `ramp_down` *starts out at 1, and shrinks to 0. How will these variables affect your sound?*

- Try the above filter on your sound. If you get something interesting, make note of it in **einstuff.wn**. If you've learned how to save sounds produced with ein, you can save this sound for submission.
- Imagine that you want to "echo" a sound from the right speaker to the left. You need to 1) feed the `y` input directly into the right speaker (that's the first thing you did in this section), and 2) feed a *delayed* `y` value to the left speaker (you learned how to produce a delayed `y` value earlier). Give it a shot, and save anything you come up with to **einstuff.wn**. If it doesn't work, explain what it did.

Feedback

- While your feed forward filter may have sounded neat, this feed-back loop will sound ugly. To make a feed-back filter, all you have to do is switch the order of the two statements in your feed forward script. Try it and wince! (Actually, it's not that bad with the volume turned down.)
- Can you explain why switching the order of the two statements produces such drastic results? Think about the significance of adding something to `y` *before* it is saved in the `tap` buffer. Compare this to the feed forward loop, which adds something to `y` *after* `y` has been saved in the `tap` buffer. Look at the following diagram, then save your insights into **einstuff.wn**.

		the ein script	
sample #	y input value (random in this case)	y = y + S1;	tap 1 3
1	27	y = 27 + 0;	S1 → 27 0 0 0
2	11	y = 11 + 0;	S1 → 11 27 0 0
3	30	y = 30 + 0;	S1 → 30 11 27 0
4	29	y = 29 + 0;	S1 → 29 30 11 27
5	8	y = 8 + 27;	S1 → 35 29 30 11
6	12	y = 12 + 11;	S1 → 23 35 29 30
2	19	y = 19 + 30;	S1 → 49 23 35 29
3	26	y = 26 + 29;	S1 → 55 49 23 35
4	14	y = 14 + 35;	S1 → 49 55 49 23
5	32	y = 32 + 23;	S1 → 55 49 55 49
6	27	y = 27 + 49;	S1 → 76 55 49 55

In this feed-back filter, first the delayed y value (S1) is added to the y input, then y is saved back into the tap buffer. Greater and greater values of y are added back in, until eventually all that's left is loud noise.

FM — Frequency Modulation

When Wynton Marsalis applies vibrato to a note, he lets it oscillate around some basic frequency — the carrier frequency. *Vibrato is basically all there is to Frequency Modulation; your favorite FM station merely applies (extremely fast) vibrato to the frequency you tune in at... the carrier frequency.*

- Select **Open...** from ein's menu, and find **~cs111/Labs/8-ein/fm.ein** .

In the ein script you just opened, the variable `carrier_frequency` *controls the carrier frequency, and is initially set to 523 Hertz (roughly an octave above middle C). The variable* `modulation_frequency` *is the rate of vibrato, measured in hertz — the initial setting of 4 hertz produces four oscillations per second. Finally, the variable* `modulation_amplitude` *controls how much the vibrato varies from the carrier frequency. At the modulation amplitude's initial setting of 1 hertz, the vibrato will vary from 522 Hertz to 524 Hertz.*

If you have learned how to save sounds you produce with ein, feel free to save anything funky for submission. Keep an eye out for pretty Spectrum windows, and feel free to copy any into ***einstuff.wn*** *using* ***Grab****.*

- **Compile / Run** the ein script as shown (you don't need an input file).

- Increase the amplitude of modulation (vibrato), so that the modulation is larger and more noticeable. **Compile / Run** it again. What happens if you make the modulator amplitude larger than the carrier frequency?

- Increase the modulator frequency to obtain faster vibrato. What happens if you make the modulator frequency larger than the carrier frequency?

- Play around with all three variables and try to come up with both pleasant and horrendous sounds. Save anything interesting that you come up with.

Vocal.ein

Imagine that you are standing in the middle of a street, singing "urrrrr...." At the same time, a deep-voiced guy in Athens, GA strikes in with his own rendition of "urrrrr...." Q: Frequency-wise, what do the two of you have in common? A: a combination of three "formant" frequencies which are buried in your "urrr" sounds. "Formants" are different resonant frequencies that show up in each human vowel sound.

Here is an ein script which attempts to simulate human vowel sounds using these three formant frequencies.

- From within ein, select **Open...** from the **File** menu and find **~cs111/Labs/8–ein/vocal.ein** .

- **Compile / Run** the script as you see it. Can you hear the "urrr" sound?

- From within a Terminal window, sing an "urrr" sound and save it into a sound file. Make it as close as possible to the "urrr" you heard in ein. After preparing it with `mu2linear`, open the sound in ein with the **Open input file** button.

- Click the **analyze input** button, and drag the Spectrum window so you can see the Spectrum you got when you ran the **vocal.ein** script. Do you notice any similarities? One of the problems with computer-generated voices is that they are sometimes too "pure." Can you see this in the difference between the two Spectrums?

- See if you can recreate any other human vowel sounds by changing the three formant frequencies and the fundamental frequency. Any non-human vowel sounds?

For further exploration...

- Look in the library for articles on computer voice recognition. Now that you (hopefully) have an idea what frequency spectrograms and FFT's are, the articles should make sense.
- In **/Net/dobro/musr/Apps**, look through some of the ein scripts in the **ein.scripts** and **ein2.scripts** directories.
- If you have a lot of time, sit down with **Bessie.app**, in **/Net/dobro/musr/Apps**. If you get through it all, you will have learned much of what there is to know about FM synthesis.

Lab #8 — TA Sheet

8-30-91 Rich Feit

Before the Lab

- Reboot each NeXT machine to clear **/tmp**. If you want, you can just check each one to make sure there are a few megabytes free on each machine.
- Bring extra earphones.

Things to Watch for

You might want to try the lab yourself beforehand, just because ein isn't a widely-used application, and it might take a little while to get the hang of it.

The concepts here aren't easy! Hopefully, in the section on analyzing sounds, the students will ask you questions if they're confused. More explanations appear in the cheatsheet, but sometimes it's easier to explain signal processing in person (so you can wave your hands a bit).

In the filtering section, the students should know more than enough C to be comfortable with the syntax. The difficult part may lie in understanding what happens to the `y` value. An ein script works on each sample of the sound. If you are filtering an input sound, ein takes a value of `y` from the sound, then modifies it from there. It might have been better if ein assigned separate variables to the input `y` and the output `y`, i.e.

```
y_out = y_in * 1;
```

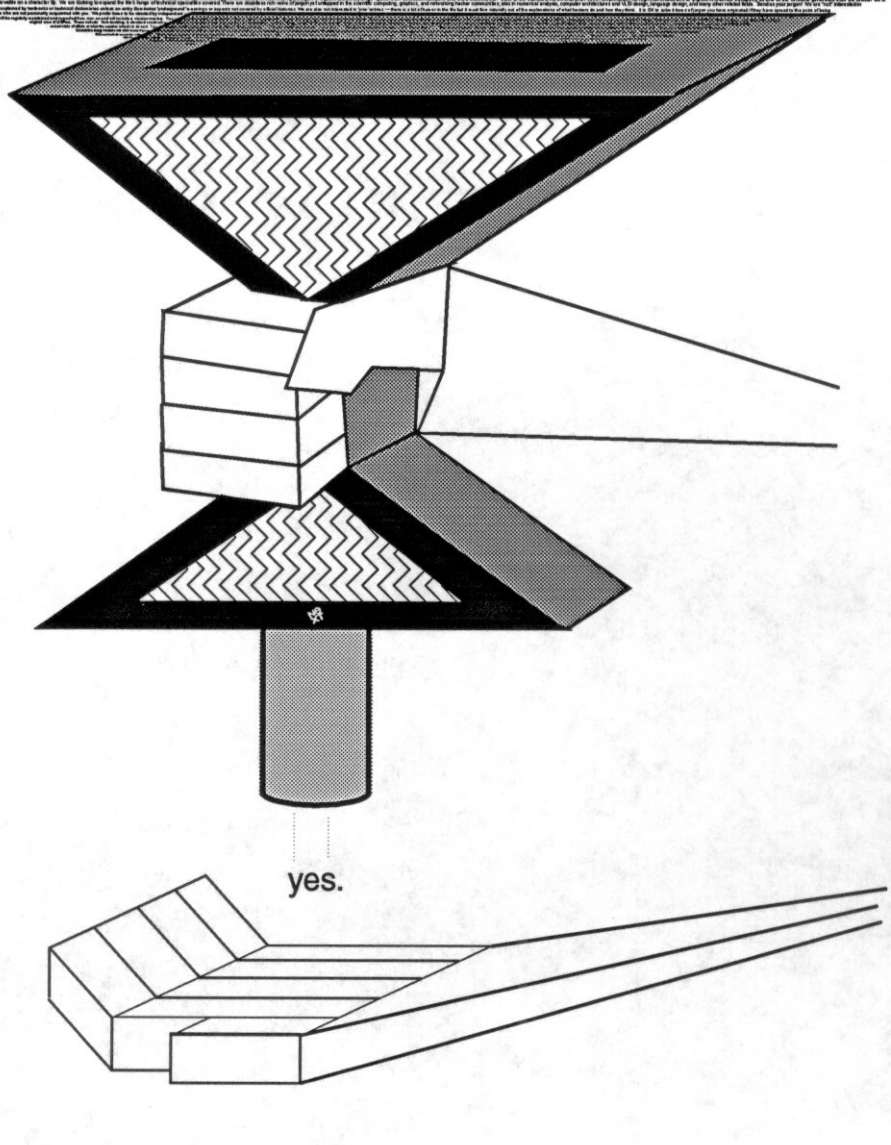
Just watch out for a little confusion there.

The `tap` statement will probably create the most confusion, at least at the beginning. This is where an explanation with a lot of handwaving will help. Maybe trace through the diagram of the feed-forward filter with any student who is confused by it. Even though it's not a terribly hard concept, I still get confused about it myself.

Finally, the two **Above and Beyond** sections for Vocal.ein and FM synthesis are mainly meant for playing around. The student need not understand every single fundamental concept for either of them.

A lot of this stuff appeared in MUS/COS 325. It's not easy, so encourage the students and remind them of this if they get frustrated.

Lab 9 - Information Processing



One of the most exciting aspects of using computers today is the enormous amount of information they make instantly available. As you may have seen in the communication lab, you can connect to a computer in Europe and retrieve a document within a few minutes.

OK, so you can grab the Bible from a computer in Finland and read it on line. You can even print it out on a laser printer. But why not find a copy in the library and read it in the solitude of the C floor?

Actually, if you merely intend to read it cover-to-cover, then the library might be a better choice. The thing that makes researching on the computer a completely different experience than a session in the depths of Firestone is the way the computer can process (sort, filter, etc.) the information it's given. If, instead of simply reading the Bible, you want to find out what it has to say about a specific subject, then you may want to consider taking a trip to the NeXT cluster.

Given a computer and an Internet connection, you have at your fingertips more text than you could read in a lifetime. And, you probably have more power and tools available than you care to learn about. Don't worry about it— this lab is more like a scavenger hunt than in-depth training. Just keep your eyes open for something you feel you can use in real life.

Purpose

- To learn how to search for specific subjects using NeXT's Digital Librarian.
- To learn a little more about UNIX — enough to help you in your researching.
- To understand why people bother to use UNIX instead of fancy, fast, easy-to-use programs like Digital Librarian.

Materials

- Patience.
- Nelson's concordance of the King James Bible, if you want to make sure the computer doesn't miss anything. Your TA will have it if Firestone lets it out of the building.

References

Required CheatSheets

- Digital Librarian
- UNIX
- Grab

Online Help

- Librarian's **Help** menu
- UNIX `man` pages for `grep`, `wc`, `sort`.

Procedure

Setup

- Log into a NeXT.
- Use the File Viewer to find **Librarian.app** in **/NextApps**. You might want to drag it to the dock on the right-hand side of the screen. Otherwise, just remember where it is.
- We have a folder called **data** deep in the **cs111** home directory which is full of materials for this lab. Because it can be a pain to find it each time you need it, here's a way to "fool" the computer into thinking that **data** is inside *your* home directory: Pop open a Terminal window, and type

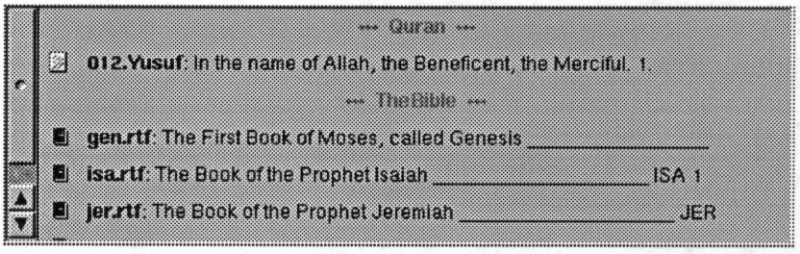
  ```
  ln -s ~cs111/Labs/9-info/data ~/data.
  ```

  You just made a "soft link" from the **data** directory in **~cs111** to a **data** directory in your home directory.
- Open up a WriteNow document and save it as **datastuff.wn**. You'll use this to paste in pictures and text for submission. **Save it every once in a while.**

Using Digital Librarian

Librarian is great if you need to quickly find a word in a set of documents. If you find yourself surprised by how quickly Librarian finds information, read about indexing *in the Librarian CheatSheet.*

- Inside your **data** directory there is a Librarian bookshelf called **data.bshlf**. Double-click it to open it up, and read enough about Librarian to be able to search for a word.
- Find the only document in any of the folders that contains the word "harbinger". Open up the document from Librarian, find "harbinger" in the text, and **Copy** the line which contains it into **datastuff.wn**.
- Imagine that you are doing a study of wolves and how they are depicted in the Bible and the Quran. Instead of sitting down with a copy of each in Firestone, you decide to trek down to the NeXT cluster (it's worth the hike!) to gather your information. Find every occurrence of the word "wolf" in both texts, and think about doing this by hand. Here's the catch: you must search both texts at the same time, so your list looks like this:



Check out the Librarian cheatsheet or use the **Help...** menu if you don't know how to do this. When you get a similar list, use the Grab application to copy the whole window into **datastuff.wn**.

Using UNIX to deal with information

Although Librarian is super-fast, it's not much good for anything but simple searches. Hopefully this next section will fulfil both the second and third purposes of this lab.

- Get to a Terminal window, `cd` to your **data** directory, and then into the **shakespeare.txt** directory.
- First you'll learn how to match — and surpass — Librarian's search capabilities with a single command: `grep`. Read about it in the UNIX CheatSheet, then find the line which contains the phrase "honey secrets" in the file **Venus_and_Adonis**. Already you have exceeded the abilities of Digital Librarian, which refuses to look for whole phrases when it uses its indexing. Copy the "honey secrets" line into **datastuff.wn**.
- Read about the use of "wildcards" in filenames (UNIX CheatSheet), and use your newly-acquired knowledge to search for the only occurrence of "wary" in *any* of the poetry in this directory. Copy the output into **datastuff.wn**.
- Read about the `wc` command, and use it to count the words in the file **A_Lovers_Complaint** . Copy the number of words you get into **datastuff.wn**.
- `cd` to **~/data/bible.txt** . Ancient scholars devoted years to counting the words in the Bible, or analyzing the frequency of certain phrases. Using what you know about filename "wildcards," count all the words in the Bible, using one command. It should take the computer about two minutes. Copy the *total* number of words into **datastuff.wn**.
- If you haven't already, you're about to discover the beauty of UNIX. Read about "pipes" in the UNIX cheatsheet. Is this sufficiently beautiful to excuse UNIX's surface unsightliness? (yes.)
- Find the total number of lines in the Bible which contain the word "silence". First you need to think of a command to search all of the files in this directory for "silence," — then you can pipe this command into `wc`. Copy the output into **datastuff.wn**.
- `cd` to **~/data/democracy** and try `grep`ping all of the documents for the string "free". You will need to save this output into a file, BUT there's an alternative to Copy&Paste... read the tiny section of the UNIX CheatSheet about *output redirection*. Then, save the output of this `grep` command into a file called **free.grep**. (You'll hand this file in.)

Scavenger Hunt

Here's a list of things to find. Use any tool you know of and skip any difficult item before you get frustrated with it! *Use output redirection and/or Copy&Paste to add anything and everything you find into a file called* ***scavenger.txt*** *(you can open up a text document with the Edit application). Also,* include the method you used to obtain your results. *And they're off...!*

- Remember the questionaire you filled out during the first lab? Your TA's compiled and typed in all of the data, in this form:

```
<word>: <list of people -userID's- who said it this way>.
```

So, an entry for the word "heron" might look like this:

```
heron: jmstipe, mikemlls, pterbuck, billbery, jeffholt.
```

The file is called **~cs111/data/questions.txt**. Your assignment: get a list of people who share words in common with you.

- Find the name of the only document in the whole **data** directory which contains the word "vicious".
- Count the number of words in Shakespeare's "The Phoenix and the Turtle" (it's in the directory **~cs111/data/shakespeare.txt**, and it's called **The_Phoenix_and_the_Turtle**)
- Find the line in Shakespeare's poetry which contains the word "murmur". Do the same for "reckoning" and "pelleted".
- Somewhere in the **~/data/democracy** directory we have hidden a small smiley-face, like this guy: " :) " . Find the line that contains it.
- Find the number of lines in the Bible which contain the string "swan".
- Count the number of books in the Bible, using UNIX commands and pipes.
- Find the two lines in Shakespeare which have both "mind" and "body" in them.
- Using pipes, find the one book of the Quran which has 564 words.

What should be handed in

- **datastuff.wn**, including:
  - The name of the document which contains the word "harbinger".
  - A copy of the Librarian window which you used to find "wolf" in the Bible and the Quran.
  - The line in **Venus_and_Adonis** which contains the phrase "honey secrets".
  - Every line in the directory of Shakespeare's poetry which contains the word "wary."
  - The number of words in "A Lover's Complaint."
  - A word count of the Bible.
  - The number of lines in the Bible which contain the word "silence".
- **free.grep**, containing every occurrence of the word "free" in the **democracy** directory.
- **scavenger.txt**, containing anything you found in the scavenger hunt.
- A file called **beyond.infolab.txt**, containing text output from any of the **Above and beyond...** sections, if you get to them.

Above and Beyond

Try anything you have time for, in any order.

More grep

- Type `man grep`, and find the section of the manual page that lists all the various "flags" you can supply. The first flag you should see is `-v`. Copy the list and save it for use in this section (you can print it out if you want).

  To include "flags" in the grep statement, just type `grep` followed by a dash and all the flags you want to use. For example, if I wanted to find just the names *of the files in the* ***Yeats*** *directory that contained the word "bid," and I didn't want to worry about whether it was upper- or lowercase, I would type*

  ```
  grep -li 'bid' data/Yeats/*
  ```

- Get a count of the number of lines with "dream" in MLK's "I Have a Dream" speech. The catch: you're not allowed use `wc` to count the lines, and you're not allowed to do it manually.
- Find the *line number* and the name of the document where this phrase can be found: "I am from those rivers".
- Find the only line with the word "me" in it. Unfortunately, there are more than 20 lines with words which *contain* "me," like "me<u></u>ntioned" and "governme<u></u>nt." Use one of `grep`'s flags to find just "me".
- Using two flags, get a count of the number of lines in Yeats's "A Dramatic Poem" which contain the *word* "a". The poem is in a file called **DramaticPoem** in the **~/data/Yeats** directory.
- Get a list of all the *file names* in the **~/data/Yeats** directory which contain the word "want". Don't list the actual text... just find the names of the files.

Regular expressions

- Read about regular expressions in the UNIX CheatSheet.
- In the **data** directory, there is a subdirectory called **Yeats**. Find all the lines of Yeats's poetry which *end* in "run". Paste everything appropriate into **beyond.data.txt**.
- Find all the lines in the Constitution with 5-letter words which start in "s" and end in "e". Put spaces around your search-string if you don't use `grep -w`!
- Find every line Shakespeare wrote that contains a word which starts with the first letter of your first name, and ends with the last letter of your last name. Hint: if you haven't learned to use the `-w` option with grep, put spaces around your search-string to make sure the computer only looks for whole words. In **beyond.data.txt**, paste in your method and your results.
- In **~/data/democracy/timeline**, there is a timeline of events which the author decided were relevant to democracy in the U.S. Here's a sample line:

```
1692  Witch hunts in Salem, MA; 19 die
```

Your task is to find every event that occurred in a year ending in '0', e.g., 1690. You will have to use a regular expression that 1) looks at the beginning of each line (where the year is), and 2) looks only at the last digit of each year.

Aliases

At some point in your UNIX career you might come across a sequence of commands that you repeat often, and you'll wish for a way to execute these commands quickly and easily. Fortunately there is *a way: aliases.*

- Read all about aliases in the UNIX Cheatsheet, and pay special attention to argument-passing.
- The `w` command tells you what everyone is doing. Make an alias called (for instance) `doing` which will tell you what a friend is up to when you type

  ```
  doing <friend's userid> .
  ```

  You'll need to use `w`, `grep`, and pipes.
- Make an alias that will play **/LocalLibrary/Sounds/Uh,_no,_no,....snd** every time you type `oops` .
- The `echo` command just prints out whatever you type (in quotes) after it. Using pipes and the `write` command, make an alias that will send a friend the message "What is up?" when you type

  ```
  whatsup <userid>.
  ```

For Further Exploration

- Besides everything you can find in cs111's **data** directory, Princeton has Shakespeare's complete works in the **/NextLibrary/Literature/Shakespeare**.
- Princeton also has *The Oxford Dictionary of Quotations* and *Webster's Dictionary* online. Both applications are in **/NextApps**, and both are intuitive and easy to use. Although you probably won't need help, both have **Help** screens, too.
- If you're interested in learning more about UNIX, get Brian Kernighan and Rob Pike's *The UNIX Programming Environment.* Look for it in the U-Store. It's very well-written and clear, and the first few chapters assume you are a complete novice (even if *you* are not).

  Although you used UNIX quite a bit in this course, no labs were completely dedicated to teaching it. Even if you just read the very first chapters of the book, within a week UNIX will make more sense to you and seem much friendlier than it possibly could now.
- At McGill University in Canada, there is a machine called Archie which can help you find documents or programs at ftp sites. To get there, type `telnet quiche.cs.mcgill.ca` , and log in as `archie`. To search for a word, type `prog <word>`, or type `help` to get help. When you get a list of `ftp` sites that have what you want, type `exit` to leave.

Lab #9 — TA Sheet

8-30-91 Rich Feit

Before the Lab

- If you can, get a copy of Nelson's concordance to the King James Bible. It might be interesting to compare this to `grep`ping from the digital Bible.

- Remember those questionaires from the first lab? You get to enter all their data into a file. Yeah! Actually, Jon wrote an interactive program called **dataenter** (in our **bin** directory) that should make it a little less tedious. Supply this filename as the output-file: **~/Labs/9-Info/data/questions.data**. Check it when you're done to make sure its format agrees with the format I described in the lab.

Things to Watch for

Originally, this lab was supposed to be about databases. It didn't have much UNIX in it. I realized, though, that everyone really shouldn't get the impression that fancy GUI applications are always better to use than UNIX. So... I added the third purpose: to show the students why anyone would bother using UNIX tools instead of NeXTstep applications. Hope it works...

One source of confusion I can think of is the directory structure. When they are using `grep`, the students have to search directories of text files. They'll get strange results if they search through RTF files. Be prepared to navigate them out of a wrong directory if they're lost.

Also, I didn't always give instructions step-by-step in the **Procedure**, especially in the **Scavenger Hunt** section. This way, when they figure out how to do something, they experience the Thrill of discovery. Frustration is our enemy, though... remind everyone to skip anything they find too difficult.

I hope the soft directory link I have them create at the beginning isn't more trouble than it's worth. You might have to help them if they put their link in the wrong place.

One more thing: Librarian may seem quirky, but it's not too bad. If a search doesn't produce anything, make sure the Preferences are set right.

CheatSheet

Table of Contents

Binary Numbers

Learning to Count

Counting is something you have done since you were a small child. But whether you realize it or not, you have been using only one of an infinite number of counting systems. We have ten fingers, so we count in base ten, where ten is the first number that needs multiple digits. In a similar vein, computers have two "fingers:" 1 and 0. Electricity dictates this state of affairs since it is easy to have electricity in one of these two states. Therefore, when a computer counts, it uses only two digits…0 and 1, or off and on. Let's take a look at how we count, then compare this to how a computer would do it in binary.

We can break up any number into the values of its individual digits. For example:

$324 = 3*100 + 2*10 + 4*1$
$= 300 + 20 + 4$
$= 324$ (yup, it's still 324!)

Any number can be broken up this way. Another way of writing this is to use powers of ten, the base we are counting in.

$324 = 3*10^2 + 2*10^1 + 4*10^0$
$= 300 + 20 + 4$
$= 324$ (zowee! still 324!)

As you can see, a number can be broken up into its component digits multiplied by the base with an exponent. (Remember 1's place, 10's place, 100's place, etc?) Base two (binary) numbers are broken up in exactly the same way. Let's stick with the same example, but this time write it in base two:

324 (base 10) $= 101000100$ (binary, base 2)
$= 1*2^8 + 0*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0$
$= 256 + 0 + 64 + 0 + 0 + 0 + 4$
$= 324$ (base 10) (lo and behold…still 324!)

OK. Hopefully you have experienced a glimmer of understanding so far. Let's change gears and look at this from another angle. Near the end of this CheatSheet is a table of numbers in decimal and their binary equivalents. Try to count along with it, and try translating other binary numbers you make up. A scientific calculator can check your answers for you.

If this makes perfect sense to you, you have the idea. Great. If not, that's OK too. Just trace through the binary numbers and break them down. 1011 in binary, for example, is 8 + 0 + 2 + 1 = 11. Try it for the other numbers. Here is a handy chart of the powers of 2 which you can use to convert binary numbers to decimal and vice-versa.

2 to the:	0	1	2	3	4	5	6	7	8	9	10	11	12
Decimal:	1	2	4	8	16	32	64	128	256	512	1024	2048	4096

If You Understand Binary But Still Haven't Had Enough

Try base 16 counting! This is called hexadecimal and is commonly used to represent numbers to programmers. A hexadecimal number has fewer digits than its binary equivalent, so programmers often prefer to see data in hexadecimal rather than the space-inefficient binary format. If you are sharp, you are probably wondering what happens when you get to 10 in hexadecimal. Since it is base 16, there should be single digits all the way through 15. The remaining digits are A, B, C, D, E and F. See the chart below and try to follow what is going on.

The Chart

Decimal (Base 10)	Binary (Base 2)	Hexadecimal (base 16)
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
:	:	:
26	11010	1A
27	11011	1B

New numbering systems are not easy if you are seeing them for the first time. The trick for most people seems to be translating numbers between systems and looking at charts like the one above until the idea clicks and they understand.

C THE PROGRAMMER'S LANGUAGE OF CHOICE - 1

Although no programming language is perfect, C is regarded by many as coming pretty darn close. C is used for programming a large portion of today's commercial software, is supported on almost every type of computer system, and is in fact the language that was used to write the majority of UNIX. The main power of this language is its modularity; there is not too much involved in the actual language, but a large number of libraries have been created which include many prebuilt functions that are used over and over again (for a definition of "library" and "function", refer to your class notes). This CheatSheet will attempt to explain some of the basics of the language, enabling you to write some simple programs.

Where to Write Your Program

The simplest place is in an Edit window. Type your program as regular text and save it to a file with a descriptive name and a ".c" on the end.

Basic Program Structure

All C programs have the same basic structure: they start off by saying what libraries will be needed (if any), then go into definitions of constants, variables and functions (if any), followed by the program itself (known as "main"), followed by the programmer's own functions which were defined at the top of the program. As an example, look at the following skeleton:

```
#include <such_and _such_library.h>            /* library section */

#define BIG  987654321                        /* definition section */
int i;                       /* everything in /*...*/ is a comment! */
void my_function();

main()                   /* this is the 'main' part of the program */
{
    printf("blah");
    my_function();
}

void my_function()                          /* I wrote this function! */
{
    printf("hi");
    printf("how are you?");
    i = i + 1;
}
```

Notice that every command must be terminated by a semi-colon (but `include` and

`define` statements do not). If groups of commands go together, such as the three in `my_function` or the commands in `main`, they need to be contained inside braces. This tells the program that they go together.

The `/*...*/` structure is a comment. Anything inside these will be ignored by the compiler. Please note, the more *good* comments that you put in your program, the easier it will be for both yourself and others to go back later and look at what you did. Useless commenting, such as mentioning something that is very obvious, is not good programming style. A good comment is something that explains how a tricky command works or a description of what a function does.

The `#define` is a way that the computer can assign a word to a number value that you might have to repeat in many different places throughout the program. In this example, instead of typing 987654321 over and over, you just type `BIG` and the computer automatically replaces it with whatever value was defined at the top. This not only makes things go faster when you program, but it also makes things easier to read. For example, the statement

```
circumference = 2 * PI * radius
```

is much better than

```
circumference = 2 * 3.1415926535 * radius.
```

To do this, just type the statement

```
#define PI 3.1415926535
```

at the top of your program. The asterisk is the computer's way of multiplying.

(By the way, it is common practice to capitalize definitions, just so you or others don't get confused later while reading the program.)

Another *very big* advantage of definitions is that it is possible to change their values at the top of the program and they will change everywhere in the program. That is to say, if the value of π were suddenly changed to 4.1415926535..., you would just change it at the beginning of your program in the definition section instead of having to go through the entire program and changing each "circumference = ..." line to the new value.

Finally, a definition is known as a constant, which means its value can't be changed while the program is running. If you think you will need something whose value can change, look at the next section...

Variables

A variable is an item in a C program that can take on different values throughout the program. Whereas the constant `PI` above could only have one value during the execution of the program, a variable `pi` could have many different values, changing whenever the program told it to. We will only be using integers in our programs. To define an integer variable named bar, we would type:

```
int bar;
```

Right beneath the `#define` section. This tells the program to set aside enough room for an integer to be stored at a place called 'bar'. Anytime we want to access this

integer, all we have to do is call it by its name, much like we did with the definition of `PI` above:

```
total = 7 * bar
```

This will place 7 times the value stored in `bar` into a different variable named `total`. If we just want to store a value in `bar`, we use the equal sign:

```
bar = 10;
```

Now, anytime we ask what value is in `bar`, the computer will respond '10'. We could of course change it to whatever we want, and the computer will reassign the variable to the new value.

How to Put Words on the Screen

Information is sent to the screen via the `printf` command. This command has fairly straightforward syntax, but when you first look at it, it may appear complex. A simple example is to print out a phrase:

```
printf("this is a phrase");
```

This will print the words in quotations on the screen. However, any future `printf`'s will just get appended to the same line because we didn't print out a carriage return after the line. To do this, we have the special symbol `\n` which stands for "newline". Therefore, to have our statement be on a line of its own, we have to type:

```
printf("I am a phrase and I am on my own line!\n");
```

Notice that no spaces are needed between the end of the line and the `\n`. Other special characters are `\t` for tab and `\\` to print the backslash itself.

Sometimes, you will want to output the value of an integer variable using the `printf` statement. To do this, we use another special symbol, `%d`. Assuming we have two variables `side`, and `area`, we could have some snazzy looking output by using the `printf` command:

```
printf("The side of your square is %d centimeters\n",side);
printf("The area of the square is %d centimeters squared\n",area)
```

Each time the `printf` comes across a `%d`, it looks for something to put in its place, namely the items following the quotation marks, separated by commas. It might be easier to have this instead:

```
printf("side:  %d\tarea:  %d\n",side,area);
```

Although it looks quite confusing, it's quite simple when you take a closer look. It starts out by printing the word "side" followed by a colon and then two spaces. The command then comes across a `%d` so looks past the entire statement to find the first variable, `side`. Next, it reads the `\t` and tabs forward to put some space between the first and second group, and prints out "area: " followed by the variable `area` (because that is the variable to replace the second `%d`) followed by a newline to prevent further output on the same line.

The For Loop

There are several different loops in C, but the `for` loop is probably the most frequently used and hence the one we discuss here. A loop is a method to have the program do something over and over again. It is a lot better to enter code one time and have the computer do it over and over again than to have to type it in multiple times. If we wanted to print out a phrase 10 times, we would type this:

```
for (i = 1; i <= 10; i = i + 1)  {
    printf("This isn't so bad.\n");
}
```

The integer variable `i` (declared at the top of the program) gets set to 1 when the loop is first run. Next, the loop tests to see if `i` is less than or equal to 10. Since it is, the body of the loop is executed. It now goes to the third part of the loop which will increase `i` by one, and then back to the second part to see if it still is less than or equal to 10. This will continue until `i` has been increased to greater than 10.

Since the variable that we use as a counter is a regular integer variable, it can be used in any expression in the body of the loop. So if we wanted to print out the numbers from 1 to 10, we could type:

```
for (i = 1; i <= 10; i = i + 1)  {
    printf("%d  ",i);
}
printf("\n");
```

This will print out each number from 1 to 10 followed by two spaces, all on the same line. Then, when it is finished, a newline character will be printed. (The newline is not printed after each number because the `for` loop will only execute those commands inside the braces.)

In case you haven't fully gotten it yet, the `for` loop has the following structure:

```
for (<initialization> ; <test for stopping> ; <update>)  {
    body of loop;
}
```

One last example is the loop

```
 for (;;)  {
     printf("Are we having fun yet?\n");
}
```

Which prints the same line over and over again, and since there is no test for stopping, will continue forever (typing control-c will cancel the program should you ever do this).

Compiling and Running

After you have written a C program using the Edit window, save it using a fairly descriptive filename with a ".c" on the end. (The ".c" ending is necessary to tell the compiler that this file is a C program.) The next step is to actually compile the

program. When you typed your file, you made an ordinary text file. The compiler reads that text file and translates what you typed into something the computer can understand. The command to compile is fairly simple; from a Terminal window, type:

`lcc <filename>` (remember the ".c"!)

`lcc` stands for "local C compiler," convenient. Assuming there are no errors, the compiler will create an executable file called **a.out** that you can run. Note that every time you compile a program, it creates a file called **a.out** which will overwrite your previous version of the program. If you want to keep an older version of your program, type `mv a.out <new_name>`. This will move a.out to the new name that you picked. For more about `mv`, type `man mv` or see the UNIX CheatSheet.

If there are any errors in your program (a forgotten semi-colon or a misplaced bracket for example), the compiler will print out an error message saying what was wrong, what was expected and where the error occurred. Unfortunately, it is up to you to go back and fix the problem. This step of going back and fixing problems is called *debugging* and is what most programmers spend the majority of their time on. Fortunately, it is a tad bit easier on the NeXT because you can keep both a Terminal window and your Edit window open at the same time, allowing you to see your code and the error messages at the same time. One other nice feature is that the Edit program allows you to immediately jump to a particular line number in your program. Choose **Line Range...** under the **Find** menu which is located in the **Edit** menu. Trust me, having multiple windows is a whole lot better than switching back and forth on a non-windowing system!

The Programmer's Drinking Song:

"100 little bugs in the code,
100 bugs in the code,
Fix one bug, compile it again,
101 little bugs in the code.

101 little bugs in the code..."

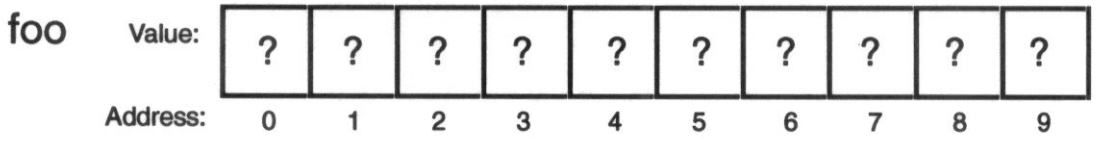
- From a post to rec.humor

C THE PROGRAMMER'S LANGUAGE OF CHOICE - 2

Now that you know some basic C commands, you should be able to write several simple programs. There are a few more items, however, which can be very useful in writing more complex programs.

Arrays

An array is a grouping of variables that has a fancy method of accessing the information it contains. An entire array can be thought of as a street, and each space in the array can be thought of as a house. If you want to leave something in a particular mailbox, you just tell the computer which street and address you want. For example, if this:

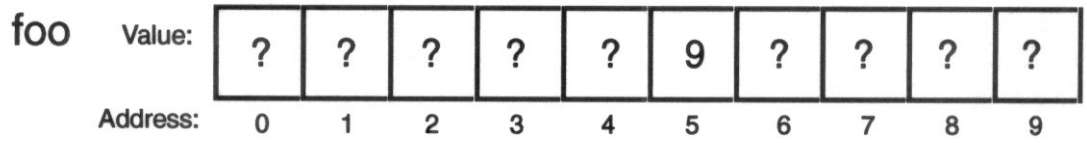


was our array `foo` which has ten elements (0..9), we could leave something in spot 5 by saying:

```
foo[5] = 9;
```

Street Address

Which would make the array look like:



We could later say

```
height = foo[5];
```

And the value returned to `height` would be whatever is stored inside the array at that address (in this case, 9). Note that until we actually place a value in an array location, we have no way to know what is there—hence the question marks.

If we wanted to switch elements in an array with each other, we need to do a little more fancy work. Suppose that in `foo[5]` we have the number 9 and in `foo[8]` we have 12. To interchange these values, we need to:

- Take one value and store it in a temporary location (i.e. a variable called 'temp').
- Copy the second value into the first value's location.
- Place the value in temp into the second's location.

In code it would look like this:

```
temp = foo[5];
foo[5] = foo[8];
foo[8] = temp;
```

If you think about it for a moment, you will realize that `foo[8]` is now the location with the value 9 and `foo[5]` has the number 12.

The way to declare an array is similar to the way we declare a variable, except we say how large we want the array structure to be. For example, to define the array `foo` as it was described above, we would type:

```
int foo[10];
```

beneath the `#define` statements.The number in the brackets tells the computer that you want to have an array of ten elements. *In C, all arrays start at 0 and go up to one less than the number you declared in the definition.* Although this seems strange, it really means that you are allocating 10 memory spots, labelled 0 through 9.

Conditional Statements

A conditional statement is a logical statement that can have one of two outputs, either true or false. We've all seen them, even used them in our day to day life. An example of a conditional is: IF it is raining. The statement is of course true if it is precipitating, and false if it is not. Although this is fine and dandy, it is pointless unless we have a course of action depending on the veracity of the statement. A better statement is: IF it is raining, THEN I will use my umbrella. In C, there is a similar structure that is used to check if a statement is true and then act on it. One example is to see if a variable is less than some number. If it is, we can, for example, print out the value of the number:

```
if (number < 20)  {
    printf("The number is %d.\n",number);
}
```

The expression inside the parentheses is the condition. If it is true, then the number will be displayed. But what happens if the statement is false? In the above case, nothing happens – the program will just skip the entire command much like the 'raining' example. If it isn't raining (the statement is false) all we know is that we won't use our umbrella; we're not quite sure what we will be using. If we want to specify some alternative action, we would say: IF it is raining, THEN I will use my umbrella, ELSE I will use my sunglasses. Now we know what happens if the statement is either true or false. In C, it is the same:

```
if (number < 20)  {
    printf("The number is %d.\n",number);
}
else  {
    printf("That number is too big!\n");
}
```

Now, no matter what `number` is, we know what will happen.

There are six different 'operators' that can be employed in a conditional expression:

(a == b)	true if a is equivalent to b
(a != b)	true if a is not equivalent to b (! means not)
(a < b)	true if a is less than b
(a > b)	true if a is greater than b
(a <= b)	true if a is lesser than or equal to b
(a >=b)	true if a is greater than or equal to b

Functions

A function in C is a self-contained segment of code that does "something." The `printf` command is a good example of a function: it takes a phrase in quotes as input, and perhaps several variables that will be used in conjunction with the phrase, and outputs the full phrase which goes to the screen. It is self-contained, because we don't see any of the underlying workings, just the end result. To see why we use functions, read on!

Suppose that we want to have a program that tells a joke. A good design would be to have the program pretend that it was actually talking to the user. To do this, we need to employ a loop that has delay function, allowing the user to have a pause before hearing the punchline. This could be an example:

```
#include <stdio.h>                                  /* standard library */

#define DELAY 5000000                          /* delay (about 1.5 sec) */
int i;                                      /* variable for the counter */

main()
{
    printf("Hey, do you want to hear a joke?\n");
    for (i = 1; i < DELAY; i = i+1)                  /* first delay loop */
        ;                                                /* does nothing */
    printf("Why did the whale cross the ocean?\n");
    for (i=1; i < DELAY; i = i+1)                   /* second delay loop */
        ;                                                /* does nothing */
    printf("To get to the other tide!  (hehehe)\n");
}
```

This program prints out the first line, then has a delay, prints out the actual joke, has another delay, and then reveals the punchline. Although this is a very short program, there is a repetition which, in other circumstances, could make the program much longer than it needs to be (imagine 10 jokes). A much better way to write this

program would be:

```
#include <stdio.h>                                    /* standard library */

#define DELAY 5000000                               /* delay (about 1.5 sec) */
void pause();
int i;                                            /* variable for the counter */

main()
{
    printf("Hey, do you want to hear a joke?\n");
    pause();                                        /* call the pause function */
    printf("Why did the whale cross the ocean?\n");
    pause();                                                    /* pause again */
    printf("To get to the other tide!  (hehehe)\n");
}

void pause() /* delay function */
{
    for (i = 1; i < DELAY; i = i+1)          /* for loop counts to DELAY */
        ;                                       /* but does nothing else */
}
```

Now, the program is easier to read (a descriptive function name is much better than a `for` loop with a comment after it) and if anything needs to be changed in the function, you don't have to go hunting through all the different areas in your program. You change the function and you are finished.

Random Numbers

Random numbers can be very useful in many different types of programs, and if nothing else, they provide an ideal way for programmers to generate large amounts of varied data for testing algorithms. Even though these numbers are called random, they are in fact anything but that. The computer uses a preset method for getting the numbers, so although the numbers appear random the first time you see them, if the same code is executed again, the numbers will reappear. Here is our "randomizer":

```
#include <stdio.h>

#define MAXNUM 10                        /* number of randoms to generate */
#define RANGE 100                       /* highest possible random number */
int i;                                                    /* loop counter */
int num;                                             /* holds random number */

main()
{
    for (i = 1; i < MAXNUM; i=i+1) {  /* loop to make and print nums */
        num = rand() % RANGE;      /* actual "randomizer": [0..RANGE] */
        printf("%d  ",num);       /* print out the number with spaces */
    }
    printf("\n");                        /* put the output on its own line */
}
```

And here is our output:

```
tome% lcc rand.c
tome% a.out
75  15  40  27  54  8  99  61  78
tome% a.out
75  15  40  27  54  8  99  61  78
tome%
```

For most applications, this non-randomness is not a problem. (Even though there is a way to generate real random numbers if you actually need to.) To create an array of random numbers, we could put the value `num` into an array of size `MAXNUM` and we would have some easily generated data for testing programs (hurrah!).

Copy&Paste

Moving data from place to place is a fundamental task for computers. Copy&Paste is a primary means of data movement on the NeXT—especially between applications. (Look at the fronts of the C and V keys!) The idea is simple—you have a "Pasteboard" that you can copy data (text, pictures, etc.) onto. You can then take this data off of the pasteboard and plop it down almost anywhere. The method is simple and applies almost everywhere. There are four basic steps:

- *Select the data you wish to Copy&Paste.*
- **Copy** *this data to the Pasteboard.*
- *Move to the area you want the data pasted.*
- **Paste** *the data.*

From the NeXT Basics document you already know how to select text—drag over it. In drawing programs, usually clicking an object will select it. Copying to the pasteboard is simple—choose ***Copy*** *from the* ***Edit*** *menu, which appears in nearly all applications. Now move to the area where you want the data placed (start the application, enter the document, whatever it takes) and choose* ***Paste*** *from the* ***Edit*** *menu. The text, picture, or whatever should appear in a few moments. Done!*

Example

Here is a quick, hands-on example of Copy&Paste for you to try:

- Open a new document in WriteNow.
- Type something random. (e.g. glurphlewambasmooklebrimbfoodle)
- Select part of the text by dragging over it with the mouse.
- Choose **Copy** from the **Edit** menu.
- Open up a new document.
- Choose **Paste**. The text you copied should appear.

This can be done between applications as well as between documents—just use **Copy** as you normally would, switch to the application you want to **Paste** into, and choose **Paste**. The data should soon appear in the document.

*We used Copy&Paste extensively to make the labs and CheatSheets. All of the graphics you see embedded in these documents were brought to you courtesy of two simple operations—****Copy*** *and* ***Paste****. Hopefully you are getting the point that this concept is very important to getting the most out of the NeXT computer. (And vital to getting anywhere in this course...) Take the time to understand it fully. If you have questions, ASK!*

Cut&Paste

There is a close relative of Copy&Paste called Cut&Paste that you may also find useful. The only difference between the two is that **Cut** will delete the selected item/text, while **Copy** won't. This makes sense, of course, if you think about the names. **Copy** leaves the original, making a duplicate, while **Cut** snips the original out. **Cut** is always found in the **Edit** menu near **Copy**. Try them both to see the difference. To use it, just substitute **Cut** wherever you see **Copy** in the instructions for Copy&Paste.

Diagram!

/LocalApps/Diagram.app

Diagram! does what its name implies—it creates diagrams. There are two main features that distinguish Diagram! from a typical drawing program:

- If you draw a diagram interconnected with lines, then move any of the objects in the diagram, the lines adjust so everything is still connected properly.
- The ability to link documents to the objects in your diagrams, creating a hypermedia environment. Hypermedia is the idea that linear reading is old–fashioned...instead of going from one page to the next, you follow a thread of thought, jumping from place to place in different media.

Even though you can create the same results that Diagram! produces in other applications, Diagram! is specifically tailored to the needs of creating a blueprint, design, game plan, project, strategy, presentation, proposal, proposition, suggestion, arrangement, order, ordering, contrivance, device, expedient, cabal, conspiracy, covin, intrigue, machination, or practice. (Yes, I am using **/NextApps/Webster** and not being at all subtle about it.)

Diagram! is object-oriented (just like TopDraw). It provides you with "Palettes" of objects which you can drag into the document you are creating, then link them up with lines. To make a line between two objects, drag from one to the other. Now triple–click–drag one of the objects. The line will adjust according to where you move the object! (If you do not understand what I am talking about, just try it; it is self–explanatory once you see it.) To link a file (ie. text, sound, drawing) to an object, simply find the file's icon in the File Viewer and drag it onto the object to which you want it linked. Voilá! Now double-click the  that appears by the object and Diagram! will load the linked document into the correct application and display it! Quite a bargain.

A good example of all these features is the Diagram! help system, which is done using these methods. Choose **Help...** from the **Info** menu and follow the instructions. There are also several Diagram! examples in **/LocalLibrary/Diagram_Samples**.

Edit

/NextApps/Edit

Edit is very useful application that acts much like a stripped down word processor. Why would anyone want a stripped down word processor, you might ask? Word processors such as WriteNow and WordPerfect insert lots of extra information dealing with how the document should look, where tab stops are, where the margins are, etc.—not just the text itself. Therefore, if you want to write a program or a file that is going to be used by a UNIX tool such as `lcc` or `grep`, you don't want to deal with extraneous stuff...you just want text, which is what makes Edit so practical.

You might want to move Edit to your dock, since you will probably find yourself using it quite often.

Emergency

What do you do when the machine either stops dead or starts going wild? It is pretty difficult to mess up a machine permanently, so don't worry about that. Just follow these steps, and you should be able to solve most common problems. If you have tried these solutions and the problem still persists, ask a TA if one is around, and if not, call CIT at

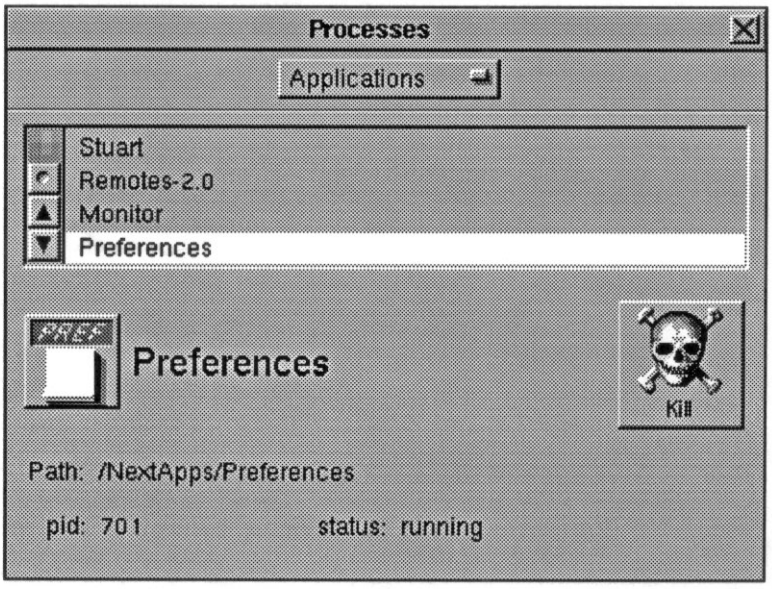
8-6028.

Problem 1:

An application is just sitting there, and the disk wheel keeps spinning. I can still get to the Workspace Manager, but I cannot quit the application since as soon as I click that application the disk wheel starts spinning again.

Solution 1:

Go to the Workspace Manager, and choose **Processes...** from the **Tools** menu. A window will pop up with a list near the top which says, "Background." Switch this to "Applications." It should now look like this:



Good. Find the offending application in the list and select it by clicking on it. Click the kill icon, wait for the dialog box to pop up asking you if you really want to kill the application, and click **OK**. In a few seconds, the application should be dead.

Problem 2:

The machine is totally dead. It will not do anything, and I have a window in the middle of the screen saying, "System Panic." Frankly, I just want to start over again from scratch.

Solution 2:

The easiest way to fix these problems is to reboot the machine (tell it to start over again from scratch). Since you will be holding down three keys to do this, it is commonly known as a "three finger salute" or a "Vulcan neck pinch." To do it, hold down *both* green command keys and press the single back quote key (the upper left key on the keypad). A box will spring up on the middle of the screen. Type `reboot` and press <Return>. It should reboot. If it does not, try this twice more. If it still does not reboot, more drastic action is called for.

Try the following only as a last resort. This sequence will take the battle-axe approach and *really* reboot the machine. Hold down the *left* Alternate and Command keys, then press the asterisk (*) on the upper right of the keypad. The screen should go blank, and the machine should reboot. This will take a while...be patient.

Problem 3:

I am in a Terminal window and a UNIX program I am running will not stop.

Solution 3:

The traditional means of telling a UNIX program to stop executing is to press ctrl-c. (Hold down the control key while pressing "`c`".) If this does not work, you will have to kill the process. This is a bit involved, so follow closely!

- Open another Terminal window by choosing **New** from the **Shell** menu. Type the following:

```
ps -ux
```

- A list of your running processes will appear.
- Find the line with the runaway program's name in the far right column.
- Read the process number in the second column on the same line. For example:

```
ford  4996  0.0  0.0   88    0 pb IW   16:57   0:00 ftp macbeth
```

is a process belonging to `ford` with process number 4996. His `ftp` to macbeth has gone bonkers, so he has looked up this process and intends to `kill` it.

- Knowing the process-id from the second column, you can now type:

```
kill -9 4996
```

- POOF! The offending process is dead. Isn't UNIX violent?

ftp

`ftp` stands for "file transfer protocol," which is just a way of moving files from one place to another. Why would you want to move files to and from different computers? Because there is TONS of information out there that can be obtained for free...if you know how to get it.

The command itself is simple. Type `ftp <address>` in a Terminal shell window. This will connect you to the site named `<address>`. To find a list of `ftp` site addresses, look in the **~cs111/misc** directory for the file **ftplist.txt**. `grep` will be an invaluable aid if you know how to use it to search for a certain subject. Otherwise, just page through the list until you find something that might be interesting.

After giving the `ftp` command, you will soon be given a `login:` prompt; type `anonymous`. After you press <Return> it will tell you what to put for the password (usually your userid). Congrats! You are logged into the site and ready to locate and transfer the files you want. To locate documents, you can use several familiar UNIX commands: `cd`, `ls`, and `ls -F` will all work. Most other commands will not.

Before discussing how to transfer files, you must first understand the `binary` command. It is useful because there are two types of files you can transmit, and each must be handled in a different way. To make a long story short, if the file is a program you can run or it has a ".Z" on the end of its name, you must type `binary` before starting the actual transfer. If you forget, everything will appear fine until you try and use the file, when it will suddenly give you error messages. More later.

`get` is the `ftp` command that does all of the real work. Type `get <filename>` and `ftp` transfers the file to your computer into whatever directory you happened to be in when you gave the `ftp` command. Beware—large files can take quite a while to transfer. Just be thankful that the NeXT is a multitasking machine so you can still work.

To leave the `ftp` site and quit the `ftp` program all in one fell swoop: type `quit`.

What do you do with the file once you have it? If it does not have a .Z or .tar on the end of the filename, it is ready for use. If it does have a .Z, then it must be `uncompress`ed. Type, `uncompress <filename>` and the file will be ready to go. If you forgot to type `binary` before `ftp`ing a file with a .Z on the end, the `uncompress` will fail and tell you that the file is corrupt. Re-`ftp` the file and don't forget to type `binary` before `get`ting the file! If at any point the file has a .tar on the end, type `tar xf <filename>`. This will extract many files from the .tar file, making them usable. Besides the list of `ftp` sites we have already mentioned, we have also listed a few favorites below:

Address:	**Comments:**
`wuarchive.wustl.edu`	HUGE!
`sonata.cc.purdue.edu`	NeXT stuff galore.
`utsun.s.u-tokyo.ac.jp`	Well, it's in Japan!
`princeton.edu`	Rather close to home...

To look for a specific file, or files on a certain subject, you probably do not want to try all the possible sites at random. The `ftp` list will give you a general idea of what is on each site, but this information is sketchy at best. There is another resource you can use called "archie." See the `telnet` CheatSheet for more information, paying special attention to the last paragraph.

Grab

/NextDeveloper/Demos/Grab

Grab has got to be one of the handiest applications on the NeXT. With it, you can copy to the Pasteboard part of a screen image, a window from some application, or the entire screen. Look throughout this lab manual — we used Grab to capture just about every screen graphic we used!

Getting Started

- **Grab** is located in **/NextDeveloper/Demos**. After deciding what it is you want to copy, start up the application.

Choosing What to Grab

- If you choose **Selection...** from the **Grab** menu, the upper-left corner of a rectangle will appear in place of the arrow cursor. You can click-drag over any section of the screen to grab just that part.
- To copy a window, choose **Window** from the **Grab** menu. A camera will appear in the upper-right corner of the screen, covering up the NeXT logo that's normally there. After you find the window you want, click the camera (whose background should turn white), then click the title bar of the desired window.

- If you want to a snapshot of the entire screen, choose **Screen...** from the **Grab** menu. The camera icon will again appear in the upper-right hand corner. Position everything on the screen to your liking, then double-click the camera icon.

Saving Your Selection

- After you have grabbed something, you will be asked where you want to save the image.
- To place a copy onto the Pasteboard (so you can **Paste** it somewhere else), click the **Copy to Pasteboard** button.
- Otherwise, type in a name and save the image as you would save any other file — it will be saved as a "tiff" image.

handin

~cs111/bin/handin

The `handin` *program makes it easy for you to turn in material electronically. With* `handin`*, both you and your TA have copies of your work. So save a tree...*

Handing in a copy of your work

- First, open a Terminal window (see the Terminal CheatSheet if you don't know how).
- Type

  ```
  handin <lab number> <list of files or directories>
  ```

  So, to hand in the files **foo.wn** and **bar.snd** for lab 5, you would type

  ```
  handin 5 foo.wn bar.snd
  ```

  You can hand in entire directories the same way.

  One more thing... read this if `handin` didn't work! `handin` is just a program we wrote to make life easy for you. It's not a normal UNIX command, but just a program we keep in the **~cs111/bin** directory. If you haven't ever run our `addpath` program, you've noticed that `handin` doesn't work for you. That's because UNIX isn't looking in the **~cs111** directory for the `handin` command.

  If you haven't run `addpath` yet, do it now if you want to use `handin`. Type

  ```
  ~cs111/bin/addpath
  ```

  This is the only time you should have to do it. All `addpath` does is add the **~cs111/bin** directory to a list which the computer checks each time you type in a command. You should be able to use `handin` normally, now.

  That's it! `handin` *whisks compressed versions of your files into a directory named after your userid within* ***~cs111****. Your submissions will be kept until the end of the course. If, when the end of the semester rolls around, you want a copy of your entire subdirectory of submissions, follow the instructions below.*

Getting back a copy of your material *(at the end of the semester)*

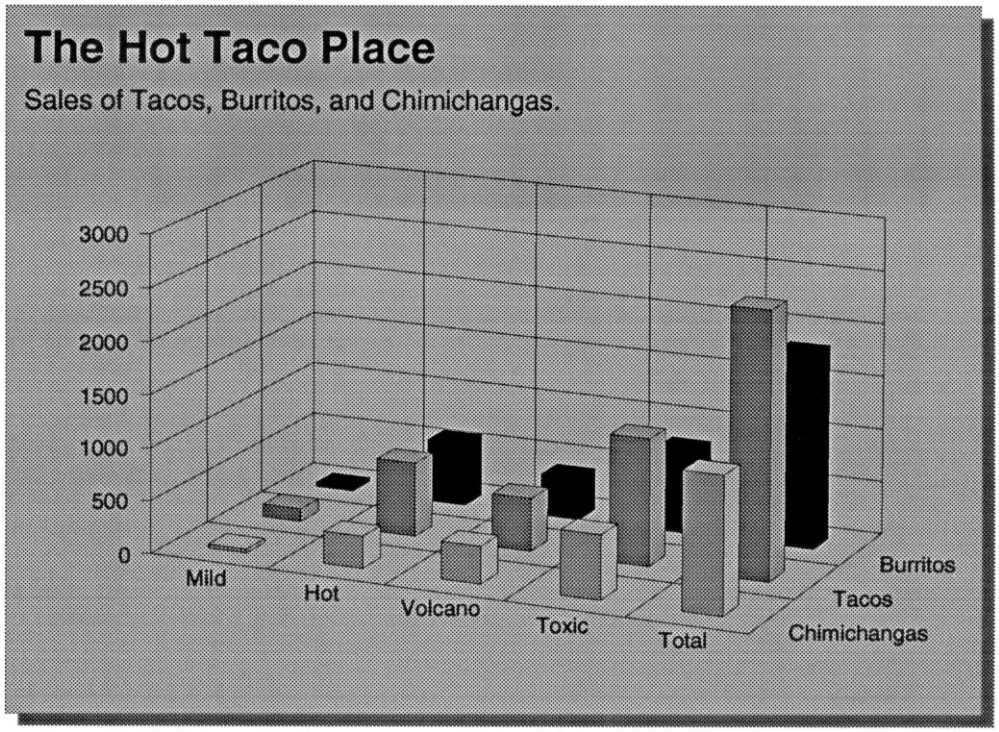
If your TAs are very nice, they will have saved all of your work for the entire semester. Near the end of the course they will give you instructions on how to get a copy of everything you have done, so pay attention if you want it.

Improv

/LocalApps/Improv.app

Introduction

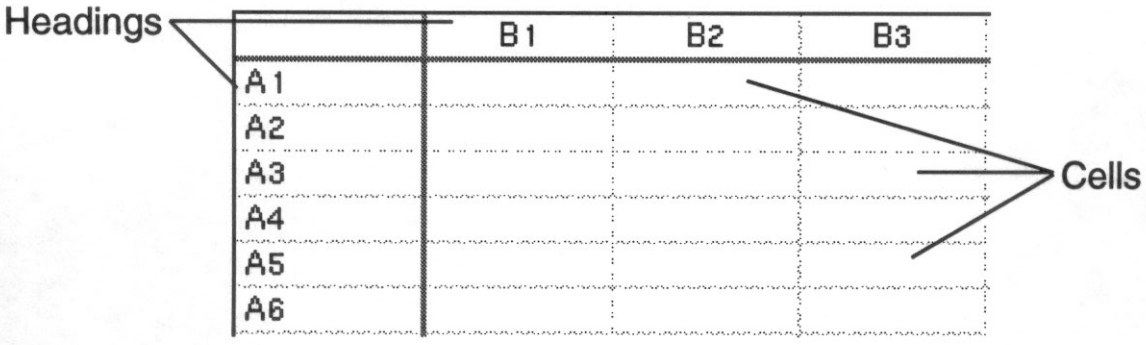
Improv is a new type of spreadsheet that lets you create great-looking graphs and charts in no time at all. If you complete the tutorial in this CheatSheet you will produce the following chart:



A spreadsheet is good at handling large amounts of data that can be represented in a table. It makes life easier for you by doing all of your calculations automatically—rather convenient when you have long columns of numbers to add up.

If you have not used a spreadsheet before, this application can seem quite daunting. But although it is harder to learn at first, the rewards are proportionately larger. Most people have no concept of how graphs of this quality can be produced; hence, when they see them in a paper or report, they are blown away. Once you can do even basic graphs, including them in papers is *extremely* impressive.

Most spreadsheets have this general form:

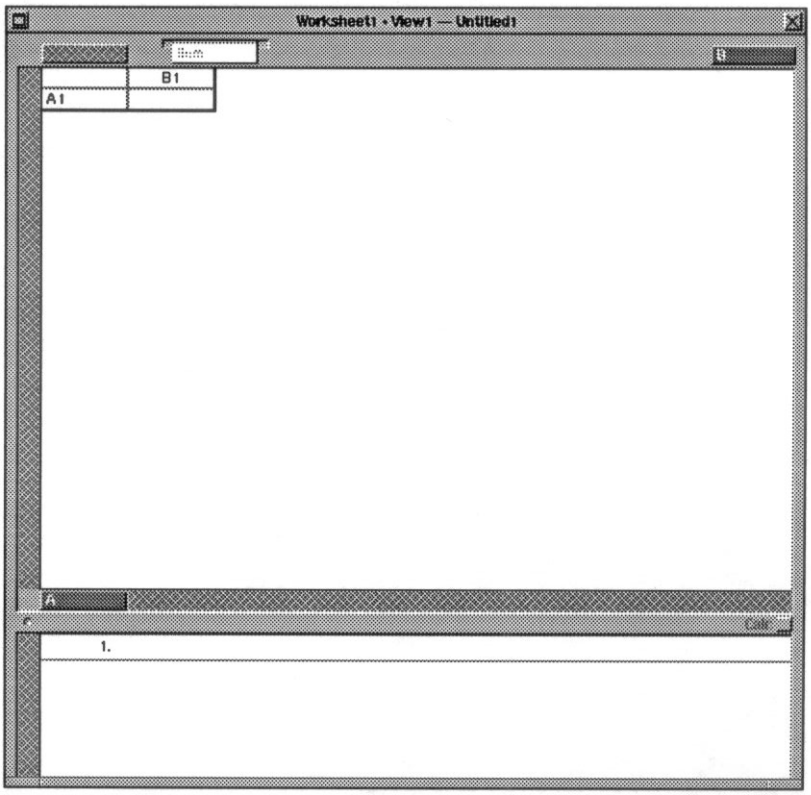


A1 and B3 are hard to remember when they are actually "Profit" and "Gross earnings," so you can rename the headings to something that makes sense. The boxes that make up the bulk of the spreadsheet are called "cells." You can either place data in cells or tell a formula to put something in them. Data are simply numbers that you collect—population data, for example, while formulas calculate a value based on the data that is entered in other cells. Adding up a column of numbers, for example, would require a formula.

If you wish to explore on your own, or want more information on any subject, Improv has its own online help system, accessible by clicking **Context Help...** in the **Info** menu. The **help** window will appear, and you can click the topic you would like help on. There is also an Improv demonstration program located in **/NextDeveloper/Demos** called **ImprovDemo.app**. It's kind of fun to watch even if it isn't all that informative.

How to Create the Sample Graph—A Tutorial

- Start Improv, which is found in **/LocalApps**.
- You will be greeted with this window:



This is a blank template for your data. Since you have not entered anything yet, there is only one column and one row. Of course, more can be added as you need them. You will also notice that the screen is split into two major sections, the large one containing the cell, and the small section at the bottom. This bottom section is where you will enter your formulas. But first let us deal with the columns and rows.

- Rename the column to something useful:
  - Select the column by clicking it. (Clicking the B1 will work nicely.)

- Type "Burritos."

- Add another column by choosing **Add Item** from the **Edit** menu. *Since you already had a column selected, when you chose **Add Item** it created another column. If you had had a row selected, **Add Item** would have created a new row. Remember this for the future... :-)*

- Rename this new column "Tacos."

- Add one more column and name it "Chimichangas."

- You will notice that "Chimichangas" will not fit in the cell. To make this column of cells wider, place the pointer over the vertical line directly after the name. The pointer will change into a left-right arrow. Once it does, drag to the right to make the column wider. Let up on the mouse when the space looks large enough. If you cannot seem to get the arrows to appear, make sure you are directly on the line. The placement must be extremely precise.

- Onward to the rows. Use what you have learned about columns to make these rows: "Mild", "Hot", "Volcano" and "Toxic."

- Add one more row and call it "Total."

Your spreadsheet should look similar to this:

	Burritos	Tacos	Chimichangas
Mild			
Hot			
Volcano			
Toxic			
Total			

You are now ready to enter the data.

- Click the cell you want to put the number into, then type the number. Here is the data as it should appear in your spreadsheet:

	Burritos	Tacos	Chimichangas
Mild	40	120	35
Hot	600	700	300
Volcano	400	500	350
Toxic	800	1200	600
Total			

You could now fill in the "Total" row by hand, but it is easier to let the computer do it. We will need a formula to tell the computer how to total up each column.

- Double-click the formula space at the bottom of the Worksheet window.

- A "1." will appear along with a cursor. Type in:

```
Total = Mild + Hot + Volcano + Toxic
```

Computers are picky beasts. Make sure you have typed this exactly as given, down to the capitalization of each word.

- Press <Return> to tell Improv that you are done entering the formula.
- Instantly, Improv will calculate the `Total` line and fill it in.
- You can change any of the values in the spreadsheet, and Improv will automatically change the Total line to reflect those changes. Try entering a few different values.

Now for the fun part – charting what you have entered.

- Select all of the cells in your spreadsheet by click-dragging over them. (It is easiest to start at one corner and drag to the opposite one.) Your data should now resemble this:

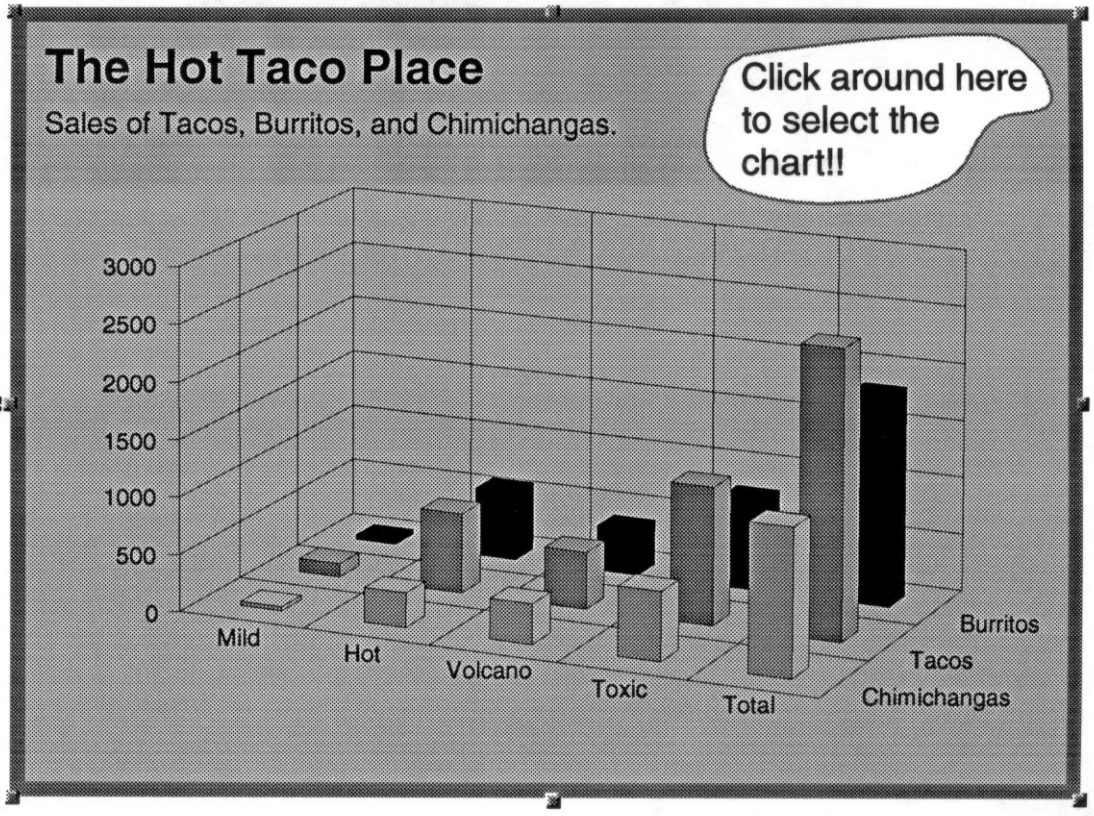
	Burritos	Tacos	Chimichangas
Mild	40	120	35
Hot	600	700	300
Volcano	400	500	350
Toxic	800	1200	600
Total	1840	2520	1285

- Choose **New Graph** from the **Graph** menu.
- The charting companion to Improv, called Presentation Builder, will load and create a generic graph using your selected data.
- Substitute your own text for the generic text "Graph" and "Subtitle of graph." by double-clicking the text, dragging over it, and typing your replacement text.
- Now that the captions are complete, you can choose the type of chart that looks the best to you. Click the **Inspect...** menu item to get a list of graph choices. *Make sure you have the graph selected so the Panel shows the correct information. To select the graph, click near its edge. Small squares should appear at each corner and the middle of each side to show it is the selected item. (See below.)*
- You can click any of the graph choices at the top of the Graph Inspector panel, and the graph will change to that type. IMHO (In my humble opinion) 3D Bar charts are quite impressive, but do whatever looks good to you.
- Our finished graph appears at the top of this CheatSheet. Yours may look very different if you chose a different type of graph. This one happens to be a 3D bar chart.

Using Improv With Other Applications

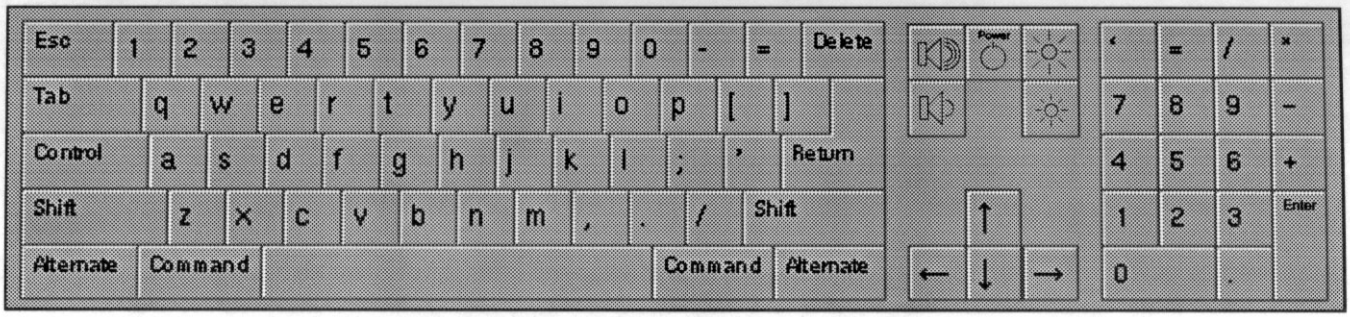
After creating a graph of any sort, you can use standard Copy&Paste to put it into another program such as WriteNow, TopDraw, Diagram!, Mail.app, etc. Select the graph by clicking it on the inside near an edge. Once again the small grey boxes should appear, indicating selection. Choose **Copy** from the **Edit** menu, then move to the application you want to paste the graph into and choose **Paste** from the **Edit** menu. Done!

A square indicating The chart is selected. →



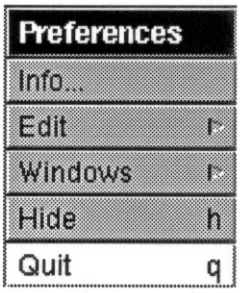
Keyboard

The keyboard. If you don't get anything else out of this class, hopefully you will at least be passingly familiar with one of these. The NeXT keyboard is shown below. You will recognize most of the keys immediately.



The configuration is much like that of a typewriter, with a few extra keys (like Alternate, Command and Control), to tell the computer to do special things. For example, if you are in WriteNow and you need to type an accented "a" (á), first type Alternate-e (press and hold down the Alternate key and press the "e" key) then press "a". To find out about other snazzy symbols, experiment with other Alternate-<key> combinations, or try the Preferences application (refer to the CheatSheet for more information)

The Command key is useful for keyboard shortcuts to menu commands. If you see a letter next to a menu item, you can hold down the Command key and press the letter to choose that item. For example, in the following menu, Command-q will quit the Preferences application.



Besides Alternate and Command, there is also a Control key. This is less commonly used in the NeXT environment than in a UNIX shell. See the Emergency CheatSheet for the use of Control-c, one of the most common control key sequences.

The arrow keys (the keys with the arrows on them!) can be used to move the cursor in some applications, most notably word processors.

Above the arrow keys, there are five keys which have special meanings. The center key is the power key, used (obviously) for turning the machine on and off. The left two keys are the volume controls. The top one will increase the volume, and the bottom one will decrease it. Holding down Command and pressing the bottom one turns the speaker mute on and off. You can control the screen brightness with the right set of buttons (the ones that look like suns). The top one will brighten the screen; the bottom one will dim it.

Librarian

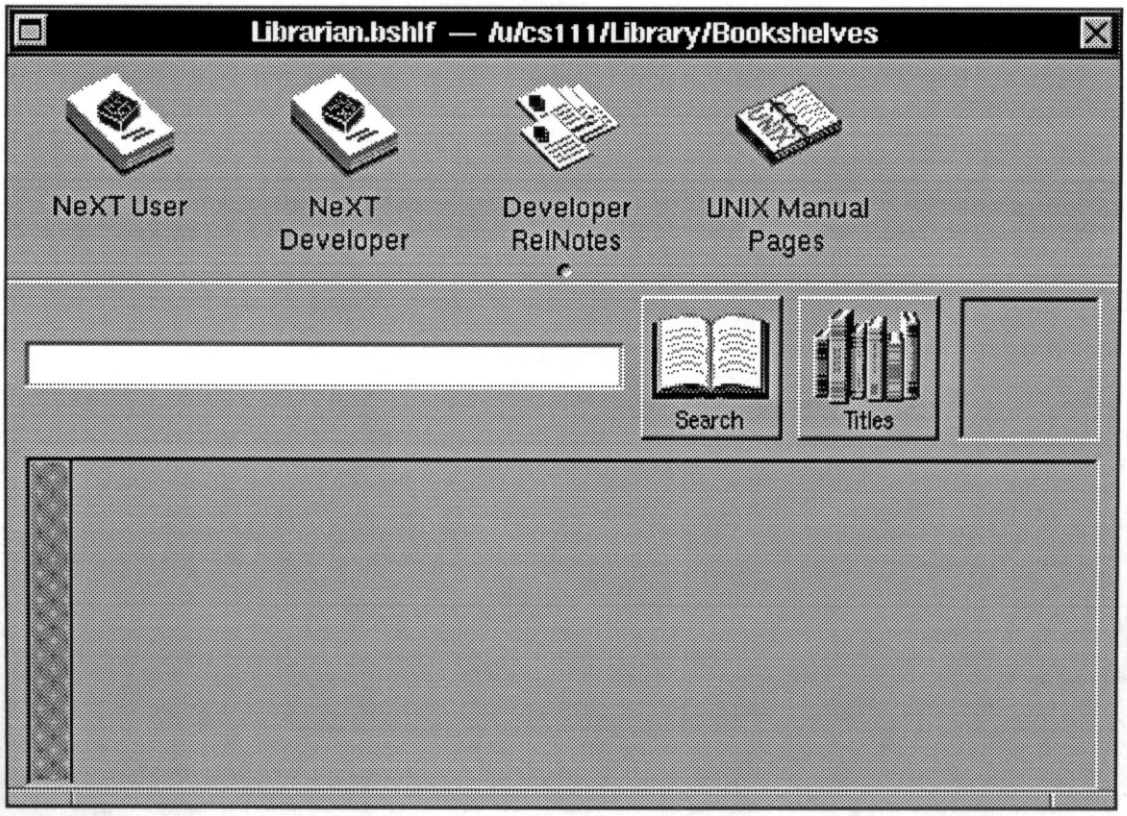
/NextApps/Librarian.app

Using a computer to search for information saves you from tedious manual searching. UNIX tools let you get very close to your data and manipulate it in many different ways, but there is also an application for the NeXT that allows you to search files using a graphical interface. This application, called Librarian, goes a step further than UNIX tools such as `grep` *by allowing you to build an "index" of words you are likely to search for in the future. By making this list once then using it over and over, Librarian saves itself the trouble of re-searching the files every time you look something up. It searches once, remembers where each word was found, and spits the results back out whenever you ask for them.*

Unfortunately, there is a disadvantage to this scheme. If Librarian were to index every single word it came across, it would generate huge files to just hold the index. To limit the amount of data it generates, Librarian chooses which words it will index, leaving out words that are likely to be very common. For example, the words "the," "and" and any single letters are left out. These would be silly to index anyway since 99% of text documents include them somewhere. The sad part about indexing is that it does miss some words you might want to search for. More on this later. First, let's learn how to do searches.

Searching

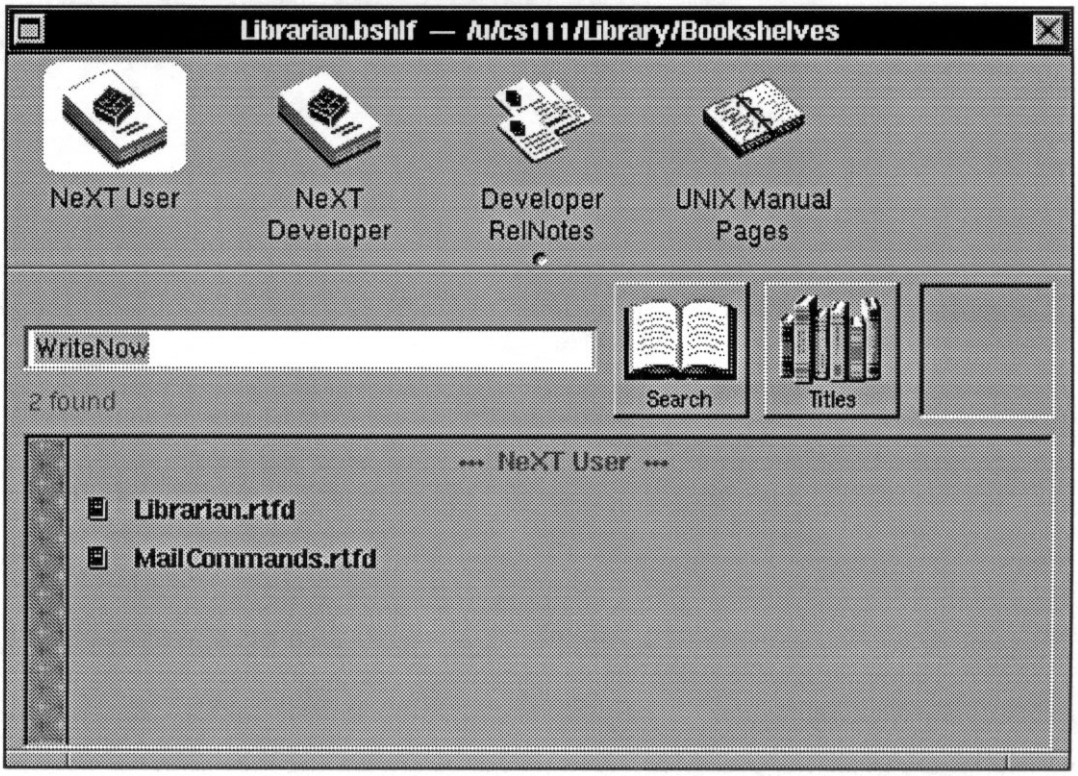
When you start Librarian (in **/NextApps**), you will be faced with this window:



The four icons across the top of the window are the directories or files you are able

to search, represented by icons. Each icon is referred to as a "target." The group of all the targets that appear is called a "bookshelf." Anyway, let's say that we want to find out more about WriteNow. We would follow these steps to obtain the information we need:

- Click the NeXT User icon, since the material on WriteNow is likely to appear in this target. (WriteNow is a User application.) The icon should now be highlighted.
- Type the word we want to search for into the text field on the left of the **Search** button. In this case, we type "WriteNow."
- Click the **Search** button.
- In a few moments, a list of documents will appear in the lower part of the window, like this:



- Librarian found two documents containing the word "WriteNow." To see one of the documents, just double-click its name. Librarian will load it in so you can read it. There! You have successfully completed a search!

Searching Multiple Targets

If you want to search in more than one file and/or directory simultaneously you can easily do so. Simply shift-click all the icons you want to use and follow the rest of the procedure exactly as above.

Searching Targets Not Shown in the Bookshelf

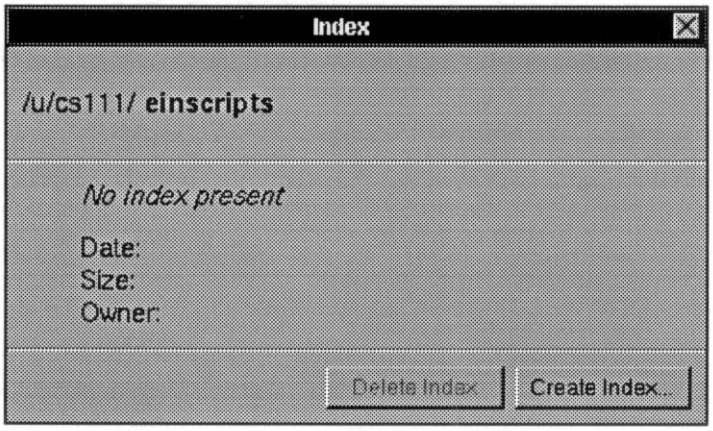
If you want to search a directory or document that is not shown at the top of the

window, you can drag its icon into Librarian from the WorkSpace. It works very much like the shelf in the File Viewer—just drag until the icon shows up lightly near the top of the Librarian.bshlf window, then release it. Select it by clicking it, and it is ready to be searched.

Indexing Your Own Targets

There may be only one small problem with dragging your own targets into the bookshelf area—they are probably not indexed. Look directly above the text field. If it says "Unindexed target," well, the target is not indexed. The search will work, but it will be slower than an indexed search. If you wish to index the directory, you must do the following:

- Make sure you have write permission in the directory. (If you don't, consult the UNIX CheatSheet to learn about soft links, then create a soft link to the directory you want to index and index your link...definitely not for the faint of heart.)
- Select the target you want to index by clicking it, then choose **Index...** from the **Target** menu. A window will appear:



- Click **Create Index...** and wait a few moments while the indexing takes place. Done!

Now you have a bookshelf of your very own. You can save this bookshelf for future use by selecting **Save** or **Save As...** from the **Bookshelf** menu. When you double-click the file in the Workspace, it will automatically start Librarian and open your custom bookshelf. Great!

Another alternative to indexing new targets is just leaving them unindexed and enduring a slower search; this is not all bad, and even offers a few advantages, such as better ways of describing what to search for, not missing words that aren't indexed, and several others. I will not go into detail here, but point you toward the NeXT Applications manual, which resides in the manual rack near the door of the NeXT cluster. Check out pages 85-100, which cover the entire Digital Librarian application in all of its splendor and glory.

mail

Note: the man page for `mail` on the NeXT is quite well done,
so for more information, type `man mail` in a NeXT terminal session.

`mail` is one of the most useful computer tools available here on campus. Using it, you can send messages to people throughout the world. Most universities are hooked up to the same major network that we are (the Internet) so you can even send mail to friends at other colleges for free!

`mail` is also very easy to use. Say you want to send mail to your friend Fred Lemonjello who also goes to Princeton:

- The first step is to find out his userid. The easiest way is to just ask him. Let's say his userid turns out to be `fredlemo`.
- Armed with this information, you can sit down at a computer, get to a UNIX shell (see the Terminal CheatSheet), type `mail fredlemo` and hit the <Return> key.
- The `mail` program will ask you for a subject. Enter a short phrase describing the content of the mail you are about to type.
- After pressing <Return> you can begin typing the body of the message. Just type away.
- When you are finished, hold down the control key and press "d" to send the mail. (This operation is usually written as Ctrl-d.)

The best way to illustrate `mail` is to show a simple example. (Note—anything between <> and in this "non-computer" font is a comment):

```
phoenix% mail djdobson
Subject: Your life is scheduled for termination.

I am sorry to inform you that Lord Vader has decided your flaming red hair
is a disgrace to the Empire.  Hence, you are scheduled for termination
at 0500 hours in Detention Block A6.

Thank you for your time.

                                --Vice Admiral Jeff Blum

EOT   <press control-D, do not type EOT to send the mail!  The computer types the EOT.>
phoenix%
```

Soon djdobson will log in and the system will tell him that he has mail. To read it, he will type `mail` and press <Return>. Here is a transcript of what his session would look like: [Note: the "&" is the `mail` prompt, so everything after an & is what djdobson typed.]

```
phoenix%  mail
Mail version SMI 4.0 Sat Oct 13 19:57:42 PDT 1990  Type ? for help.
"/usr/spool/mail/djdobson": 1 message 1 new
>N  1 jeffblum           Fri Jun 28 14:38   16/311   Your life is scheduled...
& 1      <display message number one>
Message  1:
```

```
From jeffblum Fri Jun 28 14:38:20 1991
Received: by phoenix.Princeton.EDU (4.1/1.110)
        id AA28102; Fri, 28 Jun 91 14:38:19 EDT
Date: Fri, 28 Jun 91 14:38:19 EDT
From: Jeffrey R Blum <jeffblum>
Message-Id: <9106281838.AA28102@phoenix.Princeton.EDU>
To: djdobson
Subject: Your life is scheduled for termination.
Status: R

I am sorry to inform you that Lord Vader has decided your flaming red hair
is a disgrace to the Empire.  Hence, you are scheduled for termination
at 0500 hours in Detention Block A6.

Thank you for your time.

                                    --Vice Admiral Jeff Blum

& r
```

<r means reply to the message that was just read>

```
To: jeffblum
Subject: Re:  Your life is scheduled for termination.

You sir, are a hoser.

-Wolff

EOT
& q
phoenix%
```

Several commands were used in this example. Here is a brief list of useful ones you can use use once inside the mail program, ie. when you see the "&" prompt. Parts of this are taken from the NeXT man page for mail, so look there for more commands. The letters in parentheses are abbreviations you can use for the command.

Anywhere you use a message-number, you can use a list of message-numbers, separated by spaces.

reply(r)	Takes a message (or message list) and starts a letter addressed to the original sender.
delete(d)	Takes a list of messages and deletes them.
help	fairly intuitive...
save(s)	Takes a message list and a filename and appends each message in turn to the end of the file.
<Return>	Read the next message addressed to you.
<Space>	Scroll the message you're reading down one screen.

example: `r 2` will begin a letter to whoever wrote you message number 2.
example: `s 5 foo.txt` saves message 5 as **foo.txt**.

One other thing to note about `mail` is that you must press <Return> at the end of each line. If you continue typing past the edge of the window, your words will be broken up and look quite poor. Eventually, after about 250 characters, `mail` will not let you type until you either backspace or press <Return>. If you don't touch-type, look up at the screen every few words and you can easily avoid this problem.

A (brief) summary for those who are still confused:

To send mail

- Type `mail <userid>`.
- Enter a subject.
- Type in your message
- When you are finished, hit Control-d to send the mail.

To read mail

- Type `mail` and after the "&" appears, hit the <Return> key.
- Hit <Space> to scroll the message you are reading.
- Press <Return> at the "&" prompt to read the next message.
- When you are done reading and see the "&" prompt again press `q` to exit `mail`.
- Done.

News

News is a medium that allows people to exchange messages with each other on hundreds of different topics. To make this information easier to find, the topics are separated into different groups, which are in turn organized into different categories. The main categories are `alt` *(alternative),* `rec` *(recreation),* `comp` *(computing), and* `misc` *(miscellaneous). There are many others, but these few seem to attract the most interest. Princeton receives almost every possible newsgroup, so you can explore to the limits of netnews.*

To start the newsreader, type `rn` while in a Terminal session. The first time you do this it will set up news for you, which can take a little while. Be patient. After a short introduction to the net, the names of the first 5 or so newsgroups in your list of newsgroups will appear, along with how many articles are unread. Initially, you are subscribed to *all* of the newsgroups—over *900* of them. You will have to press `u` *many* times to get rid of the groups you are not interested in. For now, you can just use the searching commands listed below to get to the groups you want to read.

There are two basic modes in `rn`. One is listing the various groups; the other is reading articles within a group. The first mode looks like this:

```
********   3 unread articles in princeton.china--read now? [ynq]
******** 38 unread articles in princeton.forsale--read now? [ynq]
********   9 unread articles in princeton.general--read now? [ynq]
********   1 unread article  in princeton.grad--read now? [ynq]
******** 18 unread articles in princeton.wanted--read now? [ynq]
```

The second mode is effective when you are actually reading articles. It is easily recognizable since you will see the text of the article.

Commands

In the following command list, the first effect of the command is for when `rn` is listing the groups, and the second is effective when reading articles.

Command	Group Mode	Article Mode
`y`	switches to article mode	scrolls forward in article or goes to next article
`n`	moves to the next group	skips to next article.
`q`	exits `rn`	exits the current message or group
`<space>`	same as `y`	same as `y`
`u`	unsubscribe from the current newsgroup	(both modes)
`g <string>`	go to group <string>	search for <string> in current article
`h` (subtle hint)	group mode help screen	article mode help screen
`/<string>`	search for <string> in a newsgroup name, starting with current group.	Scan forward for article containing <string> in the subject.

Organizing Your Directory

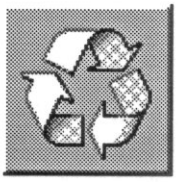
One of the most annoying things on a computer is having so many files in one place that you have to scan through a huge list to find what you're looking for. Because of their built-in directory structure, UNIX computers make it quite easy to keep things clean and allow easy access to the files you are looking for.

Suppose that you have several files from different places, such as a few letters from friends, several files from cs111, and two graphics images. The logical thing to do would be to have one directory called **letters**, another one called **cs111**, and maybe a third called **images**. Underneath the **cs111** directory, you might want to even have separate directories for each lab. To make a new directory:

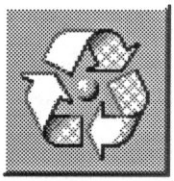
- Go to the directory in which you want to create your new directories
- Move to the Workspace Menu and choose **New Folder** from the File menu. This will place a folder called, appropriately enough, **New Folder**, in your directory.
- You can double-click the name portion of the folder icon in the icon path and type a new name for it, such as 'letters'.
- Drag the new folder icon up to the shelf.
- Move your letters to the new directory by using the mouse and dragging each letter on top of the folder icon. When you do this, the folder will 'open' up, so just drop the file (release the mouse button) and you are finished.
- After finished moving your files, you can keep your directory on the shelf, or remove it, your choice.

You can repeat the above process for the **cs111** and **images** folders.

After a while, you might find that you have a lot of files that you don't need anymore. Whenever you want to delete a file, just drag it over to the Recycler, the icon on the lower right hand of the screen.



Empty Recycler



Full Recycler

After you put something into the Recycler, a dot will appear in the center signalling that you have something there. Should you have second thoughts about getting rid of the item(s), you can double click the Recycler icon and scroll through the files and folders you were going to get rid of. If you want to get anything back, just grab it out of the Recycler and drag it back up to the directory you want. If you decide that you would rather not keep the files, go to **File** under the Workspace menu and choose **Empty Recycler**. (If you accidentally delete a file, empty the Recycler, and then realize that you desperately needed that file, all is not lost. *Quickly* write e-mail to lrr@phoenix.princeton.edu asking for a file restore.)

PostScript

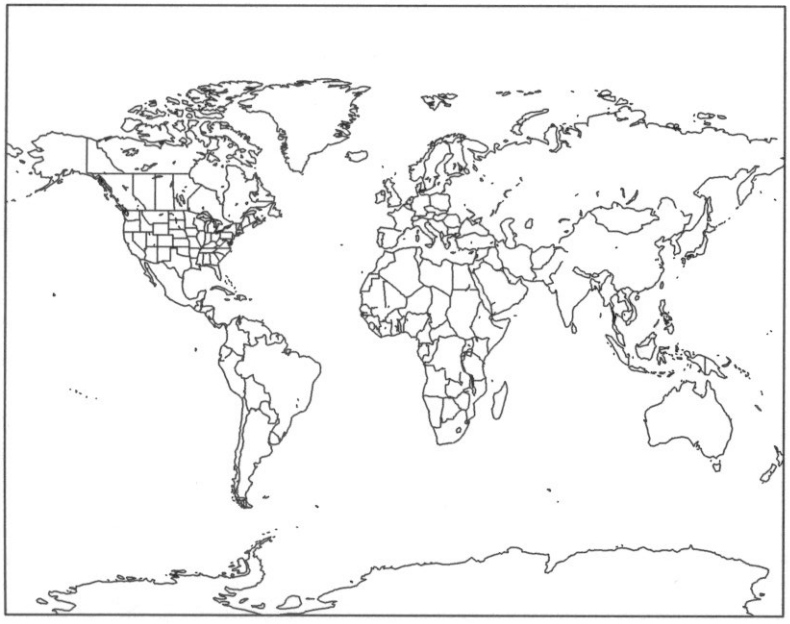
What Is It?

PostScript is a language used to describe how a picture looks. Some computers describe an image by simply recording all the dots that make it up. This is still used in many places, but PostScript takes image description one step further. With PostScript, you can ask for an object such as a circle of a certain radius, filled with a certain color, and it will understand what you mean.

Why Use It?

A computer can only perform actions that it has all the facts for. If a computer merely manipulates dots, all it knows is where those dots are. If these dots are one-hundredth of an inch on a side (or 100 dots per inch) then when it is printed on a printer that can print at 400dpi (dots per inch), each of the dots in the picture must be made up of *16* printer dots. This is bad since the resulting image is just as coarse as it would be on a 100dpi printer. Result: the image appears jagged instead of smooth even though the printer can print at much higher quality.

PostScript removes this restriction by not describing images as a bunch of dots. Instead, it describes objects—circles, squares and lines, for example. Therefore, although an image will appear at 72dpi on your screen, when you print it, the printer knows enough to print it as smoothly as it possibly can—400dpi.



A Postscript image.

A TIFF image
(non-Postscript).

Note: The TIFF is blown up to about 2X it's normal size.

This is why high quality laser printers use the PostScript language—they can print at their maximum resolution. In fact, most typesetters (upwards of 2400dpi!) also use PostScript since it is so powerful.

Why do I need to know all of this?

Normally (whether you realize it or not) when you create a drawing on a computer you use a paint program that lets you draw objects, then translates them into PostScript for you. If you like, however, you can simply create the PostScript directly, and never touch a drawing program again. The rest of this CheatSheet will show you the basics of the language and teach you how to draw pictures using it.

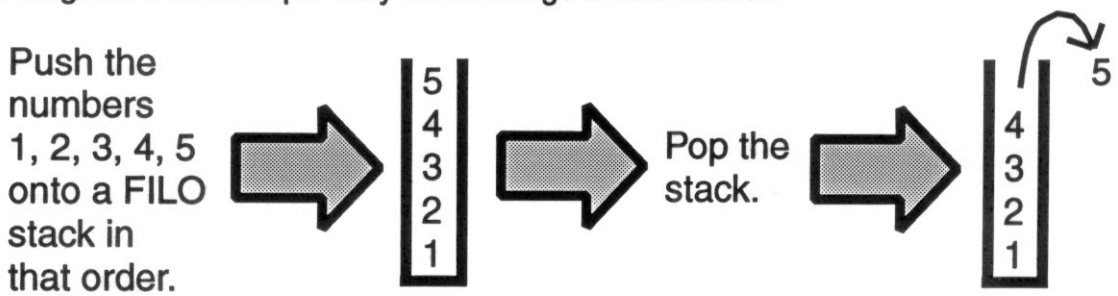
One quick note: PostScript is most commonly used for printing images on paper. This means that if you want to play with it, you normally have to print out the image to see what it looks like. Fortunately, the NeXT has an application that lets you type in a postscript program and view it directly on the screen. To use it, start **/NextDeveloper/Demos/Yap** and wait for a window to appear in the upper right-hand corner. For further information about using Yap, consult the Yap CheatSheet.

The Important Stuff

PostScript is what is known as a stack-based language. A stack is a method for storing data in an efficient manner. This method shows up again and again in Computer Science in many different applications.

The basic idea of a stack is this: You have a bunch of data that you want the computer to remember, then access later. How do you store this data? You can think of a stack as a large hopper that you can toss the data into. When you want to access the data, you simply reach in and pull out the piece that is on top. Hence, the first piece of data you put into the stack will be the last one to come out. Placing a piece of data onto the stack is called a "push", and pulling it off is called a "pop."

A diagrammed example may make things a little clearer:



In order to use PostScript, you must always keep stacks in mind. For example, if you want to draw a line from the point (1,5) to (10,15) you would use the following PostScript code:

```
1 5 moveto
10 15 lineto
stroke
```

If we trace this program step-by-step, we get the following:

1. Push the number 1 onto the stack.
2. Push 5 onto the stack.
3. Call the `moveto` routine, which pops one number off the stack for the y-coordinate then another for the x-coordinate, leaving the stack empty. (1,5) is now the current point.

4. Push 10 on the stack.
5. Push 15 on the stack.
6. Call the `lineto` routine, which pops the 15 off the stack for the y-coordinate, and the 10 as the x-coordinate. The stack is empty, and there is an invisible line drawn from (1,5) to (10,15).
7. `stroke` paints this invisible line black.

Easy enough? It should now make sense why the coordinates are given before the commands (known as operators)—it is because the operators must get the data they need off of the stack. *Every PostScript operator that needs data pops it off of the stack, so remember to put it there!*

You are now ready to learn about the different commands you can use. The format will show what must be on the stack when you call the operator. To put something on the stack, just type it—there is no command to place an item on the stack. A fine line lies in the fact that operators are not normally placed on the stack. They operate on data...so they take data *off* of the stack as they need it. Only in specific cases do operators get placed onto the stack.

Since we are dealing with simple graphics here, we are leaving out many of the built–in PostScript operators. To give you an idea of the entire operator set, we have enclosed a *PostScript Reference Sheet* which lists all of the PostScript commands known to mankind. Enjoy!

Generalities

- Remember that PostScript is stack based! Put the data *before* the operator!
- Any path you make will be invisible until you give a `stroke` command.
- Stacks can rapidly become terribly confusing. Always keep track of what you have put onto the stack, and make sure you know what your operators are putting there as well...

The commands

Note: This format is taken from the *PostScript Language Reference Manual* by Adobe Systems. In this book there is a list of ALL PostScript commands in this same general format. The first column is what you need to place on top of the stack before the operator is called. (The element furthest to the right is on top of the stack.) The next column is the operator, and the third column is the stack after the operator. (A "—" means no elements.) The final column is a short description of the operator. "Bool" stands for boolean, either the number 1 (true) or 0 (false). "proc" stands for procedure, or a block of commands that can be executed. The syntax for creating a block will be covered in Example 4.

- **Drawing operators:**

#1 #2	**moveto**	—	makes the point (#1, #2) current
#1 #2	**lineto**	—	Draws an invisible line from the current point to (#1, #2)
#1 #2	**rlineto**	—	Like **lineto** but current point is treated as (0,0)
—	**stroke**	—	paints an invisible path black
—	**fill**	—	fills an object with black
—	**newpath**	—	makes sure there are no current objects
#1 #2 #3 #4 #5	**arc**	—	makes an invisible arc of radius #3 at point (#1,#2) over the angle from #4 to #5 degrees

- **Stack operators:**

#1	**pop**	—	throws away the top item on the stack
#1 #2	**exch**	#2 #1	reverses the top two elements on the stack
#1	**dup**	#1 #1	puts two of the top element onto the stack

- **Arithmetic operators:**

#1 #2	**mul**	#1*#2	places the product of #1 and #2 on the stack
#1 #2	**add**	#1+#2	places the sum of #1 and #2 on the stack
#1 #2	**div**	#1/#2	places the quotient of #1 and #2 on the stack
#1 #2	**sub**	#1-#2	places the difference of #1 and #2 on the stack
#1	**neg**	-#1	reverses the sign of #1

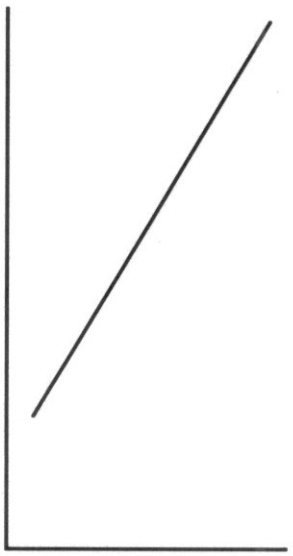
- **Control operators:** (Example #4 will be a tremendous help here)

#1 #2	**eq**	bool	test if #1 & #2 are equal
#1 #2	**ne**	bool	test if #1 & #2 are not equal
#1 #2	**gt**	bool	true if #1 is greater than #2
#1 #2	**ge**	bool	true if #1 is less than or equal to #2
#1 #2	**lt**	bool	true if #1 is less than #2
#1 #2	**le**	bool	true if #1 is less than or equal to #2
—	**true**	bool	push true onto the stack
—	**false**	bool	push false onto the stack
bool proc	**if**	—	execute proc if bool is true
#1 proc	**repeat**	—	execute proc #1 times

Examples

Example 1

`moveto, lineto, stroke, showpage`



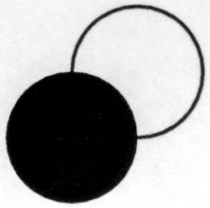
Note: The axis is drawn merely to show the coordinate system the line is drawn in. It are not produced by the PostScript code shown.

```
%!                  % standard beginning of a PostScript file
10 50 moveto        % make point (10,50) current
100 200 lineto      % draw an invisible line from the current point to (100,200)
stroke              % paint line black
showpage            % show this page
```

When drawing your picture, PostScript will ignore anything after a "%" on a line. This is called a "comment" and simply makes the code easier for us humans to read.

Example 2

rlineto, fill, arc



```
%!
100 150 25 0 360 arc        % draw an invisible circle radius 25 at (100,150)
stroke                      % paint the circle black
75 125 25 0 360 arc         % draw an invisible circle radius 25 at (75,125)
fill                        % fill this circle with black

40 dup moveto               % put 40 on the stack, duplicate it, moveto (40,40)
0 100 exch rlineto          % make line to (100,0)  (points are exchanged!)
stroke                      % paint this line black

showpage                    % show this page
```

Example 3

Shows a neat quality of stack-based languages—you can build up the stack and *then* call all of the operators...even though this is not good programming practice since it is very difficult for humans to read.

```
%!

250 250 100 0 360
210 290 20 0 360
210 285 8 0 360
290 290 20 0 360
290 285 8 0 360
250 250 75 190 -10

arc stroke newpath
arc fill newpath
arc stroke newpath
arc fill newpath
arc stroke newpath
arc stroke newpath

showpage
```

Just keep the stack in your mind, and you will get the idea of what is happening.

Example 4

Conditional statements and looping (`repeat`, `if`, etc.), procedures

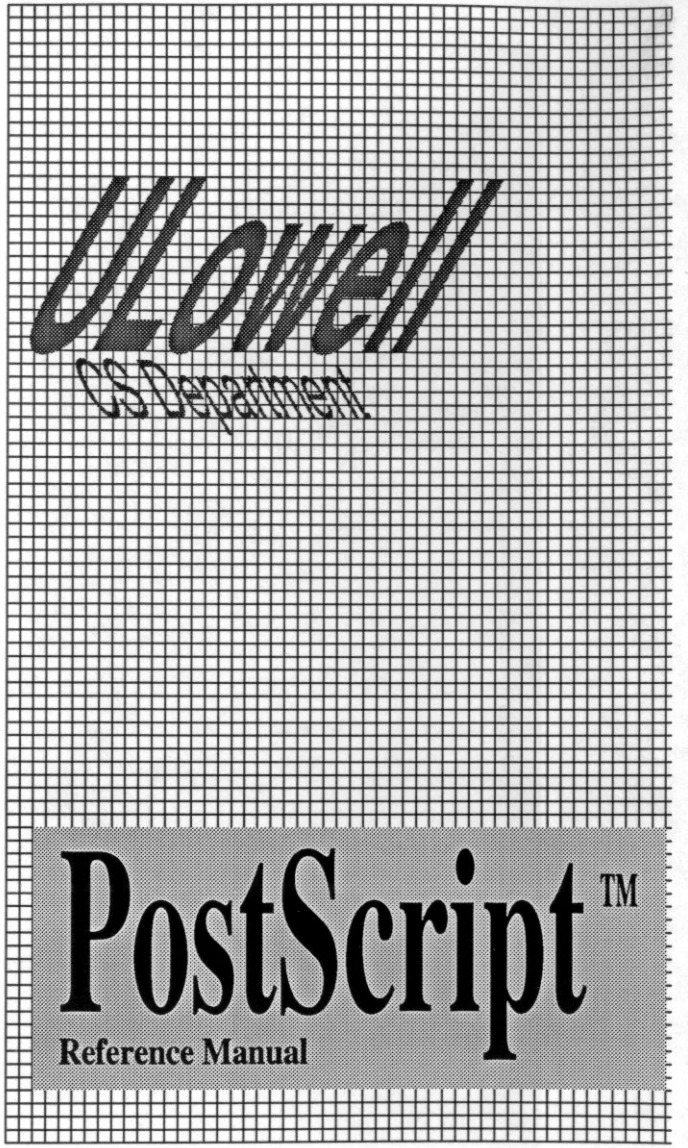
```
%!
/line { moveto 0 25 rlineto stroke } def      % define line to draw a vertical line 25
                                              % points tall starting at a point on the stack.
/variable 10 def                              % variable = 10

10 {                                          % place 10 on the stack for the repeat
    4 variable le {                           % check if 4 is less then variable
        variable 10 mul 10 line               % draw line at (variable*10, 10)
    } if                                      % perform previous step only if
                                              % 4 variable le was true.
    /variable variable 1 sub def              % subtract 1 from variable
} repeat                                      % repeat the loop (only 10 times)
```

This is a rather simple program that shows looping and conditionals. The picture is stupid, but the number of lines drawn is significant, since each time the program goes through the loop, it draws one. Trace through this code and make sure you understand why it is drawing 7 lines.

This program also uses the `def` operator several times. All that the first line of the program does is say that wherever the program sees the word `line` it should act like `moveto 0 25 rlineto stroke` is there instead. Likewise, wherever the program sees `variable`, it should act like there is a `10` in its place. Notice also that `variable` is redefined later in the program; you can define `variable` by saying "subtract one from whatever `variable` currently is, then make `variable` equal to that." In short, subtract one from `variable`. `def` is an extremely useful operator—well worth the time spent to understand it.

print	string >> --
prompt	-- >> --
pstack	a..b..c.. >> --
put	array index value >> --
putinterval	arry1 beg arry2 >> arry1
quit	-- >> --
rand	-- >> int
rcheck	array >> bool (true if readable)
rcurveto	dx0 dy0 dx1 dy1 dx2 dy2 >> --
read	file >> byte bool (false if EOF)
readhexstring	file string >> substring bool
readline	file string >> substring bool
readonly	array >> ReadOnly-array
readstring	file string >> substr bool (false if EOF)
repeat	count proc >> --
restore	save-objct >> --
reversepath	-- >> --
rlineto	dx dy >> --
rmoveto	dx dy >> --
roll	a..b..c.. N R >> a..b..c.. (top N elems rolled by R)
rotate	angle >> -- (or, angle mtrx >> mtrx)
round	num >> num-rounded
rrand	-- >> current-random-nr-seed-state
run	string >> --
save	-- >> save-object
scale	sx sy >> -- or sx sy mtrx >> mtrx
scalefont	font-dict number >> transformed-font-dict
search	string
setcachedevice	wx wy llx lly urx ury >> --
setcachelimit	maxbytes >> --
setcharwidth	wx wy >> --
setdash	array offset >> --
setflat	num >> --
setfont	font-dict >> --
setgray	num >> --
sethsbcolor	hue satur bright >> --
setlinecap	integer >> --
setlinejoin	integer >> --
setlinewidth	num >> --
setmatrix	matrix >> --
setmiterlimit	num >> --
setrgbcolor	red green blue >> --
setscreen	freq rotation spot-function >> --
settransfer	gray-transfer-funct >> --
show	string >> --
showpage	-- >> --

sin	num >> sine(num)
sqrt	num >> square-root-of-num
srand	int >> --
stack	a..b..c.. >> a..b..c..
start	-- >> --
status	file >> bool (true if open)
stop	-- >> --
stopped	a >> bool (false if a was terminated normaly)
store	key value >> --
string	int >> string
stringwidth	string >> wx wy
stroke	-- >> --
strokepath	-- >> --
sub	num1 num2 >> num1-num2
systemdict	-- >> system-dict
token	file >> bool (true if found)
token	string >> if found: s-post token true not found: false
transform	x y >> xt xy or x y mtrx >> xt yt
translate	tx ty >> -- or tx ty mtrx >> mtrx
true	-- >> true
truncate	num >> num-truncated
type	a >> type-name-of-a
userdict	-- >> user-dict
usertime	-- >> time-in-msecs
version	-- >> soft-&-hard-version-string
vmstatus	-- >> level-of-save bytes-used total-bytes-avail
wcheck	array >> bool (if writeable: true)
where	key >> if found: dict true not found: false
widthshow	dx dy char-code string >> --
write	file byte >> --
writehexstring	file strig >> --
writestring	file string >> --
xcheck	a >> bool (true if a is executable)
xor	a b >> aXORb (bitwise if a,b are integers)
=	a..b..c.. >> --
==	a..b..c.. >> --



FOLD ALONG THIS LINE

CUT ALONG THIS LINE

FOLD ALONG THIS LINE

abs num >> lnuml
add num1 num2 >> (num1+num2)
aload array >> elem1..elem2.. array
anchorsearch string seek >> found: spost smatch true
not found: string false
and a b >> aANDb (bitwise if a,b are integers)
arc x y r ang1 ang2 >> --
arcn x y r ang1 ang2 >> --
arcto x1 y1 x2 y2 r >> xt1 yt1 xt2 yt2
array int >> arry-of-size-int
ashow ax ay string >> --
astore elem1..elem2.. arry-size >> arry[elem1..elem2]
atan a b >> angle-whose-tang-is-(a/b)
awidthshow ax ay string >> --
begin dict >> --
bitshift int shift >> int-shifted (right: +, left: -)
bytesavailable file >> int (-1 if cannot be determ)
cachestatus -- >> bsize bmax msize mmax csize cmax maxbits
ceiling number >> least-integ-grtr-than-or-eq-to
charpath string strokepath-bool >> --
clear a..b..c.. >> ....
cleartomark mark a..b..c.. >> --
clip -- >> --
clippath -- >> --
closefile file >> --
closepath -- >> --
concat matrix >> --
concatmatrix mtrx1 mtrx2 mtrx3 >> mtrx3 (=mtrx1*mtrx2)
copy a..b..c.. int >> a..b..c.. a..b..c.. (top -int- elem)
copypage -- >> --
cos a >> cosine(a)
count a..b..c.. >> a..b..c..count
countdictstack -- >> count
countexecstack -- >> count
counttomark mark a..b..c.. >> mark a..b..c..count
currentdash -- >> array offset
currentdict -- >> dict
currentfile -- >> file
currentflat -- >> number
currentfont -- >> font-dict
currentgray -- >> number
currenthsbcolor -- >> hue satur bright
currentlinecap -- >> integer
currentlinejoin -- >> integer
currentlinewidth -- >> number
currentmatrix matrix >> CTM-matrix

curentmiterlimit -- >> number
currentpoint -- >> x y
currentrgbcolor -- >> red green blue
currentscreen -- >> freq rot spot-funct
currenttransfer -- >> gray-tansf-funct
curveto x0 y0 x1 y1 x2 y2 >> --
cvi num >> integ or strng >> int
cvlit a >> literal (not-exec)
cvn string >> name
cvr num >> real
cvrs num base string >> substring
cvs a string >> substring
cvx a >> executable
def key value >> --
defaultmatrix matrix >> def-matrix
definefont key dict >> font-dict
dict int >> dict (maximum-capacity: int)
dictstack array >> subarray
div num1 num2 >> (num1/num2)
dtransform xd yd >> xdt ydt
or xd yd matrix >> xdt ydt
dup a >> a a
echo bool >> --
end -- >> --
eoclip -- >> --
eofill -- >> --
eq a b >> bool (true if a=b
erasepage -- >> --
exch a b >> b a
exec a >> --
execstack array >> subarray
executeonly arry >> exec-only-arry (or string)
exit -- >> --
exp num1 num2 >> num1-to-the-num2-pwr
false -- >> false
file string1 string2 >> file (str2: r, w)
fill -- >> --
findfont key >> font-dict
flattenpath -- >> --
floor number >> greatest-int-less-than-or-eq-to
flush -- >> --
flushfile file >> --
for init incr limit proc >> --
forall array proc >> elem1..elem2.. (& executes proc)
framedevice mtrx wid height proc >> --
ge num1 num2 >> bool (true if num1>=num2)

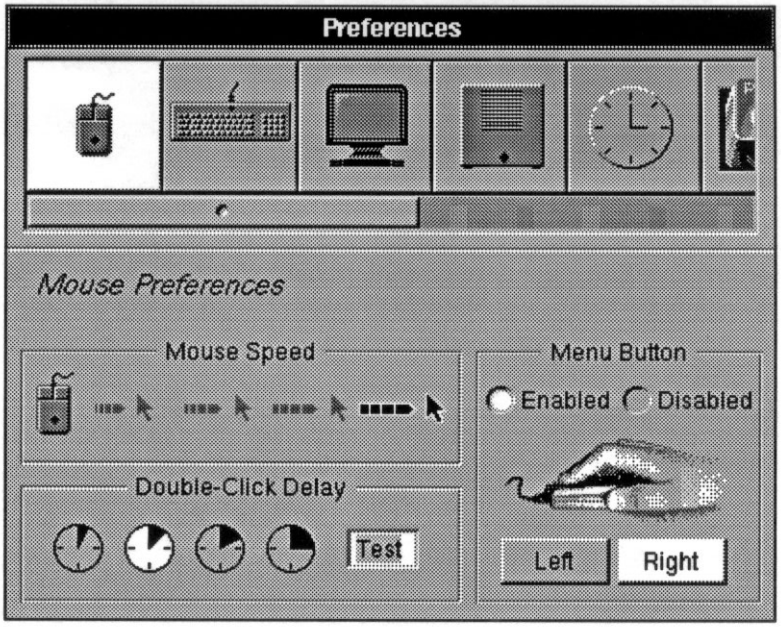
get array index >> element
getinterval arry beg len >> subarry
grestore -- >> --
grestoreall -- >> --
gsave -- tab --
gt num1 num2 >> bool (true if num1>num2)
identmatrix matrix >> id-transf-mtrx
idiv int1 int2 >> int-part-of(int1/int2)
idtransform xdt ydt >> xd yd (xdt ydt mtrx >> xd yd)
if bool proc >> --
ifelse bool proc1 proc2 >> --
image scan-len scan-lns bits/pixl mtrx proc >> --
imagemask scan-len scan-lns invrt mtrx proc >> --
index a1..a2..a3...ak t >> a1..a2..a3..ak a(k-t)
initclip -- >> --
initgraphics -- >> --
initmatrix -- >> --
invertmatrix mtrx1 mtrx >> mtrx (contents-of-mtrx1-inverted)
itransform xt yt >> x y (xt yt mtrx >> x y)
known dict key >> bool
kshow proc strinng >> --
le num1 num2 >> bool (true if num1<=num2)
length array >> length-of-arry
lineto x y >> --
ln num >> natural-logar-of-num
load key >> value
log num >> common-logar-of-num
loop proc >> --
lt num1 num2 >> bool (true if num1<num2)
makefont font-dict matrix >> transformed-font-dict
mark -- >> mark
matrix -- >> matrix
maxlength dict >> int
mod int1 int2 >> int1MODint2
moveto x y >> --
mul num1 num2 >> num1*num2
ne num1 num2 >> bool (false if num1=num2)
neg num >> -num
newpath -- >> --
not a >> NOTa (bitwise if a is integer)
null -- >> null
nulldevice -- >> --
or a b >> aORb (bitwise if a,b are integers)
pathbbox -- >> lo-left-x lo-le-y upr-rgt-x upr-rgt-y
pathforall mveto-proc lneto-proc crveto-proc clsepth-proc >> --
pop a >> --

Preferences

/NextApps/Preferences

Preferences is the application which lets you customize your NeXT machine. With it, you can tell the mouse how fast to move, configure your system to beep at you with your favorite tune, and tell the computer how loud you want the beep to be.

- In the File Viewer, go to **/NextApps** and open up **Preferences**. This application is so useful that you might want to drag it over to the dock on the right-hand side of the screen. When you open Preferences, this window will pop up:



Mouse: The icon that's selected is the *mouse*... you can set how fast it moves across the screen and how fast you have to double-click to open things. Look under the section called **Menu Button**. If you click **Enabled**, every time you hold down the right-hand mouse button the menu will pop up under it. Try it— this way you don't have to travel all the way to the upper-left corner of the screen to use the menus.

Keyboard: Click the icon of the keyboard. Here you can decide how fast a letter will repeat when you hold down a key, and how long the NeXT waits before letters start repeating. You can also find out how to get special characters like ¥ or ¶ by clicking the **Keyboard Panel** button and holding down the Alternate and/or Shift keys.

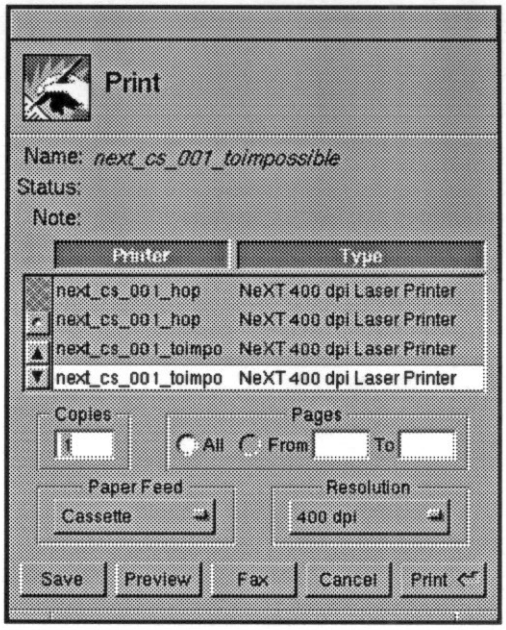
Monitor: Click the monitor to set the brightness and volume. Drag the brightness all the way to the right and blast your eyes (now put it back!). Do the same thing with the volume, and turn the mute on and off. Finally, use the volume and brightness controls on the keyboard to do the same thing. You can turn the mute on and off by pressing Command-  ... see the Keyboard CheatSheet for more info.

Setting UNIX Expert mode: Drag the scroll bar to the right until you can see this picture: 

Click the icon, and then select the check box labelled **UNIX Expert**. UNIX Expert mode lets you see files which are normally hidden from view.

Printing

Whenever you feel that you need a hard copy of something you are working on, whether it be a WriteNow document or a TopDraw picture, the NeXT has a universal command that is 99.99% guaranteed to work. By choosing **Print...** from the menu, a window similar to this will pop up:

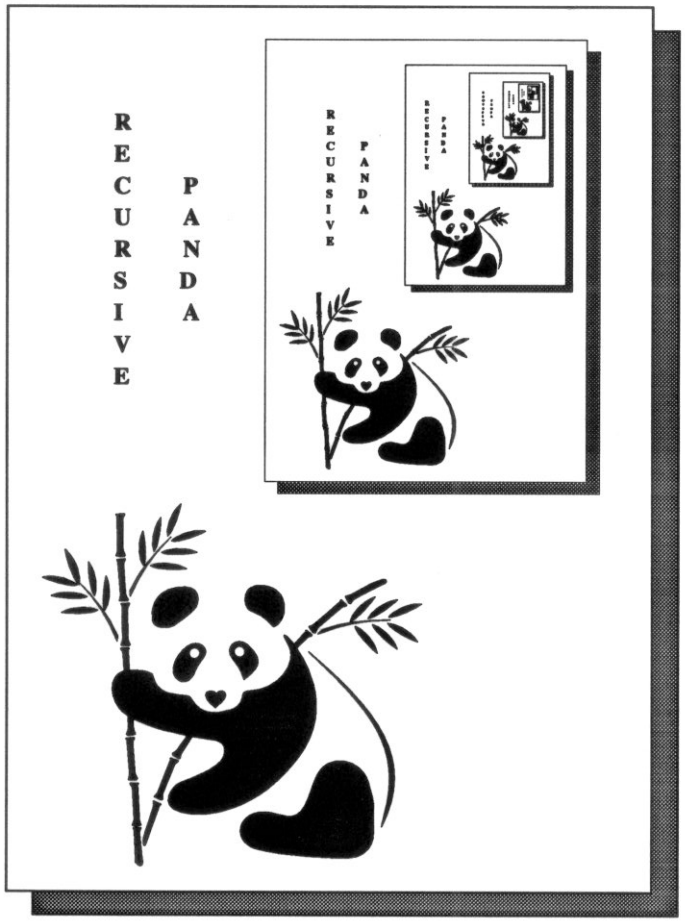


Originally, the local printer will be selected so you may want to move the scroll bar up and down until you find another printer that you would like. Probably the best choice is one of the printers that starts off with "next_cs_001" in the name because these are all located in the NeXT cluster with you. The printers are named according to the computer that they sit next to, so if you select next_cs_001_toimpossible, it would be the printer in the third row next to toimpossible. (Please don't ask us the difference between the two identically named listings; we have no clue.)

When you want to print something again, it will remember what printer you selected last time, so there will be no need to choose a printer again. However, you are more than welcome to change the printer to another one if it seems that a lot of people are all trying to use it (or some hoser is printing out 20 copies of his 80 page thesis so every single one of his relatives knows what he has been doing for the last 9 months).

If you want just one copy of the entire document, you can click the **Print** button. If you need multiple copies, you can type the number of copies you will need in the **Copies** box. If you only want to print out part of a document, click the **From** button and enter the pages that you want to be printed out. For example, to print pages 2, 3 and 4 of your midterm project, you would click the **From** button, then enter 2 as the start and 4 as the end. You will most likely want to keep the resolution set at 400 dpi (dots per inch) because that is the highest quality the printer can print at. If you need to cancel the print job at any time, just click the **Cancel** button. Good luck, don't waste paper and if you have old printouts that you don't need anymore, please recycle them.

Recursion



Introduction

Computer folks go to extreme lengths to avoid writing any more code than they have to. Recursion is a rather clever way of saving yourself tons of work—once you understand how it works.

Recursion is used when you want to do a task over and over again with slightly different parameters. For example, the above drawing is simply the same picture done over an over, only smaller and moved up to the right each time. It could be done by drawing each picture separately, but it takes much less typing to use the same code over and over again. This may seem to be exactly what a function would do, but recursion is subtly different. Hopefully by the end of this CheatSheet you will understand this difference.

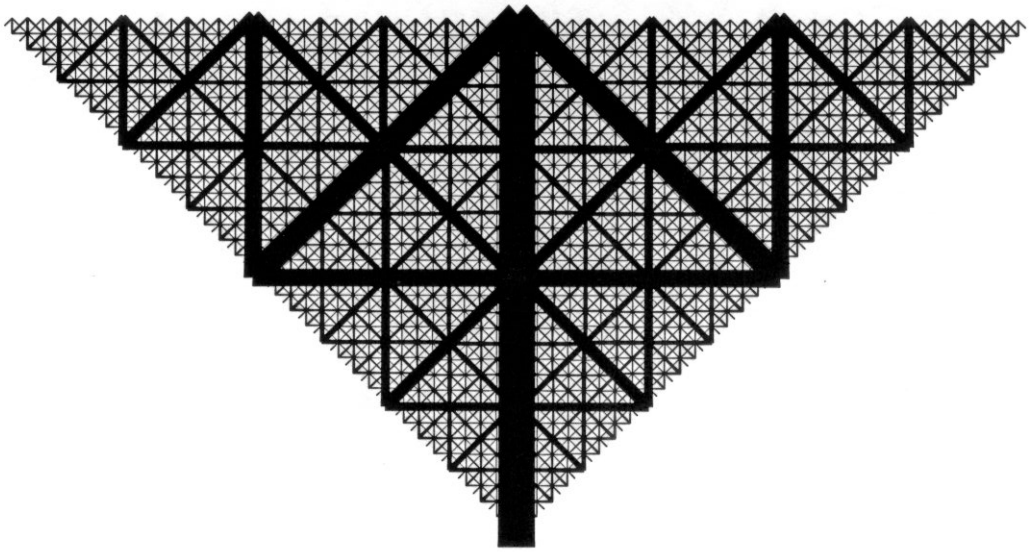
You may have seen recursive effects with two mirrors set up face to face, where the image goes back and back into infinity. In computers, there is a fixed number of times that an image can be repeated; otherwise, a recursive program would go on forever. In the above example, the pandas do not get infinitely small since the computer stops drawing them after a while.

A simple (for recursion anyway!) example

The following drawing is a fractal. We will not go into fractals here in any detail, but you may find it interesting. To see truly beautiful fractals, try ***/NextDeveloper/Demos/Mandlebrot.app****. If you run this on a NeXTdimension system, you can obtain fantastic color results.*

This program is in PostScript, so if you are not familiar with the language, you will probably have trouble following this example. The PostScript CheatSheet can give you an idea of what is going on.

Here are the results of the program:



And here is the PostScript code that produced it:

```
%!
/depth 0 def
/maxdepth 11 def          % Play with this number for different results.
/down {/depth depth 1 add def} def
/up {/depth depth 1 sub def} def

/DoLine
{ 0 144 rlineto currentpoint
  stroke translate 0 0 moveto} def

/FractArrow
{gsave .7 .7 scale        % change the .7s to drastically alter the drawing!
10 setlinewidth
down DoLine
depth maxdepth le
{135 rotate FractArrow
  -270 rotate FractArrow} if
up grestore} def

2 2 scale  % change the two numbers here to resize the picture.
```

```
150 200 moveto
FractArrow
stroke
showpage
```

In order to *fully* understand this program, you will have to learn a little more PostScript. If you like, however, you can simply play with the code found here, seeing what sort of havoc you can wreak by changing numbers in various places. Your choice. Anyway, upward and onward to more PostScript!

You will notice several new operators in this program. Basically:

- `gsave` will place the current size of the image and the way the page is rotated onto the stack.
- `grestore` takes such a state off of the stack and causes that state to take effect once again.
- `scale` scales the space the drawing is taking place in. For example, `.5 .5 scale` makes everything drawn from then on half the current size. The first number is the horizontal scale, the second is vertical.
- `currentpoint` is the point that the program is currently "thinking" about.
- `rotate` rotates the paper that the drawing is being produced on.

If you continue using PostScript, you will find that there are certain things you would like to do over and over. PostScript has an easy way to do this without typing a routine over and over. It is the `def` operator, which allows you to `def`ine a short string as equivalent to a long list of commands, a number, or a combination of both. In the above program the first line `/depth 0 def` defines the word depth to mean the number 0. Wherever the program sees "depth" from now on, it will replace it with a zero. The longest `def` statement in the fractal arrow program is the FractArrow procedure.

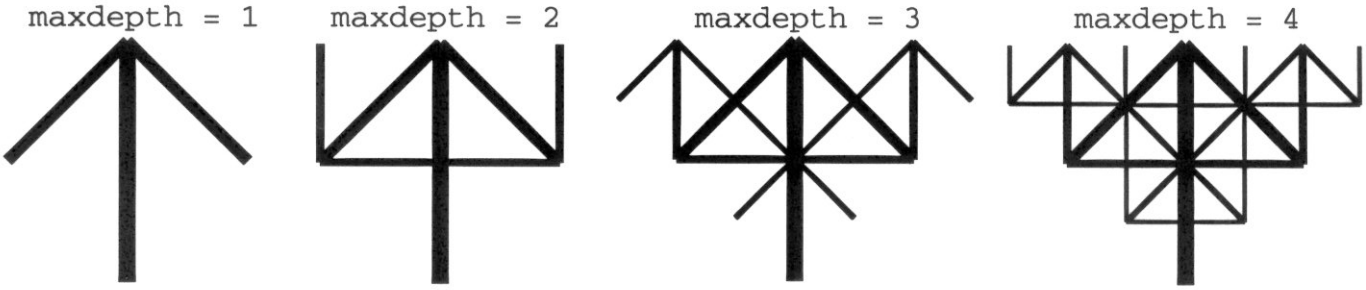
```
/FractArrow
{gsave .7 .7 scale
10 setlinewidth
down DoLine
depth maxdepth le
{135 rotate FractArrow          %note that both FractArrow calls are within
  -270 rotate FractArrow} if    %an if statement, providing a condition for
up grestore} def                %exiting the recursion.
```

After this procedure is defined, every time the program sees the string "FractArrow" it acts as if everything between the {}'s were there instead. Basically, procedures are simply shorthand for longer strings.

All of this becomes much more interesting when you see that the FractArrow procedure *contains* FractArrow! So FractArrow contains a FractArrow, but that FractArrow contains a FractArrow, etc. This would go on forever, but the Fractarrows within FractArrow are in an `if` statement, so there is a way to break the cycle. If this conditional statement was not there, the program would run forever.

What this program does is call FractArrow after rotating the piece of paper it is

working on and changing the scale of what it does. Until it gets to `maxdepth`, it will execute `line` twice, creating a new arrow on the ends of a larger arrow. It then calls FractArrow again for each end of that arrow, and so forth, scaling down each time it does a new layer. Look again at the pattern and see if you can make out the pattern. Here it is in stages:



If all of this looks terribly confusing, relax. You are dealing with a fairly advanced topic in Computer Science. If you are grasping the general concepts here, excellent! Try entering different numbers into the program and running it, guessing what will happen. Good luck!

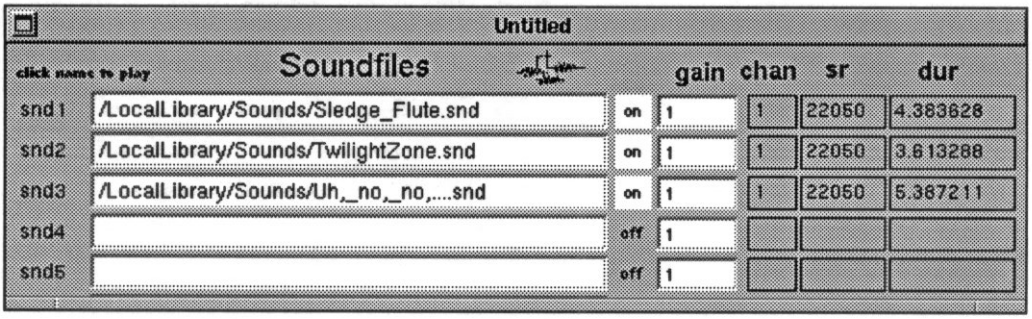
rt

/Net/dobro/musr/Apps/rt.app

Princeton's own Kent Dickey '92 and Professor Paul Lansky wrote this sound-mixing application. rt *stands for* real-time, *which is the funkiest thing about rt.app — it mixes sounds by sending them directly through to the speaker, as opposed to first figuring out the final result, then sending it to the speaker. This is instant gratification!*

Opening rt.app and loading in sounds for mixing

- In the File Viewer, find **rt.app** in **/Net/dobro/musr/Apps**. Open it up, and when it has launched, choose **Open new scratch pad** from the **Document** menu.
- To open sounds for mixing, choose **Open Soundfile(s)** from the **Document** menu. You can select more than one file at a time by shift-clicking each one.
- You should see something like this in the Soundfiles window:



If you see the word **off** next to any of your sounds, click once on the word **off** to turn the sound **on**. If you ever want to ignore any one of your sounds while mixing, click again on the **on** button to switch it **off**.

- Click **Load Driver** in the controls window to load in all of your sounds.

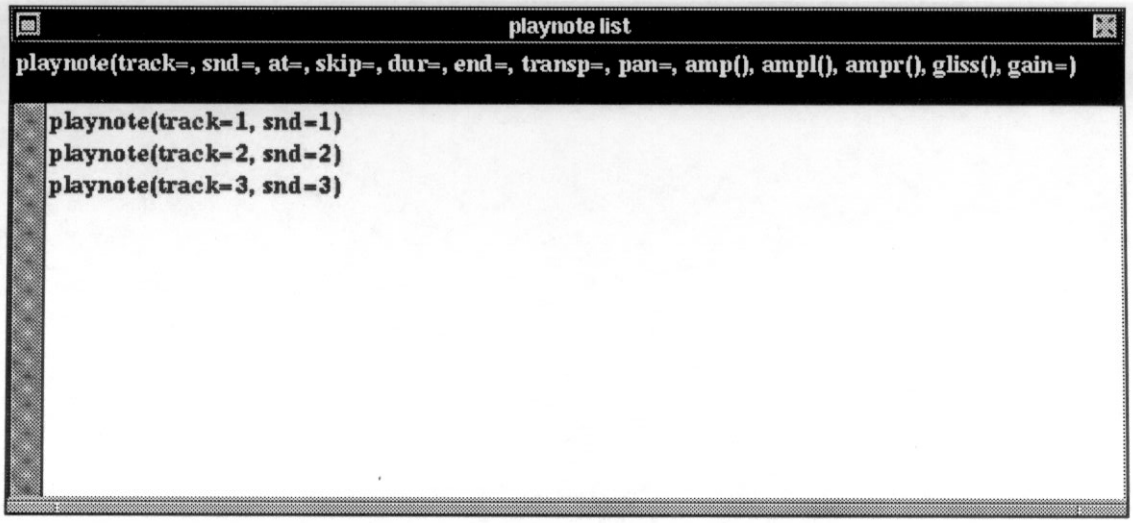
Preparing sounds with playnote *commands*

Each **playnote** *command tells the computer to play one sound in a certain way. For instance, one* **playnote** *command might wait two seconds, then play the second sound in the Soundfiles list. When you play the entire mix, all of the* **playnote** *commands are effectively executed together.*

You type **playnote** *commands in the* **playnote list** *window. Here is the very simplest form of the command:*

playnote(track=<track number>, snd=<sound number>)

If I wanted to include all the sounds in the above window in my mix, my playnote list window might look like this:



Each track is just a layer that gets added onto the final sound. There can be only one sound played on each track. You could, however, play the same sound on several different tracks. So, you could add this command: **playnote(track=4, snd=2).**

- Make playnote statements for each of your sounds.

- There's a lot more you can do, but for now... Click **Reload playnotes** in the control window. This button makes rt.app aware of any changes you made in the playnote list window.

- Click **play** in the controls window. Neat, huh? But there's more...

Arguments to playnote

Here is an incomplete list of arguments to **playnote** *and their effects on your mix. To add another argument to a* **playnote** *statement, just add it within the parenthesis of the existing statement, and separate it from the other arguments with a comma. Arguments can be in any order.*

at= the time (in seconds) to start playing the sound after the mix has begun. For example, **playnote(track=1, snd=1, at=1.5)** will start playing the sound 1.5 seconds into the mix.

dur= the amount of time (in seconds) during which rt.app reads the sound. For example, **playnote(track=1, snd=1, dur=5)** will read sound 1 for 5 seconds. If you hear your sound being cut off, make **dur** longer than the length of the original sound.

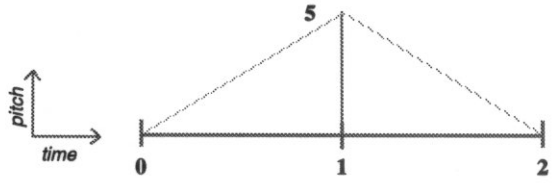
transp= how many notes up or down to transpose the sound. For example, **playnote(track=1, snd=1, transp=-3)** will transpose the sound 3 notes down the scale.

gliss(<relative time>, <transposition>, <relative time>, <transposition>, ...)
gliss is similar to **transp**, except that it "slides" a sound from one transposition to another. For example, **gliss(0,0, 1,5 ,2,0)** will slide the

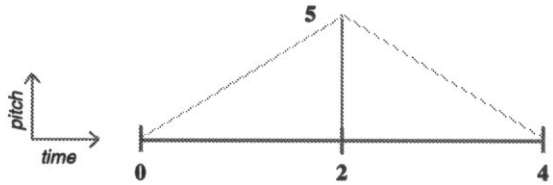
sound from no transposition (0) to 5 notes up the scale, then back down to 0 again.

Times are relative here. This means that gliss thinks of the first time you give — 0 in this case — as the beginning of the sound, and the last time you give — 2 in this case — as the end of the sound. Anything between is scaled accordingly. So these two **gliss**'s would accomplish the same thing:

gliss(0,0, 1,5, 2,0)



gliss(0,0, 2,5, 4,0)



amp(<relative time>, <volume>, <relative time>, <volume>, ...)

amp works much the same way as **gliss** does, but instead of controlling pitch, you control the volume. Normal volume is 1 (volume * 1), silence is 0 (volume * 0), and twice the normal volume is 2. So, to go from normal volume to silence to triple volume, you might type **playnote(track=1, snd=1, amp(0,1, 4,0, 7,3))**.

Saving the mix

- When you have something that sounds good, choose **Write mix to disk...** from the **Document** menu and save the file into your home directory.

Getting more help

- rt.app has an excellent **help...** section in the **Info** menu, complete with descriptions of every command and window, and even a short tutorial. If you want to spend some time here, you'll emerge with enough expertise to produce scripts for real digital music — complete with fade-ins and stereo.

General procedure

Keep in mind this general procedure for mixing sounds:

General Procedure for Mixing with rt.app

- load or change any sound file names in the Soundfiles window, and turn them on.
- click **Load Driver** in the controls window.
- add or change **playnote** commands in the playnote list window.
- click **Reload Playnotes** in the controls window.
- click **play** in the controls window. If you get weird results, click **Load Driver** again.
- when you have a sound you like, choose **Write mix to disk...** from the **Document** menu.

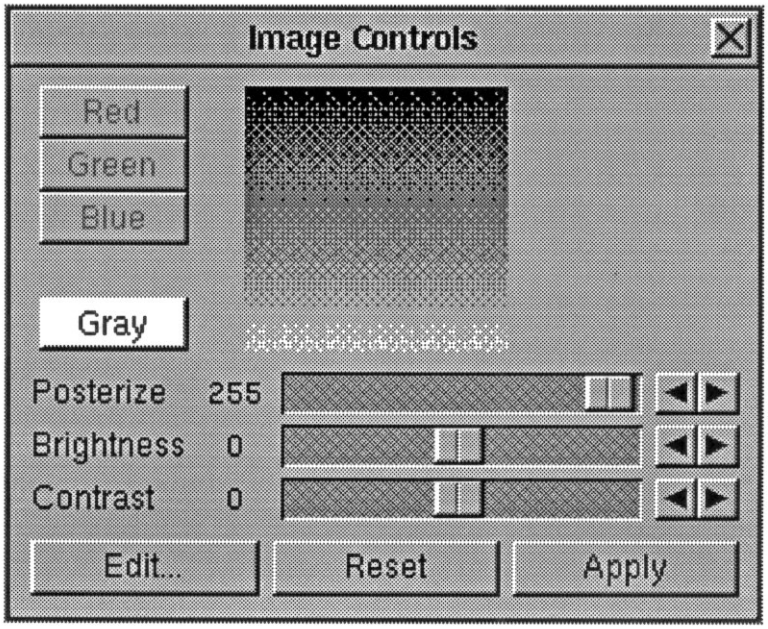
Scene

/NextDeveloper/Demos/Scene

Basics

In programming, a good maxim to follow is "don't re-invent the wheel." When it comes to pictures, Scene can let you take what others have done and use it in your own work. (Only uncopyrighted material, of course!) Scene manipulates pictures stored in the tiff format (files ending in .tiff), which is the most common type of bitmap graphic file on the NeXT. There is an entire directory of tiff images in **/LocalLibrary/Images/Scene_screens**. If you load them into Scene (**Open...** from the **Window** menu), you can play with their contrast and brightness as shown below.

To manipulate the picture you **Open...**, you will primarily use the Image Controls window, found by choosing **Adjust...** from the **Panels** menu.



Since you will probably be working on a greyscale screen, you don't need to worry about the color choices in the upper left corner. The three important things in this panel are the slider bars toward the bottom. To see what effect changing them has on the current picture, click **Apply**. The easiest way to see what each of them does is to just try them. Posterize seems to have little effect; Brightness and Contrast, on the other hand, can make a marginal image into a much more pleasant one. If the picture seems too grey, for example, increasing the contrast can make it more "snappy".

Using Scene With Other Applications

You can use standard Copy&Paste to take an image and place it in another document. Just perform these three easy steps:

- Make sure that the window containing the picture you wish to Copy&Paste is in the currently selected window. (Its title bar should be black.)

- Choose **Copy** from the **Edit** menu.
- Move to the application you want to **Paste** the image into and choose **Paste** from that application's **Edit** menu. Voilá!

Unfortunately, it seems that you can no longer save your *altered* images. Why this is I have no idea. It had worked previously, but when we tried it recently, we could no longer save a changed image. When you load it back in, it looks identical to the original. Go figure. Likewise, you cannot Copy&Paste an altered image—it still looks like the unmodified version. So why even bother with Scene? You can still Copy&Paste the original tiff images. This is normally fine, and it is a fairly easy way to perform the task. Good luck!

Sound

When the lead singer of Meatloaf belts out a note, he's just creating a wave of air pressure disturbances, which your ear interprets as music (or noise). Because air pressure is measurable at any instant in time, the sound is called continuous.

A computer, however, has no way of storing sound information for every *instant in time. Just as the computer must represent a continuous curve with a bunch of tiny dots on the screen, so must it represent a continuous sound with* samples *of sound information.*

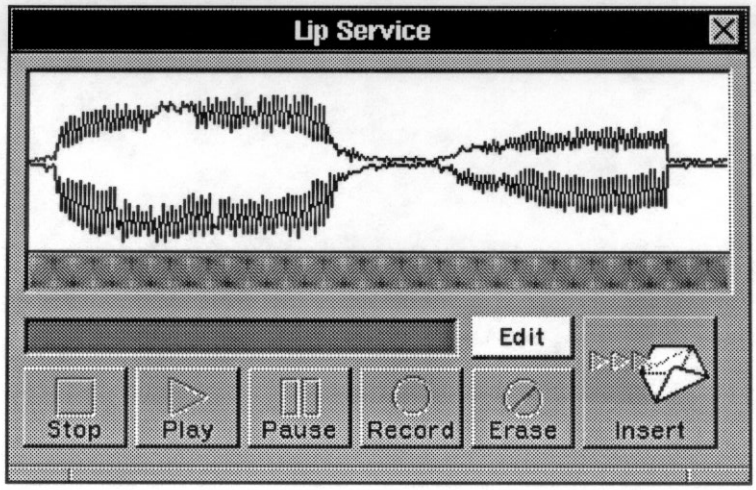
Try to familiarize yourself with these terms and concepts of digital sound. More importantly, come back to this sheet if you come across anything you don't understand.

analog signal	A continuously-defined signal. Real-world sounds are analog signals.
digital signal	A signal which is defined only for certain values of time, e.g. every ten-thousandth of a second. Computers store sounds digitally... in tiny chunks of information called samples.
sample	One little chunk of sound information used by a computer. Computers store each sample as a number, typically an integer.
hertz, or **Hz.**	One cycle per second. If a signal repeats itself (goes through one cycle) 22,050 times each second, then its frequency is 22,050 Hz. A thousand hertz equals one **kilohertz**, or **kHz**.
sampling rate	The rate at which a computer gathers sound information. At the 44,100 Hertz sampling rate common to compact discs, 44,100 samples are collected every second.
speaker cone	The physical object that vibrates back and forth inside a speaker to produce a sound.
frequency	The "pitch" of a sound. Higher frequencies occur as the speaker cone vibrates faster.
waveform	The wave of sound as it travels through the air. It is usually represented by a graph of the position of the speaker cone over time.
FFT	"Fast Fourier Transform"... an oh-so-important method for figuring out how much of each frequency appears in a signal over an time interval.
spectrogram	Usually a graph of how much of each frequency appears *at each instant* in the sound, as opposed to an FFT of the entire sound. The spectrogram is a just a series of FFT's taken in rapid succession.
DSP	"Digital Signal Processor" — a chip inside the NeXT that makes for *fast* processing of sounds and other digital signals.
channels	A one-channel sound is monophonic, and two-channel is stereo.

Digital Sound is the secret to the high-quality music of Compact Discs. Record labels just put sound sampled at 44,100 hertz onto the CD's. In fact, Apple™ CD-ROM disk drives can play music right off audio CD's! It's all headed in the same direction...

Visual Representation

When you work with sounds on the NeXT machine, you will commonly see this representation of the sound waveform:



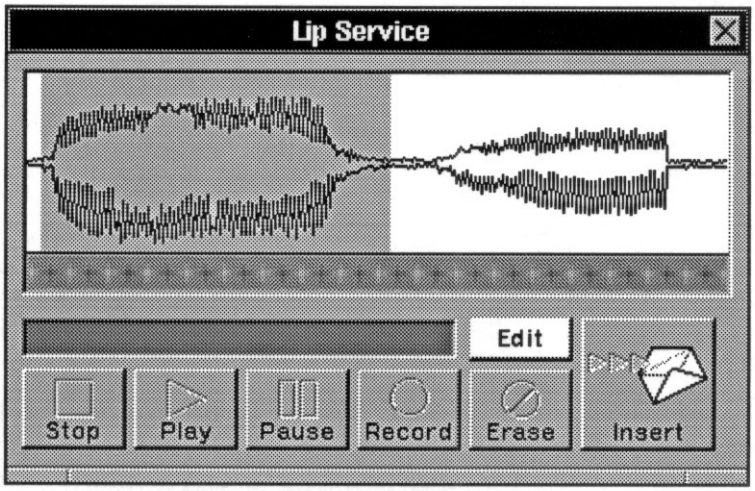
Since the speaker cone is responsible for creating the air disturbances (sound) which you hear, it's fair to show a graph of the speaker cone versus time, and claim that it represents the sound waveform.

Look carefully at the picture. Imagine that a horizontal line through the center represents a "resting" position for the speaker cone. Think of the speaker cone, then, as vibrating between the top and bottom lines of the graph. That's what produces the sound, and that's all you need to understand!

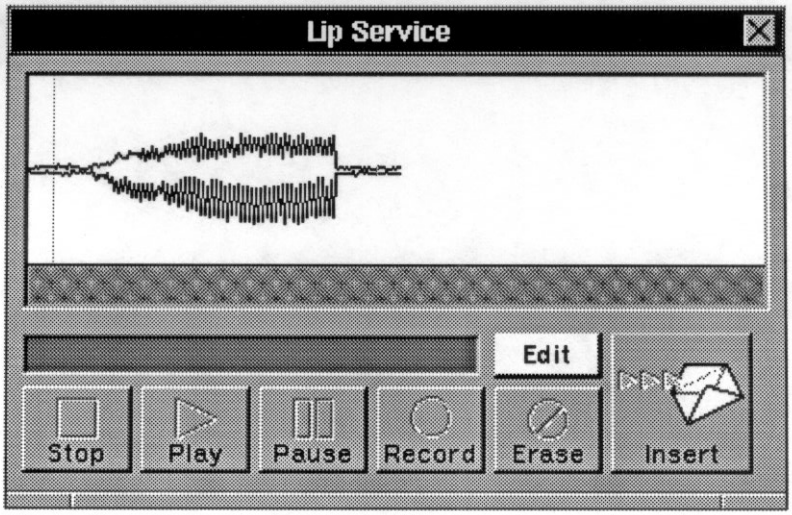
Editing

Editing sound is much like editing text. You can ***Cut***, ***Copy***, *and* ***Paste*** *to your heart's content. For instance, to move the beginning of the above sound to the end, I might follow this procedure:*

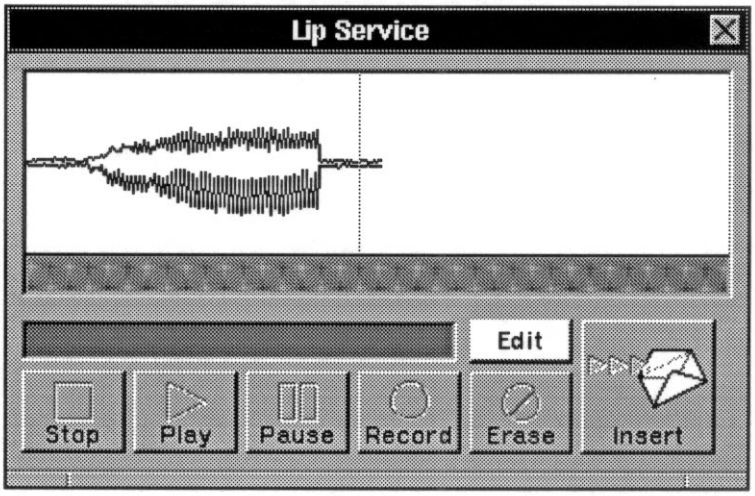
- Select the beginning of the sound (drag over it with the mouse), like this:



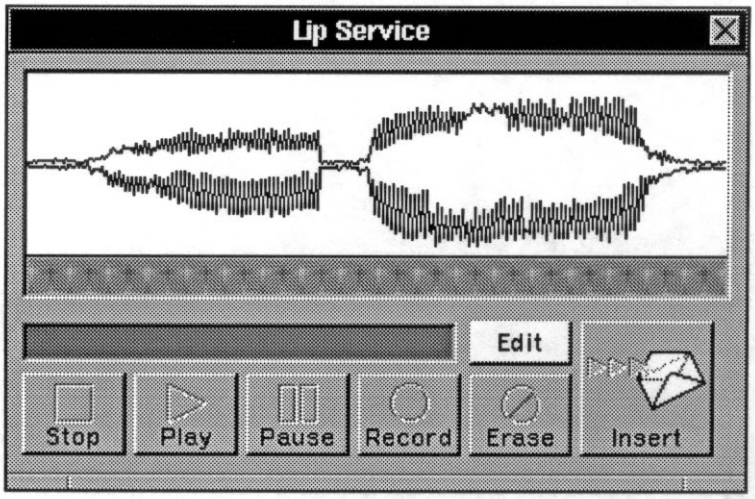
• Choose **Cut** from the **Edit** menu, to get this:



• Click once near the end to put in an "insertion point" (just like a text insertion point):



• Choose **Paste** from the **Edit** menu, to end up with this:



There ya go! Play the sound. If you are just reading along, look in Lab 5 for sound tools. Have a blast...

Storage

The smallest piece of data that a computer can deal with is a single signal which is either on or off. This signal, called a bit, obviously can't hold too much information, so bits are usually grouped together in larger chunks, called bytes. A byte, which is comprised of 8 bits, is the basis for storing computer data. What makes the byte so popular is that by using various on/off combinations of the bits in a byte, we can hold up to 256 different combinations, which is plenty sufficient to allow us to have a unique combination for each of the letters, numbers, punctuation marks, etc. that we commonly use.

When talking about big a file is, we usually need bigger units to describe it than bytes. Some of the more common ones are:

- A kilobyte (abbreviated 'KB') is 2^{10}, or just over a thousand, bytes (1024 to be exact).
- A megabyte (or just 'MB') is 2^{20} bytes which is 1024 kilobytes or 1048576 bytes.
- A gigabyte ('GB') is a billion (2^{30}) bytes.
- A terabyte (yes...'TB') is a *trillion* (2^{40}) bytes.

To get an idea of the amount of information that exists around us, let us do a few 'back of the envelope' calculations:

- A full page of single-spaced text is roughly 400 words at 5 letters per word. This is a total of 2,000 bytes, or a little less than 2 KB.
- A ten second sound sample using the built-in NeXT command 'sndrecord' takes up about 80 KB.
- A high-density 3.5 inch disk can hold 1440 KB or 1.44 MB.
- The Bible is 4.4 megabytes.
- A compact disc records music at 44,100 samples per second, with each sample composed of 16 bits. Therefore, a recording of Beethoven's Ninth Symphony which lasts 66 minutes 40 seconds equals 2,822,400,000 bits or 352,800,00 bytes or 344,531 KB or 336 MB of information.
- A floptical disk (a floppy disk for a NeXT that is somewhat similar to a cd) can store 512 megabytes of data.
- Firestone library has roughly 4 million volumes at 500 pages per volume at 400 words per page at 5 letters per word equalling a total of 4 trillion bytes or 3,906,250,000 kilobytes or 3,814,697 megs which is the equivalent of 7,451 floptical disks. Since each disk costs around $175, it would take $1,303,925 to store it all (however, we could probably get a bulk rate :-)). For a typist who can type 100 words per minute to convert all of the information into electronic form, it would take 15,210 years of typing 24 hours a day, 7 days a week, 365.25 days a year.

talk

Although e-mail is a great way to communicate with your friends, it can be a bit slow if you just want a quick answer to a question. The solution to this problem is to use the UNIX command `talk`, which will allow you to communicate with a friend interactively.

What `talk` does is divide up your screen by placing a horizontal line in the middle and allows you to type in the top part while the other person's typing echoes in the bottom. When you enter a character, it immediately shows up on the other person's screen and when the other person types, it immediately shows up on yours. Therefore, both of you can see everything that is typed by either person in real-time.

If someone wants to `talk` with you, you will receive a message that looks similar to this:

```
Message from Talk_Daemon@week.Princeton.EDU at 14:54 ...
talk: connection requested by jonthomp@palette.Princeton.EDU.
talk: respond with:  talk jonthomp@palette.Princeton.EDU
```

To start `talk`ing with jonthomp, just follow the instructions given by the message and type `talk jonthomp@palette` (you don't have to type `.Princeton.EDU` because both of your machines have the same suffix.)

After you are finished talking, type control-c (hold down the control key while typing the 'c' key) to quit.

If you want to request a `talk`, type:

```
talk <userid>@<the machine they are on>
```

Therefore, typing:

```
talk aaron@toyou
```

would request a talk session with `aaron` on the NeXT named toyou.

Your terminal window will draw the `talk` screen and start sending a message similar to the one above to the person you want to talk to. At the top of your screen, you will get a status message that at first says `[Waiting for your party to respond]`. If after a few seconds there is no response, the computer will send another talk-request message and your status line will change to `[Ringing your party again]`. If no one answers after a couple of rings, there is a pretty good chance that they are away from their computer, perhaps getting a late-night snack from the Wa.

If you are really busy and don't want to be disturbed by `talk` requests, you can tell the computer that you don't want to receive messages. Type `mesg n` at a prompt and you won't be bothered. If you want to start receiving `talk` requests again, type `mesg y`.

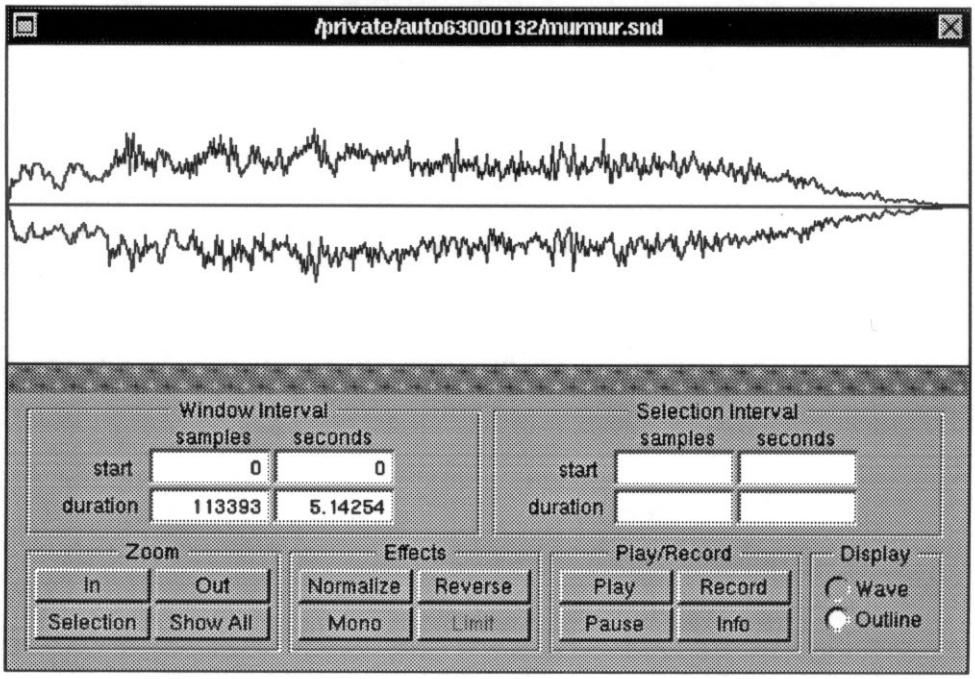
SoundEditor

/LocalApps/SoundEditor

SoundEditor is your basic tool for hacking around with sounds. There's nothing fancy here, but if you need to quickly record a sound, or if you need to do some minor editing, SoundEditor is the way to go.

Opening and Playing a Sound

- SoundEditor is located in **/LocalApps**. After you open it up, choose **Open** from the **Window** menu to select your sound. After SoundEditor loads the sound, you should see a sound waveform graph which looks something like this:



The Play/Record Area

- Click **Play** to hear the sound. If you drag over a part of the sound to select it, you can hear just that part by clicking **Play**.
- To record a sound, choose **New** from the **File** menu, and click **Record** to start. Click the same button (now the **Stop** button) again to stop recording.

  If you really want to be tricky, you can insert your own recording into an existing sound by clicking once at the point you want to insert at, then recording as you normally would with a blank sound.

The Zoom Area

- You can zoom in on a section of the sound by selecting that part and pressing **Selection**. Zoom back out to view the complete sound with the **Show All** button.
- Alternatively, the **In** and **Out** buttons just zoom in or out by a certain amount each

time you click them.

The Effects Area

- The **Normalize** button scales your sound to the maximum possible volume that doesn't introduce distortion.
- Guess what the **Reverse** button does to your sound? Ask a TA if you need help.
- **Mono** converts a stereo file to monophonic format — useful if you're short on disk space.
- If you ever figure out what the **Limit** button does or how to use it, tell the TA.

Working with a Sound... General Tips

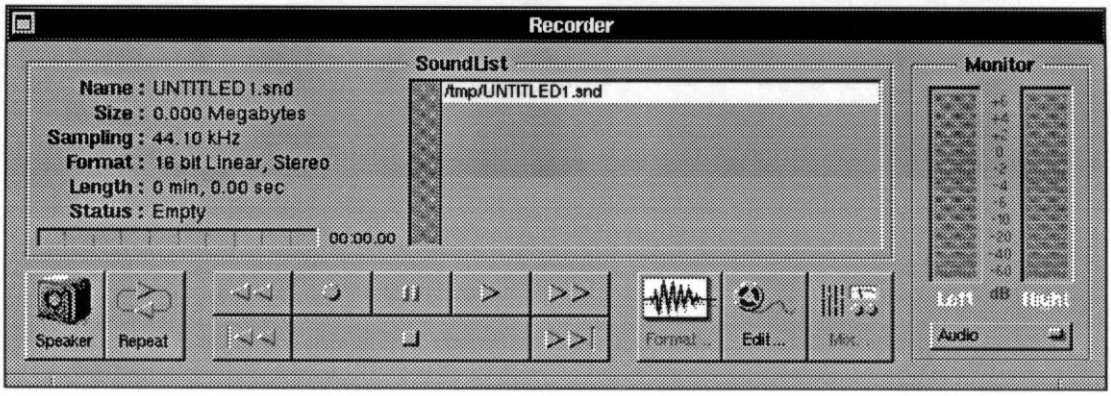
- You can select a section of the sound just like you select a section of text: by dragging over it. Then you can listen to that section, or use **Cut**, **Copy** and **Paste** just like you would with text.
- For easier access to them, you can grab hold of the **Edit** and **Effects** menus and drag them to a convenient place.
- Remember that you have to select part of a sound before you can do anything to it.
- *See the second part of the Sound CheatSheet for step-by-step help.*

SoundWorks

/Apps/Soundworks.app *only on the NeXT machine* **page**.

Getting Started

- **SoundWorks.app** is in the **/Apps** directory of the NeXT machine **page**. Find it in the File Viewer, and double-click it to open it.
- You'll be confronted by a window titled Recorder, which resembles a tape deck:

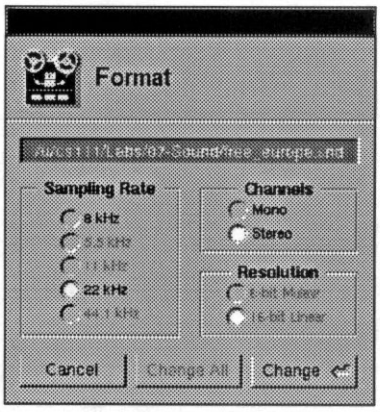


- First, choose **Open...** from the **Sound** menu, and select any sounds you want to work on (perhaps from **/LocalLibrary/Sounds**). To select more than one sound at once, shift-click each sound you want.

Mixing Sounds

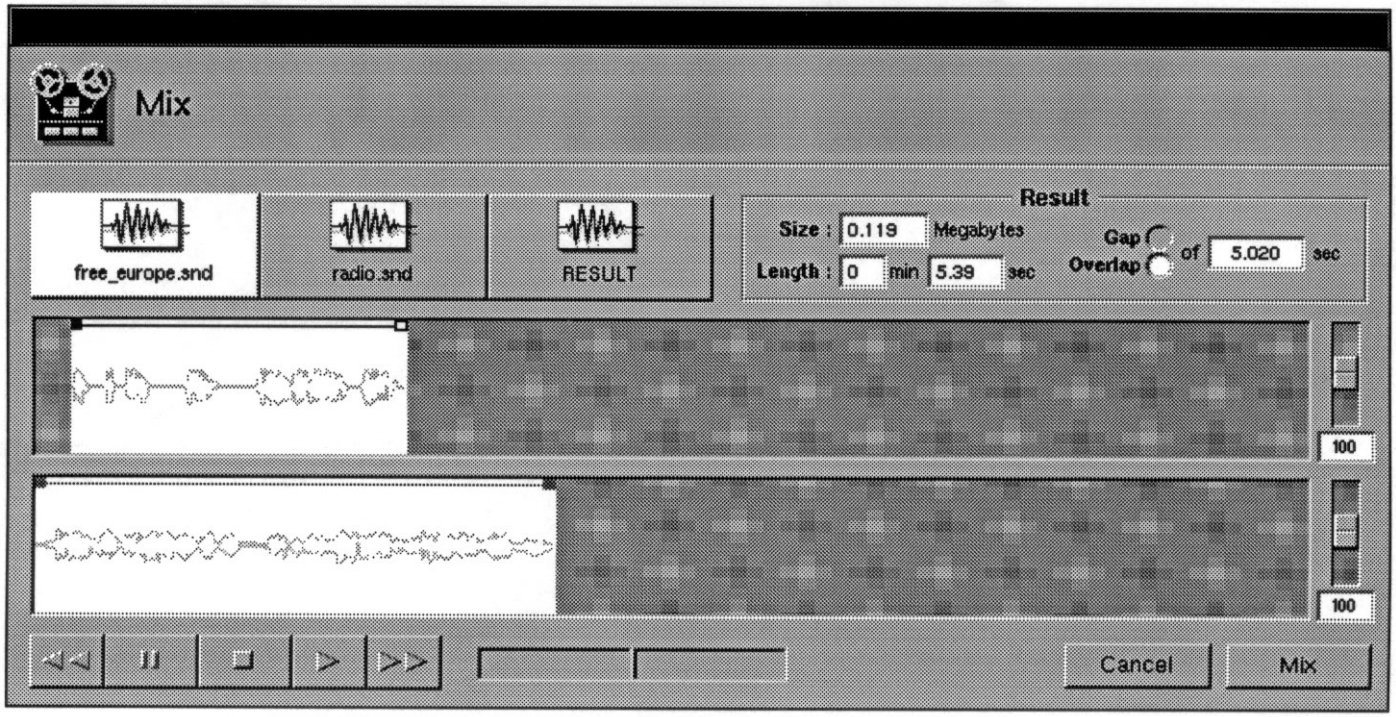
SoundWorks has an excellent and intuitive interface for mixing sounds. Although you can only mix two sounds at a time, it's probably the easiest way to do it. Give it a try.

- Both sounds must be saved in the same format, i.e. they must both be stereo, or both monophonic, and they must play at the same Sampling Rate. To check this, shift-click both of the sounds to select them. If the **Mix...** button is dimmed (so you can't click it), then you need to change the format of one of the sounds so it matches the other. If you can't press the **Mix...** button, perform the following step.
  - To change the format of a sound, select the sound in the Recorder window. Click the **Format...** button, and this window will pop up:

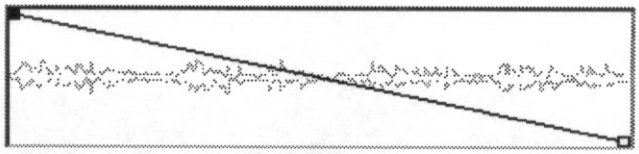


The only things you can change here are the Sampling Rate and the number of channels (whether the sound is stereo or monophonic). Remember that converting to a lower Sampling Rate reduces the sound quality, because the computer sends information to the speaker at a lower rate. *Make sure the Sampling Rates of the two sounds are the same.*

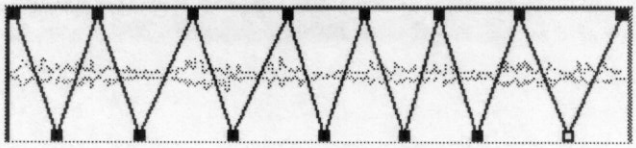
- Now you're ready to mix! In the Recorder window, select the two sounds by dragging over both of them or by shift-clicking. Press the **Mix...** button, and up pops this window:

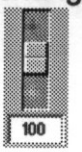


- The highlighted button at the top tells you which sound is selected, e.g. the sound you are currently manipulating and which you will hear if you press the **play** button ( ). By selecting the **RESULT** button and pressing **play**, you can hear both sounds played together. You can decide where to mix in a sound by dragging the picture of its waveform horizontally.
- That's not all! The line across the top of each waveform represents the sound volume. Try this: on one of the sounds, drag the tiny "selection point" in the upper-right corner to the bottom, so it looks like this:



Now play the sound and listen to it fade out. The selection points on the ends aren't the only ones, either... a new selection point — which you can drag up and down — will appear wherever you click the volume-line. This sound will fade in and out and give you a headache:



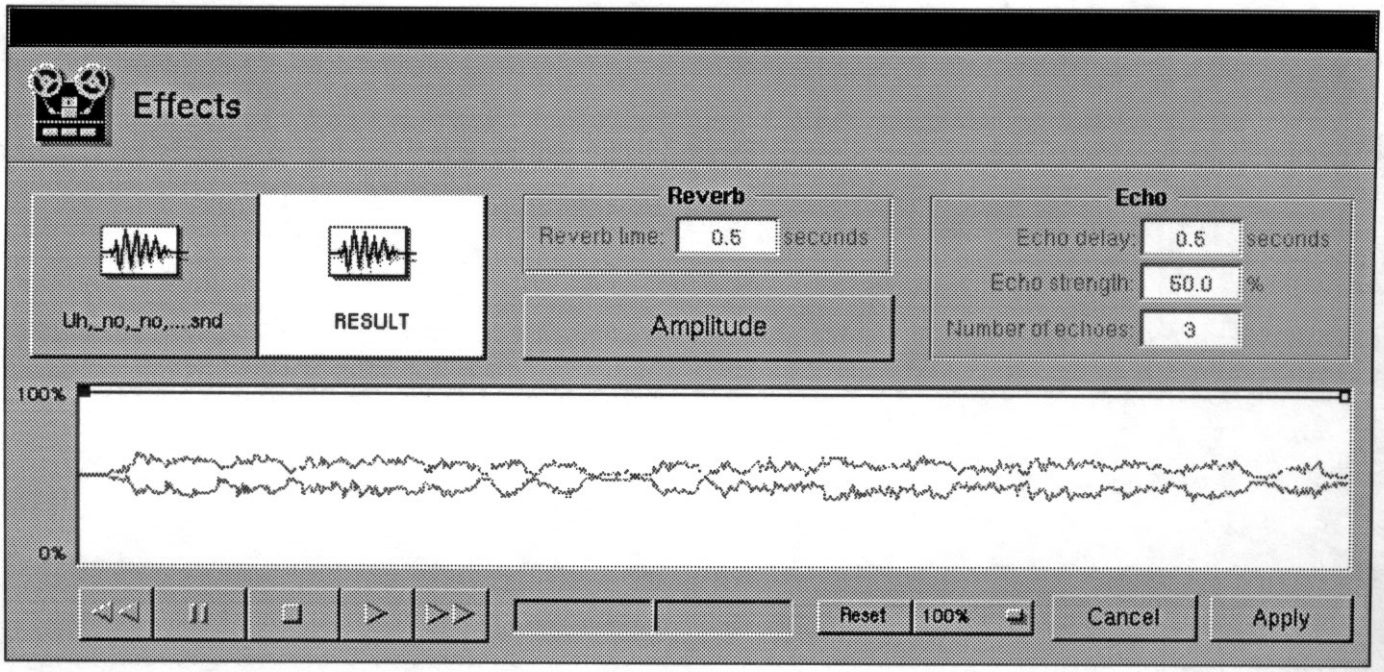
- Finally, you can change the relative volume of each sound with the slider on the right-hand side: 

- When you are done, click **Mix**, and your shiny (happy) new mix will appear in the Recorder window as something like ***/tmp/MIX_1.1.snd***. The filename is italicized because you still have to save it; choose **Save** from the **Sound** menu.

The Effects Panel

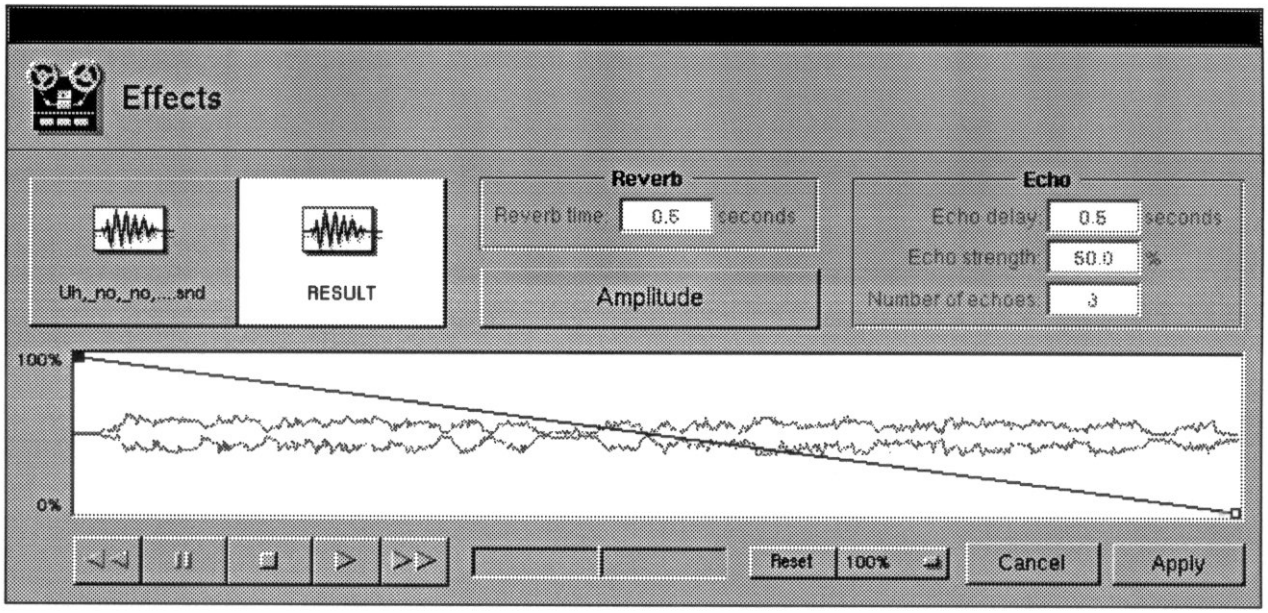
Say you want to make a glissando out of your voice. (For those of us who aren't musically inclined, a glissando is a quick slide up or down the musical scale.) Or, to be mundane, you just want to hear a sound fade in or out. SoundWorks provides you the opportunity to do either, with a utility for manipulating both pitch and amplitude (volume).

- First, select a sound to work on in the Recorder window. Click the **Edit** button to open up to a view of the sound waveform.
- To select a part of the sound to work on, click-drag over a section or choose **Select All** from the **Edit** menu.
- Choose **Effects Panel...** from the **Effects** menu. A window like this one should pop up:

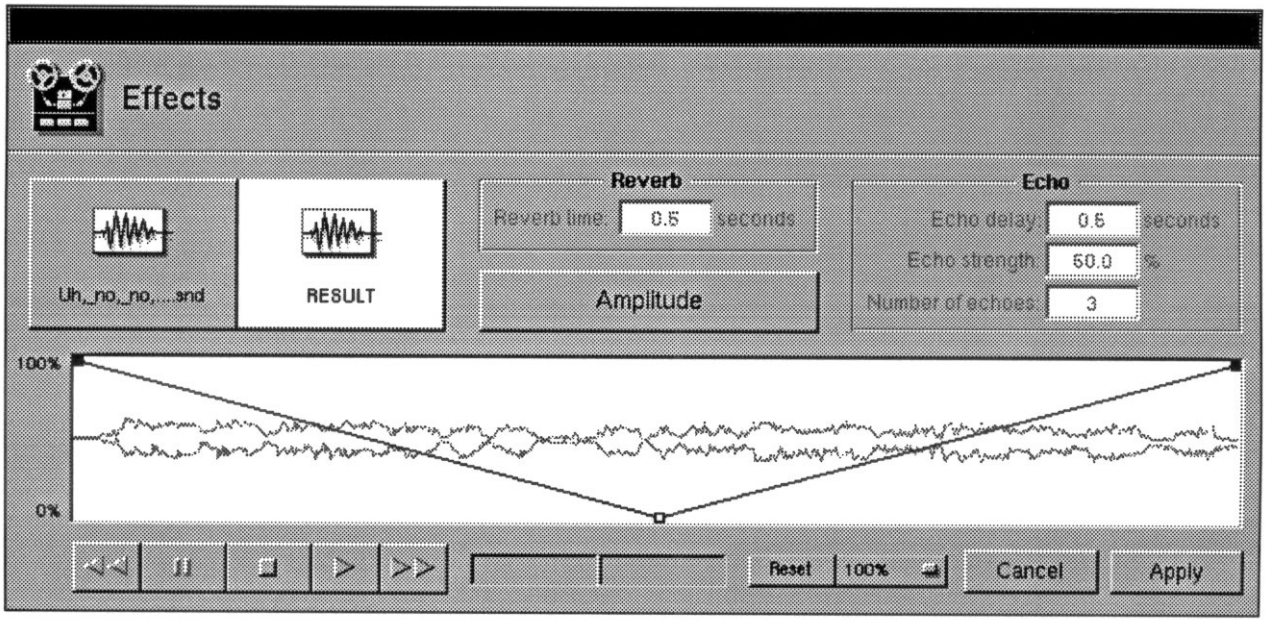


Amplitude Effects

- If the button at the top of the window reads **Amplitude**, you can play around with the sound volume. The line across the top represents the volume, and you can move it by dragging one of the little "selection points" in the upper corners. To create a fade-out effect, you might drag the selection point on the right so it looks like this:



- To hear how it will sound, make sure the **RESULT** button in the upper-left is selected, and press the **play** button.
- You can also create new selection points by clicking on the line itself. To make the following example, I clicked once in the middle of the line, and dragged it to the bottom of the window.



- When you're done fiddling with the volume, click the **Apply** button and save your sound with the **Save** menu option.

Pitch Effects

- To play with the pitch, click the button that reads **Amplitude**. It should change to read **Pitch**.
- Pitch Effects works much the same way as Amplitude Effects, except the line starts out in the center of the window and represents... pitch. Play around, and click **Apply** when you're done.

Echo and Reverb

- To activate either **Echo** or **Reverb**, click once in the appropriate section. Once you have activated something, you can turn it off by clicking it again.
- These options are fairly self-explanatory. Type numbers into the text boxes, listen to their effects with the **play** button, and click **Apply** when you're finished.

Other Effects

- Play around with other choices from the **Effects** menu yourself. Just select part of a sound in the editing window and choose any of the effects.

telnet

`telnet` allows you to "teleport" around the globe, moving from one computer to another. There are computers that specialize in certain games, computers that contain discussion groups on specific topics, whatever. Now here comes the best part—many of them are completely FREE! (Oh boy!) Below, there is a short list of some of the places that will allow you onto their system. One of the most interesting (and largest!) ones is the Cleveland Freenet, which is also very easy to use—a good candidate for your first `telnet` attempt. To get to any of these machines, type:

```
telnet <address>
```

When you find the name of a new site to `telnet` to, you should also be given instructions on how to log in. (Such as what to type in at the `login:` prompt.) This information varies for each site, so we can give you no general rules.

`telnet` is also usable within Princeton for logging into other machines you have access to. Phoenix, for example, is a much larger machine than a NeXT, and often has many users logged in. To `telnet` to phoenix type `telnet phoenix`. No sweat. Login with your normal userid and password, and you will be using phoenix! It is often nice to keep a phoenix `telnet` session going so your friends can find you while you are online. You can also `telnet` to another NeXT in the lab if you so desire. For example, typing `telnet toyou` would begin a `telnet` session to the NeXT named toyou.

Some interesting addresses to `telnet` to:

`freenet-in-a.cwru.edu`	Cleveland Freenet; instructions online
`quartz.rutgers.edu`	login as 'bbs'
`testsun3.nersc.gov`	login as 'bbs'; relatively new.
`forest.unomaha.edu 2001`	note space between edu and 2001; login as 'bbs'
`quiche.cs.mcgill.ca`	search the archives! log in as 'archie'
`mud.iastate.edu 1991`	a good game in Iowa; log in with a fictional name

Of all these, Cleveland Freenet is probably a good first try. It is huge, has some fairly interesting stuff (including USA Today summaries every morning at 8am), and is easy to use. Plus, if you register with them, they will give you an e-mail address in Cleveland. Whatever makes you happy.

Archie

There is one other site worthy of special mention. At `quiche.cs.mcgill.ca` you can search for specific files and find out what `ftp` sites they reside on. (See the `ftp` CheatSheet for an explanation of the `ftp` command) This is extraordinarily useful since it takes the place of manually searching 600 locations! Once you are `telnet`ted to the site (remember to log in as 'archie'), you can use `prog <string>` to perform a search. Typing `prog hat`, for example, would give you a list of all files with names containing "hat." The information archie produces includes the size of the file, its full name, and the site at which it is located. You can then `ftp` to that site and transfer the file(s) you want.

When you are finished searching, type `quit` to leave archie. If a search is going crazy and you want it to stop, you can press ctrl-c. Just hold down the control key while pressing the "c" key. Good luck!

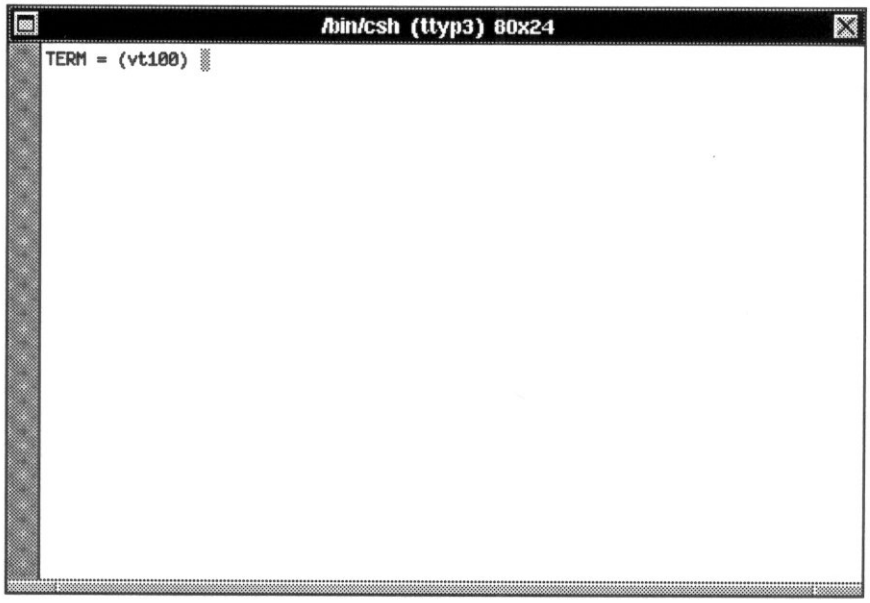
Terminal

/NextApps/Terminal

The Terminal window is where you type UNIX commands. It is your window into the guts of the system.

Opening a Terminal Window

- To open a Terminal window, use the File Viewer to find the application **Terminal** in **/NextApps**. You may want to drag it over to the Application Dock on the right-hand side of the screen, where you keep all your frequently-used applications.
- Double-click the Terminal icon. After a few seconds a window pops up with the message `TERM = (vt100).`



The computer is asking you what kind of keyboard you're using. Hit <Return> to accept the default `vt100`. From now on, the computer will think you are sitting in front of an old terminal called... the "vt100."

- A prompt such as

```
toyou%
```

should appear, with the name of your NeXT machine in place of `toyou`. That's your cue — you're talking with the computer!

Don't be afraid of making mistakes... the worst that can happen is this:

```
toyou% help
help: Command not found.
```

(I typed `help` at the prompt, and the NeXT told me it didn't know what I was talking about.) Go to it!

TopDraw

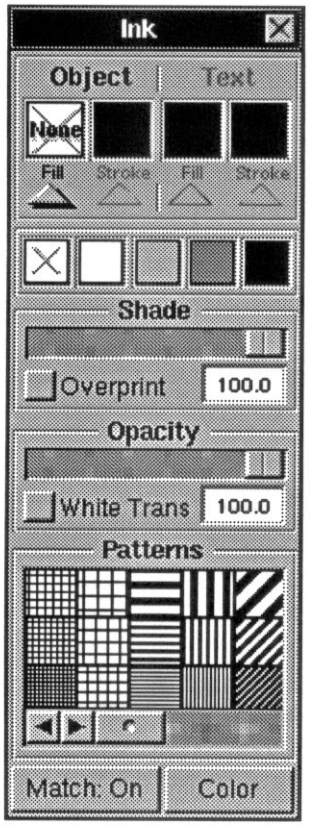
/LocalApps/TopDraw.app

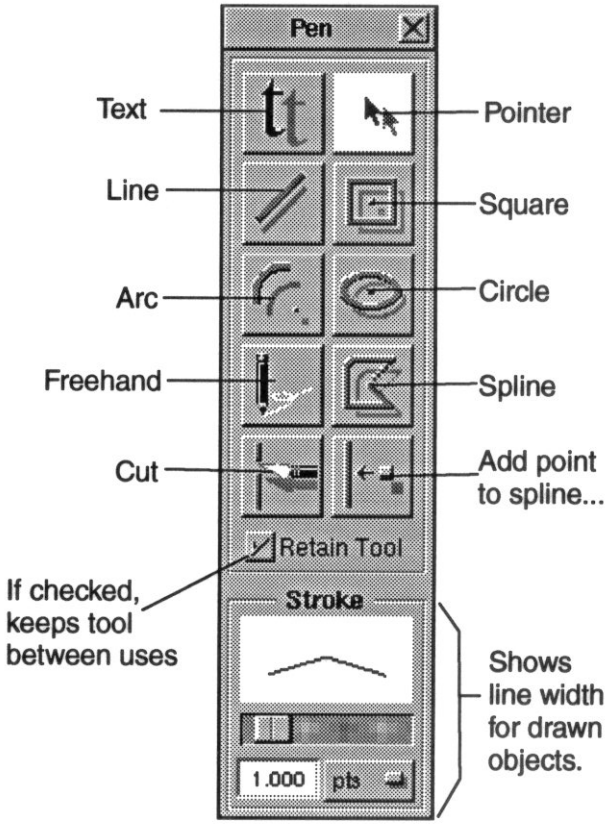
*TopDraw is an extremely powerful drawing program that is rather simple to use. This CheatSheet will be short since there is a fairly good tutorial/demonstration of this application in **/LocalDemos**. It takes quite a while to load, but it does a pretty good job of introducing the basic concepts of TopDraw. One note: make sure that you do not click the "Auto" button at the beginning of the demo. Page through the demo manually—otherwise the video will run faster than the audio, and the speech will be clipped off as it tries to keep up with what is happening on screen. Your alternative to the demo is to just plunge right in. This is probably the easiest way to learn TopDraw. Once you see a few basic concepts in action, you can begin designing your own drawings, so go for it. If you get stuck you can always go back and watch the demo.*

Basics

*Note: To get a drawing space after starting TopDraw, choose **New Document** from the **Window** menu and click the **OK** button in the window that appears.*

TopDraw is an object-based drawing application, which means that it deals with objects like circles, squares and lines rather than individual dots on the screen. To make these objects, there are many different "tools" you can use. There are tools to draw lines, squares, ovals, text, etc. To select a tool, click it in the Pen window. Most tools are used by dragging in the drawing area. The best way to learn is to try—give it a whirl. To manipulate what color (shade of grey) the objects are drawn and filled with, use the Ink window. The Ink and Pen windows (which appear when you start TopDraw) look like this:





The four squares near the top of the Ink window are the colors representing their respective elements. In the above picture, Objects are filled with "None" (transparent) and stroked (outlined) with black. Text is filled with black and stroked with black. Below this area of the window there are five squares of different colors. These are color wells, and are like tubes of paint. To change the color for the Object Filling, for example, you would drag from the color well to the square above Fill in the Object section, thereby dropping the paint into the appropriate well. Patterns (located at the bottom of the Ink window) are handled in the same way.

A fundamental concept in drawing programs such as TopDraw is the selection of objects. Selecting an object makes it the object that you are currently dealing with. Therefore, if you change color in the ink window, all selected objects will be affected, or if you choose **Copy** from the **Edit** menu, all selected objects will be put onto the Pasteboard. A selected object will have eight little boxes around it to show that it is selected. Try clicking a few objects; drag them around; change their stroke or fill. To select multiple objects, either drag a box around them using the pointer tool or shift-click each of them individually.

Beyond these conceptual generalities, it is nearly impossible to give a good overview of how to use TopDraw. A demonstration is far more effective at teaching how to actually make your own drawings. Better yet is for you to brave the unknown and try it out. You can't hurt it and it can't hurt you. The only warning I will give you is to save often since we have found TopDraw to be slightly more crash-prone than most other applications.

To aid you in your endeavors, there is a help facility within the program—the entire set of TopDraw documentation is accessible by choosing the **Help...** menu item.

Using TopDraw With Other Applications

TopDraw is good at drawing. (Surprise.) We used it extensively for this lab manual—the above Ink and Pen window diagrams, for example, were retouched and edited in TopDraw, then Copied and Pasted into WriteNow. We found TopDraw to be the easiest and most powerful "temporary space" for our diagrams. We could capture an image using Grab or some other application, **Paste** it into TopDraw, add text to it, edit it, then **Paste** it into the final WriteNow document. Remember: Copy&Paste! Copy&Paste! Just remember that an object must be selected in order to be put onto the Pasteboard using **Copy**.

troff

(pronounced Tee-roff, as in Tee-shirt)

Introduction

`troff` is a text formatting program that allows you to write good-looking papers using only standard UNIX tools. If you have previously written papers using a computer, you will probably find `troff` *very* unlike what you are used to. Instead of showing you what the paper will look like when it is printed, `troff` looks like a confusing tangle of text. Yet when you print it out, it somehow looks good. Welcome to the *text formatter.*

Look at the sample `troff` document at the end of this CheatSheet. There are many (apparently) non-sensical lines beginning with periods. These are `troff` commands, telling `troff` what to do with the text. That is basically all there is to producing a `troff` document. Simply look up the command you want to use, find out how it works, then place it before the text you want it to act upon. Easy.

To print out the results, type:

```
cat <filename> | eqn | troff -ms -t  | lpr -t -P<name of printer>
```

The formatted document should soon appear.

The commands (at least a few useful ones)

`troff` *commands can be rather strange. A few rules should help to clarify things:*

- *All commands begin with a period (.)*
- *All commands go at the beginning of a line.*
- *Continuous blocks of text are put together and formatted unless a command tells* `troff` *to do it differently. See the sample* `troff` *file and its corresponding output for an illustration.*

Titles and Section headings

.TL	Title follows
.AU	Author section follows
.tl /left/center/right/	Three part heading follows
.ce [n]	center the next [n] lines

Paragraphs

.PP	Normal indented paragraph
.LP	Left-aligned (block) paragraph
.sp	blank line

Emphasis

.R [text]	Print [text] in Roman font (normal)
.I [text]	Print [text] in Italic font
.B [text]	Print [text] in Bold font
.UL [word]	underline word

Equations

.EQ	Begin writing an equation
.EN	Stop writing an equation

The following operators will affect the next item or group of items enclosed in {}:

sqrt	next item is placed under a square root sign
over	places the left section over the right item
sup, sub	superscripts or subscripts next item, respectively
+ - * /	addition, subtraction, multiplication and division signs (appear as shown)

Note: most of this command list was taken from the book:

Preparing Documents with UNIX
Constance C. Brown, Jack L. Falk & Richard D Sperline
Prentice-Hall, Englewood Cliffs, NJ 07632
©1986

It is located in the Engineering library, and provides an excellent reference for doing much more with `troff`.

Sample troff document

```
.sp
.sp
.tl /cs111/troff sample/1/

.sp
.TL
Sample Troff Document--Printed output
.LP
.ce 1
.I by

.AU
Sebastian DeBroglie

.sp
.sp

.PP
Notice how this line
is all broken
up in the troff file (the one with all the troff
commands in it), yet
when printed, everything comes out
together.

After skipping a line in the
troff file, the text is printed flush
with the left margin.

This is the same as using .LP
before the start of a paragraph...
But now we will put .PP before the
next paragraph...

.PP
And last but not least, an
.UL underlined
word!!!!!  (Plus the first line is
indented, since we put the .PP
before the beginning
of the paragraph....)

.sp
.sp
.ce 4
centered stuff!
.sp
.UL underlined
.B Bold!!

.EQ
1 * {{sqrt {3x sup 2 + 4y +3}} over {sqrt {19x sup 2 + 13y +6}}} - 1395
.EN
```

Sample Troff Document--Printed output

by

Sebastian DeBroglie

Notice how this line is all broken up in the troff file (the one with all the troff commands in it), yet when printed, everything comes out together.

After skipping a line in the troff file, the text is printed flush with the left margin.

This is the same as using .LP before the start of a paragraph... But now we will put .PP before the next paragraph...

And last but not least, an underlined word!!!!! (Plus the first line is indented, since we put the .PP before the beginning of the paragraph....)

centered stuff!

underlined
Bold!!

$$1*\frac{\sqrt{3x^2+4y+3}}{\sqrt{19x^2+13y+6}}-1395$$

UNIX

Every computer you use has an Operating System *which completes the tasks you send it with menu selections, mouse clicks, and typed commands. The operating system of the NeXT computer is, as you might have guessed, UNIX.*

Up until now, you may have only used menus, windows, and buttons in your interaction with UNIX. This Graphical User Interface (GUI) is one of NeXT's strengths. But a far more common interface that you will find available on every UNIX system is the Command Line Interface (CLI). In this situation you exchange information with the computer solely through text input and output — you type commands and the computer prints responses.

Why would you want to work with just text when you have fantastic menu-driven programs at hand? That's for you to find out in this course. Suffice it to say that many people enjoy the power and flexibility of UNIX's Command Line Interface, and prefer to use it over menus, windows, and mice wherever possible.

Hopefully the following information will get be enough to get you started. Don't worry about learning every single thing... just absorb as much as you can and use this CheatSheet for reference as you work with UNIX in the labs.

Where to get help

Manual Pages

These handy guys may well be one of your most useful assets for finding UNIX information quickly. In a Terminal window, simply type:

```
man <command>
```

This will bring up help on the command you type in place of <command>.

If you want help on a particular subject, use the following:

```
man -k <subject>
```

This will bring up a list of commands relevant to <subject> You can then use the normal `man` command above to get more information on any of those listed.

CIT documentation

The packet from CIT entitled, "Introduction to Phoenix" has many nitty-gritty details that all of us often forget. It's a good thing to bring to lab for reference. Anyway, we all know how much fun these things are to read when you get bored. You can pick these up in the CIT InfoCenter, at 87 Prospect Street.

Concepts

Userids and Addresses

Every UNIX user has a unique userid ("user I.D."). From the computer's point of view, your userid is your only real name. People who want to contact you from outside Princeton must type `<your userid>@phoenix.Princeton.edu` in order to reach you. This whole "phrase" is your own unique *Internet address*.

Passwords

Nothing but common sense here. Everyone has their own password, and *no one* should know yours. When you choose a password, *don't use a real word.* If you do, there's a chance that someone will discover it, because with a little UNIX knowledge, it's not too hard to write a program that will discover every password that can be found in a dictionary.

Why does it matter if someone guesses your password? Unfortunately, there are scum who would love to enter your account, using your password, and break into other computers. You would be blamed.

If you want to change your password right now, ask your TA for help changing it with the Preferences application.

Files

Files are "collections of related data records" (Webster's) They are more of an intuitive concept than anything else. A word-processing document, for example, is a file.

Directories

UNIX allows you to organize your files into directories. A directory is like a folder, which (usually) contains related files. You can have directories within directories, forming an entire "tree" of directories. (If you were to draw out the directory structure on paper it would resemble a genealogical tree.)

Along with every other UNIX user, you have a **home directory**. This is where you keep all your own files (and directories) It's also the place you find yourself when you log in. The computer's shorthand for "your home directory" is "~", and its shorthand for someone else's home directory is `~<userid>`. For example, you would refer to `omkensey`'s home directory as `~omkensey`.

Paths

Whenever you want to run an executable file, you usually need to tell the computer exactly where the file is. For example, throughout this course you will use the `handin` program to submit your work electronically. Since `handin` is in the **~cs111/bin** directory, to run it you would type

```
~cs111/bin/handin.
```

Luckily, there's another way. If you type a command which isn't in the current

directory, the computer will search through a list of other directories that contain commonly-used commands. This list is called your *path*. If **~cs111/bin** is in your path, you could merely type

```
handin
```

to run the program. To see your list of paths, type `echo $path`.

For more information on files and directories, see "Welcome to NeXT."

Disk Quotas

Your *disk quota* is the maximum amount of information you are allowed to store in your home directory. To get information about how much storage you have used, and how much space you have left before you hit your disk quota, type `quota -v`.

If you use the `quota -v` command, you will see a number labelled "limit." This is the absolute, unyielding maximum amount of information you can store. If you approach this limit, the computer will let you know, and will ask you to remove some files.

Commands

This section will give an overview of some of the more useful UNIX commands. It is by no means comprehensive, but it should give you what you need to know for basic survival. To use these commands, type them while in a Terminal window (see the Terminal CheatSheet).

Necessary commands

- Type **`ls`** to get a listing of what is in your current directory. You should know that UNIX hides files beginning with dots (like **.login**) — this is just a convenient way to hide files you know are there and don't want to see each time you type `ls`. If you *do* want to see the hidden files, type `ls -a`.

  While `ls` gives you just the listing, `ls -s` will tell you the size of each file:

```
prompt> ls -s
total 1001
    4 hummingbird.wn     1 now.wn          1 swan2.wn
  411 hurrah.snd         2 swan1.wn      584 were_free.snd
prompt>
```

  The file sizes next to each file are given in kilobytes(KB) — roughly a thousand characters. A single-spaced page in WriteNow, like **hummingbird.wn** above, weighs in at about 4 kilobytes.

  One important point: if `ls -s` comes across a directory, it *will not* tell you the size of everything inside it. It will give you something like 1 KB — that's how much space the directory needs to keep an index of everything that's inside.

  Finally, to see which elements of a listing are directories and which are

executable programs, type `ls -F`. Directories are followed by a slash ("/"), and executable files are followed by an asterisk ("*"). In the following example, **TA** is a directory, **mumble** is a normal file, and **hello** is an executable file.

```
TA/                    mumble                    hello*
```

- **cd** is used for changing directories. To move into a directory, just type `cd <directory name>`. Typing `cd ..` will take you up to the directory which contains the current one. If you get lost, `cd` by itself will take you back to your home directory.

  REMEMBER THIS: *use ~ as an abbreviation for your home directory. Typing*

  ```
  cd ~/songs
  ```

  will send you to the subdirectory `songs` *in your home directory. You can also get to someone else's home directory by typing* `cd ~<userid>`*, where <userid> is the userid of the person you are looking for.*

- **cat** spits out the contents of a file or files. Type `cat <filename(s)>` and watch the file(s) scroll up the screen, one after the other. (This is where the command got its name... from con*cat*enating files and sending them to the screen or another file.)

- **more** lets you view a file one screenful at a time. Type `more <filename(s)>` to view the contents of the files listed in <filename(s)>, in small chunks. Pressing the spacebar will take you to the next page.

- **rm** will erase any files you tell it to. Type `rm <filename(s)>`. Be careful with this command! If you do erase something by accident, all is not lost. Send mail with a request to restore your file to info@phoenix.

- **cp** copies a file to a new name or place. The format is

  ```
  cp <original file> <new file>.
  ```

  Typing `cp ~/wolves ~/lower` will make a copy of the file **wolves** in your home directory and will name the copy **lower**.

- UNIX has a fast way to send a file to the printer: **lpr**. If you just type

  ```
  lpr <filename>
  ```

  your file should go to the nearest printer (the "local" printer).

  If that doesn't work, or if you want to send your file to a specific printer, use

  ```
  lpr -P<name of printer> <filename>.
  ```

  for example, `lpr -Pnext_cs_001_toimpossible laughing` would send the file **laughing** to the printer to the left of the NeXT machine "toimpossible." See the Printing CheatSheet for more on printer names.

- To print a file with half the paper **lpr** uses, use the `enscript` command. The format is the same as that of `lpr`, except after `enscript` type -2rG (**2** columns, **r**otated 90 degrees, in **G**audy, fancy, format). For example,

  ```
  enscript -2rG -Ptoimpossible laughing
  ```

Not-Absolutely-Necessary-But-Still-Very-Interesting Commands

- `finger` will tell you information about a user. Give it a try by typing `finger <name>`. If you know a person's userid, then `finger -m <userid>` is many times faster since the `-m` tells `finger` just to search through a list of userid's.
- Typing `who` will tell you who is currently logged onto the system and where they are logged in from.
- `w` is like `who`, but the last column of its output tells you what each person is doing. Try it — it's legal to be a voyeur on the network!

Advanced Commands

- `grep` is not a command; it's a way of life. In its most simple usage, `grep` finds a search-string in a text file. Type `grep 'foo' <filenames>` to find the string "foo" in the file(s) <filenames>. `grep` will output every line containing "foo" to the screen.

  To find "flowers" in the files **guatemala** and **amanita**, type

  ```
  grep 'flowers' guatemala amanita
  ```

- `wc` counts words, lines, and characters in a file. Type `wc <filenames>`, and `wc` will give you something like this:

  ```
  toyou% wc foo bar
      904    7950   51288  foo
      165    1348    8432  bar
      1069   9298   59720  total
  toyou%
  ```

  The first column is the number of lines, the second is the number of words, and the third, the number of characters.

- `compress` shrinks a file's size down to, on average, about a half of its previous size. There are many reasons for compressing files. One is simply to save disk space, which is expensive and sometimes scarce. The other is to make a file smaller so it can be transmitted faster over the network. Compressed files all have a `.Z` on the end. To compress a file, type `compress <filename>`.
- `uncompress` uncompresses a compressed file. Any file which ends in `.Z` is compressed. Type `uncompress <filename.Z>`.

For those who want to become UNIX wizards

Pipes

Here lies the beauty and power of UNIX. With a pipe (|), you can send the output of one command into the input of another. For example, if you come across a directory containing a huge number of files, you might type

```
ls | more
```

This will send the output of the `ls` command into `more`, which lets you view it one screenful at a time.

If you think of each UNIX command as a flow of data, pipes are the means of controlling that flow. Normally, the effect of a command flows right out onto the screen, the simplest case of data flow. By using a pipe, you can make that flow of data go someplace else...to another UNIX command, for example. This is the area in which `grep` truly shines. Let's look at another example:

The `w` command lets us see what each person on the system is doing at the moment. Unfortunately, if you are on a large UNIX system such as phoenix, there are bound to be many, many users. It becomes tedious to look through all of the people to find your friends each time you check up on them. `grep`, as we know, will find any lines containing a certain string. So we want to take the flow of data from `w` and "pipe" it into `grep`, like this:

```
w | grep <name of friend>
```

This series of commands will execute `w`, but instead of printing its output on the screen, gives it to `grep`. `grep` in turn looks for your friend's name and since there is no pipe after grep, the flow goes onto the screen, printing out every line with your friend's name in it. Voilá!

Output redirection

- This isn't too hard. If you follow a command with a greater-than sign (>) and a filename, the output of the command is saved into that file. For example,

  ```
  ls > listing
  ```

  saves the output of the `ls` command into a file called **listing**.

Filename expansion

- This isn't too hard, either. Say you want to count the words of every file in your current directory. Luckily, you have a special symbol which makes it easy. The * symbol "expands" to a mean any combination of characters of any length. So typing

  ```
  wc *
  ```

  would count the words in all the files of the current directory. It works!

- Here's another example: typing

  ```
  grep 'buck' a*
  ```

  will search for the string "buck" in all the files that begin with "a."

Aliasing

- UNIX allows you to build "aliases" for those long commands which you find yourself typing over and over again. For example:

  ```
  alias fb "who | grep foobar"
  ```

  will check to see if foobar is on the system whenever you type `fb`.

- You can **pass arguments** to the alias with the sequence \!* . If you type

```
alias hello "who | grep '\!*'"
```

the alias will substitute, in place of \!*, anything you type after the `hello` command. So if you type

```
hello eedecker
```

the computer will actually receive the command "who | grep 'eedecker'". (This command finds out who is on the system, and only prints out `eedecker`, if she is on.)

- That's all! To make an alias permanent, type the alias command (as above) at the end of a file called **.cshrc** in your home directory. The **.cshrc** file is run every time you log in.

Regular Expressions

Say you want to search a text file for every line which begins with 'a'. You can't just use `grep` *to search for 'a' — you'll get hundreds of lines! Fortunately, grep accepts certain special characters — like one that represents the beginning of a line — in its search-strings. Expressions which use these special characters are called* ***regular expressions****, and they pop up often in UNIX.*

Here is a tiny, incomplete list of regular expression characters and what they do. Include any of these in search strings just as you would include a normal character.

^	at the beginning stands for the beginning of a line. To find all lines in the file **foo** that begin with the letter "a", you might type `grep '^a' foo`.
$	at the end stands for the end of a line (it "anchors" the expression to the end), and it's used just like ^. Type `grep 'a$' foo` to find every line in **foo** which ends in "a".
.	stands for *any* single character, so 'r.m' could mean 'ram' or 'rem'.
.*	will match *anything*. So m.*e fill find *anything* which starts in m and ends in e.

For a more complete list, look for regular expressions in the `man` *page for* `ed`.

UNIX Cultural note: In the old days, UNIX users had to use `ed` as a text editor (*you* have Edit or `vi`). `ed` is only cute until you try it. In `ed`, you can't look at your document while you edit it. Instead, you type commands to edit your file, and use your imagination to see how it looks as you make changes.

Anyway, one of ed's commands searches for text, and it accepts regular expressions. The command is g/<regular expression>/p, and it does the same thing as grep. Actually — and this is where the culture shows itself— "grep" can be found in the abbreviation for ed's search command: "g/re/p".

Another related note... some UNIX wizards know `ed` so well that they use it by choice. To them, arrow keys and pointing devices are cumbersome and annoying.

Useful Stuff

This is probably the most random CheatSheet in the Universe...It is a hoge–podge assortment of general wisdom we thought you might appreciate having.

CIT

It's worth a call to **8-6028** for help if you are really stuck. The people that answer the phones just sit around bored if you don't give them a call. Be friendly.

TAs

Love 'em, hate 'em, use 'em. They probably know what's going on, and even if they don't, they can probably make up something good. Also don't forget the Professor's office hours for those fundamental problems. ("What's a computer?") Remember the TAs will be there for the lab periods, so don't hesitate to ask them for help!

Your Neighbor

"Love thy neighbor as thyself" Or perhaps moreso when they save your project masterpiece by telling you not to empty your recycler. Seriously, the other people in this class probably don't know any *more* than you, but they may know *different* information. Give them a try. (Ask them, but whatever you do, don't let them *do* anything for you!). Likewise, when someone asks you what a keyboard is, don't poke them in the eyes for being stupid. Someday you may forget.

The goals of the course

One of our main goals here is to rid you of computer fear and make you a competent computer user. We realize that not all of you will become Computer Science majors, but we do not want to lie to you about anything in the field. Unfortunately, in many cases reality is often less pleasant to swallow than a well told lie. As long as you come away with more than you came in with, you are probably doing all right. In the same vein, however, don't slough off! A good showing of effort on a difficult concept, or extra submitted material (especially of high quality!) certainly reflects well on you.

WriteNow

/NextApps/WriteNow.app

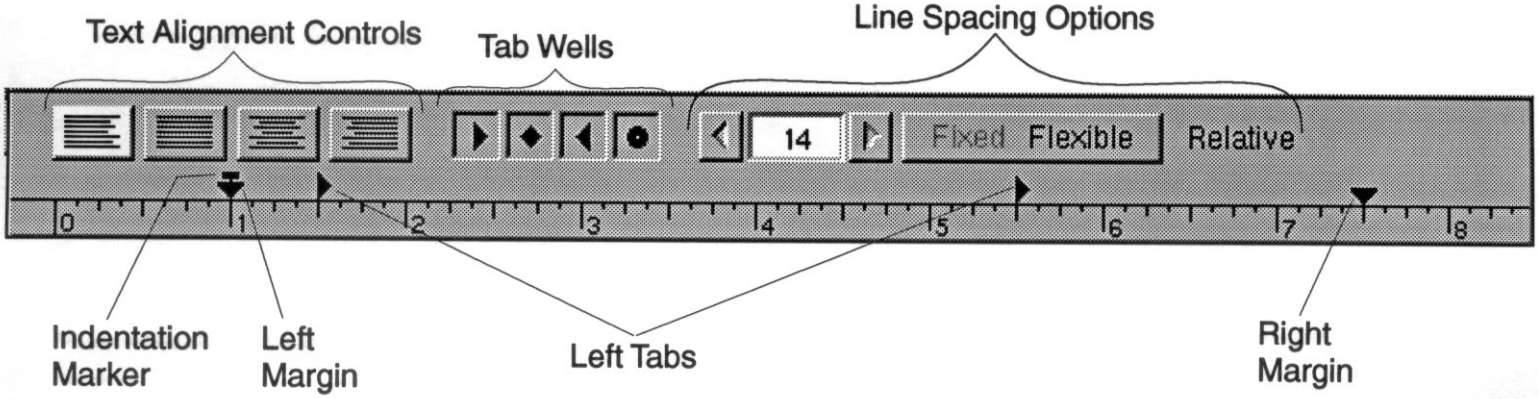
WriteNow is the standard word processing program for the NeXT computer. It allows you to format text, place graphics, and print out your final product. Fortunately it is quite easy to use, and is very similar to Macintosh™ word processors. To get a feel for its power, just look at this lab manual—most of it was done in WriteNow.

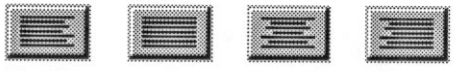
Getting Started

When you start WriteNow, an empty document window will appear toward the right–hand side of the screen. You can begin typing here. If you had double-clicked a WriteNow document (a file ending in ".wn") then that document would have appeared instead of a blank one. You are now ready to learn the details of producing a document.

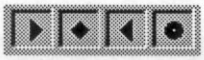
The Ruler

One of the first things you will probably want to do is find the ruler. This controls things like spacing, margins, and centering. To make it appear, choose ***Show Ruler*** *from the* ***Format*** *menu. The ruler should appear at the top of your document.*



Text Alignment 

Each of these will align text as it appears in the respective icon. The one that is highlighted in the ruler picture, for example, will left justify text. The second one will fully justify text, the third will center text, and the last will right justify text. Click the type of alignment you want. To change the alignment of a block of text, select the block with the mouse, then click the new alignment.

Tab Wells 

To make a new tab stop, drag a tab marker from one of the tab wells onto the ruler's tic marks. In the above example, there are left tab stops at 1.5" and 5.5". The four different types of tabs are:

 left justified tab stop—lines up the left sides of the text

Example:
Left justified
Text.

 center justified tab stop—centers the text at the tab

Example:
Center justified
Text

 right justified tab stop—lines up the right sides of the text

Example:
Right justified
text

 decimal tab—lines up the decimal points in the data at the tab stop

23.42
1213.457
345.4358

Line Spacing Options 

The number between the two arrows (in the white area) is the number of vertical points taken up by each line. A point is a unit of measurement equal to 1/72 of an inch. On the ruler shown, each line of text is 14 points high, so if we are using 12 point type, there will be 2 points of space between lines. To double space a document of 12 point text we would set the line spacing to 24—12 points for the text itself, and 12 points between the lines.

The other section is the Fixed/Flexible option. If Fixed is in black, then the line spacing number is an absolute. No matter what, it will be that number of points. Therefore, if you set the line spacing number smaller than the point size of the type, the text will overlap between lines, like this:

This is 12 point text, but only 8 points for each line.
Fixed spacing is selected, causing this overlap.

If Flexible is in black (selected), then WriteNow will automatically prevent this sort of thing from happening. Click either side of the bar to select that part.

Tab, Indentation and Margin markers

Tab markers show where the cursor will jump when the Tab key is pressed. There are four different types of tab stops, each described in the above section "Tab Wells."

Text is typed between the right and left margin markers. To change the margins, drag the downward-pointing triangles to where you want the new margins.

The indentation marker is where you want the first line of each paragraph to

begin. You can move this marker to the right of the left margin marker to create a standard indent, or to the left to make a "hanging indent" like this:

> This is an example of a hanging indent, done at great expense and at the last minute in a really nifty font. The first line will be hanging off the end of the paragraph, while all of the subsequent lines will be indented one-half inch.
>
> However, when a new paragraph is begun, the first line is once again hanging out, but all the rest of the lines are indented. This is particularly nice for bibliographies and lists of points, since they can be so easily delimited. Enjoy!

If you want to change any of these settings for a group of text you have already typed in, you must select that text before making the change.

Using WriteNow With Other Applications

WriteNow is a good application to use as a base for importing graphics from other applications. It handles text well, and allows you to create that much of your document. Then you can use drawing applications, spreadsheets, etc... to add flair to the paper. On the other hand, you can also use WriteNow's text capabilities to do your writing, then import that text into some other application for further manipulation. Your choice. For projects that are mostly text, many people use WriteNow as the gathering point for all of the other items rather than as just another tool.

To Insert Graphics

To insert a graphic into a WriteNow document, use the standard Copy&Paste routine. **Copy** the graphic onto the Pasteboard, then move to the WriteNow document. Position the cursor wherever you want the graphic located and choose **Paste** from the **Edit** menu. The graphic will soon appear. You may treat the graphic as any other single character; it can be backspaced over, centered, etc.

For more information

This CheatSheet is by no means comprehensive. For more information, see the manual in the back of the NeXT cluster titled "Applications." It is a white softcover book with the NeXT logo on the cover, located in the manual rack along with all of the other white manuals with the NeXT logo on the cover. :-) Try the left-hand side.

Yap

/NextDeveloper/Demos/Yap

Note: This CheatSheet assumes basic familiarity with PostScript.
See the PostScript CheatSheet for more information.

Yap is a tool for displaying PostScript on the screen, thereby avoiding having to print out each new creation you make. It's extraordinarily simple to use, so I will keep this short. In addition, you can watch a brief ShowAndTell script showing Yap in action that we have created.

The ShowAndTell Script (Demonstration)

And now for the most important thing you will ever be told. Before you start this demonstration, you MUST put the Yap icon in the second to lowest place in your dock. This is the slot directly above where the recycler normally is. If you forget to do this, you are hosed and will probably have to reboot your machine. It stinks, but that's the way it is. If you want an explanation of why this happens, see the ShowAndTell section of the Presentation Lab. (It is in the Above and Beyond section.) At any rate, you would be wise to save all your work before running this demonstration. Good luck!

Look for the demo in the **cs111/Labs/3-Graphics** directory. Double-click **Yap.st** to begin. (Don't forget the Yap icon in the dock!) This demo is geared for the Graphics Lab, but it contains what you need to get started no matter what you are trying to accomplish.

The 7 Easy Steps To Using Yap

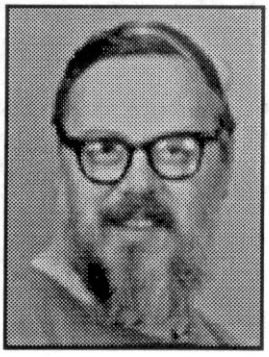
1. Start the application
2. Either:
   A. Open a PostScript file (Choose **Open...** from the **Document** menu), or
   B. Start a new file (Choose **New** from the **Document** menu)
3. Type and modify your PostScript code in the text window.
4. Choose **Execute** from the **Document** menu.
5. The image should appear in the Yap PostScript Output window.
6. If it looks the way you want it to, you are finished! Otherwise return to step 3.

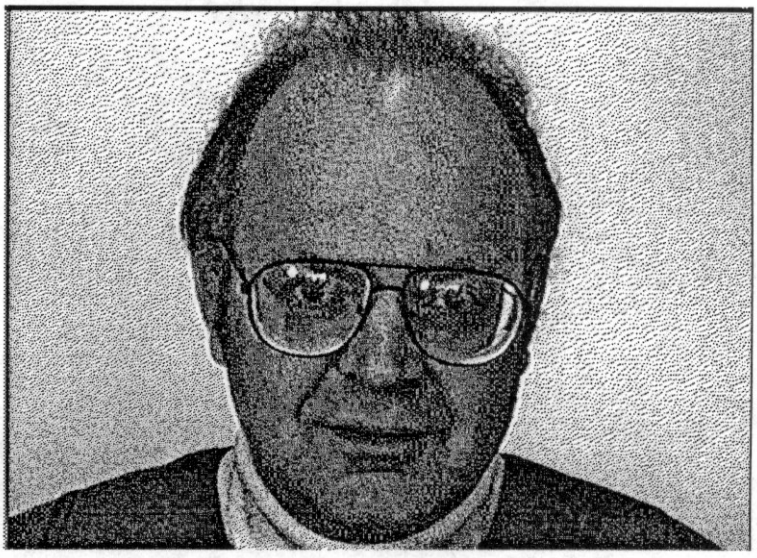
Using Yap With Other Applications

Yap is different than many other applications in that you cannot directly Copy&Paste the pictures you make. You must save your PostScript code in a file ending in ".ps", then go to a drawing application that can import PostScript. We used TopDraw, so will explain how to do it from there. Simply open a new document and choose **Import...** from the **Window** menu. Find and double-click your document. A window will appear; make sure the **Copy imported file into TopDraw file** and **Create new object for imported image** buttons are selected and click the **OK** button. Your document will appear as an object, ready for normal Copy&Paste operations.

COS 111

Handouts





Clockwise from upper-left: Alan Turing, Ada Lovelace, Donald Knuth, Dennis Ritchie