Milieu

METAFONT and LINUX: A Personal Computing Milieu

---

 tdunbar@vtaix.cc.vt.edu

Contents

A Fresh Start

This text aims to be a serious introduction to academic computing. It is intended to prepare students for a comprehensive introduction to computer science but is not, itself, such a course. Rather, it provides an introduction to programming and accustoms students to using complex programs. My approach is baroque rather than classical, exploratory rather than axiomatic, organic rather than architectural. In teaching such a course, even to well prepared and motivated students, the fundamental problem is to find meaningful examples of programs and tasks. This is my attempt to solve this problem in a way that provides a foundation for further studies.

In calling my approach baroque, I mean to suggest that we will be working with large, complex programs with many convolutions and interconnections. Not only are such programs and systems very common but also, in my opinion, they are more interesting. Life, itself, is very baroque and art *should* be imitative as it trys to create patterns and structures in order to cultivate this vitality. Rigid abstract structures developed for their own sakes, while modern, are boring.

On the other hand, this very complexity can be intimidating. Thus, students need help in learning how to 'explore the country' whether this is in finding paths to the heartland, backtracking out of dead ends, or planning for logistic support. We're more concerned with the search for fundamentals than in relying upon presently accepted building codes.

In fact, we're not really concerned with building stuff at all. Rather, this text just provides basic materials that can be used as resources as the course grows. This growth should be controlled by the interests and abilities of the students whom it is the responsibility of the teacher to lead. The course should follow the students who are following the teacher's lead.

The instruction is meant to follow an apprenticeship model more than a scholastic model. Hence, we want to look at code produced by skilled professionals (but which is still understandable and significant to beginning young students).

So, what region should we explore? Although there are many appropriate text-processing tasks (and suitable languages, e.g. AWK, for this programming), graphics-based tasks are more interesting and vivid to the students. However, none of the standard languages are really appropriate for such work. While one can make a special graphics package for, say, Pascal, the resources provided are typically not extensive enough nor sufficiently integrated into the basic syntax of the language.

METAFONT

Donald Knuth designed MetaFont as a fontmaking language to support his TEX typesetting language. Thus, all the graphics support one needs is built into the system. At the same time, MetaFont is a fullscale programming language with assignments, loops, scoping and parameter passing. Furthermore, MetaFont's equations provide an introduction to declarative programming languages and its macro facilities enable one to glimpse what is going on "inside" a programming language. Source code for MetaFont is freely available and free executables are available for personal computers.

	´0	´1	´2	´3	´4	´5	´6	´7	Γ
´00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	″0x Δ
´01x	Φ	Ψ	Ω	ﬀ	ﬁ	ﬂ	ﬃ	ﬄ	
´02x	ı	ȷ	`	´	ˇ	˘	¯	˚	″1x Θ
´03x	¸	ß	æ	œ	ø	Æ	Œ	Ø	
´04x	-	!	”	#	$	%	&	’	″2x Λ
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	″3x Ξ
´07x	8	9	:	;	¡	=	¿	?	
´10x	@	A	B	C	D	E	F	G	″4x Π
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	″5x Σ
´13x	X	Y	Z	[	“	]	ˆ	˙	
´14x	‘	a	b	c	d	e	f	g	″6x Υ
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	″7x
´17x	x	y	z	–	—	˝	˜	¨	
	″8	″9	″A	″B	″C	″D	″E	″F	

Computer Modern Roman 10 point

MetaFont is the primary programming language used in this text. I focus primarily on simple drawing; however making a complete font provides a good opportunity to present the complications of scale that accompany a large programming project. In addition, such a project enables one to appeal to a broader range of student talents and interests. The course works best, I think, if students' artistic skills are cultivated as well as their ability to think analytically. Different students will, of course, have different interests and abilities. Learning how to work together and make the best advantage of this is also very useful and provides a good introduction to software engineering and related issues. I hope this text will be of use for a course at our school, Dayspring Christian Academy, and that we will be able to include a sample of designing a font for the Wycliffe Bible Translators.

LINUX

However, besides a programming language, one needs an operating system. The system must support our programming well; in particular, one must be able to interactively write MetaFont code and look at the results simultaneously. Hence, some sort of graphical windowing system is essential. We will be working with many files and directories and will have need for a wide variety of support programs. In addition, we want to use a system that will not be 'outgrown' and which will also provide a foundation for using and studying operating systems themselves. Furthermore, in our school environment, a multi-user system is most efficient. All of this leads one to want a Unix workstation environment; however, the cost is prohibitive.

Rather, we use Linux, a largely POSIX-compliant multi-user operating system whose kernel is written by Linus Torvalds, a graduate student at the University of Helsinki. Linux is freely redistributable (under the GNU copyleft) and runs on INTEL 386 cpu/IBM AT bus architectures. It is sufficiently compatible with BSD and SYSV UNIX that all of the software we need compiles "out of the box." As with Unix, the word Linux is often used to refer not just to the operating system kernel itself, but also to the various systems and applications software commonly available with the system. Much of Linux's system software comes from the Free Software Foundation's GNU project. GNU Emacs is our standard text editor (and lisp interpreter, calculator, and symbolic math solver) and gcc is the standard C compiler. TEX and MetaFont, which provide a state of the art typesetting environment, are also a standard part of Linux and many other programs of interest to the academic community (graphical FEM, 2 and 3D plotters, GUI builders, etc.) are freely available. MIT's X Windows (X11R5 in its XFree86 version for 386 compatible machines) provides the graphical interface and various graphic utilities. Linux also has full TCP/IP networking support and there is a very active user/developer community available via the Internet.

All of the above software is free; not only does this save money, it makes technical support, bug fixes and program updates more readily available. Since the programs are free, and since they are widely available, this text can refer to more programs and speak more specifically and concretely than otherwise possible. Furthermore, since source code is available for all the programs, the same system that the students are first introduced to can also serve them well if they continue to courses in operating systems, language theory, systems engineering, etc.

MetaFont itself is very well documented by its author, Donald Knuth: *The METAFONTbook*, a comprehensive user's guide and description of the programming language and its standard macro package, *Computer Modern Typefaces*, attractively typeset source code for all the Computer Modern fonts together with proof samples for font designers, and *METAFONT: The Program*, which thoroughly documents the code of the program itself–a wide range of programming paradigms are discussed here for professional programmers.

MILIEU

The close association of MetaFont and Linux is specific to this text; many Linux users may use TEX and/or MetaFont little, if at all. The Linux Documentation Group is using TEX for their work. I hope to write in a way that will be useful even if one is only interested in Linux and also to provide useful information on Emacs, Emacs Lisp, and, perhaps, even Cweb. Nevertheless, my discussion of these various topics will always be centered around MetaFont-related tasks. **Warning:** I certainly don't claim any expertise regarding Linux, MetaFont, or any of the other programs I discuss. I teach English as a Second Language and Technical Writing; I'm very much an amateur programmer. I'm teaching myself as I write this; please join me in this endeavor. One reason that this text is publically available for ftp is to see if the same advantages that I claim for publically available source code also apply to manuscripts. The free flow of information is important to academic computing; most of the tension between academic computing and business computing is that the two areas have different aims and values. This text's focus is on academic computing. For example, suppose one wants to make a black and white logo. If it's just for some specific hardware, one could use a bitmap editor such as MacPaint. If output device independence is needed, then one could use Corel Draw, etc. In both cases, however, one needs a certain program to determine what the logo looks like and still one has no understanding as to the rationale behind the design. On the other hand, with MetaFont one has a program that anyone can read which precisely defines the logo in a device independent fashion and through which one can explain various design considerations. Furthermore the source code for the necessary programs to output the logo to various devices is freely available. This portability and archivibility is important to academic enterprises.

It is true that our system places restrictions on the hardware; I hope that this does not place too much of a financial burden on people that would otherwise be happy with our approach. In calling this text, *METAFONT and Linux: a Personal Computing Milieu,* the word personal refers not to PC's but rather to the effort to personalize one's computing system. There is an international 'community' involved in making such systems possible, hence the word *Milieu.*

Hardware

First we will discuss building a particular type of computer suitable for Linux. This will also provide the necessary information one needs if one prefers to buy the computer. There are, however, advantages to building it oneself: first and foremost, it is a good learning experience — it is interesting and also removes some of the mystery which prevents students from being comfortable with computers; furthermore, it allows one to have a system with precisely what one needs and can afford; finally, one can, perhaps, save a slight bit of money (especially if one can buy parts in bulk). We will be building a 486/33, ISA bus computer with 16 megs of ram (a 386/40 would also be fine). A 200 meg fixed disk is more than adequate; half that would be fine but wouldn't be as good a buy. Eight megs of ram is sufficient; however, with the memory designs commonly used, adding more memory may then mean scraping what one has.

Of course, how you will be using the system determines where you should spend your money. In deciding, one major concern should be the upgradability of the the system; don't put much money into something that will have to be scrapped later. Browse in the various trade magazines for product reviews; however, bear in mind that you will be using the hardware for a different OS than what the reviewers had in mind.

Suggestions

1) If your primary concern is that the system be fast, get a 486/66 with 16 megabits of ram (i.e. 4 4meg simms). If you plan on making major use of gcc, for example, frequent kernel recompiles, etc., this is a good choice. Although Linux uses memory very efficiently, programs such as gcc and X Windows use lots of memory and 8 megs isn't quite enough. If you will be doing significant graphical work, you may want to get one of the video accelerators such as ATI's Graphic Ultra or Orchid's Fahrenheit (make sure that your card is supported by the X server that you will be using, see Appendix 1). A Tseng ET4000 based card on a VESA local bus is another, but slower, choice (XFree86 includes special optimizations for the ET4000 chip).
2) If graphics are important, get a good monitor. Whatever monitor you get, if you are going to use X Windows at all, make sure the monitor can do 1024x768 extended VGA with an adequate refresh rate (at least 70Hz, in my opinion). Thus, the monitor needs to be multisyncing and to support a horizontal scan rate of at least 56khz (65khz would be better). Then, get the biggest monitor that you can afford. Don't believe monitor specs; the diagonal sizes that are advertised are usually optimistic. Frequently the image can't be stretched to fill the entire screen. At any rate, take a ruler with you and look at the monitor, if possible. While writing this text, I used a 14" Viewsonic 5e (about $420) and a 17" CTX 1760DF (about $820). If a good color monitor is too expensive, I suggest that you get the best monochrome monitor you can find.
3) Linux makes very efficient use of disk space so unless you are doing something like compiling the entire X11 distribution yourself, a big fixed disk is not needed. An 80–130 meg disk is fine; my personal opinion is that two small disks are better than one large disk. Thus my recommendation is: if needed disk space is $<= 130$, get one IDE fixed disk, if $<= 450$, get two. Linux also supports scsi disks–if you will also have scsi tape, cdrom, etc., this is a better option.

Exercise: You're building a 486-based computer for your boss; what component will have the most influence in conveying an impression of quality construction?

Software

Next, we need to get our operating system. There are various packaged collections of Linux kernel, systems and applications software available. The most widely available is the SoftLanding System (SLS) archive which is mirrored at various sites and has a home at tsx-11.mit.edu, dir pub/linux/SLS. Other packages include Lu's rootdisk/basedisk set (tsx-11.mit.edu, pub/linux/GCC), and LeBlanc's excellent MCC-Interim package (hpb.mcc.ac.uk, pub/linux).

We will, however, be using the Milieu package especially designed to go with this text. This package has a home at tsx-11.mit.edu, dir pub/linux/packages/TeX/Milieu. If you have use of a machine with Internet access, you might try `archie Milieu` to get a list of various other sites archiving this package.

The files we need break down to four categories: 1) basic kernel (`0.99.14`) and utilities (`gcc 2.4.5`), 2) MetaFont and TEX(`2.71 & 3.1415`), 3) basic X Windows (`XFree86 2.00`), and 4) GNU Emacs (`19.19`). For each of these categories there are certain essential files and then lots of enhancements and frills one can choose. The Milieu package was put together from various sources with the following constraints: the entire system (including an emergency dos disk) must fit on 5 3.5″ floppies and it must include all the programs and files needed for this text.

The Milieu package is certainly not comprehensive. For example, the TEX package doesn't include any LaTEX support, many basic file and system utilities are missing, and there is no support for X Windows programming. All these are widely available elsewhere. On the other hand, the Milieu package permits you to carry one of those little 5 disk boxes and change any willing 386/AT bus computer with an IDE fixed drive[1] into a multi-user workstation.

Installing Linux via the Milieu package

0 Get the Milieu files.

1 Make a dos bootable floppy (with format.exe and fdisk.exe on it) and put `bootlin.com, config.lnx, gzip.exe, rawrite.exe, and vmlinux` on it.

2 On four other dos floppies, put 1) `tex.tz`, 2) `xfree86.tz`, 3) `emacs.tz`, and 4) `rootdisk.z`, which will be overwritten during the installation. There are also some additional files you may want: `usrlib.tz,usrbin.tz` contain the GNU C compiler and miscellenious support files, `gnuplot.tz` contains a scientific plotting program (with MetaFont support), `i-calc.tz` has the Calc info files and `elispman.tz` is the GNU Emacs Lisp manual.

You may want another disk for the kernel source. You can, and may need to, arrange the disk contents differently, of course.

3 You will need at least a 40meg partition on the fixed disk. If necessary, backup your files now.

4 Copy `bootlin.com, config.lnx, gzip.exe, rawrite.exe, vmlinux`, and `rootdisk.z` to C:\. Now, do `gzip -d rootdisk.z`, then run `rawrite`; tell rawrite which floppy to write to and write `rootdisk` to the floppy that has `rootdisk.z` on it. Next, copy `config.lnx` to `config.sys` and reboot.

[1] Linux also supports SCSI drives, I just haven't compiled in support

5 Bootlin will boot the vmlinux kernel and prompt you for a location for the root filesystem. Insert the rootdisk floppy and respond appropriately. Once the system loads, login as root. You are now running Linux from the floppy.

6 Run `fdisk`, leave /dev/hda1 for dos, make /dev/hda2 for linux, /dev/hda3 for swap (say 10megs), and the rest as you please. On my maxtor 213, I have:

```
/book > fdisk
Using /dev/hda as default device!
Command (m for help): p
Disk /dev/hda: 16 heads, 38 sectors, 683 cylinders
Units = cylinders of 608 * 512 bytes
Device Boot Begin Start End Blocks Id System
/dev/hda1 * 1 1 213 64733 6 DOS 16-bit >=32M
/dev/hda2 214 214 426 64752 81 Linux/MINIX
/dev/hda3 427 427 470 13376 82 Linux swap
/dev/hda4 471 471 683 64752 81 Linux/MINIX
```

7 Now you can install to your fixed disk. Reboot and mount the rootdisk floppy. Use mkfs to put a minix filesystem on /dev/hda2 (or wherever). Also use mkswap to make swap and move rootdisk stuff to your new partition. For example:

```
mkfs -c -n30 /dev/hda2 64752
mkswap -c /dev/hda3 13376
swapon /dev/hda3
mount /dev/hda2 /mnt
cd /
for i in bin dev etc lib root tmp
do
tar rvf /mnt/t.t $i
done
cd /mnt; tar xvf t.t;rm t.t
```

8 Reboot with the dos floppy disk and format the dos partition (if you didn't make the partition active with linux's `fdisk`, use the dos fdisk) with `format c:/s`. Then restore files to your dos partition (including vmlinux and bootlin.com). Again use bootlin to boot the linux kernel but now use /dev/hda2 for your root filesystem. Untar the various other tz's (*note:* they are compressed with gzip, to which uncompress, etc. are linked), for example:

```
cd /
mkdir /fd
mount -t msdos /dev/fd0 fd
tar zxvf /fd/tex.tz
sync
umount /dev/fd0
```

and you're all set (mread could also be used; in that case, don't mount the dos floppy. That's faster; I just wanted to show that Linux can read/write dos filesystems). To automate the bootup, either: 1) install `lilo`, or 2) recompile the kernel or use rdev, Norton Utilities, etc to change the root device in vmlinux (for example: `rdev` vmlinux /dev/hda2) and then continue to use bootlin. You may want to put the floppy installation files on your dos partition. To remake the rootdisk.z, do

```
dd if=/dev/fd0 of=rootdisk bs=1024 % or /dev/fd1, as appropriate
compress -9 rootdisk; cp rootdisk.z /dos/linux/backup %for example
```

Operation System Basics

We just discuss here the bare minimum needed to use Linux. We want to start programming as soon as possible.

To get a brief overview of the files in the Milieu package that you have installed, enter `pushd /; du -b|more;popd`, this will show you the size (in bytes) of each directory. Press the spacebar to page through the file (b to backup one page, q to quit). Pushd is an internal command for the bash (Bourne-Again SHell) shell which saves the current directory on a stack and moves to the directory specified. Multiple commands can be given on a line by separating the commands with a ';'. Du gives you a listing of the Disk Usage and the '|' pipes its output into the `more` file viewer. Finally, we pop the directory off the stack and are back where we were. You might also want to sort the result by size by using `sort -nr` (numeric, reverse).

See Appendix 2 for a brief summary of usage for various executables in the package. In particular, I assume that you can use one of the text editors on your system. Speaking of editors, the Milieu package includes three: `ed` the line/script oriented editor[2], *jove* an emacs clone which is nice for quick little editing jobs, and GNU Emacs.[3] To view a reference card on Emacs, start X Windows by entering `startx` and then hold down the right mouse button and move down to the Xterm menu selection. Inside the xterm window, do

```
cd /usr/emacs/etc
sed s/columnsperpage=1/columnsperpage=3/ refcard.tex >emacs.tex
tex emacs
xdvi -s 2 -paper usr -offset 1in -l emacs
```

You can use the arrow keys to move around on the page (press d to move down, 4s to zoom out; if there were more pages, `n,p,3g` could be used to move to next, previous, and page 3).

See Appendix 4 if you have trouble starting X.

We will discuss customizing your `.emacs` file and also `elisp` programming in a later chapter. You should look at `/root/.emacs` (see Appendix 3) now, however, to see what default keybindings I've given the function keys, etc.

I usually log in as `root`; it teachs one to be careful! It is safer to log in as, say, `user` and just log in as root when necessary in another virtual console.

[2] See Appendix 1 of Kernighan and Pike's *The UNIX Programming Environment.*

[3] See Abrahams and Larson's standard UNIX reference, *UNIX for the Impatient.*

GNU Specifics

Basic information on `bash`, is available in Gnu Emacs info format. Start emacs and press the F1 key and then press i to start the info reader. The `features` and `history` menu entries discuss the bash shell. The `usrman.tz` package contains some important manual pages.

Many of the standard utilities included with Linux are GNU enhancements of standard Unix programs. The number of options can be useful, overwhelming, or amusing, depending on your point of view. If, for example, you enter `ls -h`, the program will respond:

```
/book > ls -h
ls:  unrecognized option ‘-h’
Usage:  ls [-abcdgiklmnpqrstuxABCFLNQRSUX1]
    [-w cols] [-T cols] [-I pattern] [--all] [--escape]
    [--directory] [--inode] [--kilobytes] [--literal]
    [--numeric-uid-gid] [--hide-control-chars] [--reverse] [--size]
    [--width=cols] [--tabsize=cols] [--almost-all] [--ignore-backups]
    [--classify] [--file-type] [--ignore=pattern] [--dereference]
    [--quote-name] [--recursive] [--sort=none,time,size,extension]
    [--format=long,verbose,commas,across,vertical,single-column]
    [--time=atime,access,use,ctime,status] [--no-color] [path...]
```

See any Unix reference for the standard options and Appendix 2 for GNU specifics. Better, experiment with it yourself: `pushd /root; ls -saFx;cd /usr/TeX/lib`
`ls -FCR --format=across --sort=size|more; popd` (press enter after each line). One can, of course, make little shell scripts for frequently used options. For example,

```
cat >/bin/lss
ls -lar --sort=size $*
d̂
```

`chmod 755 /bin/lss` (d̂, the unix end-of-file, is my notation for holding down the control key and pressing d). Emacs has an excellent file management macro package–inside emacs, press x̂d to access `dired` (ĥm for help). There are also X utilities, such as xdtm, for file management. Metrolink sells a Motif package for Linux and has generously made xmfm and other sample Motif applications freely available (it is statically linked so one doesn't have to have Motif to run xmfm).

MetaFont Basics

Let's plunge right in with some sample MetaFont code. The boldface section headings are what you type. The subsequent paragraphs provide more details; sometimes they explain the previous code or introduce the following code and other times they introduce topics that may be skipped on first reading.

Run MetaFont by entering (in a `xterm` window! Since we will want to view our work onscreen, a graphic display is needed) the command, `mf`:

%intromf.mf

```
/book > mf
This is METAFONT, Version 2.71 (C version d)
**
```

The web2c implementation of MetaFont and TEX we use adopts the convention that the executables look for similarly named formats. Thus, `mf` (which is just a symlink to `virmf`, the 'virgin' MetaFont binary) will try to load the `mf.base` format.

To make a new format called pmf.base, creat a file called pmf.ini containing: `input plain; input modes; input plocal; dump` and then run: `inimf pmf.ini`. Both `inimf` and `virmf` will try filename with an added default extension of `.mf` first when searching for files. Here `modes.mf` is Karl Berry's mode definition file for various output devices and `plocal.mf` contains some macros you want to compile into the default base. To use this format, `ln -s /usr/TeX/bin/virtex /usr/TeX/pmf`; then running `pmf` will load the `pmf.base` file (which should be in the `/usr/TeX/lib/mf/bases` directory.

Part of the plain.mf file that you may want to personalize is

```
newinternal screen_rows, screen_cols, currentwindow;
screen_rows:=400; % these values are changed in
screen_cols:=500; % the mode.mf file read in next
def openit = openwindow currentwindow
from origin to (screen_rows,screen_cols) at (-50,300) enddef;
```

The 'at' coordinate is the position of the top left corner of the MetaFont window (not the X window) with respect to the origin. Notice that, unlike the Cartesian 'at' coordinate, the 'from' and 'to' values are in `(row,column)` format in which the *first* coordinate is the number of rows *down* from the top (Knuth *claims* that "they seem appropriate when applied to screens" – the two conflicting conventions in one statement seem confusing). You may want to put different values into `modes.mf` or some local file, for example:

local.mf

```
input modes;
screen_rows:=500; screen_cols:=700;
pair screen_topleft;
screen_topleft:=(-0.1screen_cols,screen_rows-100);
def openit = openwindow currentwindow from origin
to (screen_rows,screen_cols) at screen_topleft;
enddef;
```

```
localfont:=CanonCX;
base_version := base_version & "mybase";
```

NOTE: Something similar to the `screen_topleft` variable is necessary to do screen-oriented programming with MetaFont. Although the base format includes the screen size, a program can not otherwise find out the position of the origin within the screen window. The MetaFont programs in this text will assume that `screen_topleft` has been assigned an appropriate value.

MetaFont and X Windows

You may want to run MetaFont itself with, say, `xterm -sb -geometry 80x25+0-0 -fn 9x15 -e mf &`. The `-sb` instructs xterm to use a vertical scrollbar. You control the X window that MetaFont will open by entries in your `.Xdefaults` file:

```
mf*geometry:  -0+0
mf*height:  500
mf*width:  700
```

Note: one should have `screen_rows=mf*height, screen_cols=mf*width`. Also, the mf resources specified above for X Windows refer not to the xterm window in which MetaFont is running, but rather to the window it will open to display graphics. See Appendix 4 for a sample .Xdefaults file and related information on customizing X Windows (and fvwm, xterm, etc.); we will have little else to say about X. **Note:** in fact, to save time & space, currently `fvwm` is the window manager and `rxvt` is also included as is the `ksh (pd)` shell.

**\relax;

Initially, MetaFont expects to read the name of a font file. The backslash says that instead, you will be using MetaFont interactively and the `relax` just does nothing (any other command would do as well).

To setup for the current mode (i.e., whether `proof` or `localfont`, etc. one uses:

mode_setup;

drawdot(0, 0);
showit;

Note: Normally, MetaFont inserts a blank line and then prints an asterick prompting for input. To make cut and paste operations in Emacs or `xterm` easier, the `mf` binary in the milieu package has been made from a modified `mf.web`.[1] In section 679 (*see METAFONT: The Program*), I've deleted `print_ln;` and use a space, instead of an asterick, as `prompt_input`'s argument.

The first time that `showit` is executed, MetaFont will call the openit macro which will cause the X server to open a window for the MetaFont drawings. Note that drawn

[1] Web is Knuth's programming system. A web source file is `tangled` to produce the program and `weaved` to produce its documentation. The web system provides excellent support for true "top-down" writing of programs.

objects must be explicitly shown. To change this default so that curves, etc. are shown immediately, one says:

screenstrokes; %I usually use this to begin with %instead of 'relax'

clearit;
showit; **drawdot** *origin* + (70, 10);

First macro

Since clearing isn't shown automatically, we have to show the clear; then we draw another dot. The origin can be referred to by name or value and one can do vector addition (and much else) with the points. There isn't a way to have `clearit` take effect immediately; instead, let's write our first macro:

def cc = **clearit**; **showit**;
enddef;

Thus when we enter `cc;` MetaFont translates it into `clearit; showit;;` so the screen is erased. Statements in MetaFont end with a ';' (extra semi-colons are fine since the empty statement is a statement in MetaFont) and white space is ignored.

Equations

$x_1 := x_2 := 50;\ y_1 + y_2 = 100;\ y_1 = 60 + y_2;$ **drawdot**(x_1, y_1);

In MetaFont only numeric variables need not be declared. Variables can be *assigned* values with `:=`, as in Pascal. However, one can also declare systems of linear equations* and leave their solution to MetaFont. This declarative nature of MetaFont leads to a different style of programming than in procedural languages such as `C`.

Variables can have their values *reassigned* with '`:=`', we don't use equations to do this since an inconsistent set of equations would result. Unlike assignments, equations can have unknowns on both sides.

show z_2; **path** *tst*, *p*[], *p*[]*line*; **pair** *w*[]; $z_3 = (20, 40)$;

Variable types such as `path` or `pair` (i.e., point) must be declared. Other variable types are `boolean, string, pen, picture, transform`. Variable names consist of three parts: the essential part is the `tag`, then there may be a numeric subscript which may be followed by another tag. The subscript part, which we declare with `[]`, enables us to creat arrays of related variables. The subscript (possibly empty!) together with any following tag is called the `suffix` of the original tag.

Why didn't the `z3` variable above have to be declared? Well, z is in fact a macro in MetaFont which converts z$ into (x$,y$) for any suffix $. This is convenient since we will be working with points a lot.

---

* Via it's `solve` macro, MetaFont also has a limited ability to solve nonlinear equations. Namely, if one can consider one of the variables to have a fixed value so that only one of the resulting equations is nonlinear (in the remaining variables), then the original ($N \times N$) system can be solved.

```
pickup pencircle scaled .4pt; cc; drawdot origin;
```

One has a wide variety of pens available; we want a fine one now. The dimensions that one gives are converted into pixel units for the currently defined mode. By default, MetaFont is in 'proof' mode which has 36 pixel 'dots' per point. If one wants a 'real' point (i.e., one 72.27th of an inch), one uses pt#. These 'sharped' units are interpreted in a device independent fashion and are usually used during initial mode/font setup. If no units are given, the number is interpreted directly as pixel units. We will discuss mode setup, etc. later. The units which MetaFont knows about are the printer's point (pt), pica (pc=12pt), inch (in=72.27pt), big point (bp=1.00375pt), centimeter (cm=28.45276pt), millimeter (mm=2.84528pt), didot point (dd=1.07pt), and cicero (cc=12.8401pt)–note that this last unit isn't available right now since we have redefined `cc` to be a clearscreen macro. The printer's point is the basic unit, as you can see if you do a `show in#;` command.

The following code is what I use in my customized version of modes.mf. It is for writing the screen to a character in order to illustrate this code. The `charscreen` macro is basically just a `beginchar` without the default clearings. Don't worry about it for now.

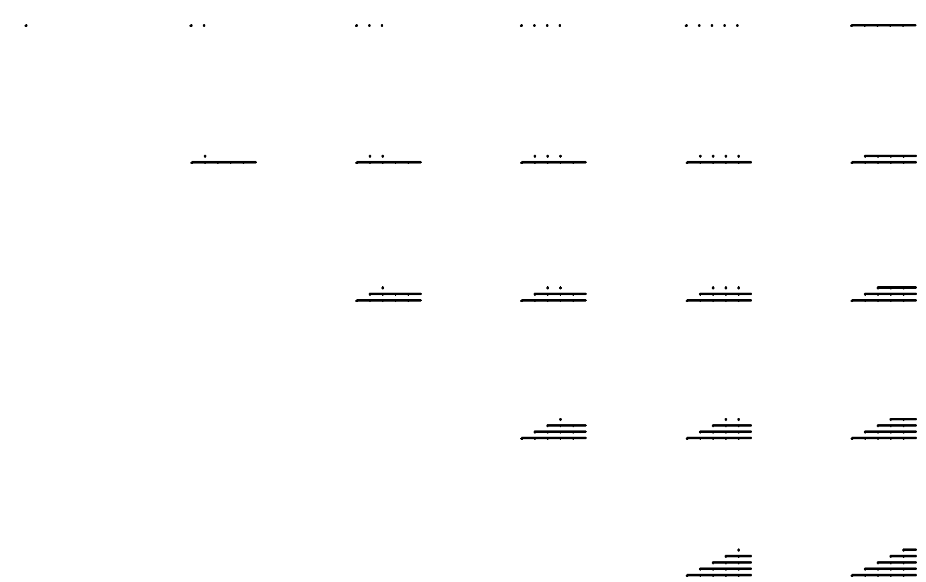
```
picture picvar; numeric charnr; charnr := ASCII("A") - 1;

def sscreen(expr w_sharp, h_sharp, d_sharp, c_scale) =
  picvar := currentpicture;
  charscreen(incr charnr, w_sharp, h_sharp, d_sharp)
    currentpicture := currentpicture scaled c_scale;
  endchar;
  currentpicture := picvar;
enddef;

def charscreen(expr c, w_sharp, h_sharp, d_sharp) =
  begingroup
  charcode := if known c: byte c else: 0 fi;
  charwd := w_sharp; charht := h_sharp; chardp := d_sharp;
  w := hround(charwd * hppp); h := vround(charht * hppp);
  d := vround(chardp * hppp);
  charic := 0; scantokens extra_beginchar;
enddef;
```

Watching loops

Now to get back to our introduction: MetaFont has various commands that enable one to repeat similar tasks. For example:

```
for i = 1 step 20 until 100:
  for j = i step 20 until 100:
    drawdot(j, i/2); sscreen(25pt#, 20pt#, 0, 1);
  endfor;
  draw(i, i/2) .. (100, i/2); sscreen(25pt#, 20pt#, 0, 1);
endfor;
```

The above code produces the following figures:

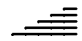
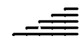


Watching the picture as this code executes is more valuable than these remarks. In fact, that is one of the reasons I'm using MetaFont; the visuals assist one's first encounter with programming. If you want to slow things down to look at what happens at each step of the loops, put `s:=readstring;` between the statements. This just sits and waits for the enter key to be pressed. Note that a colon (not a ;) ends the `for` statement conditions. The inner loop draws a row of dots whose height and starting x coordinate are controlled by the outer loop which also draws a line over the dots. Each row of figures above shows one iteration through the outer loop.

MetaFont's `for` statement can execute 0 times if the condition is not initially true. At first, you might think that's pointless; however, it enables one to program for the general case and not have to deal specially with certain cases in which the loop shouldn't execute. For example, we might have a loop: `for i=curve[k] step -node_step until limit[k]:` which applies to many path nodes, and on only some of which do we want the loop to operate.

```
undrawdot(41, 21/2); sscreen(25pt#, 20pt#, 0, 1);
undrawdot(41, 21/2); sscreen(25pt#, 20pt#, 0, 1);
```

The undrawdot must be done twice, once for the drawdot and once for the draw which drew a line on top of it.

Pixels in the current picture have integers associated with them; if the pixel value is greater than zero, it is blackened. In the process of drawing, and undrawing, pixel values may be increased or decreased by more than one. The macro `cullit` will change all negative values to zero and all positive values to one. If you give the command `show currentpicture`, the log file will include the current picture's definition. Note that only the places where pixel values change are recorded.

cc; **draw** z_2 .. z_3; $z_4 = .5[z_3, z_1]$;

We can draw lines through points and we can specify a point as being a certain proportion of the way along a line segment (`z2=0[z2,z1]` and `.99[z2,z1]` is close to z1) by using what is called the mediation (or: of-the-way) function.

Declarations

$z_5 = whatever[z_1, z_2]$; $z_4 - z_5 = whatever * (z_3 - z_2)$; **draw** z_4 .. z_5;

An anonymous numeric variable, `whatever`, which assumes a different value with each use, can be used to define properties. The first equation above says that z5 is on the line defined by z1 and z2 (note that `2[z1,z2]` is not in the segment `[z1,z2]`). The second equation specifies that the segment (ie, *vector*) from z2 to z3 has the same direction as the line from z5 to z4 (in MetaFont, one can't *multiply* unknowns, so one can't use `whatever*(z4-z5)`). These two equations define the point z5 so that we can then draw from z4 to z5. This style of declaring equations which define the properties of what is to be drawn is typical of MetaFont.

We've just covered a great deal of material that should be worked on. If you read a certain piece of code and wonder—"What would . . . do?"—try it out! We're using MetaFont precisely because it encourages such experimentation. If your code is invalid, has a *bug*, no problem! Just press <`enter`> until you get the prompt back, press `h` for help, or if all else fails, press `x` to exit MetaFont and start again. If you are reading this on your own, I assume this isn't your first programming language so probably only the material in *The METAFONTbook,* chapter 13 on filling need be read now.[1]

Editing

This is perhaps a good time to discuss entering code in MetaFont. As you may have learned by now, one uses the `DEL` key to erase the previous character when running `mf` in an `xterm`. The `^h` character (i.e., the control-h character) which `BACKSPACE` inserts confuses

[1] See Donald Knuth's *The METAFONTbook.*

MetaFont. The standard `xterm` cut and paste operations can be used: use the left mouse button and drag over a line of MetaFont code (either in the xterm mf window or in, say, emacs), then move to the `mf` prompt and click the middle mouse button (or both buttons if you are using three button emulation) to paste the line. This compensates a bit for MetaFont not having its own editor build in. An even better solution, described later, is to use an Emacs shell.

One might start with `**\screenstrokes; input mymacs;` or even load your macros into a precompiled base. If you set `pausing:=1`, MetaFont will pause after each line of the input file; in addition, you can insert statements, for example `show` commands, after the line. **Pausing** is a very useful feature when writing a program.

Let's get back to our introductory overview to MetaFont. First, I declare x and y as numeric arrays so that later we can redefine `z1`, say. Declarations erase (unbind) previous values of their variables.

numeric $x[]$, $y[]$;

Curves

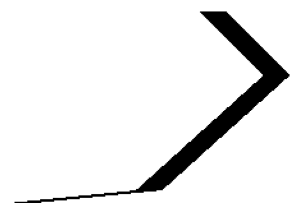
draw$(0, 0) \mathinner{..} (100, 10) \mathinner{..} (200, 100) \mathinner{..} (150, 150)$;
undraw$(0, 0) \mathinner{..} (100, 10) \mathinner{..} (200, 100)$;
$sscreen(30pt^{\#}, 30pt^{\#}, 1pt^{\#}, 2)$;

MetaFont draws a smooth curve through the points. For now, pretend that MetaFont takes three adjacent points and replaces them by the unique circular arc through them. When there are more than three points on the curve, the resulting arcs are blended in a smooth fashion, keeping all the points on the curve. To be more precise: MetaFont uses the control points (some of which it may compute itself) as knots for cubic spline approximations where special attention is paid to minimizing discretization errors. The curvature of the curve at any point depends on *all* the control points in the curve as the above code illustrates. However, as the knots step away from a given point, their influence is roughly halved. Parts 18 and 19 of *METAFONT: The Program*, should be consulted for the exact algorithms.

$npen := savepen$;
pickup *penrazor* scaled $2pt$;
draw$(0, 0) \mathinner{..} (100, 10) \mathinner{..} (200, 100) \mathinner{..} (150, 150)$;
$sscreen(30pt^{\#}, 35pt^{\#}, 5pt^{\#}, 2)$;

If desired, one can save the current pen to a variable (actually just a reference number) before picking up another.

tst := (0, 0) .. (100, 10) -- (200, 100) .. (150, 150);
cc; **draw** tst;
draw tst --- cycle;
undraw tst -- cycle;
$sscreen(30pt^{\#}, 30pt^{\#}, 0pt^{\#}, 2)$;



It is convenient to assign curves to variables to save typing. In actual font production, this isn't done so often.

When drawing curves, '--' specifies a straight line segment and '---' specifies a straight segment that smoothly connects at its ends even though the radius of curvature may be tiny.

cc; **draw**(0, 150);

We can use `draw` to draw points; however, the special routine `drawdot` is designed to better cope with the rounding problems related to printing with raster devices rather than analog devices such as an ink pen. Of course if the discrete values of a digital device are fine enough, it will seem continuous. However, MetaFont is designed to have a high degree of device independence and to make the best of low resolution devices.

pickup pencircle scaled .4pt;
$penpos_1(30, 110)$; $penpos_2(20, 90)$; $penpos_3(50, 70)$;

The first argument to `penpos` is a length and the second is a direction (in degrees). This specifies the type of pen that we are using at a given point. Actually, the number suffix to penpos is the first argument, giving which point the penpos is for. For example, `penpos1(30,110)` says that at point (x1,y1), pickup penrazor scaled 30 rotated 110.

$z_1 = (50, 100)$; $z_2 = (120, 80)$; $z_3 = (200, 130)$;
cc; **penstroke** $z_{1e} \mathinner{..} z_{2e} \mathinner{..} z_{3e}$;
$sscreen(40pt^{\#}, 20pt^{\#}, -10pt^{\#}, 2)$;

Once the pens and positions have been specified, one can fill inside the edges of the resulting stroke with `penstroke`. This enables MetaFont to simulate broadnibbed pens.

cc; **draw** flex(z_1, z_2, (150, 150), z_3);
charnr := ASCII("a") − 1; *sscreen*($40pt^{\#}$, $15pt^{\#}$, $0pt^{\#}$, 2);

MetaFont's `plain` base predefines various functions especially for drawing. For example, `flex` takes three or more points where the curve has direction `z3-z1` (ie, from z1 to z3) at the specified points in the middle. Speaking of directions, when defining a curve, you can put a direction specification before or/and after a point. For example, one can say `draw origin..{(2,1)}(90,90){dir -30}..(200,0)`. If there is only one dir for a point, MetaFont will put the same dir both before and after the point. Any pair variable can be used to give the direction; the `dir` function takes a numeric argument which it considers as the number of degrees and returns a unit vector in that direction. There is also an `angle` function which takes a point, i.e., a vector, (not necessarily of length one) and returns its direction in degrees.

numeric *ht*, *wt*;
cc; $ht = .7wt$; $wt = 200$;
p_1 := *fullcircle* xscaled w yscaled h shifted $(.5wt, .5ht)$;
draw p_1;
p_2 := $(.2wt, .5ht)$\{dir −30\} .. $(.8wt, .5ht)$\{dir 30\}; **draw** p_2;

A Donut

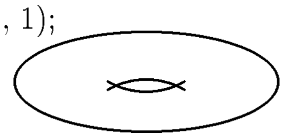
We want to draw a donut (where `w` controls the size) and to conclude this MetaFont overview with some of the functions introduced in chapter 14 of Knuth's book.

%Various transforms are predefined.

p_3 := p_2 reflectedabout($(0, .5ht)$, $(100, .5ht)$);
show directionpoint *right* **of** p_2;

The numbers that MetaFont prints out in the xterm window tell us where the curve p2 is horizontal. We use this so we can draw the upper part of the donut's hole in a way that doesn't explicitly depend on the dimensions of the donut.

s := ypart **directionpoint** *right* **of** p_3 − ypart **directionpoint** *right* **of** p_2;
p_4 := p_3 shifted $(0, -2s/5)$; **draw** p_4;
sscreen($30pt^{\#}$, $10pt^{\#}$, $0pt^{\#}$, 1);



undraw p_4; $z_4 = p_2$ intersectionpoint p_4;

We draw p4 just to test out if it is ok. Then we use it to find the point on p2 where we want the upper curve to start.

$z_5 = z_4$ reflectedabout$((.5wt, 0), (.5wt, 100))$;
$p_5 := (.2wt, .5ht)\{\mathrm{dir}\ {-30}\} \mathinner{\ldotp\ldotp} z_4 \mathinner{\ldotp\ldotp} (.8wt, .5ht)\{\mathrm{dir}\ 30\}$;

The curve p5 traces the same points as p2, the bottom curve of the donut's hole. We just added a point to the curve so that we can get the direction there:

$z_6 =$ **direction** 1 **of** p_5;

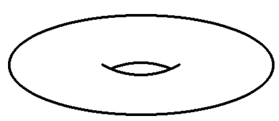
The nodes of a curve are naturally numbered starting with zero; consider a bug moving along the curve: the above `direction` macro gives the direction it is traveling at the second control point.

We could have gotten the above information in other ways, of course. For example

```
  show p2 intersectiontimes p4;
>> (0.1127,0.1127)
  show point .1127 of p2;
>> (52.85632,63.56888)
  show z4;
>> (52.85632,63.56886)
  show direction .1127 of p2;
>> (38.85165,-16.60434)
  show direction 1 of p5;
>> (39.15742,-16.69167)
  show angle direction 1 of p5;
>> -23.08713
  show angle direction .1127 of p2;
>> -23.1408
```

As the above illustrates, due to rounding errors one can get different answers if the computation is done differently. Furthermore, the effect of small rounding errors may be magnified later on. At any rate, let's draw the top of the donut hole:

show angle z_6;
draw $z_4\{\mathrm{dir} - \mathrm{angle}\ z_6\} \mathinner{\ldotp\ldotp} z_5\{z_6\}$;
$sscreen(30pt^{\#}, 10pt^{\#}, 0pt^{\#}, 1)$;



Usually, we consider `pair` variables as points; however, they can also be considered as vectors which give directions, or even as complex numbers (`z1 zscaled z2` does complex multiplication).

Finally, we draw the upper curve which we have designed so that if it were reflected across its end points, the resulting curve would overlay the lower curve of the donut's hole.

As I hope this overview has shown, MetaFont is a high level language well designed for doing two dimensional black and white glyphs. All beginning programming concepts can be illustrated via MetaFont code, as well as more advanced concepts. The following sections present a through introduction to programming via MetaFont. But, for now, let's go get some real donuts! To exit MetaFont enter:

end

Dots, etc.

First, let's set up our programming environment. Start X (`startx`) and then launch Emacs (with the default Milieu setup, press the right mouse button). Press `^X2` to split the Emacs screen and then drag one of the corners (put the cursor over the corner and hold down the left mouse button as you move the mouse cursor) to make the Emacs window higher. To change Emacs font, hold down the control key and press the right mouse button to raise a menu of fonts from which to choose.

Enter `^[x shell` (hit the escape key and then x) to run an interactive shell and start MetaFont by entering `mf`.

I find running MetaFont inside Emacs is more convenient than using an xterm. Emacs' editing facilities are useful and compensate for MetaFont's lack of a builtin editor (*Note:* in writing this, I have a TEX buffer and a `shell` buffer and cut and paste between them. To copy, left-click on the first character and then right-click just after the last character you want to copy. The region currently selected will be highlighted. To delete the region, right-click again. To paste the region, left-click where you want it and then middle-click to paste.

You can, of course, also use Emacs keyboard commands. The main point is that the full power of the Emacs editor is available not only in the TEX text buffer, but also in the MetaFont shell buffer). Various macro packages provide helpful tools; in particular, the Calc macro package is a powerful calculator/symbolic mathematics system for Emacs.

Respond to `mf's **` prompt with \`relax`.

screenstrokes;
show *mode*;
>> mode **mode_setup**; **show** *mode*;
>> 1

Initially, mode doesn't have a value although it has been declared numeric. When `mode_setup` is run, if `mode` doesn't have a value, it is set to `proof` mode.

show *mode_name*[*mode*]; %proof_
drawdot *origin*; **showit**;
numeric *mode*; *mode* = *CanonCX*; **mode_setup**;
drawdot *origin* + (50, 50); **showit**; **show** *pt*; %4.1511

numeric *mode*; *mode* = *proof*; **mode_setup**;
drawdot *origin* + (50, 50); **showit**; **show** *pt*; %36

When `mode_setup` is called, among other things, it sets the device dependent values of the various units. In other words, units like `mm` or `in` are just numerics which give the number of pixels in that unit for the currently defined device (i.e., mode). The default pen is .4pt in diameter; thus, in proof mode, a dot is $round(36*.4) = 14$ pixels wide. CanonCX is for 300dpi LaserJets, etc. where the corresponding dot is only $round(4.1511*.4) = 2$ pixels wide. To reset `mode`, clear the previous equation by declaring `mode` as `numeric`.

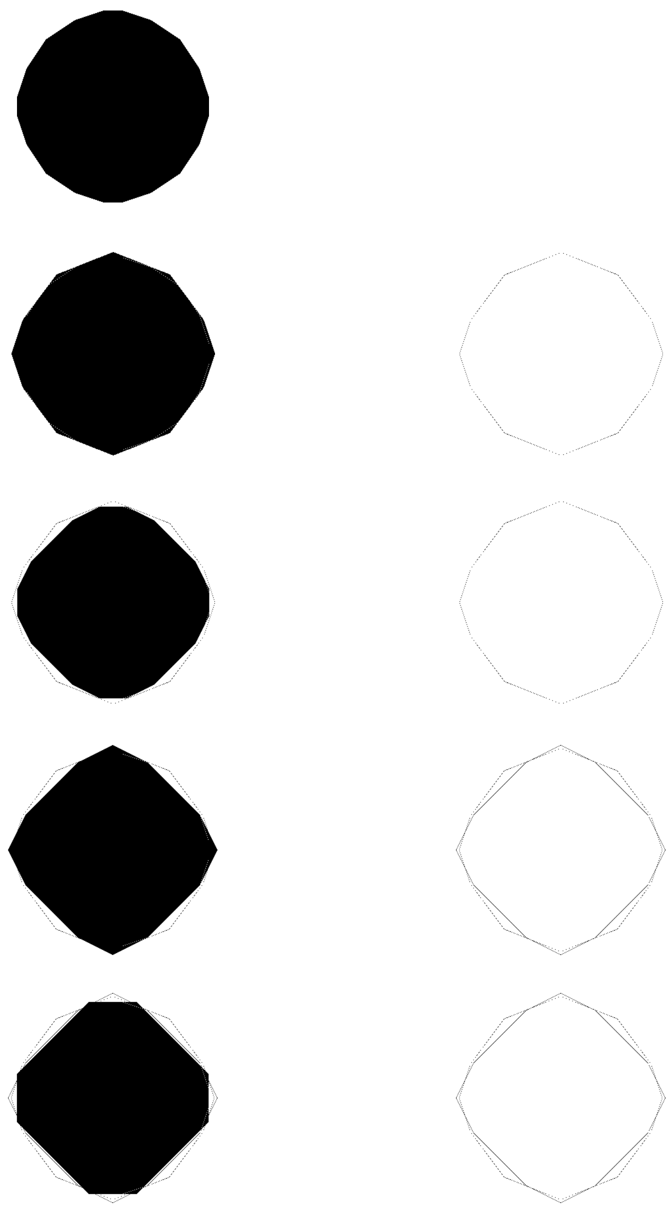
To see the names of the various modes, one can do:

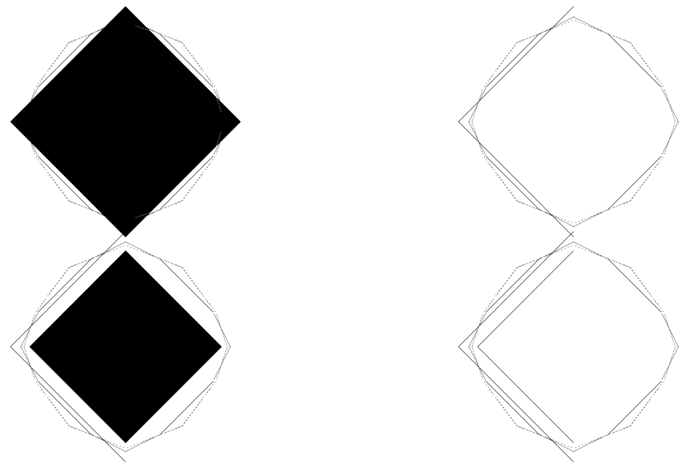
for $i = 1$ **upto** *number_of_modes*:

message "`mode `" & decimal i & "` is `" & *mode_name*[i];
endfor;

Lets look at some pens:

string s;
for $i = 10$ **step** -1.5 **until** 1:
  $a := 300/i$; %want pens about the same size
  pickup pencircle scaled i; **pickup** *currentpen* scaled a;
  drawdot(200, 100);
  undraw(200, 100);
endfor;





We want to look at small pens; thus we first create the pen to define its shape and then magnify it. As you can see, MetaFont's pencirle isn't really a circle, but rather a polygonal approximation to one. When one is outputing to a particular discrete device, the resulting rounding errors are reduced by using such convex polygons. Any convex polygon can be used as a pen (and vice-versa); MetaFont's plain format predefines `penrazor`, `pensquare`, `pencircle` which can be scaled in `x,y` directions separately. Also notice that `drawdot (x,y)` and `draw (x,y)` aren't exactly the same; again, discretization errors make it useful to have a special routine to draw dots. This careful attention to problems of discretization is one of the three main differences between MetaFont and Postscript.

end

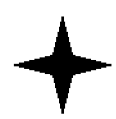
Suppose we want a non-convex dot? Let's make a macro to print a 'star' dot.

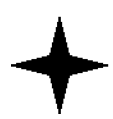
mode_setup; **screenstrokes**; **path** $p[]$, **stardot**; $p_1 := (0, 10) \mathbin{-\,-} (2, 2) \mathbin{-\,-} (10, 0)$;
def cc = **clearit**; **showit**; **enddef**;
for $i = 2, 3, 4$:
  $p[i] = p[i-1]$ rotated 90;
endfor;

%The end of one curve must be the start of the next.

$p_5 := p_4 \mathbin{\&} p_3 \mathbin{\&} p_2 \mathbin{\&} p_1$ .. cycle;

fill p_5 scaled 4 shifted (50, 50);
fill p_5 shifted (50, 50) scaled 4;
$sscreen(50pt^\#, 30pt^\#, 1pt^\#, 2)$;





These two aren't the same! Unlike Greek geometry, our geometry does not take place 'in' a void; rather, transformations are done with reference to a coordinate system.[1] The first takes $(2, 2) \mapsto (8, 8) \mapsto (58, 58)$ while the second takes $(2, 2) \mapsto (52, 52) \mapsto (208, 208)$.

[1] Just as our lives in Christ should be ordered by the fundamental coordinate system: "Take up your cross and follow me."

Now we can define:

stardot := p_5 scaled .1; **cc**;
fill stardot scaled30 shifted (100, 100);

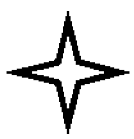
The -- operators above can't (unless we change p5 to `p5:=p4..p3..p2..p1..cycle;`) be replaced with --- as the following example shows:

pickup pencircle scaled .1pt;
numeric x, y; **path** p[];

$z_1 = (0, 10)$; $z_2 = (2, 2)$; $z_3 = (10, 0)$; $z_4 = (2, -2)$; $z_5 = (0, -10)$;
$z_6 = (-2, -2)$; $z_7 = (-10, 0)$; $z_8 = (-2, 2)$;

p_1 := z_1 --- z_2 --- z_3 --- z_4 --- z_5 --- z_6 --- z_7 --- z_8 --- cycle;
p_2 := z_1 --- z_2 --- z_3 & z_3 --- z_4 --- z_5 & z_5 --- z_6 --- z_7 & z_7 --- z_8 --- cycle;

fill p_1 scaled 5;
message "`p1 scaled 5`"; **show** *currentpicture*; **cc**;
fill p_2 scaled 3;
message "`p2 scaled 3`"; **show** *currentpicture*;
fill p_1 scaled 5;
sscreen$(50pt^\#, 20pt^\#, 1pt^\#, 2)$;



%Surprising, yes?

end;
The log file shows: ;

```
p1 scaled 5
>> Edge structure at line 32:
row 129:  | 100- 100+
row 128:  | 100- 100+
row 127:  | 99+ 101-
row 126:  | 99+ 101-
.
.
p2 scaled 3
>> Edge structure at line 34:
row 29:  | 0- 0+
row 28:  | 0- 0+
row 27:  | -1- 1+
row 26:  | -1- 1+
.
.
```

MetaFont decides whether a pixel is to be filled by keeping track of 'edge structures'– only changes in the fill number are saved. Pixels with a fill number greater than zero are blackened.

The `p2` curve has tiny 'loops' that confuse MetaFont. **Note to author:** figure out exactly what's going on here!

A First Program: Tic Tac Toe

It's high time to write our first MetaFont program. When I was a child, I saw an exhibit at the Smithsonian (or maybe it was the 1960 Worlds Fair in New York) in which one played tic-tac-toe against a computer. For our first program, let's write a program so two people can play this on-screen without needing paper.

As mentioned earlier, the default `plain.mf` file includes no way for a MetaFont program to find a point's position on the screen. We will assume that any user of this program is using a base format in which the pair `screen_topleft` has been assigned the coordinates of the top left point of the `mf` screen.

```
                                                              %ttt3.mf
mode_setup; if not known screen_topleft:
message "screen_topleft undefined, defaults to (-50,300)";
pair screen_topleft; screen_topleft := (-50, 300); fi;
```

A program should cope with inproper inputs, preconditions, etc., gracefully. For example, as said above, I assume that `screen_topleft` has been compiled into `mf.base`. If not, I set it to the value appropriate for the standard `plain.mf` which is used to creat the base file. The message is just meant to give the user a clue, more information could well be given.

Our first task is to draw the '#' we're going to play on. First, let's change the origin's position: currently it is `-xpart screen_topleft` over from the left edge and `screen_rows - ypart screen_topleft` pixels up from the bottom edge. To move the origin close to the bottom left corner, we can set:

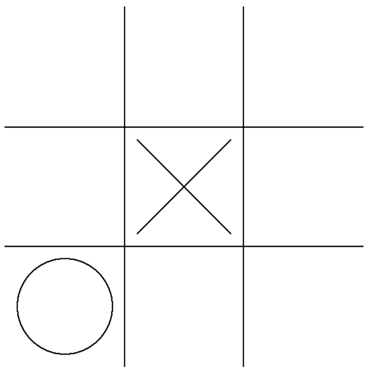
```
currenttransform := identity shifted (20 + xpart
    screen_topleft, 20 − screen_rows + ypart screen_topleft);
ttt := min(screen_rows, screen_cols) − 40;
```

Now, let's draw the 'board' using a for loop. We'll worry about generalizing later.

```
for i = 1, 2: draw(0, i ∗ ttt/3) .. (ttt, i ∗ ttt/3);
draw(i ∗ ttt/3, 0) .. (i ∗ ttt/3, ttt); endfor;
                                %We will want to do some experimenting now so
                                               %first save the current picture:
picture tttboard; tttboard := currentpicture;
path ttto, tttxa, tttxb;
ttto := fullcircle scaled (.8 ∗ ttt/3) shifted (ttt/6, ttt/6);
draw ttto;
```

The predefined `fullcircle` is a circle of diameter 1 centered at the origin. We shift it to the center of the bottom left square after scaling it to an appropriate size.

```
tttxa := (20, 20) .. (ttt/3 − 20, ttt/3 − 20);
tttxb := (20, ttt/3 − 20) .. (ttt/3 − 20, 20);
draw tttxa shifted (ttt/3, ttt/3);
draw tttxb shifted (ttt/3, ttt/3);
sscreen(80pt#, 40pt#, 2pt#, 1);
```



Now we're ready to actually play the game. The users need to be able to 1) start a new game, 2) make a move, and 3) undo a move. Let's have the user enter two digits to determine this:

message "enter 00 to quit, 10 to start new game";
message "enter 11 to place mark at bottom left square";
message "enter -33 to erase mark just placed at top right square";

The erasing is there so the user won't get frustrated by careless mistakes. The program will alternate the marks, starting with an 'X'. I purposely use Cartesian coordinates for the position; some people might prefer `(row,column)` notation.

boolean *tttmarkx*; *tttmarkx* := **true**; **string** *s*;
screenstrokes; *currentpicture* := *tttboard*; **showit**;
forever: **message** "Move? "; *s* := **readstring**;
  move := round(**scantokens** *s*); **exitif** *move* = 0;

We make a general loop in which we get input from the players and then do the appropriate action. The `numeric` is rounded to an integer; more error checking could well be done here. Non-count loops are done by using `forever: ...exitif` < *boolean* > ... `endfor;` loops.

  if (*move* = 10) or (*move* = 1): *currentpicture* := *tttboard*;
    showit; *tttmarkx* := **false**;

Here we start handling the `move` cases (included an 'undocumented' one). After each 'move', we will change the mark used so the `tttmarkx:=false` is needed here since we're not actually not making a move.

  elseif *move* > 0:
    move := *move* − 11;

We adjust `move` so that its digits give how much to shift the mark (in units of `ttt/3`).

    if *tttmarkx*:
      draw *tttxa* shifted ((*move* div 10) $*$ *ttt*/3, (*move* mod 10) $*$ *ttt*/3);
      draw *tttxb* shifted ((*move* div 10) $*$ *ttt*/3, (*move* mod 10) $*$ *ttt*/3);
    else:

draw *ttto* shifted $((move \text{ div } 10) * ttt/3, (move \text{ mod } 10) * ttt/3)$;
fi; %end if tttmarks

Since I'm writing the program using Emacs, the redundancy above needs little extra typing. It does, of course, waste CPU time so when we rewrite this, we'll want to change this.

else:
$move := \text{abs } move$;
$move := move - 11$;

Recall that when we make a move the mark type flag is changed; thus, here we want to erase the *previous* mark type so we say:

if not *tttmarkx*:
undraw *tttxa* shifted $((move \text{ div } 10) * ttt/3, (move \text{ mod } 10) * ttt/3)$;
undraw *tttxb* shifted $((move \text{ div } 10) * ttt/3, (move \text{ mod } 10) * ttt/3)$;
else:
undraw *ttto* shifted $((move \text{ div } 10) * ttt/3, (move \text{ mod } 10) * ttt/3)$;
fi;
fi; % end if (move=10

Now we flip the mark type and go back through the loop. We only end the program if `exitif` gets called.

$tttmarkx := \text{not } tttmarkx$;
endfor; **end** %quit if we exit the for loop

Program Refinement

There are several things we could do to improve the TicTacToe program. Several redundant calculations can be eliminated; while the current program is fast enough, there's no good reason to waste computer resources any more than paper. In addition, the current program can not handle incorrect user input–not only 'a3' but even '45' isn't handled smoothly. However, as far as potential users are concerned, probably the most important improvement would be to adapt the program to work with, say 5x5, or other size boards.

Instead of taking the board size to be 3, this time let's prompt the user for a size and use that (we'll continue using a square board). As the size gets bigger, screen size becomes more critical so we change the `topleft` default.

%tttn.mf

if not known *screen_topleft*:
  message `"screen_topleft undefined, (-50,screen_rows-100) used"`;
  pair *screen_topleft*; *screen_topleft* := (−50, *screen_rows* − 100);
fi;
currenttransform := *identity* shifted (20 + xpart *screen_topleft*,
  20 − *screen_rows* + ypart *screen_topleft*);
ttt := min(*screen_rows*, *screen_cols*) − 40;
message `"Boardsize? (3,...,9) "`; **string** *s*; *s* := **readstring**;

We won't insure that the user in fact inputs a number. However, we do round it to an integer. Then, if it's too small, we replace it with 3 and if too big, we replace it with 9.

boardsize := min(9, max(3, round(**scantokens** *s*)));
ss := *ttt*/*boardsize*;

If a larger boardsize were needed, it would probably be best to change the input scheme. In the following, we just have to replace `ttt/3` with `ss`, the *squaresize* (use Emacs' `query-replace`). There are a few other places we have to change also, such as the for loop for the board:

m := 3/*boardsize*; **pickup** *currentpen* scaled *m*;
for *i* = 1 **upto** *boardsize* − 1:
  draw(0, *i* ∗ *ss*) . . (*ttt*, *i* ∗ *ss*);
  draw(*i* ∗ *ss*, 0) . . (*i* ∗ *ss*, *ttt*);
endfor;
picture *tttboard*; *tttboard* := *currentpicture*;
path *ttto*, *tttxa*, *tttxb*;
%Changed shift; to keep size 3 board ratios, we scale with m.
ttto := *fullcircle* scaled (.8 ∗ *ss*) shifted (*ss*/2, *ss*/2);
tttxa := (20 ∗ *m*, 20 ∗ *m*) . . (*ss* − 20 ∗ *m*, *ss* − 20 ∗ *m*);
tttxb := (20 ∗ *m*, *ss* − 20 ∗ *m*) . . (*ss* − 20 ∗ *m*, 20 ∗ *m*);
message `"enter 00 to quit, 10 to start new game"`;
message `"enter 11 to place mark at bottom left square"`;
  tr := *boardsize* + 10 ∗ *boardsize*;
s := `" to erase mark just placed at top right square"`;

```
message "enter " & decimal -tr & s;
boolean tttmarkx; tttmarkx := true;
screenstrokes; currentpicture := tttboard; showit;
forever:
  forever:
    message "Move? "; s := readstring; move := scantokens s;
    exitif known move; message "bad input, reenter";
  endfor;
  move := round move; exitif move = 0;
```

To make sure that `move` has acceptable data, we round it if known. We've implicitly declared it as numeric; however, `known move` will be false if it contains a string token in which case ask for the move again. We haven't prevented the players from placing a marker off the board. If one wanted to do this, the input loop could be expanded to check this. Let's fix some of the redundancy:

```
  mx := (((abs move) div 10) - 1) * ss; my := (((abs move) mod 10) - 1) * ss;
  if (move = 10) or (move = 1):
    currentpicture := tttboard; showit; tttmarkx := false;
  elseif move > 0:
    if tttmarkx:
      draw tttxa shifted (mx, my); draw tttxb shifted (mx, my);
    else:
      draw ttto shifted (mx, my);
    fi;
  else:
    if not tttmarkx:
      undraw tttxa shifted (mx, my); undraw tttxb shifted (mx, my);
    else:
      undraw ttto shifted (mx, my);
    fi;
  fi;
  tttmarkx := not tttmarkx;
endfor;
end;
```

In the above, we've used the `mft` utility to convert the MetaFont source into attractively typeset code.

Complex Documents

As programs and documents become complex, it becomes useful to split them into pieces. The `make` utility is used to manage the resulting files. Here is the `Makefile` for this book:

```
MACROS=/usr/TeX/lib/mf/macros
BASE=/usr/TeX/lib/mf/bases/mf.base
FORMAT=/usr/TeX/lib/tex/formats/tex.fmt
TEXINPUTS=/usr/TeX/lib/tex/inputs
BOOK=/book/milieu.texi
INPUTS = tttn.tex eflask.tex Makefile cmd10.tex llogo.tex \
    gnuchars.tex linus.tex queen.tex ttt3.tex dotsetc.tex \
    intromf.tex stardot.tex linusa.tex geo.mf
#
#the first target is what is made by default.  After the target,
#we list its dependencies.  The following line(s), which are begun
#with a TAB, tell how to make (or update) the target.
#
milieu.dvi:  macs.tex $(BOOK) $(FORMAT) $(INPUTS) $(TEXINPUTS)/mftmac.tex
    tex $(BOOK) # note:  line begins with TAB (not spaces)
#
#We can define general rules.  The following, for example, says
#that the tex files are made by MFT from the corresponding mf files.
#
%.tex:%.mf mplain.mft
    mft $< -s ./mplain.mft
#
# Finally, some utility targets.
#
$(BASE): $(MACROS)/plain.mf $(MACROS)/modes.mf
    inimf mf.ini
    mv mf.base /usr/TeX/lib/mf/bases
$(FORMAT): $(TEXINPUTS)/plain.tex $(TEXINPUTS)/hyphen.tex
    initex tex.ini
    mv tex.fmt /usr/TeX/lib/tex/formats
clean:
    rm -vf *.log *gf *.dvi
```

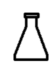
A First Font

My son, Daniel, wanted a font with chemistry symbols. First he drew a sketch on graph paper and then entered the points into MetaFont and drew the first 'curve'. Initially, there were no units; thus, when we made the font for a 300dpi printer, the characters were too big. At that point, I added the `u=pt/36;` definition. Since the point unit isn't sharped (ie, `pt#`), it gets converted for the current device. As a result, what was designed for a 2602dpi device looks ok on our printer also.

mode_setup; **path** $p[]$; $u := pt/36$;
$currenttransform := identity$ scaled .5 shifted $(0, -2pt)$;

The scale of Daniel's initial drawing was too big; rather than change each of the points, we just halve them all by means of a transform.

beginchar("e", $6pt^{\#}$, $7pt^{\#}$, $1pt^{\#}$); "eflask";
$z_1 = (170u, 500u)$; $z_2 = (40u, 190u)$; $z_3 = (40u, 140u)$;
$z_4 = (400u, 140u)$; $z_5 = (400u, 190u)$; $z_6 = (280u, 500u)$;
$z_7 = (300u, 600u)$; $z_8 = (300u, 630u)$; $z_9 = (150u, 630u)$;
$z_{10} = (150u, 600u)$; $z_{11} = (170u, 600u)$; $z_{12} = (280u, 600u)$;
$p_1 := z_{11}$ --- z_1 --- z_2 --- z_3 --- z_4 --- z_5
--- z_6 --- z_{12} --- z_7 --- z_8 --- z_9 --- z_{10} --- z_7;
pickup pencircle scaled $.4pt$; **draw** p_1;



At first, the points and curves were defined outside the character routines. But then, the `z*` definitions aren't available for the labeling routines. (The `beginchar` does a `begingroup`. It also does a `save x,y` which saves previously defined `z` points (z1 is just a macro that sets x1 & y1) so they will be available after the grouping ends. However, inside this grouping, `x*,y*` become undefined and so aren't accessible by the `label` commands. If you dislike this convention, I suggest you use `pair w[]; w1=(170u,500u);` etc.). The labeling routines insert specials into the `gf` file so that when one runs `gftodvi eflask.2602gf`, one gets a pretty picture of the character with its control points labeled.

labels($range_{1 thru 10}$); $makelabel_{lft}$("12", z_{12});
$makelabel_{rt}$("11", z_{11}); ; **endchar**;

MetaFont didn't think there was enough room for all the labels so I didn't 'autolabel' the `z11,z12` points; instead, I explicitly said where to place the labels.

beginchar("f", $6pt^{\#}$, $7pt^{\#}$, $1pt^{\#}$); "fflask";
$z_8 = (300u, 630u)$; $z_9 = (150u, 630u)$; $z_{11} = (170u, 600u)$; $z_{12} = (280u, 600u)$;
$z_{13} = (170u, 140u)$; $z_{14} = (280u, 140u)$; $z_{15} = (40u, 320u)$;
$z_{16} = (400u, 320u)$; $z_{17} = (170u, 450u)$; $z_{18} = (280u, 450u)$;
$p_2 := z_{17} \mathinner{..} z_{15} \mathinner{..} z_{13}$ & z_{13} -- z_{14} & $z_{14} \mathinner{..} z_{16} \mathinner{..} z_{18}$;
$p_3 := z_{18}$ --- $z_{12} \mathinner{..} z_8$ & z_8 -- z_9 & $z_9 \mathinner{..} z_{11}$ --- z_{17}; $p_4 := p_2$ & p_3;
pickup pencircle scaled $.5pt$;

draw p_4; **labels**$(range_{11thru18})$;
$makelabel_{rt}$("curve to z8", z_8);
$makelabel_{top}$("out to z9", z_9);
endchar;

Besides points and their labels, one can also insert `proofrule(z1,z2)` (where `x1=x2` $\|$ `y1=y2`) and general annotations with `makelabel.bot.nodot`, etc. To use this font in TEX, one first defines a command for the font:

`\font\eflaskfont = eflask.`

Then, in the text, one can say: {\eflaskfont e {\rm\ and\ } f}, for example.

While we are making characters, lets do something calligraphic. The Computer Modern Symbol fonts (cmsy10.mf, for example) include uppercase calligraphic characters designed by N. N. Billawala. I've eliminated two fitting adjustments and simplified the slanting. These characters rely on a slanted elliptical pen and carefully chosen control points. Their coding appears simpler than most of the computer modern characters which use simulated broadnibbed pens via the `penstroke` command.

beginchar("C", $9.4 * {}^{20}\!/_{36}pt^{\#}$, ${}^{246}\!/_{36}pt^{\#}$, 0);
def ....=.. tension atleast .9 .. **enddef**;
$currenttransform$:= $identity$ slanted $.2pt^{\#}$;
$join\_radius := {}^{20}\!/_{36}pt$;
$cap\_curve := {}^{35}\!/_{36}pt$; $cap\_hair := {}^{11}\!/_{36}pt$; $o := {}^{8}\!/_{36}pt$;
pickup pencircle xscaled cap_curve yscaled cap_hair rotated 30;

Thus, 30 degree strokes will be the thinest and 120 degree strokes the thickest.

MetaFont's most distinctive features

This ability to draw with sophisticated pens[4] is another principle advantage that MetaFont has over PostScript as a language for font design. That said, the comparison is not really fair; PostScript was not developed as a language for font design but rather as a general printer language in which font outlines (designed with graphical tools) could be expressed. Related to this, PostScript is meant to be written by printer drivers and other such programs. This, in my opinion, is the other main difference from MetaFont. Although some people actually write directly in PostScript, commonly programs are written to write PostScript. Metafont is meant to be written by people even though there are graphics programs such as `gnuplot` and `xfig, via fig2mf` which can output MetaFont code. MetaFont's declarative (linear) equations are a good example of its adaptation to the programmers needs. While complex pens (and even its attention to discretization errors) are of more significance to the font maker, being able to declare a font's linear relationships eliminates programming complications.

[4] "... MetaFont's most difficult task, which is to fill the envelope of a given cyclic path with respect to a given pen polygon." – Donald Knuth, *METAFONT: The Program*, section 490

By the way, if you try to format this section with `mft`, you will need to add the .... operator to `plain.mft`. This, along with using three or more %'s unintentionally, is a common source of typesetting errors when using `mft`. Also note that each space at the beginning of a line gets converted into a `quad` of space.

$x_0 = .7[x_2, x_1];\ \ \mathit{rt}\ x_1 = .85w;\ \ x_2 = .6w;\ \ \mathit{lft}\ x_3 = 0;\ \ x_4 = .5w;\ \ \mathit{rt}\ x_5 = w;$
$y_0 = .7h;\ \ y_1 = .8[y_0, y_2];\ \ \mathit{top}\ y_2 = h + o;\ \ y_3 = .5h;\ \ \mathit{bot}\ y_4 = -o;\ \ y_5 = .2h;$
draw$(z_0\{2(x_1 - x_0), y_1 - y_0\} \ldots z_1)$
  softjoin $(z_1 \ldots z_2\{\mathit{left}\} \ldots . z_3\{\mathit{down}\} \ldots z_4 \ldots z_5);$
labels$(0, 1, 2, 3, 4, 5)$; **endchar**;

The `softjoin` command joins two curves with a circular arc of radius `join_radius`, thus saving us the effort of explicitly working out the curve.

Notice the dramatic difference between this last character and the previous two. In the 'eflask' character, while we did leave the actual drawing up to `mf`, we explicitly defined all the crucial points. In this last character, the MetaFont programming language is better utilized. We declare various facts, ie equations, which determine the character and then just command MetaFont to draw so we can see it.

This character also points out the value of more interactive tools. This character is easy to code; the real work was in defining the control points for the pen to use. One might want a tool enabling one to grab a control point and move it around while viewing the results. One can not do this with MetaFont (however, provided one's computer is fast enough, we can 'prompt and show' to somewhat simulate this). On the other hand, for real interaction, no computer tools can compare to real pens and brushes.

Even with a simple character like the flask where equations don't need to be solved, it is easier to define the character in terms of a parameter:

$\mathit{currenttransform} := \mathit{identity};$
beginchar("F", $8pt^\#$, $9pt^\#$, $1pt^\#$); `"Fflask"`;
  pickup pencircle scaled $.4pt$; **path** $p[]$;
  $m = {}^1\!/\!_8 w;\ \ x_1 = x_3 = x_4 = 5m;\ \ x_2 = 7.3m;$
  $y_1 = 0;\ \ y_2 = 3m;\ \ y_3 = 6m;\ \ y_4 = h;\ \ z_5 = z_4 + (.05w, .05h);$
  $p_1 = z_1 \mathrel{..} z_2 \mathrel{..} z_3\ \&\ z_3 \mathrel{---} z_4 \mathrel{..} z_5;$
  $p_2 = p_1$ reflectedabout$((.5w, 0), (.5w, h))$;
  draw p_1 -- reverse p_2 -- cycle;
endchar;

Modifying Computer Modern Parameters

Start Emacs in its own X-window and enter `M-x shell` and `cd macros` (see Appendix 2 for setting `CDPATH`) to move to the MetaFont font sources. Suppose you want to make a variation on `cmr10` with thicker hairlines, etc. To look at several parameter files side-by-side in Emacs, press `C-x 3`. Use `C-x C-f` to load several parameter files (`C-x o` to change emacs window, or left-click). You may want to use a smaller font: `M-x x-set-font 6x13`. As I write this, I have four parameter files (cmd10,cmr10,ccr10,cmb10) across the top of the Emacs screen, then a horizontal border with two more windows below: one with the TEX source for this book and the other a shell window.

Use `M-x rename-buffer` to make `cmd10.mf` from `cmr10.mf` which is our model. We will only change a few of the parameters from their cmr10 defaults:

% Computer Modern Demi-Roman 10 point: cmd10

if unknown *cmbase*: **input** `cmbase` **fi**

font_identifier := "CMR"; **font_size** $10pt^\#$;

$u^\# := {}^{20}/_{36}pt^\#$;	% unit width
$width\_adj^\# := 0pt^\#$;	% width adjustment for certain characters
$serif\_fit^\# := 0pt^\#$;	% extra sidebar near lowercase serifs
$cap\_serif\_fit^\# := {}^{5}/_{36}pt^\#$;	% extra sidebar near uppercase serifs
$letter\_fit^\# := 0pt^\#$;	% extra space added to all sidebars
$body\_height^\# := {}^{270}/_{36}pt^\#$;	% height of tallest characters
$asc\_height^\# := {}^{250}/_{36}pt^\#$;	% height of lowercase ascenders
$cap\_height^\# := {}^{246}/_{36}pt^\#$;	% height of caps
$fig\_height^\# := {}^{232}/_{36}pt^\#$;	% height of numerals
$x\_height^\# := {}^{155}/_{36}pt^\#$;	% height of lowercase without ascenders
$math\_axis^\# := {}^{90}/_{36}pt^\#$;	% axis of symmetry for math symbols
$bar\_height^\# := {}^{87}/_{36}pt^\#$;	% height of crossbar in lowercase e
$comma\_depth^\# := {}^{70}/_{36}pt^\#$;	% depth of comma below baseline
$desc\_depth^\# := {}^{70}/_{36}pt^\#$;	% depth of lowercase descenders
$crisp^\# := 0pt^\#$;	% diameter of serif corners
$tiny^\# := {}^{11}/_{36}pt^\#$;	% diameter of rounded corners
$fine^\# := {}^{10}/_{36}pt^\#$;	% diameter of sharply rounded corners
$thin\_join^\# := {}^{12}/_{36}pt^\#$;	% width of extrafine details

The `tiny` and `fine` parameters also are related to various pen sizes (for example, the crossbar of the 'e' has thickness `fine`). For the changes that I wanted to make, the `thin_join` change was especially important.

$hair^\# := {}^{14}/_{36}pt^\#$;	% lowercase hairline breadth
$stem^\# := {}^{25}/_{36}pt^\#$;	% lowercase stem breadth
$curve^\# := {}^{30}/_{36}pt^\#$;	% lowercase curve breadth
$ess^\# := {}^{27}/_{36}pt^\#$;	% breadth in middle of lowercase s
$flare^\# := {}^{33}/_{36}pt^\#$;	% diameter of bulbs or breadth of terminals
$dot\_size^\# := {}^{38}/_{36}pt^\#$;	% diameter of dots

```
                         %Various thin lines originally set to 11/36pt# are increased:
cap_hair# := 15/36pt#;                                  % uppercase hairline breadth
bar# := 15/36pt#;                                       % lowercase bar thickness
slab# := 15/36pt#;                                      % serif and arm thickness
cap_bar# := 15/36pt#;                                   % uppercase bar thickness
cap_band# := 15/36pt#;                     % uppercase thickness above/below lobes

cap_stem# := 31/36pt#;                                  % uppercase stem breadth
cap_curve# := 37/36pt#;                                 % uppercase curve breadth
cap_ess# := 35/36pt#;                          % breadth in middle of uppercase s
rule_thickness# := .4pt#;                     % thickness of lines in math symbols

                                         %the subtle dishing of the serifs is eliminated
dish# := 0pt#;                              % amount erased at top or bottom of serifs
bracket# := 20/36pt#;              % vertical distance from serif base to tangent
jut# := 28/36pt#;                                   % protrusion of lowercase serifs
cap_jut# := 37/36pt#;                               % protrusion of uppercase serifs
beak_jut# := 10/36pt#;                       % horizontal protrusion of beak serifs

                                                       %long beak serifs are shortened:
beak# := 60/36pt#;                             % vertical protrusion of beak serifs
vair# := 13/36pt#;                                  % vertical diameter of hairlines
notch_cut# := 10pt#;                    % maximum breadth above or below notches
cap_notch_cut# := 10pt#;           % max breadth above/below uppercase notches
serif_drop# := 4/36pt#;                               % vertical drop of sloped serifs
stem_corr# := 1/36pt#;                       % for small refinements of stem breadth
vair_corr# := 1/36pt#;                      % for small refinements of hairline height
apex_corr# := 0pt#;                             % extra width at diagonal junctions

o# := 8/36pt#;                                        % amount of overshoot for curves
apex_o# := 8/36pt#;                     % amount of overshoot for diagonal junctions

slant := 0;                                                          % tilt ratio (Δx/Δy)
fudge := .85;                          % factor applied to weights of heavy characters
math_spread := 0;                                      % extra openness of math symbols
superness := 1/ sqrt 2;                                   % parameter for superellipses
superpull := 1/6;                                         % extra openness inside bowls
beak_darkness := 11/30;                       % fraction of triangle inside beak serifs
ligs := 2;                                             % level of ligatures to be included

square_dots := false;                                           % should dots be square?
hefty := true;                               % should we try hard not to be overweight?
serifs := true;                                  % should serifs and bulbs be attached?
monospace := false;                      % should all characters have the same width?
variant_g := false;                                   % should an italic-style g be used?
low_asterisk := false;                   % should the asterisk be centered at the axis?
math_fitting := false;                              % should math-mode spacing be used?
```

generate$_{roman}$ % switch to the driver file

Save `cmd10.mf`; for a listing of differences, you can do: `diff cmr10.mf cmd10.mf >diffs`.

We now want to test the font; however, there's not yet a font metric for it (`cmd10.tfm`) so we can't use TeX. There are two ways to cope with this: 1) `cp cmr10.tfm cmd10.tfm` and go ahead and use it; when we preview, the real `tfm` file will be made along with the actual packed font bitmap (cmd10.300pk). 2) run `mf \\mode=localfont\; input cmd10;gftopk cmd10.300gf` (the first semicolon is for MetaFont, the second for the shell) and then `cp cmd*pk cmd*.tfm /usr/TeX/lib/tex/fonts`. **Warning:** check what path `mf` uses in searching for font files to make sure that it is using the files you intend. Also, if you are using `MakeTeXPK`, you may want to delete its change to a temp dir.

We can now run `tex testfont` to test the font. I tested at `magstep1` since I was using that. Two problems showed up: first, the cap stems were a bit too prominent so I changed `cap_stem`. The major problem, however, was that the 'w' was too dark. Although *Computer Modern Typefaces* suggests that `fudge` can be used to fix this, I found that, in addition, `hefty:=true` was needed.

This last change has an unintended side-effect on the exclamation and question marks (see the `bot_width` definition for the characters in `punct.mf` and `romanp.mf`, respectively). However, since I'm happy with the font otherwise, I'll just pretend the change was intended! While fixing this is not difficult, it would take us away from simple changes to the parameter file.

The resulting font, `cmd10 scaled 1200`, reproduces on my 300dpi LaserJet better than the standard `cmr12`. While not as legible as Knuth's Concrete, `ccr10`, it has more variety than that font. In fact, this text is set with this Computer Modern Demi-Roman at 12 points.

A Logo for Linux

Logos are meant to give simple and concise symbolic expression of the nature of some group, person, etc. The cross, for example, is the simplest anthropomorphic symbol I can think of, thus symbolizing Christ's humanity. Furthermore, it also represents Jesus' divinity in that the fact of His death is only of ultimate importance to us as we recognize God reaching out to us through His Son. As Paul says, "If Christ has not been raised, our preaching is useless and so is your faith." Our conviction of this historical reality by the Holy Spirit is what gives the cross its significance in our own lives.

Regretfully, few of Linux's users or developers give much regard to the eternal logos. Nevertheless, I want to make a logo for Linux, one that will embody its nature. It seems appropriate for the logo to be a program; I'll just use MetaFont to write the letters in a stylized fashion. L is for Linux and its creator, Linus Torvalds; in addition, the U indicates the close association with various other 'unix' systems. I make the bottom stroke of the L so that it cradles the U. The X could refer to Linux's POSIX compliance; instead, I use it to say that X Window support is standard. Linux is an 'in' unix with X. Having the 'in' separate also conveys that the 'i' is short as in 'lint' rather than as in the famous cartoon character (whose picture we will also show an encoding for if copyright restrictions permit).

%llogo.mf, v:0.1p2a - tdunbar@vtaix.cc.vt.edu

```
mode_setup;
beginchar("L", 1.4in#, 1in#, .1in#); "The letter L (Linux logo)";
a := h/100; pickup penrazor xscaled 2a; path p[];
p1 := (15a, 10a) -- (15a, 40a); draw p1;
draw p1 shifted (8a, 0); draw p1 shifted (25a, 0);
p2 := (23a, 40a) -- (40a, 10a); draw p2;
z1 = (0, 0); z2 = (0, 70a); z7 = z2 shifted (5a, 0);
z3 = (-1a, 80a); z6 = z3 shifted (6.5a, 0);
z4 = (-10a, 90a); z5 = z4 shifted (17a, 0);
z8 = (5a, 5a); z9 = (55a, 5a); z10 = (70a, 6a); z11 = (85a, 15a); z12 = (85a, 0);
p3 := z1 --- z2{up} .. z3 .. {left}z4 & z4 -- z5 & z5 .. z6 .. z7{down} --- z8 -- cycle;
p4 := z8 --- z9 .. z10 .. {up}z11 -- z12 -- z1 -- cycle;
pickup pencircle scaled 1.2a; filldraw p3; filldraw p4;
```

The L needs to be broken; otherwise there is a bad turning number

```
pickup pencircle scaled 3a; p4 := (50a, 50a) -- (50a, 25a) .. (51a, 20a) ..
  (65a, 11a) .. (79a, 20a) .. (80a, 25a) -- (80a, 50a); draw p4;
pickup penrazor xscaled 4.5a rotated 20;
p5 := (90a, 40a) -- (110a, 15a) -- (90a, -10a); draw p5;
p6 := (130a, 42a) -- (110a, 15a) -- (130a, -8a);
draw p6 shifted (2a, -2a);
endchar;
```

```
                                                  %a smaller version:
beginchar("l",.7in#,.5in#,.05in#); "The letter l (linux logo)";
⋮
```

Electronic Documents

Let's diverge from MetaFont programming to consider the electronic production of documents. Certainly, different document types need different tools for their production. In particular, I want to discuss Knuth's tools for producing scientific texts such as this book.

Ease of Use: The Primary Criteria

As in any other endeavor, we want our tools to be as easy to use as possible given our requirements. Namely, although our page layout requirements are minimal (unlike 6 column newspapers or advertizing graphics), we do require full support for the two dimensional nature of typesetting. In addition, we want to be able to compose and edit as efficiently as possible. Furthermore, the finished document files should precisely specify the document independently of any specific equipment. Knuth's TEX typesetting system, which he designed specifically for the production of beautiful books, especially ones containing much mathematics, meets these requirements well.

Two Dimensional Type

When using a typewriter (proto-typical wysiwyg system!), type barely has one dimension: we type a letter and then the next has the same size and its placement doesn't usually depend on the previous letter. Simple word processing software extends this slightly to allow the use of letters which vary in height and, more significantly, width. Still, however, the placement of a character is largely independent of the previous character. Finally, sophisticated word-processors may provide horizontal kerning support so that, for example, an AV is set tighter than a TV.

So far we've considered text as merely a linear string of characters. But even if we are just setting plain text, this viewpoint is inadequate. For example, how do we decide where to insert hyphens and line breaks? All the commercial wordprocessors I'm familiar with just go to the end of the line and then hypenate, if necessary, and insert space between the words of that line in order to fill the line. The resulting variation in word spacing between lines can look very sloppy. Consequently, one often has to be satisfied with left justified text. However, then (and even assuming good hyphenation routines) the lines frequently still look bad since usually no word spacing is then added; as a result, line lengths often vary wildly.

TEX copes with this problem by taking a broader viewpoint. TEX, in addition to having very good hyphenation algorithms, looks at a paragraph as a whole and inserts word spacing and hyphens so as to make the paragraph look as good as possible. TEX makes good choices by default; moreover, it's algorithms can be customized by the user by setting various parameters.

The two dimensional nature of type really comes to the fore when we come to typesetting mathematics. Now, in addition to horizontal kerning, we often want to adjust the vertical position of a character depending on the previous character. Or rather, we want the program to do the vertical kerning automatically instead of having to place the character by hand and merely getting what we see. The optimal position may, in fact, depend on more than the previous character's height–it's 'center' or some other parameter may be significant. By the way, TEX can use standard PostScript fonts; however, such fonts are too naively parameterized to typeset equations, etc. attractively. There are a few PostScript fonts (Lucida Bright, Math Times) that have had TEX font metrics added.

Efficient Editing

TEX is a 'markup' language: we do the data entry with our favorite text editor and embed formatting commands that TEX will use to format the text. The widely used wordprocesser, *Word Perfect*, does the same thing; however, it hides its commands and uses a proprietary format. TEX, on the other hand, uses plain ascii file format and 'backslash' commands such as `\nopagenumbers`. One can, of course, define a TEX macro (or Emacs Lisp macro and Emacs key binding) to save keystrokes, if desired.

As mentioned above, we want our system to be as easy to use as possible. This is not the same thing as *easy to learn!* TEX is widely regarded to be hard to learn to use; however, there are several good books, at several levels, to help in the learning process. Furthermore, TEX provides a good return on one's investment–once learned, one can write scientific articles, etc. faster with TEX than with *wysiwyg* programs such as *Microsoft Word*. One can also more easily maintain and modify such documents: one may use Emacs to do a sophisticated regular expression replacement throughout the text or change various TEX macros that determine the overall style of the document.

Standard TEX archive sites such as `pip.shsu.edu` have a large collection of TEX packages and macros to cope with a wide range of formatting problems or ease producing various styles of documents.

Permanent Portability

The plain text format of TEX document source files ensures that spacetime separations need not interfere with our sharing of documents. The public availability of the code for the TEX programs themselves makes TEX available on more hardware platforms than any other typesetting software.

All the TEX[1] programs are well documented. In fact, Knuth developed his `web` composite programming language specifically for this purpose. For example, `mf.web` is the `WEB` source for the MetaFont program. By running `weave` on this source, one gets the document *MetaFont, The Program* (minus introduction, etc.); running `tangle` on it instead gives `PASCAL` source code (with Linux, and other unixlike systems, we instead use `web2c` to generate C code). A crucial feature of `web` is that it enables skilled programmers to write and document code in whatever combination of 'topdown' and/or 'bottomup' style they prefer.

To conclude with a rather impractical, but significant, thought problem: suppose one has a only a document's source in microfiche format. What is essential to answer the question: if printed on 8.5x11 paper (assume the printer can precisely align the paper), what are the coordinates of the top right corner of the 77th character on page 2? In theory

---

[1] The word TEX (ie $\tau\epsilon\chi$) is sometimes used to refer specifically to the `virtex` text formatter. It is also used, as here, to refer to the entire set of typesetting programs and utilities which Knuth designed to work together: `initex, virtex, inimf, virmf, gftopk, tangle, weave, plain.tex, cmbase.mf, etc.`

at least, with TeX one does not even need the TeX program if one has Knuth's *Computers and Typesetting* texts and sufficient patience to calculate the answer.

In contrast, with a typical word processor one:

1) Could not conveniently put the source on microfiche but would have to use hexidecimal, etc. to represent the special control characters..
2) To read the source, one needs the wordprocessor program (or some conversion utility–all of which, to my knowledge, lose data).
3) To try to answer the question, one must use the program. One could use a `print preview` feature to approximate the answer.
4) To actually answer the question, print one the first two pages and use calipers, etc. to measure the position.

While perhaps seeming a bit absurd, this example illustrates features which have significant bearing on one's day to day work, especially as documents become more electronically based. For a more farfetched example, can your word-processor and font tools be used to specify the circuits to 'burn' onto microchips so they will be able to execute that same program?

The TeX files

This section gives a brief listing of the types of files used by the TeX system.

inimf We use this to make the MetaFont base files; to make the plain base, one can run (user input in bold):

```
/book > inimf
This is METAFONT, Version 2.71 (C version d) (INIMF)
** plain
(/usr/TeX/lib/mf/macros/plain.mf
Preloading the plain base, version 2.7: preliminaries,
 basic constants and mathematical macros,
 macros for converting from device-independent units to pixels,
 macros and tables for various modes of operation,
 macros for drawing and filling,
 macros for proof labels and rules,
 macros for character and font administration,
and a few last-minute items.) input modes;
(/usr/TeX/lib/mf/macros/modes.mf) dump;
Beginning to dump on file plain.base
 (base=plain 93.6.28)
1760 strings of total length 27399
8361 memory locations dumped; current usage is 2436&5709
744 symbolic tokens
Transcript written on plain.log.
/book > mv plain.base /usr/TeX/lib/mf/bases/mf.base
```

The file `/usr/TeX/lib/mf/mf.pool`, a file of strings used by MetaFont, is also needed by `inimf`. `Inimf` is a special version of MetaFont which is used to make an initialization file consisting of 'predigested' MetaFont macros that can be quickly loaded. This base file

is then loaded by `virmf`, virgin MetaFont, which, as you recall, is linked to a name with the same base as the initialization file. For example, if we'd done a `inimf cmbase\; input modes\; dump; mv cmbase.base /usr/TeX/lib/mf/bases/cmmf.base;`
`ln -s /usr/TeX/bin/virmf /usr/TeX/bin/cmmf` we could then produce a cmr10 font slightly faster by running `cmmf`.

cmbx10.mf The only other files needed for fontmakes are the various source files. These are often broken up into various components; for example, `cmbx10.mf` just sets various parameters that determine the appearance of the font and then calls `roman.mf` which in turn inputs the various character files (e.g., `romlig.mf`).

cmbx10.300gf This is an actual font file in Generic Format. Usually, these aren't used but rather are converted to packed format by running `gftopk cmbx10.300gf`.

cmbx10.300pk This is the actual font file which is used by `xdvi`, `dvips`, `dvilj2p`, etc. **Note:** `tex`, itself, does not use the font file.

cmbx10.tfm When fontmaking, MetaFont also outputs a `tfm`, TEX Font Metric, file that contains the font parameters needed by TEX. In particular, it contains 1) overall font parameters like `xheight` and `quad`, 2) ligature (ie, what converts fl into ﬂ) and kerning information, 3) charater width and height. Math fonts contain extra information needed by TEX; run `tftopl cmsy10.tfm cmsy10.pl; more cmsy10.pl` to see a sample.

initex Like MetaFont, TEX also uses an initialization file. Unlike MetaFont, however, TEX does not separate statements with a semi-colon. It does use a string file called `tex.pool`. If we use the command line, instead of a file, to make the base format we have to use a \, as with MetaFont, to protect special characters from being interpreted by the shell. This we might enter: `initex plain \\input eplain \\input local \\dump` if we want our default format to include Berry's `eplain` macros (described in *TEX for the Impatient*). If run interactively, a sample run might look like: `/book > initex`

```
initex
This is TeX, Version 3.141 (C version d) (INITEX)
** mplain \input eplain \dump
(/usr/TeX/lib/tex/macros/mplain.tex Preloading the plain format: codes,
registers, parameters, fonts, macros, math definitions, output routines,
hyphenation (/usr/TeX/lib/tex/macros/hyphen.tex))
(/usr/TeX/lib/tex/macros/eplain.tex)
Beginning to dump on file mplain.fmt
 (format=mplain 93.6.28)
2512 strings of total length 34832
15059 memory locations dumped; current usage is 118&14924
1500 multiletter control sequences
\font\nullfont=nullfont
\font\tenrm=cmd10
\font\sevenrm=cmr7
\font\fiverm=cmr5
\font\teni=cmmi10
\font\seveni=cmmi7
\font\fivei=cmmi5
```

```
\font\tensy=cmsy10
\font\sevensy=cmsy7
\font\fivesy=cmsy5
\font\tenex=cmex10
\font\tenbf=cmbx10
\font\sevenbf=cmbx7
\font\fivebf=cmbx5
\font\tentt=cmttss10
\font\tensl=cmsl10
\font\tenit=cmti10
\font\manfnt=cmss10
\font\manfntsl=cmssi10
\font\tenln=line10
5761 words of font info for 19 preloaded fonts
14 hyphenation exceptions
Hyphenation trie of length 6075 has 181 ops out of 751
  181 for language 0
No pages of output.
Transcript written on mplain.log.
/book > mv mplain.fmt /usr/TeX/lib/tex/formats/tex.fmt
```

In addition to the three TEX files mentioned, `initex` also reads 19 `tfm` files (note: my tenrm and tentt are non-standard). The actual font files themselves are not used by `tex`, ie `virtex`, itself.

mplain.log Both `tex` and `mf` generate log files; the amount of detail (for debugging, etc.) contained in them can vary depending on various parameter settings.

MakeTeXPK Finally, I should mention the `MakeTeXPK` script. If, when using `xdvi` or a printer driver, a font file is missing, the driver will call this script to make the font. The script in the Milieu package also updates the `tfm` file.

If the `tfm` file doesn't exist in the first place, either 1) make it with `mf` or 2) `cp cmsy10.tfm foo.tfm` to fake TEX into thinking you have a `foo` font, preview (which will make the font and correct the tfm file) and then run `tex` on the source again. Actually, you shouldn't have to do this, but some versions of web2c have a bug that prevents TEX from calling MakeTeXTFM as it should. See the chapter on Hacking on patching `openinout.c`.

Basic Use

Now that we've covered the main types of files used by TEX, how do we use it? I recommend Ramond Seroul and Silvio Levy's *A Beginner's Book of TEX* for an introduction. Here, I just give a small example, one which is not very representative of TEX:

Memos with TEX

Suppose you want to use TEX to print memos satisfying the following conditions:

- One page
- Output to postscript printer
- Two line 'from' and 'to' addresses

First let's write a sample:

```
\def\footline{\hfil\ifnum\pageno=1 \else\folio\fi\hfil} %1st page
%a \nopagenumbers would have been easier..this is for 2+ page letters
\def\getfromaddress{}
\def\gettoaddress{}
```

Testing fonts with TeX

The standard TeX distribution includes a file testfont.tex which one uses to test character spacing, etc. Since that file is closely related to this text's topics, we will use it as an example of writing TeX macros for specialized tasks. It also can serve, perhaps, to scare off the fainthearted.

```
% A testbed for font evaluation (see The METAFONTbook, Appendix H)

\tracinglostchars=0
\tolerance=1000
\raggedbottom
\nopagenumbers
\parindent=0pt
\newlinechar=`@
\hyphenpenalty=200
\doublehyphendemerits=30000

\newcount\m \newcount\n \newcount\p \newdimen\dim
\chardef\other=12

\def\today{\ifcase\month\or
  January\or February\or March\or April\or May\or June\or
  July\or August\or September\or October\or November\or December\fi
  \space\number\day, \number\year}
\def\hours{\n=\time \divide\n 60
  \m=-\n \multiply\m 60 \advance\m \time
  \twodigits\n\twodigits\m}
\def\twodigits#1{\ifnum #1<10 0\fi \number#1}

\def\init{\message{@Name of the font to test = }
  \read-1 to\fontname \startfont
  \message{Now type a test command (\string\help\space for help):}}
\def\startfont{\font\testfont=\fontname
  \leftline{\sevenrm Test of \fontname\unskip\ on \today\ at \hours}
  \medskip
  \testfont \setbaselineskip
  \ifdim\fontdimen6\testfont<10pt \rightskip=0pt plus 20pt
  \else\rightskip=0pt plus 2em \fi
  \spaceskip=\fontdimen2\testfont % space between words (\raggedright)
  \xspaceskip=\fontdimen2\testfont \advance\xspaceskip by\fontdimen7\testfont}
```

```
{\catcode`\|=0 \catcode`\\=\other
|gdef|help{|message{%
\init switches to another font;@%
\end or \bye finishes the run;@%
\table prints the font layout in tabular format;@%
\text prints a sample text, assuming TeX text font conventions;@%
\sample combines \table and \text;@%
\mixture mixes a background character with a series of others;@%
\alternation interleaves a background character with a series;@%
\alphabet prints all lowercase letters within a given background;@%
\ALPHABET prints all uppercase letters within a given background;@%
\series prints a series of letters within a given background;@%
\lowers prints a comprehensive test of lowercase;@%
\uppers prints a comprehensive test of uppercase;@%
\digits prints a comprehensive test of numerals;@%
\math prints a comprehensive test of TeX math italic;@%
\names prints a text that mixes upper and lower case;@%
\punct prints a punctuation test;@%
\bigtest combines many of the above routines;@%
\help repeats this message;@%
and you can use ordinary TeX commands (e.g., to \input a file).}}}

\def\setbaselineskip{\setbox0=\hbox{\n=0
\loop\char\n \ifnum \n<255 \advance\n 1 \repeat}
\baselineskip=6pt \advance\baselineskip\ht0 \advance\baselineskip\dp0 }

\def\setchar#1{{\escapechar-1\message{\string#1 character = }%
  \def\do##1{\catcode`##1=\other}\dospecials
  \read-1 to\next
  \expandafter\finsetchar\next\next#1}}
\def\finsetchar#1#2\next#3{\global\chardef#3=`#1
  \ifnum #3=`\# \global\chardef#3=#2 \fi}
\def\promptthree{\setchar\background
  \setchar\starting \setchar\ending}

\def\mixture{\promptthree \domix\mixpattern}
\def\alternation{\promptthree \domix\altpattern}
\def\mixpattern{\0\1\0\0\1\1\0\0\0\1\1\1\0\1}
\def\altpattern{\0\1\0\1\0\1\0\1\0\1\0\1\0\1\0\1\0}
\def\domix#1{\par\chardef\0=\background \n=\starting
  \loop \chardef\1=\n #1\endgraf
  \ifnum \n<\ending \advance\n 1 \repeat}
```

```
\def\!{\discretionary{\background}{\background}{\background}}
\def\series{\promptthree \!\doseries\starting\ending\par}
\def\doseries#1#2{\n=#1\loop\char\n\!\ifnum\n<#2\advance\n 1 \repeat}
\def\complower{\!\doseries{`a}{`z}\doseries{'31}{'34}\par}
\def\compupper{\!\doseries{`A}{`Z}\doseries{'35}{'37}\par}
\def\compdigs{\!\doseries{`0}{`9}\par}
\def\alphabet{\setchar\background\complower}
\def\ALPHABET{\setchar\background\compupper}

\def\lowers{\docomprehensive\complower{`a}{`z}{'31}{'34}}
\def\uppers{\docomprehensive\compupper{`A}{`Z}{'35}{'37}}
\def\digits{\docomprehensive\compdigs{`0}{`4}{`5}{`9}}
\def\docomprehensive#1#2#3#4#5{\par\chardef\background=#2
  \loop{#1} \ifnum\background<#3\m=\background\advance\m 1
  \chardef\background=\m \repeat \chardef\background=#4
  \loop{#1} \ifnum\background<#5\m=\background\advance\m 1
  \chardef\background=\m \repeat}

\def\names{ {\AA}ngel\aa\ Beatrice Claire
  Diana \'Erica Fran\c{c}oise Ginette H\'el\`ene Iris
  Jackie K\=aren {\L}au\.ra Mar{\'\i}a N\H{a}ta{\l}{\u\i}e {\O}ctave
  Pauline Qu\^eneau Roxanne Sabine T\~a{\'\j}a Ur\v{s}ula
  Vivian Wendy Xanthippe Yv{\o}nne Z\"azilie\par}
\def\punct{\par\dopunct{min}\dopunct{pig}\dopunct{hid}
  \dopunct{HIE}\dopunct{TIP}\dopunct{fluff}
  \$1,234.56 + 7/8 = 9\% @ \#0\par}
\def\dopunct#1{#1,\ #1:\ #1;\ `#1'\ ?`#1?\ !`#1!\ (#1)\ [#1]\ #1*\ #1.\par}

\def\bigtest{\sample
  hamburgefonstiv HAMBURGEFONSTIV\par
  \names \punct \lowers \uppers \digits}

\def\math{\textfont1=\testfont \skewchar\testfont=\skewtrial
 \mathchardef\Gamma="100 \mathchardef\Delta="101
 \mathchardef\Theta="102 \mathchardef\Lambda="103 \mathchardef\Xi="104
 \mathchardef\Pi="105 \mathchardef\Sigma="106 \mathchardef\Upsilon="107
 \mathchardef\Phi="108 \mathchardef\Psi="109 \mathchardef\Omega="10A
 \def\ii{i} \def\jj{j}
 \def\\##1{|##1|+}\mathtrial
 \def\\##1{##1_2+}\mathtrial
 \def\\##1{##1^2+}\mathtrial
 \def\\##1{##1/2+}\mathtrial
 \def\\##1{2/##1+}\mathtrial
 \def\\##1{##1,{}+}\mathtrial
```

```
 \def\\##1{d##1+}\mathtrial
 \let\ii=\imath \let\jj=\jmath \def\\##1{\hat##1+}\mathtrial}
\newcount\skewtrial \skewtrial='177
\def\mathtrial{$\\A \\B \\C \\D \\E \\F \\G \\H \\I \\J \\K \\L \\M \\N \\O
 \\P \\Q \\R \\S \\T \\U \\V \\W \\X \\Y \\Z \\a \\b \\c \\d \\e \\f \\g
 \\h \\\ii \\\jj \\k \\l \\m \\n \\o \\p \\q \\r \\s \\t \\u \\v \\w \\x \\y
 \\z \\\alpha \\\beta \\\gamma \\\delta \\\epsilon \\\zeta \\\eta \\\theta
 \\\iota \\\kappa \\\lambda \\\mu \\\nu \\\xi \\\pi \\\rho \\\sigma \\\tau
 \\\upsilon \\\phi \\\chi \\\psi \\\omega \\\vartheta \\\varpi \\\varphi
 \\\Gamma \\\Delta \\\Theta \\\Lambda \\\Xi \\\Pi \\\Sigma \\\Upsilon
 \\\Phi \\\Psi \\\Omega \\\partial \\\ell \\\wp$\par}
\def\mathsy{\begingroup\skewtrial='060 % for math symbol font tests
 \def\mathtrial{$\\A \\B \\C \\D \\E \\F \\G \\H \\I \\J \\K \\L
  \\M \\N \\O \\P \\Q \\R \\S \\T \\U \\V \\W \\X \\Y \\Z$\par}
 \math\endgroup}

\def\oct#1{\hbox{\rm\'{}\kern-.2em\it#1\/\kern.05em}} % octal constant
\def\hex#1{\hbox{\rm\H{}\tt#1}} % hexadecimal constant
\def\setdigs#1"#2{\gdef\h{#2}% \h=hex prefix; \0\1=corresponding octal
 \m=\n \divide\m by 64 \xdef\0{\the\m}%
 \multiply\m by-64 \advance\m by\n \divide\m by 8 \xdef\1{\the\m}}
\def\testrow{\setbox0=\hbox{\penalty 1\def\\{\char"\h}%
 \\0\\1\\2\\3\\4\\5\\6\\7\\8\\9\\A\\B\\C\\D\\E\\F%
 \global\p=\lastpenalty}} % \p=1 if none of the characters exist
\def\oddline{\cr
  \noalign{\nointerlineskip}
  \multispan{19}\hrulefill&
  \setbox0=\hbox{\lower 2.3pt\hbox{\hex{\h x}}}\smash{\box0}\cr
  \noalign{\nointerlineskip}}
\newif\ifskipping
\def\evenline{\loop\skippingfalse
 \ifnum\n<256 \m=\n \divide\m 16 \chardef\next=\m
 \expandafter\setdigs\meaning\next \testrow
 \ifnum\p=1 \skippingtrue \fi\fi
 \ifskipping \global\advance\n 16 \repeat
 \ifnum\n=256 \let\next=\endchart\else\let\next=\morechart\fi
 \next}
\def\morechart{\cr\noalign{\hrule\penalty5000}
 \chartline \oddline \m=\1 \advance\m 1 \xdef\1{\the\m}
 \chartline \evenline}
\def\chartline{&\oct{\0\1x}&&\:&&\:&&\:&&\:&&\:&&\:&&\:&&\:&&}
\def\chartstrut{\lower4.5pt\vbox to14pt{}}
\def\table{$$\global\n=0
  \halign to\hsize\bgroup
```

```
    \chartstrut##\tabskip0pt plus10pt&
    &\hfil##\hfil&\vrule##\cr
    \lower6.5pt\null
    &&&\oct0&&\oct1&&\oct2&&\oct3&&\oct4&&\oct5&&\oct6&&\oct7&\evenline}
\def\endchart{\cr\noalign{\hrule}
  \raise11.5pt\null&&&\hex 8&&\hex 9&&\hex A&&\hex B&
  &\hex C&&\hex D&&\hex E&&\hex F&\cr\egroup$$\par}
\def\:{\setbox0=\hbox{\char\n}%
  \ifdim\ht0>7.5pt\reposition
  \else\ifdim\dp0>2.5pt\reposition\fi\fi
  \box0\global\advance\n 1 }
\def\reposition{\setbox0=\vbox{\kern2pt\box0}\dim=\dp0
  \advance\dim 2pt \dp0=\dim}
\def\centerlargechars{
  \def\reposition{\setbox0=\hbox{$\vcenter{\kern2pt\box0\kern2pt}$}}}

\def\text{{\advance\baselineskip-4pt
\setbox0=\hbox{abcdefghijklmnopqrstuvwxyz}
\ifdim\hsize>2\wd0 \ifdim 15pc>2\wd0 \hsize=15pc \else \hsize=2\wd0 \fi\fi
On November 14, 1885, Senator \& Mrs.~Leland Stanford called
together at their San Francisco mansion the 24~prominent men who had
been chosen as the first trustees of The Leland Stanford Junior University.
They handed to the board the Founding Grant of the University, which they
had executed three days before. This document---with various amendments,
legislative acts, and court decrees---remains as the University's charter.
In bold, sweeping language it stipulates that the objectives of the University
are ``to qualify students for personal success and direct usefulness in life;
and to promote the publick welfare by exercising an influence in behalf of
humanity and civilization, teaching the blessings of liberty regulated by
law, and inculcating love and reverence for the great principles of
government as derived from the inalienable rights of man to life, liberty,
and the pursuit of happiness.'' \moretext
(!`THE DAZED BROWN FOX QUICKLY GAVE 12345--67890 JUMPS!)\par}}
\def\moretext{?`But aren't Kafka's Schlo{\ss} and {\AE}sop's {\OE}uvres
often na{\"\i}ve  vis-\`a-vis the d{\ae}monic ph{\oe}nix's official r\^ole
in fluffy souffl\'es? }
\def\omitaccents{\let\moretext=\relax}

\def\sample{\table\text}
```

Note: this file's table macro was used for the table on page 2.

The Andrew Toolkit, a WYSIWYG alternative

Now, while TEX is faster, produces better output, and is much more configurable, there is certainly still a need for wysiwyg style word processors. This is especially true if one needs to produce 'multimedia' documents.

The Andrew Toolkit (ATK) is a portable user-interface toolkit that runs under X11. It provides a dynamically-loadable object-oriented environment wherein objects can be embedded in one-another. It includes an editor (ez) that one expects in an editor but it also can contain embedded raster images, spreadsheets, drawing editors, equations, simple animations, clocks, calendars, etc. The system also includes an extensive help system. All files are stored in ascii format and one can include `troff` commands for special needs. Postscript printing of the multimedia documents is supported via troff (or, in the case of Linux, GNU's groff).

The Andrew binaries (at sunsite.unc.edu: /pub/Linux/X11/andrew or in the Milieu package) can be unpacked at the root directory and used immediately. A few points should be noted:

1) Since bash interprets `help` as a shell command, write a little shell script called ahelp: `/usr/andrew/bin/help $*`
2) For a tour of features, run: `ez /usr/andrew/doc/AtkTour/Tour`.

Simple Hacking

By hacking I mean the modification of someone's source code to fix bugs or to add some needed feature. Often one needs to do this inspite of not fully understanding the source code; or even the language it's programmed in! This section gives two examples: 1) modifying the web source for MetaFont to change the prompt character and 2) fixing web2c's routine that TEX uses to try to make a missing `tfm` file.

Modifying mf.web

I've mentioned the needed changes earlier; suppose one didn't know how to change the input prompt but just knew that the default "*" interfered with interactive cut-and-paste use.

1) First, we need to try to find the relevant section of source code. Several options are possible; one could, as I did, be reading *METAFONT: The Program* and notice the relevant code. Another method is to use grep (or fgrep, if regular expression matches aren't needed) to search for key phrases. One could search for `prompt` or `input`, with, say, `fgrep prompt mf.web|more`. A better 'phrase' to search for, however, is a quote followed by an asterick– `fgrep "\"*" mf.web` – which gives one:

```
  l:=xord[m]*10+xord[n]-"0"*11; {compute the length}
  c:=c*10+buffer[first+1]-"0"*11
times:print_char("*");
char_class["*"]:=13;
    begin print("*4"); v:=v-2;
    begin print("*4"); w:=w-2;
    prompt_input("*"); {input on-line into |buffer|}
  else fatal_error("*** (job aborted, no legal end found)");
if p<>info(loop_ptr) then fatal_error("*** (loop confusion)");
  fatal_error("*** (job aborted, file error in nonstop mode)");
print_nl("**");
    begin print("*4"); value(p):=value(p)-2;
primitive("*",secondary_binary,times);@/
  fatal_error("*** (cannot readstring in nonstop modes)");
```

Looking through this, the relevent source line is obvious so one loads a copy of mf.web into Emacs and searches for `prompt_input("*")`. After changing the prompt to a space, and also deleting the newline, one can run diff which will list the change:

```
*** mf.web~        Sun Mar 29 17:06:18 1992
--- mf.web        Tue Aug 31 09:06:19 1993
***************
*** 14127,14136 ****
    if interaction>nonstop_mode then
      begin if limit=start then {previous line was empty}
        print_nl("(Please type a command or say `end')");
  @.Please type...@>
!     print_ln; first:=start;
```

```
!     prompt_input("*"); {input on-line into |buffer|}
  @.*\relax@>
      limit:=last; buffer[limit]:="%";
      first:=limit+1; loc:=start;
      end
--- 14127,14136 ----
    if interaction>nonstop_mode then
      begin if limit=start then {previous line was empty}
        print_nl("(Please type a command or say `end')");
  @.Please type...@>
!     first:=start;
!     prompt_input(" "); {input on-line into |buffer|}
  @.*\relax@>
      limit:=last; buffer[limit]:="%";
      first:=limit+1; loc:=start;
      end
```

(Of course, having the diff, one can use `patch`, rather than an editor, to change the source.)

Fixing openinout.c

If one runs TeX and a tfm file is missing, TeX normally stops with an error; in an analogous situation, when xdvi is missing a pk bitmap, it just calls `mf` to make it and then proceeds. Wouldn't it be nice if TeX could do the same? It can! Searching the top of the web2c-5.851d source directory, the NEWS file says:

```
  - If a .tex, .tfm, or .mf file can't be found, MakeTeXTeX,TFM,MF is
    invoked before giving up, as in dvips' MakeTeXPK.
```

Thus we are lead to use `fgrep` to search source files for, say, `MakeTeX`. The relevant file turns out to be `lib/openinout.c` where grep reports:

```
/* Wrap another sh around the invocation of the MakeTeX program, so we
   can avoid `sh: MakeTeXTFM: not found' errors confusing the user.
return make_tex_file ("MakeTeXTeX");
return make_tex_file ("MakeTeXTFM");
return make_tex_file ("MakeTeXMF");
```

Further snooping in the configuration file lib/c-auto.h.in (which is the first thing that the INSTALL info file mentions to edit) shows that this feature is enabled by default (despite the misleading comment in c-auto.h.in!).

So. We need to see 1) is TeX, in fact, trying to call mf to make the tfm file and 2) if so, what's going wrong. An easy way to do this is to run `top` which shows the current process activity (you may need to slow down your computer to see the call). With `top` running in one xterm window, run TeX on a file that needs a non-existant tfm file. Then as tex forks the call to MakeTeXTFM, top will (among other things) report:

```
 PID  SIZE  RES SHRD STAT %CPU %MEM  TIME COMMAND
  161   89  300  316 R     5.8  1.9  0:30 top
```

```
   21 1582 1328  496 S     2.9  8.7  4:29 X :0
  204  285  340  404 S     1.5  2.2  0:00 sh -c sh -c MakeTeXTFM fakef.tfm>/dev/nu
```

Now, `sh -c` *string* means that the shell is to to execute *string*, unfortunately, since the above isn't quoted, `MakeTeXTFM` doesn't get passed it's needed argument of `fakef.tfm`.

To fix openinout.c, we just need to add quotes to the call. Diff --context=5 reports:

```
*** openinout.c~        Sun Feb 21 09:48:14 1993
--- openinout.c        Tue Aug 31 09:06:14 1993
***************
*** 131,150 ****

    /* Wrap another sh around the invocation of the MakeTeX program, so we
       can avoid `sh: MakeTeXTFM: not found' errors confusing the user.
       We don't use fork/exec ourselves, since we'd have to call sh anyway
       to interpret the script.  */
!   strcpy (cmd, "sh -c ");

    strcat (cmd, program);
    cmd_len = strlen (cmd);
    cmd[cmd_len++] = ' ';

    while (nameoffile[i] != ' ')
      cmd[cmd_len++] = nameoffile[i++];

    /* Add terminating null.  */
    cmd[cmd_len] = 0;

    /* Don't show any output.  */
    strcat (cmd, ">/dev/null 2>&1");

--- 131,151 ----

    /* Wrap another sh around the invocation of the MakeTeX program, so we
       can avoid `sh: MakeTeXTFM: not found' errors confusing the user.
       We don't use fork/exec ourselves, since we'd have to call sh anyway
       to interpret the script.  */
!   strcpy (cmd, "sh -c \"");

    strcat (cmd, program);
    cmd_len = strlen (cmd);
    cmd[cmd_len++] = ' ';

    while (nameoffile[i] != ' ')
      cmd[cmd_len++] = nameoffile[i++];
```

```
    /* Add terminating null.  */
+   cmd[cmd_len++] = '\"';
    cmd[cmd_len] = 0;

    /* Don't show any output.  */
    strcat (cmd, ">/dev/null 2>&1");
```

Now a `make clean; make TeX` will make corrected tex binaries.

Educational hacking

1) Read documentation files that a program's author includes with the source code. Read the source code and it's comments.
2) This sort of hacking is an alternative way of learning programming. Rather than listening to orderly lectures, one is apprenticed to experienced programmers. Hopefully, their programs can speak for themselves.
3) For that matter, much of this text is concerned with this sort of training. I think it much more fun and useful but it does need a rather different set of skills, e.g., browsing around systems involving hundreds of files and looking for ways to 'get into' the code.

Embedding characters in METAFONT

Suppose you want to make some labeled drawings in MetaFont; how can you get the characters easily? The GNUPLOT 2–3D scientific graphing program points out an easy way to do this: just use some already defined font! The `gnuplot` MetaFont output loads `cmr10` characters for each plot and uses them for various labels.

It is more efficient to make a special format with these 'pictures' included.

%gnuchars.mf

```
if unknown cmbase: input cmbase fi;
mode = localfont; picture r[];
mode_setup;
```

First, the function to print a string:

```
def put_text(expr ts, xstart, ystart, rot, justification) =
```

The `rot`, rotation, should be a multiple of 90. Justification is 1,2, or 3 for left, center, and right justified text with respect to the `(xstart,ystart)` position.

```
begingroup
  text_width := 0; text_height := 0;
```

Compute the width of the text string (ts):

```
  for ind := 0 step 1 until length(ts) − 1:
    dec_num := ASCII substring(ind, ind + 1) of ts;
    if unknown r[dec_num]: dec_num := 32; fi
```

r is a picture array to be filled later. The space character is used for missing characters.

```
    if dec_num = 32:
      text_width := text_width + wd[65];
      text_height := max(text_height, ht[65] + dp[65]);
    elseif dec_num ≥ 0:
      text_width := text_width + wd[dec_num];
      text_height := max(text_height, ht[dec_num] + dp[dec_num]);
    fi
  endfor
```

Now that we have the text width (and height), we can compute the x and y coordinates for the starting position, given the specified justification (1=left, 2=centered, 3=right)

```
  if rot = 90:
    if justification = 1: ynext := ystart;
    elseif justification = 2: ynext := round(ystart − text_width/2);
    else: ynext := round(ystart − text_width);
    fi
    xnext := xstart + text_height/2;
  else:
```

if $justification = 1$: $xnext := xstart$;
elseif $justification = 2$: $xnext := \mathrm{round}(xstart - text\_width/2)$;
else: $xnext := \mathrm{round}(xstart - text\_width)$;
fi
$ynext := ystart - text\_height/2$;
fi

All of that was just to get the starting position. Now we again step through the string character by character, adding the characters' picture to the current picture. The `wd` array contains the characters' lengths.

for $ind := 0$ **step** 1 **until** $\mathrm{length}(ts) - 1$:
$dec\_num :=$ ASCII **substring**$(ind, ind + 1)$ **of** ts;
if unknown $r[dec\_num]$: $dec\_num := 32$; **fi**
if $dec\_num = 32$:
$xnext := xnext + wd[65] * \mathrm{cosd}\ rot$;
$ynext := ynext + wd[65] * \mathrm{sind}\ rot$;
elseif $dec\_num \geq 0$:
% We add the text character by character to the current picture.
$currentpicture := currentpicture + r[dec\_num]$ shifted $(xnext, ynext)$
rotatedaround$((xnext, ynext), rot)$;
$xnext := xnext + wd[dec\_num] * \mathrm{cosd}\ rot$;
$ynext := ynext + wd[dec\_num] * \mathrm{sind}\ rot$;
fi
endfor
endgroup
enddef;

In order to save the character pictures and widths, we redefine the `endchar` macro:

def endchar =
$r[charcode] := currentpicture$;
$wd[charcode] := w$; $ht[charcode] := h$; $dp[charcode] := d$;
message `"Picture of charcode no."` & decimal $charcode$;
endgroup;
enddef;

The plain base adds a _ to the end of all its private tokens. This reduces the likelehood of users redefining those tokens. However, that's precisely whate we want to do here:

let $endchar\_ :=$ **endchar**;

Negate the commands that would normally generate a font:

let $o\_generate := generate$; **let** $o\_roman := roman$;
let $generate :=$ **relax**;
let $roman =$ **relax**;

The let macro, like the def macro, can use either an = or a := symbol. The code generated by gnuplot uses a = which I change to := which seems less confusing.

Rather than cmr10, I use a sans serif font. The `cmssb10.mf` parameter file is just `cmssdc10` with sharp corners: `crisp:= tiny:= fine:= 0`.

input `cmbx5.mf`
if $ligs > 1$: **font_coding_scheme** := `"TeX text"`;
  $spanish\_shriek$ = oct `"074"`; $spanish\_query$ = oct `"076"`;
else: **font_coding_scheme** :=
  if $ligs = 0$: `"TeX typewriter text"`
  else: `"TeX text without f-ligatures"` **fi**;
  $spanish\_shriek$ = oct `"016"`; $spanish\_query$ = oct `"017"`;
fi;
font_setup;
input `romanu.mf` %Roman uppercase.
input `romanl.mf` %Roman lowerrcase.
input `romand.mf` %Numerals.
input `punct.mf` %Punctuation symbols.

You can, of course, include other characters if needed; in fact, you can include characters from several fonts (parameters and font_setup may need redoing).

Next we add some extra characters:

$minus$ = ASCII `"-"`; $cmchar$ `"Minus sign"`;
$beginarithchar(minus)$;
  pickup $rule_{nib}$;
  $lft\ x_1$ = hround $1.5u - eps$;
  $x_2 = w - x_1$; $y_1 = y_2 = math\_axis$;
  draw z_1 -- z_2; % bar
  labels(1, 2);
endchar;

$cmchar$ `"Period"`;
  numeric $dot\_diam\#$; $dot\_diam\#$:= **if** $monospace$: 5/4 **fi**\ $dot\_size\#$;
  define_whole_blacker_pixels(dot_diam);
  beginchar(`"."`, $5u\#$, $dot\_diam\#$, 0);
  $adjust\_fit(0, 0)$; **pickup** $fine_{nib}$;
  $pos_1(dot\_diam, 0)$; $pos_2(dot\_diam, 90)$;
  $lft\ x_{1l}$ = hround($.5w - .5dot\_diam$); $bot\ y_{2l} = 0$; $z_1 = z_2$; $dot(1, 2)$; % dot
  $penlabels(1, 2)$;
endchar;

Now we reset endchar back to normal; there isn't really any need to do this.

def endchar =
  $l := 0$; $r := w$;
%Include the next two lines if you want to rotate

```
                                         %the picture 90 deg.(Portrait to Landscape)
                              %currentpicture:=currentpicture rotated 90 shifted (h,0);
                                       %tmp:=charht; charht:=charwd; charwd:=tmp;
   scantokens extra_endchar;
   if proofing > 0: makebox(proofrule); fi
   chardx := w;
   shipit;
   if displaying > 0: makebox(screenrule); showit; fi
   endgroup
enddef;
let endchar_ := endchar;

let generate := o_generate;
let roman := o_roman;

font_identifier := "GNUPLOT";
font_size 72pt#;
th# = 0.4pt#; define_whole_pixels(th);

path arrowhead;
arrowhead = (-7pt, -2pt){dir 30} .. (-6pt, 0pt) .. {dir 150}(-7pt, 2pt)&
      (-7pt, 2pt) -- (0pt, 0pt) -- (-7pt, -2pt) & cycle;
```

Save this in a base file for future use (the mode=localfont at the beginning was needed since the pictures are for paper; and even for on-screen use, proof mode characters are too big).

Geometric Graphics

%geo.mf; assumes gnuchars.mf has been loaded.

input `gnuchars`; %otherwise, add this

Make a special format:

```
inimf plain\; input modes\; input gnuchars\; dump
mv plain.base /usr/TeX/lib/mf/bases/gmf.base
ln -s /usr/TeX/bin/virmf /usr/TeX/bin/gmf
gmf geo
```

Now that we know how to embed characters in MetaFont, let's do a bit of geometry. At the beginning of his *Elements,* Euclid shows that the local symmetry of space (ie the postulated constructibility of circles) leads to the global property that lengths can be 'moved.' First some preliminaries:

screenstrokes; **path** $p[]$;
pair A, B, C, D, E, F, G, H;
  pickup pencircle scaled $.06pt$;
  $spen = savepen$;
  pickup pencircle scaled $.12pt$;
  $mpen = savepen$;

We want to:

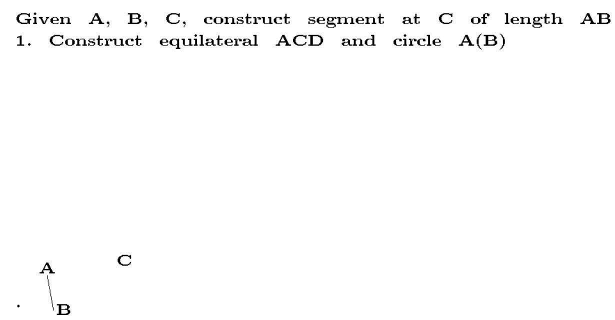
put_text(`"Given A, B, C, construct segment at C of length AB"`, 0, 450, 0, 1);
showit; $sscreen(80pt^\#, 80pt^\#, 2pt^\#, 1)$;

Recall that the `gnuchars.mf` file defined `put_text(<string>, <x-coord>, <y-coord>, <rotation:0, 90, 180, or270>, <justification: 1,2,3 (left, center, right)>)`. I pick convenient generic points and then shift for my specific screen:

$A := (50, 50)$; $B := (60, -5)$; $C := (170, 60)$;
$currenttransform := identity$ shifted $(90, 20)$;
pickup $mpen$;
drawdot A; put_text(`"A"`, xpart A, ypart $A + 10$, 0, 2); **showit**;
drawdot B; put_text(`"B"`, xpart $B + 4$, ypart B, 0, 1); **showit**;
drawdot C; put_text(`"C"`, xpart C, ypart $C + 12$, 0, 2); **showit**;
draw A -- B;

To begin, we make use of Euclid's first theorem to:

put_text(`"1. Construct equilateral ACD and circle A(B)"`, 0, 420, 0, 1);
showit;
$sscreen(80pt^\#, 80pt^\#, 2pt^\#, 1)$;



To do this, we need a 'compass,' a way to, given a center and a point on the circumference, define a circle. Often in defining MetaFont macros, parentheses are needed to make MetaFont parse the macro correctly. For example,

def *gcircle*(**expr** *wa*, *wb*) =
  fullcircle scaled $(2 * \text{length}(wa - wb))$ shifted *wa* **enddef**;

In Euclidean geometry, lines intersect in points; however, lines weren't thought of as a potentially incomplete set of points. Hence,

$D :=$ *gcircle*(A, C) intersectionpoint *gcircle*(C, A);

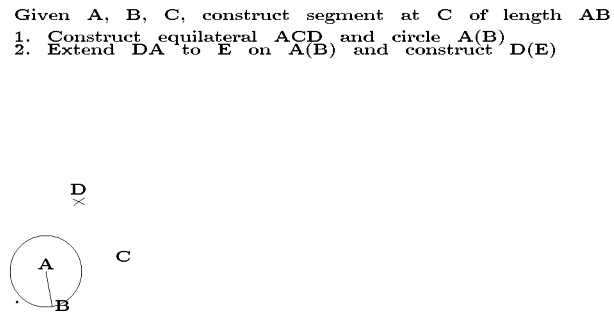
obviously exists. Moreover, `AD=AC` as radii of `gcircle(A,C)`; similarly, `AC=CD` so `AD=CD` since things equal to the same thing are the same.

To illustrate this construction, we need another macro to draw arcs such as we would draw with a compass. Knowing where the intersection is, we can make the arcs short. The `garc` macro expands to a three point curve.

def *garc*(**expr** *wa*, *wb*, *d*) =
  wb rotatedaround(wa, d) .. *wb* .. *wb* rotatedaround$(wa, -d)$ **enddef**;
pickup *spen*; **draw** *garc*$(A, D, 5)$; **draw** *garc*$(C, D, 5)$;
pickup *mpen*; **drawdot** D;
put_text(`"D"`, xpart D, ypart $D + 20, 0, 2$); **showit**;

Our idea is to use `gcircle(A,B)` to rotate `AB` until we can slide it up a line to `D` and then down to `C`. Such sliding might change the length of the segment so instead we appeal to properties of circles.

pickup *spen*; **draw** *gcircle*(A, B);
put_text(`"2. Extend DA to E on A(B) and construct D(E)"`, $0, 400, 0, 1$);
showit;
sscreen$(80pt^{\#}, 80pt^{\#}, 2pt^{\#}, 1)$;



Since we're using MetaFont to merely illustrate the abstract proof, it is convenient to have another circle macro:

def *grcircle*(**expr** *wa*, *r*) = *fullcircle* scaled $(2 * r)$ shifted *wa* **enddef**;

$r :=$ length$(A - D)$ + length$(A - B)$;
$p_1 :=$ *grcircle*(D, r);
draw p_1;

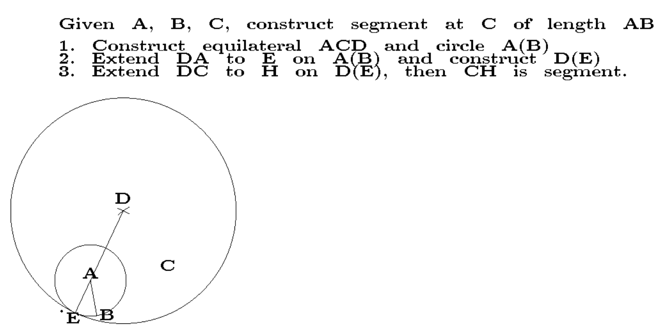
The point `E` is where `A(B)` intersects `p1`. However, due to rounding errors, `E` needs to be specified differently. We instead make use of the fact that the direction in which a point on a circle is traveling, the tangent, is perpendicular to a radius at that point:

$F = (A - D)$ rotated 90;
$t :=$ **directiontime** F **of** p_1; $E =$ **point** t **of** p_1; **drawdot** E;

%show E; uncomment following for proof of rounding error.
%show p1 intersectionpoint gcircle(A,B);

put_text("E", xpart $E + 8$, ypart $E - 8$, 0, 3); **showit**;

draw D -- E; **pickup** *mpen*; **draw** A -- E;
put_text("`3. Extend DC to H on D(E), then CH is segment.`", 0, 380, 0, 1);
showit;
sscreen$(80pt^\#, 80pt^\#, 2pt^\#, 1)$;



%We repeat the above procedure for `H`.

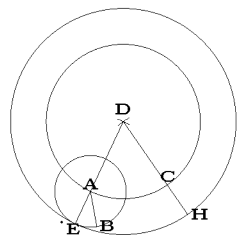
$G = (C - D)$ rotated 90;

$t :=$ **directiontime** G **of** p_1; $H =$ **point** t **of** p_1;

pickup *spen*; **draw** D -- H;
draw $gcircle(D, C)$;
pickup *mpen*; **drawdot** H;
put_text("H", xpart $H + 5$, ypart H, 0, 1); **showit**;

draw C -- H;
$sscreen(80pt\#, 80pt\#, 2pt\#, 1)$;

Given A, B, C, construct segment at C of length AB
1. Construct equilateral ACD and circle A(B)
2. Extend DA to E on A(B) and construct D(E)
3. Extend DC to H on D(E), then CH is segment.



As radii, DH=DE so, subtracting the equilateral's sides DC=AC, we have CH=AE. Thus CH=AB and the construction is complete. **end**;

Linus

This is very tedious and ugly code; it does, however, point out that one can use brute force methods for line drawings. About the only point of it, besides its original use for announcing Linux's release, is to provide the reader opportunities for improvements.

The main 'problem' is that this section is about graphic art rather than computer programming. Let's organize the file to emphasize the essential features. First, we adjust for the screen:

screenstrokes; $mode = localfont$; **mode_setup**;
$currenttransform := identity$ shifted $(10 + \mathrm{xpart}$
$screen\_topleft, 10 - screen\_rows + \mathrm{ypart}\ screen\_topleft)$;

beginchar("l", $100pt^\#$, $100pt^\#$, 0); **pickup pencircle** scaled $.8pt$;

%eyebrows

draw$(.49w, .86w)$ -- $(.555w, .89w)$; **draw**$(.66w, .91w)$ -- $(.725w, .895w)$;
pickup pencircle xscaled $.035w$ yscaled $.07w$ rotated 10;

Minimalist art, in my opinion, is really more a scientific investigation. It is interesting to see the maximum one can suggest with the minimum of lines and colors.

%eyes

drawdot$(.49w, .63w)$; **drawdot**$(.68w, .65w)$; **pickup pencircle** scaled $.4pt$;

%eye sockets

draw$(.43w, .59w)\{\mathrm{dir}\ 135\}$ .. $(.42w, .63w)\{up\}$ .. $\{\mathrm{dir}\ 65\}(.46w, .72w)$;
draw$(.7w, .76w)\{\mathrm{dir}\ {-45}\}$ .. $\{down\}(.75w, .68w)$ .. $(.745w, .645w)$;

Linus is the most interesting of Charles Schultz's characters. The distinctive feature is the outline of his head. On the left we have a 'Cro-Magnon' skull:

draw$(.3w, .51w)$ .. $\{up\}(.29w, .54w)$ .. $(.36w, .66w)$ .. $(.4w, .73w)\{up\}$
.. $(.36w, .87w)\{up\}$ .. $(.46w, .97w)\{\mathrm{dir}\ 20\}$ .. $(.7w, 1.1w)\{right\}$ %Cro-Magnon
.. $(.83w, 1.07w)\{\mathrm{dir}\ {-30}\}$ .. $\{\mathrm{dir}\ 265\}(1.03w, .73w)$; %down to Charlie's ear

with an ear that, from a different angle, also functions as a nose.

draw$(.29w, .54w)\{\mathrm{dir}\ 170\}$ .. $(.26w, .57w)\{up\}$ .. $\{right\}(.36w, .66w)$;

Now let's draw some other stuff to give the image time to sink in. The left hand could definitely be improved. One might want to use the 'gnuchars' technique to put an announcement on the paper (or, use kerns with TEX–this is more flexible).

draw flex((.2w, .15h), (.14w, .3h), (.1w, .45h)) %front page
  & flex((.1w, .45w), (.4w, .42w), (.63w, .46w), (.68w, .45w))
    & flex((.68w, .45w), (.69w, .30w), (.7w, .18w));
draw flex((.11w, .47w), (.3w, .46w), (.55w, .48w), (.71w, .47w))
  & flex((.71w, .47w), (.72w, .31w), (.73w, .19w));
draw flex((.13w, .49w), (.45w, .5w), (.75w, .49w))
  & flex((.75w, .49w), (.76w, .38w), (.77w, .25w)); %back page

draw(.24w, .16w) . . (.4w, .12w){*right*} . . (.5w, .13w); %paper creases
draw(.28w, .12w) . . (.4w, .08w){dir −15} . . (.53w, .06w);

%left hand
draw(1.01w, .01w) . . (.95w, .15w) . . (.92w, .17w){*left*} . . (.85w, .2w){*up*}
  . . {dir 200}(.77w, .25w) . . {dir −45}(.79w, .17w)
  & (.79w, .17w){dir 135} . . (.73w, .19w) . . (.7w, .18w)
  . . (.72w, .1w) - - - (.77w, .11w) & (.77w, .11w) - - - (.72w, .1w)
  . . (.76w, .02w) - - - (.8w, .05w) & (.8w, .05w) - - - (.76w, .02w)
  . . (.81w, −.02w) - - - (.84w, .01w) & (.84w, .01w) - - - (.82w, −.02w)
  . . (.84w, −.03w) . . (.88w, −.01) . . {dir 80}(.89w, .02w)
  - - - (1.01w, .01w) - - - cycle;

%right fingers
draw(.2w, .15w) - - - (.18w, .21w) . . (.15w, .22w){*left*}
  . . (.115w, .17w){*down*} . . (.18w, .1w){*right*} . . (.21w, .12w){*up*} . . cycle;
draw(.12w, .14w) . . (.11w, .07w) . . (.16w, .04w){*right*}
  . . {dir 100}(.205w, .09w);
draw(.11w, .07w){dir 230} . . (.15w, −.02w){*right*} . . (.19w, .02w);

Now finish the 'Charlie Brown' part of the face:

draw(1.03w, .73w){*right*} . . (1.14w, .63w){*down*} . . {dir 175}(1.03w, .56w); %ear
draw(1.03w, .56w){*down*} . . {*left*}(.765w, .28w);

%nose
draw(.62w, .70w){dir 170} . . (.55w, .7w) . . (.54w, .65w){*down*}
  . . (.58w, .61w){dir 20} - - - (.61w, .62w);

%neck
draw(.8w, .282w) . . (.805w, .255w);

%collar
fill(.85w, .2w){dir 40} . . (.89w, .245w) - - - (.88w, .25w) - - - (.84w, .24w)
  . . (.85w, .22w) - - - cycle;

Now Linus' trademark blanket, on the 'Charlie' side.

draw(.85w, .20w){dir 40} . . (1w, .3w) . . (1.07w, .31w){*right*}
  . . (1.23w, .06w){dir −80};

$\textbf{draw}(.9w, .2w) \mathrel{..} (1w, .24w)\{\mathit{right}\} \mathrel{..} \{\mathrm{dir}\ {-80}\}(1.14w, .01w);$

And, finally, Linus' hair, which augments the basic figure. It perhaps should have been drawn sooner; however, from an interactive point of view, this gives the viewer time to look at the figure. When MetaFont reads the `end` statement, she closes the window and exits.

$$\begin{aligned}
&\textbf{draw}(.46w, .97w)\{\mathit{left}\} \mathrel{..} (.31w, .9w) \mathrel{..} (.25w, .78w);\\
&\textbf{draw}(.7w, 1.1w)\{\mathrm{dir}\ 150\} \mathrel{..} (.46w, .97w) \mathrel{..} \{\mathrm{dir}\ 285\}(.46w, .83w);\\
&\textbf{draw}(.7w, 1.1w)\{\mathit{left}\} \mathrel{..} \{\mathrm{dir}\ 285\}(.53w, .84w);\\
&\textbf{draw}(.83w, 1.07w)\{\mathrm{dir}\ {-50}\} \mathrel{..} \{\mathrm{dir}\ 250\}(.9w, .7w);\\
&\textbf{draw}(.83w, 1.07w)\{\mathrm{dir}\ 135\} \mathrel{..} (.7w, 1.1w) \mathrel{..} \{\mathrm{dir}\ 280\}(.57w, .8w);\\
&\textbf{draw}(.7w, 1.1w)\{\mathit{right}\} \mathrel{..} (.83w, 1w) \mathrel{..} (.9w, .75w);\\
&\textbf{draw}(.535w, 1.02w)\{\mathrm{dir}\ 190\} \mathrel{..} (.37w, 1.03w)\{\mathit{left}\} \mathrel{..} \{\mathit{down}\}(.15w, .77w);\\
&\textbf{draw}(.92w, 1w)\{\mathrm{dir}\ {-55}\} \mathrel{..} \{\mathrm{dir}\ 250\}(.95w, .7w);
\end{aligned}$$

What with all the other hard-coded coordinates, there seems little point in using `intersectiontimes` or `intersectionpoint` here.

$$\begin{aligned}
&\textbf{draw}(.92w, 1w)\{\mathit{right}\} \mathrel{..} (.96w, .99w) \mathrel{...} \{\mathrm{dir}\ 250\}(.95w, .68w);\\
&\textbf{draw}(.92w, 1w)\{\mathrm{dir}\ 120\} \mathrel{..} \{\mathit{left}\}(.83w, 1.07w) \mathrel{..} (.69w, 1w)\\
&\quad \mathrel{..} \{\mathrm{dir}\ 250\}(.59w, .87w);\\
&\textbf{draw}(.92w, 1w)\{\mathrm{dir}\ 135\} \mathrel{..} (.69w, .9w)\{\mathrm{dir}\ 240\};\\
&\textbf{draw}(.92w, 1w)\{\mathrm{dir}\ 185\} \mathrel{..} (.75w, .85w)\{\mathrm{dir}\ 250\};\\
&\textbf{endchar};\ \textbf{end}
\end{aligned}$$

Run `mf linusa` to view the construction. You may need to slow you computer down to get the effect I intended–I'm viewing this myself on a 386sx/20 laptop with 4 megs ram (and with X, MetaFont, and Emacs all running).

Watching an N Queens solution

```
                                    %standard N queens programmed in MetaFont
                                    %initialize; non-numerics must be declared
screenstrokes; pickup pencircle scaled .08pt;
boolean r[], u[], d[], loop, done;
string s[]; path p[], queen, checkit;
message "how many queens? ";
n = -1 + scantokens readstring;
if n < 3: n := 3; fi; if n > 92: n := 92; fi;
message "pause after each add (y/n)?";
s2 := readstring; m := 2*n; N := n+1; w = floor(290/N);
                                    %paths and board
checkit := (-30,-30)--(0,-80)--(70,-10);
                                    %queen originally: queen:=unitsquare scaled w;
p1 := (.25w,.15w)--(.15w,.85w)--(.3w,.5w)--(.4w,.85w)--(.5w,.5w);
p2 := p1 reflectedabout((.5w,0),(.5w,w)); queen := p1..reverse p2--cycle;
                                    %draw board
for i := 0 upto n+1:
  draw(0,w*i)--(w*N,w*i); draw(w*i,0)--(w*i,w*N);
endfor;
                                    %procedures (ie macros)
                                    %mark a square
def add = b[j] := i; r[i] := true; d[j+i] := true; u[j-i] := true;
  fill queen shifted (j*w, i*w);
  if s2 = "y":
    message "press enter to continue"; s1 := readstring;
  fi;
enddef;
                                    %unmark a square
def rem = r[i] := false; d[j+i] := false; u[j-i] := false;
  unfill queen shifted (j*w, i*w);
enddef;
                                    %backtrack
def back = j := j-1; i := b[j];
  forever:
    exitif not((i = n) and (j > 0));
    rem; j := j-1; i := b[j];
  endfor;
  if i < n:
    rem; i := i+1;
  else:
    done := true;
  fi;
```

```
enddef;
                                                        %draw check
def printit =
  pickup pencircle xscaled .4pt yscaled .1pt rotated -20;
  draw checkit; message "press enter to continue ";
  s1 := readstring; undraw checkit;
enddef;
                                                        %main program
i := 0; j := 0; done := false;
for k := 0 upto n: r[k] := false; endfor;
for k := -n upto n: u[k] := false; endfor;
for k := 0 upto m: d[k] := false; endfor;
forever:
  loop := true;
  if not(r[i] or d[j + i] or u[j - i]):
    add;
    if j = n:
      printit; message "show another solution (y/n)?";
      s3 := readstring;
      if s3 = "y":
        s2 := "n"; rem; back;
      else:
        done := true;
      fi;
    else:
      j := j + 1; i := 0; loop := false;
    fi;
  fi;
  if loop:
    if i = n:
      back;
    else: i := i + 1;
    fi;
  fi;
  exitif done;
endfor;

if s3 = "y": message "no more solutions"; fi;
end;
```

Other general programming stuff with MetaFont

A Real Font

New Pens

Pens are primary

Triangular Pens

A Meta Font

Using declarations to exploit our new pens

Organization and Cooperation

Further Reading

THE BOOKS

1) New International Version. There are certainly other excellent translations of the Bible also.[1]

Knuth's books

1) The MetaFontbook. Volume C of *Computers and Typesetting.*
2) Seminumerical Algorithms. Volume 2 of *The Art of Computer Programming.*

Other Texts

1) The Unix Programming Environment *by Brian Kernighan and Robert Pike*
2) Structure and Interpretation of Computer Programs *by Harold Abelson and Gerald Jay Sussman with Julie Sussman*
3) Code Complete *by Steven McConnell*

Online References

1) /usr/TeX/lib/mf/macros
2) /usr/lib/emacs/19.19/lisp
3) /usr/src/linux
4) /usr/man/cat1, /usr/TeX/man, /usr/info

[1] A few verses are translated and commented upon in Knuth's *3:16.*

Appendix 1

Motherboards

Video Cards

Supported accelerator chips include: S3 (928, 805, 801) and ATI (Mach32 and Mach8).

Cases and Fans

Other Parts

Appendix 2

The Filesystem and ls

The standard unix utility to list directories is `ls`. As we list its options, we will also discuss Linux filesystems in general. The Milieu package uses the 'minix' filesystem with 30 character filenames (use mkfs -n30 to make such a filesystem). Linux supports several filesystems (minix,efs,efs2,xia,dos,proc,cdrom); via its Virtual File System, the user is shielded from having to be concerned with the details.

-a –**all** –List all files in directories, including all files that start with '.' (such files are used for initialization and customization and are mainly to be found in ones home directory–for example: ls -al /root).

-d –**directory** –List directories like other files, rather than listing their contents. The need for this is demonstrated by trying: `ls /usr` and then trying `ls /usr/i*` (you may also want to add a `-l` option).

-i –**inode** –Print the index number of each file to the left of the file name. **Important:** the inodes are actually the files. File names are merely pointers to the inodes. When an inode has only one filename, one can think of the filename as the file. However, one can also link a file (ie, a inode) to several filenames: `ln ls dir`. In addition to hard links, Linux also supports symbolic links: `ln -s /bin/ls /usr/bin/dir` (sometimes these links need to be forced: `ln -sf /lib/libc.so.4 /lib/libc.so.4.4.4` (**Do NOT** relink lib files unless you have floppies to reboot with in case something vital breaks. Furthermore, it's safer not to remove the old lib files for a while–something important may need the old libs)

-l –**format=verbose** –In addition to the name of each file, print the file type, permissions (see `chmod` below for information on permissions, etc.), number of hard links, owner name, group name, size in bytes, and timestamp (the modification time unless other times are selected). For files with a time that is more than 6 months old or more than 1 hour into the future, the timestamp contains the year instead of the time of day.

-r –**reverse** –Sort directory contents in reverse order. As with all the other one character options, you can conbine options, for example: `ls -ltr`

-F –**classify** –Append a character to each file name indicating the file type. For regular files that are executable, append a '*'. The file type indicators are '/' for directories, '@' for symbolic links, '—' for FIFOs, '=' for sockets, and nothing for regular files.

-R –**recursive** –List the contents of all directories recursively.

-S –**sort=size** –Sort directory contents by file size instead of alaphabetically, with the largest files listed first.

-b –**escape** –Quote nongraphic characters in file names using alphabetic and octal backslash sequences like those used in C.

-c –**time=ctime** –Sort directory contents according to the files' status change time instead of the modification time. If the long listing format is being used, print the status change time instead of the modification time.

-k –**kilobytes** –If file sizes are being listed, print them in kilobytes. This overrides the environment variable POSIXLY_CORRECT.

-m –**format=commas** –List files horizontally, with as many as will fit on each line, separated by commas.

-n –**numeric-uid-gid** –List the numeric UID and GID instead of the names.

-p –Append a character to each file name indicating the file type.

-q –**hide-control-chars** –Print question marks instead of nongraphic characters in file names.

-s –**size** –Print the size of each file in 1K blocks to the left of the file name. If the environment variable POSIXLY_CORRECT is set, 512-byte blocks are used instead.

-t –**sort=time** –Sort directory contents by timestamp instead of alphabetically, with the newest files listed first.

-u –**time=atime** –Sort directory contents according to the files' last access time instead of the modification time. If the long listing format is being used, print the last access time instead of the modification time.

-x –**format=across** –List the files in columns, sorted horizontally.

-A –**almost-all** –List all files in directories except for '.' and '..'.

-B –**ignore-backups** –Do not list files that end with ' ', unless they are given on the command line.

-C –**format=vertical** –List files in columns, sorted vertically.

-L –**dereference** –List the files linked to by symbolic links instead of listing the contents of the links.

-N –**literal** –Do not quote file names.

-Q –**quote-name** –Enclose file names in double quotes and quote nongraphic characters as in C.

-U –**sort=none** –Do not sort directory contents.

-X –**sort=extension** –Sort directory contents alphabetically by file extension (characters after the last '.'); files with no extension are sorted first.

-1 –**format=single-column** –List one file per line.

-w –**width** –cols Assume the screen is cols columns wide. The default is taken from the terminal driver if possible; otherwise the environment variable COLUMNS is used if it is set; otherwise the default is 80.

-T –**tabsize** –cols Assume that each tabstop is cols columns wide. The default is 8.

-I –**ignore** –pattern Do not list files whose names match the shell pattern pattern unless they are given on the command line. As in the shell, an initial '.' in a filename does not match a wildcard at the start of pattern.

Listing and creating files with `cat`

The command `cat` copies its arguments to standard output. One useful option is:

-b –**number-nonblank** –Number all nonblank output lines, starting with 1.

If the file is more than a few lines, you will want to pipe the output to a pager: `cat /root/.emacs || more`. You can also create files with `cat`: `cat >/bin/ll`
`ls -la $*`
d̂
This file, containing instructions to be passed to the shell, can be executed with `sh /bin/ll /bin/l* /bin/c*`. The `$*` passes the arguments given to `ll` on to `ls`. We don't have to use the file as a script given to the shell; we can make it executable itself with, say, `chmod 755 /bin/ll` (see `chmod` below).

File Permissions and chmod

Chmod will accept symbolic mode specifications. I prefer numeric mode specifications myself. A numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values 4, 2, and 1. Any omitted digits are assumed to be leading zeros. The first digit selects the set user ID (4) (for an example of this, do `ls -l /usr/X386/bin`; the suid bit is used by programs which start other programs–it passes its permissions on to its 'children') and set group ID (2) and save text image (1) attributes. The second digit selects permissions for the user who owns the file: read (4), write (2), and execute (1); the third selects permis- sions for other users in the file's group, with the same values; and the fourth for other users not in the file's group, with the same values.

-v Verbosely describe changed permissions.

-R Recursively change permissions of directories and their contents.

Kermit

Kermit is a widely available communications package. The file /root/.kermrc is the initialization file I use (both on my local linux and the remote unix site that I log into).

```
set escape 29 ;ie press ^] to escape to local kermit
set file type binary ;
set send packet-length 1000
set receive packet-length 1000
set flow none
set line /dev/modem
set baud 9600
set window 5
set block 3
```

Kermit has extensive online help. However, let's walk thru a simple session. First, run `kermit`, press 'c' at the kermit prompt. Then, if using a modem, enter `atdt 123-4567`, for example. Once connected, login and do what you need (for example:

```
ftp nic.funet.fi
anonymous
tdunbar@vtaix.cc.vt.edu
cd pub/OS/Linux/testing/Linus
get ALPHA-patch.diff.z
get linux-1.01.tar.z
quit
```

To transfer these files, one might do: `kermit -i -s *.z`, when prompted by the server, enter ^c `receive` to get the files.

Gzip compression

The `gzip` utility is linked (in the Milieu package) to `compress`, `uncompress`, and `zcat` so one can just do: `uncompress ALPHA-diff;mv ALPHA-diff /usr/src/linux`.

Tar archives

To remove the old kernel source, one can do: `rm -r /usr/src/linux` **Warning:** this can be a very dangerous command–know what you are doing. Depending on how permissions are set and your user and group id, you may also need the `-f` option to force the removal.

We can unpack the above kernel source with: `mv linux-1.01.tar.z /usr/src; tar zxvf linux-1.01.tar.z` (the z tells tar to uncompress, x=extract, v=verbose, f=file to extract from; there is also a p=permissions option to preserve the original permissions–this may be needed if you set default permissions with `umask`, I don't).

Patching source code

The above patches can be applied with `patch -p1 < ALPHA-diff` (the -p1 tells patch to strip off one subdirectory layer since we're already in `/usr/src/linux` [which we could have renamed via `mv linux linux-1.01`]. If we were in `/usr/src` instead, we'd have used `-p0`).

Finding files

The `find` utility enables one to search through a directory hierarchy for files matching certain conditions and then to perform actions on the matching files. For example,

`find . -name "*#" -exec ls -l {} \;`

will list any files unsuccessfully patched by the above patch and the old files which were modified can be deleted with:

`find . -name ''*~'' -exec rm -v {} \;`

Making a new kernel

To make a new boot image,

```
make clean;
emacs Makefile; # change video default, etc., as needed
make config;
make dep; make clean; make Image;
cd /etc/lilo; ./install
```

Booting via lilo

Lilo comes with excellent and extensive documentation which should be consulted. The above kernel make assumed 1) `/usr/src/linux/Image` was **not** the default boot kernel–if it were, and something went wrong, we'd have to reboot via floppy. 2) lilo is installed and there's an entry for `/usr/src/linux/Image`. My `/etc/lilo/install` is: `./lilo -v -v -C config` where config is:

```
install = /etc/lilo/boot.b
compact
```

```
delay = 9 # optional, for systems that boot very quickly
boot = /dev/hda
image = /etc/vmlinux
label = linux
image = /usr/src/linux/Image
label = tlinux
other = /dev/hda1
table = /dev/hda
label = dos
```

Then, when one reboots, as soon as the word LILO appears, press ALT or CTRL and then enter tlinux to boot from your new kernel.

stty

`stty 9600 cs8 ixon </dev/ttys2` will setup com2 for a serial connection to a LaserWriter.

Bourne Again SHell

The default key-bindings for bash may be changed with an ~/.inputrc file. Other programs that use this library may add their own commands and bindings.

For example, placing

```
M-Control-u: universal-argument
```

or

```
C-Meta-u: universal-argument
```

into the .inputrc would make M-C-u execute the command universal-argument.

The following symbolic character names are recognized: RUBOUT, DEL, ESC, NEWLINE, SPACE, RETURN, LFD, TAB.

The bindings are meant to be compatible with the Emacs default keybindings where appropriate.

Commands for Moving

beginning-of-line (C-a)
Move to the start of the current line.

end-of-line (C-e)
Move to the end of the line.

forward-char (C-f)
Move forward a character.

backward-char (C-b)
Move back a character.

forward-word (M-f)
Move forward to the end of the next word.

backward-word (M-b)
Move back to the start of this, or the previous, word.

clear-screen (C-l)
Clear the screen leaving the current line at the top of the screen.

Commands for Manipulating the History

accept-line (Newline, Return)
Accept the line regardless of where the cursor is. If this line is non-empty, add it to the history list according to the state of the history_control variable. If this line was a history line, then restore the history line to its original state.

previous-history (C-p)
Fetch the previous command from the history list, moving back in the list.

next-history (C-n)
Fetch the next command from the history list, moving forward in the list.

beginning-of-history (M-<)
Move to the first line in the history, the first line entered.

end-of-history (M->)
Move to the end of the input history, i.e., the line you are entering.

reverse-search-history (C-r)
Search backward starting at the current line and moving 'up' through the history as necessary. This is an incremental search.

forward-search-history (C-s)
Search forward starting at the current line and moving 'down' through the history as necessary.

shell-expand-line (M-C-e)
Expand the line the way the shell does when it reads it. This performs alias and history expansion.

insert-last-argument (M-., M-_)
Insert the last argument to the previous command (the last word on the previous line).

operate-and-get-next (C-O)
Accept the current line for execution and fetch the next line relative to the current line from the history file for editing.

Commands for Changing Text

delete-char (C-d)
Delete the character under the cursor. If the cursor is at the beginning of the line, and there are no characters in the line, and the last character typed was not C-d, then return EOF.

backward-delete-char (Rubout)
Delete the character behind the cursor. A numeric arg says to kill the characters instead of deleting them.

quoted-insert (C-q, C-v)
Add the next character that you type to the line verbatim. This is how to insert characters like C-q, for example.

tab-insert (M-TAB)
Insert a tab character.

self-insert (a, b, A, 1, !, ...)
Insert the character typed.

transpose-chars (C-t)
Drag the character before point forward over the character at point. Point moves forward as well. If point is at the end of the line, then transpose the two characters before point. Negative arguments don't work.

transpose-words (M-t)
Drag the word behind the cursor past the word in front of the cursor moving the cursor over that word as well.

upcase-word (M-u)
Uppercase the current (or following) word. With a negative argument, do the previous word, but do not move point.

downcase-word (M-l)
Lowercase the current (or following) word. With a negative argument, do the previous word, but do not move point.

capitalize-word (M-c)
Capitalize the current (or following) word. With a negative argument, do the previous word, but do not move point.

Killing and Yanking

kill-line (C-k)
Kill the text from the current cursor position to the end of the line. This saves the killed text on the kill-ring. (see below)

backward-kill-linen ()
Kill backward to the beginning of the line. This is normally unbound, in favor of unix-line-discard, which emulates the behavior of the standard Unix terminal driver.

kill-word (M-d)
Kill from the cursor to the end of the current word, or if between words, to the end of the next word.

backward-kill-word (M-Rubout)
Kill the word behind the cursor.

unix-line-discard (C-u)
Do what C-u used to do in Unix line input. We save the killed text on the kill-ring, though.

unix-word-rubout (C-w)
Do what C-w used to do in Unix line input. The killed text is saved on the kill-ring. This is different than backward-kill-word because the word boundaries differ.

yank (C-y)
Yank the top of the kill ring into the buffer at point.

yank-pop (M-y)
Rotate the kill-ring, and yank the new top. Only works following 'yank' or 'yank-pop'.

Arguments

digit-argument (M-0, M-1, ..., M--)
Add this digit to the argument already accumulating, or start a new argument. M-- starts a negative argument.

universal-argument ()
Do what C-u does in emacs. By default, this is not bound to a key.

Completing

complete (TAB)
Attempt to perform completion on the text before point. Bash attempts completion treating the text as a variable (if the text begins with $), username (if the text begins with ~), hostname (if the text begins with @), or command (including aliases and functions) in turn. If none of these produces a match, filename completion is attempted.

possible-completions (M-?)
List the possible completions of the text before point.

complete-filename (M-/)
Attempt filename completion on the text before point.

possible-filename-completions (C-x /)
List the possible completions of the text before point, treating it as a filename.

complete-username (M-~)
Attempt completion on the text before point, treating it as a username.

possible-username-completions (C-x ~)
List the possible completions of the text before point, treating it as a username.

complete-variable (M-$)
Attempt completion on the text before point, treating it as a shell variable.

possible-variable-completions (C-x $)
List the possible completions of the text before point, treating it as a shell variable.

complete-hostname (M-@)
Attempt completion on the text before point, treating it as a hostname.

possible-hostname-completions (C-x @)
List the possible completions of the text before point, treating it as a hostname.

Miscellaneous

abort (C-g)
Abort the current editing command and ring the terminal's bell.

do-uppercase-version (M-a, M-b, ...)
Run the command that is bound to the uppercased key.

prefix-meta (ESC)
Metafy the next character typed. This is for people without a meta key. ESC f is equivalent to Meta-f.

undo (C-_)
Incremental undo, separately remembered for each line.

revert-line (M-r)
Undo all changes made to this line. This is like typing the 'undo' command enough times to get back to the beginning.

display-shell-version (C-x C-v)
Display version information about the current instance of bash.

emacs-editing-mode (C-e)
When in vi editing mode, this causes a switch to emacs editing mode.

vi-editing-mode (M-C-j or M-C-m)
When in emacs editing mode, this causes a switch to vi editing mode.

Getting the MetaFont and TEX files

Suppose you want to recompile MetaFont?

FTP to ftp.cs.umb.edu

Get the latest `web.tar.Z` (Knuth's web source for TEX, MetaFont, and friends) and `web2c.tar.Z` (utility to generate C code from the web (Pascal) source). These are available at various sites; the above is Karl Berry's (the current web2c maintainer) home site. As I write this, the current version for web2c is 5.851d. Unpack these files in some convenient directory (`tar zxvf web.tar.Z; tar zxvf web2c.tar.Z`) and `cd` into the top directory (e.g., web2c-5.851d).

Edit Makefile.in

Changes you may wish to make: 1) change the prefix from `/usr/local` to `/usr/TeX`, 2) change `CFLAGS` from `-g` to `-O2` (no need to debug, optimize) and add `-s` to `LDFLAGS` (use shared libraries), and 3) add // to the end of path components if you want subdirectories to be searched (the implementation will work fastest if the top directory, say `/usr/TeX/lib/mf/macros//`, just contains subdirectories).

You may want to change the `EDITOR` default in `lib/c-auto.h` (I set it to `TeXeditor` which is a symlink to my editor.

Note: As mentioned in the text, I've modified section 679 of the source and also the openinout.c file; you may also want to.

Now run make

When (if?) the compiler exits due to the error from the `malloc` def in lex.yy.c, just run `jove +37 lex.yy.c`, for example, and delete the unneeded `malloc` def. Then you can:

make; make install-programs;

Other than the programs themselves, various TEX and MetaFont files are needed.

TEX archives

ftp pip.shsu.edu

Or other convenient site to get the needed files. For example, at shsu,
`cd tex-archive/fonts/mf; get cm.tar.Z` will pack up the `cm` subdirectory which has the needed MetaFont font source code. You may also want to get the `tfm` (TeX font Metric) files while you are there.

Only a few TEX files are essential: `plain.tex, hyphen.tex`. There are various macro packages you may also want (for plain TEX, I recommend `stables.tex, newsletr.tex`) such as LaTEX.

Certainly, it is simplest to get a prepackaged collection of all the needed TEX and MetaFont files. The SLS archive (disks t1-t3) contains an extensive package for Linux.

Initialization files

/etc/rc

```
/etc/update &
/bin/rm -f /etc/mtab~ /etc/utmp
>/etc/utmp;/bin/chmod 444 /etc/utmp
/bin/cp /etc/smtab /etc/mtab
/etc/mount -a
/etc/swapon -a
/etc/ctrlaltdel soft
/etc/hostname your.ip.name.here
/bin/rm -f /tmp/.X0-lock
#/bin/stty 9600 cs8 ixon </dev/ttys1
/bin/clock -s
/etc/setterm -store -term console \
        -foreground black \
        -background white > /dev/tty1
/etc/setterm -blank 10 -store -term console \
        -clear all -foreground black \
        -background white > /dev/tty2
echo " rc complete"
#/etc/rc.inet
/bin/rm -f /etc/nologin
```

/etc/rc.inet

To set networking, uncomment the call to `rc.inet` after you've:

1 Created a file /etc/resolv.conf containing a line with: `nameserver <your.nameserver.ip.here>`

2 Edited rc.inet for your local site editing all the $< your.ip.goes.here >$ entries for your site (search for $<$ in rc.inet).

A minimal, unedited rc.inet might look like:

```
#! /bin/sh
  # Attach the loopback device.
  /etc/ifconfig lo 127.0.0.1
  /etc/route add 127.0.0.1 #ie can look in /etc/hosts for addresses
  # Set up the Ethernet connection(s).
  /etc/ifconfig eth0 <your.ip.nr.here> #(optional) <your.netmask.goes.here>
  # Set up the primary (static) routes.
  /etc/route add <put.misc.router.here> #add as many as you wish
  /etc/route add default gw <your.default.gateway.here> metric 1
  NET="/conf/etc"
  # Start the SYSLOG daemon.  This has to be the first server.
  echo -n "INET: "
```

```
 if [ -f ${NET}/syslogd ]
 then
       echo -n "syslogd "
       ${NET}/syslogd
 fi
 # Start the INET SuperServer
 if [ -f ${NET}/inetd ]
 then
       echo -n "inetd "
       ${NET}/inetd
 else
       echo "no INETD found.  INET cancelled!"
       exit 1
 fi
#/conf/etc/httpd -d /usr/lib/httpd  #if you want that
```

/etc/profile

```
export MODEM=/dev/modem
export TERM=vt100
export DISPLAY=":0"
export PATH=/etc:/bin:/conf/bin:/usr/X386/bin/:/usr/TeX/bin:/conf/etc:/usr/andrew/bin
history_control=ignoredups
/bin/stty sane
ulimit -c 0
PS1='`pwd` > '
PS2='  > '
CDPATH=".:~:/root/work:/usr/TeX/lib/tex:/usr/TeX/lib/mf:/usr/X386/lib/X11:/usr/lib/httpd"
HISTFILESIZE=100
```

Appendix 3

Misc. Emacs Notes:

- When loading files, etc., Emacs will prompt with a default directory. If you want a different directory, just start with / (i.e., give the absolute pathname). For example, if the prompt is `/book/` you can type `/etc/rc` which will be loaded (even though the minibuffer has `/book//etc/rc`).

For reference, here are default Emacs key-bindings (this is the refcard.tex by Stephen Gildea, copyright 1993, Free Software Foundation) that comes with GNU Emacs 19:

GNU Emacs Reference Card

(for version 19)

Starting Emacs

To enter GNU Emacs 19, just type its name: `emacs`

To read in a file to edit, see Files, below.

Leaving Emacs

suspend Emacs (or iconify it under X)	`C-z`
exit Emacs permanently	`C-x C-c`

Files

read a file into Emacs	`C-x C-f`
save a file back to disk	`C-x C-s`
save **all** files	`C-x s`
insert contents of another file into this buffer	`C-x i`
replace this file with the file you really want	`C-x C-v`
write buffer to a specified file	`C-x C-w`

Getting Help

The Help system is simple. Type `C-h` and follow the directions. If you are a first-time user, type `C-h t` for a **tutorial**.

remove Help window	`C-x 1`
scroll Help window	`ESC C-v`
apropos: show cmds matching string	`C-h a`
show the function a key runs	`C-h c`
describe a function	`C-h f`
get mode-specific information	`C-h m`

Error Recovery

abort partially typed or executing command	`C-g`
recover a file lost by a system crash	`M-x recover-file`
undo an unwanted change	`C-x u` or `C-_`
restore a buffer to its original contents	`M-x revert-buffer`
redraw garbaged screen	`C-l`

Incremental Search

search forward	`C-s`
search backward	`C-r`
regular expression search	`C-M-s`
reverse regular expression search	`C-M-r`
select previous search string	`M-p`
select next later search string	`M-n`
exit incremental search	`RET`
undo effect of last character	`DEL`
abort current search	`C-g`

Use `C-s` or `C-r` again to repeat the search in either direction. If Emacs is still searching, `C-g` cancels only the part not done.

Motion

entity to move over	**backward**	**forward**
character	`C-b`	`C-f`
word	`M-b`	`M-f`
line	`C-p`	`C-n`
go to line beginning (or end)	`C-a`	`C-e`
sentence	`M-a`	`M-e`
paragraph	`M-{`	`M-}`
page	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
function	`C-M-a`	`C-M-e`
go to buffer beginning (or end)	`M-<`	`M->`

scroll to next screen	`C-v`
scroll to previous screen	`M-v`
scroll left	`C-x <`
scroll right	`C-x >`
scroll current line to center of screen	`C-u C-l`

Killing and Deleting

entity to kill	**backward**	**forward**
character (delete, not kill)	`DEL`	`C-d`
word	`M-DEL`	`M-d`
line (to end of)	`M-0 C-k`	`C-k`
sentence	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill **region**	`C-w`
copy region to kill ring	`M-w`
kill through next occurrence of *char*	`M-z` *char*
yank back last thing killed	`C-y`
replace last yank with previous kill	`M-y`

Marking

set mark here	`C-@` or `C-SPC`
exchange point and mark	`C-x C-x`
set mark *arg* **words** away	`M-@`
mark **paragraph**	`M-h`
mark **page**	`C-x C-p`
mark **sexp**	`C-M-@`
mark **function**	`C-M-h`
mark entire **buffer**	`C-x h`

Query Replace

interactively replace a text string	`M-%`
using regular expressions	`M-x query-replace-regexp`

Valid responses in query-replace mode are

replace this one, go on to next	`SPC`
replace this one, don't move	`,`
skip to next without replacing	`DEL`
replace all remaining matches	`!`
back up to the previous match	`^`
exit query-replace	`ESC`
enter recursive edit (`C-M-c` to exit)	`C-r`

Multiple Windows

delete all other windows	`C-x 1`
delete this window	`C-x 0`
split window in two vertically	`C-x 2`
split window in two horizontally	`C-x 3`
scroll other window	`C-M-v`
switch cursor to another window	`C-x o`
shrink window shorter	`M-x shrink-window`
grow window taller	`C-x ^`
shrink window narrower	`C-x {`
grow window wider	`C-x }`
select buffer in other window	`C-x 4 b`
display buffer in other window	`C-x 4 C-o`
find file in other window	`C-x 4 f`
find file read-only in other window	`C-x 4 r`
run Dired in other window	`C-x 4 d`
find tag in other window	`C-x 4 .`

Formatting

indent current **line** (mode-dependent)	`TAB`
indent **region** (mode-dependent)	`C-M-\`
indent **sexp** (mode-dependent)	`C-M-q`
indent region rigidly *arg* columns	`C-x TAB`
insert newline after point	`C-o`
move rest of line vertically down	`C-M-o`
delete blank lines around point	`C-x C-o`
join line with previous (with arg, next)	`M-^`
delete all white space around point	`M-\`
put exactly one space at point	`M-SPC`
fill paragraph	`M-q`
set fill column	`C-x f`
set prefix each line starts with	`C-x .`

Case Change

uppercase word	`M-u`
lowercase word	`M-l`
capitalize word	`M-c`
uppercase region	`C-x C-u`
lowercase region	`C-x C-l`
capitalize region	`M-x capitalize-region`

The Minibuffer

The following keys are defined in the minibuffer.

complete as much as possible	`TAB`
complete up to one word	`SPC`
complete and execute	`RET`
show possible completions	`?`
fetch previous minibuffer input	`M-p`
fetch next later minibuffer input	`M-n`
regexp search backward through history	`M-r`
regexp search forward through history	`M-s`
abort command	`C-g`

Type `C-x ESC ESC` to edit and repeat the last command that used the minibuffer. The following keys are then defined.

previous minibuffer command	`M-p`
next minibuffer command	`M-n`

GNU Emacs Reference Card

Buffers

- select another buffer — `C-x b`
- list all buffers — `C-x C-b`
- kill a buffer — `C-x k`

Transposing

- transpose **characters** — `C-t`
- transpose **words** — `M-t`
- transpose **lines** — `C-x C-t`
- transpose **sexps** — `C-M-t`

Spelling Check

- check spelling of current word — `M-$`
- check spelling of all words in region — `M-x ispell-region`
- check spelling of entire buffer — `M-x ispell-buffer`

Tags

- find a tag (a definition) — `M-.`
- find next occurrence of tag — `C-u M-.`
- specify a new tags file — `M-x visit-tags-table`

- regexp search on all files in tags table — `M-x tags-search`
- run query-replace on all the files — `M-x tags-query-replace`
- continue last tags search or query-replace — `M-,`

Shells

- execute a shell command — `M-!`
- run a shell command on the region — `M-|`
- filter region through a shell command — `C-u M-|`
- start a shell in window `*shell*` — `M-x shell`

Rectangles

- copy rectangle to register — `C-x r r`
- kill rectangle — `C-x r k`
- yank rectangle — `C-x r y`
- open rectangle, shifting text right — `C-x r o`
- blank out rectangle — `M-x clear-rectangle`
- prefix each line with a string — `M-x string-rectangle`

Abbrevs

- add global abbrev — `C-x a g`
- add mode-local abbrev — `C-x a l`
- add global expansion for this abbrev — `C-x a i g`
- add mode-local expansion for this abbrev — `C-x a i l`
- explicitly expand abbrev — `C-x a e`

- expand previous word dynamically — `M-/`

Regular Expressions

- any single character except a newline — `.` (dot)
- zero or more repeats — `*`
- one or more repeats — `+`
- zero or one repeat — `?`
- any character in the set — `[ ... ]`
- any character not in the set — `[^ ... ]`
- beginning of line — `^`
- end of line — `$`
- quote a special character *c* — `\c`
- alternative ("or") — `\|`
- grouping — `\( ... \)`
- *n*th group — `\n`
- beginning of buffer — `` \` ``
- end of buffer — `\'`
- word break — `\b`
- not beginning or end of word — `\B`
- beginning of word — `\<`
- end of word — `\>`
- any word-syntax character — `\w`
- any non-word-syntax character — `\W`
- character with syntax *c* — `\sc`
- character with syntax not *c* — `\Sc`

Registers

- save region in register — `C-x r s`
- insert register contents into buffer — `C-x r i`

- save value of point in register — `C-x r SPC`
- jump to point saved in register — `C-x r j`

Info

- enter the Info documentation reader — `C-h i`

Moving within a node:

- scroll forward — `SPC`
- scroll reverse — `DEL`
- beginning of node — `.` (dot)

Moving between nodes:

- **next** node — `n`
- **previous** node — `p`
- move **up** — `u`
- select menu item by name — `m`
- select *n*th menu item by number (1–5) — *n*
- follow cross reference (return with `l`) — `f`
- return to last node you saw — `l`
- return to directory node — `d`
- go to any node by name — `g`

Other:

- run Info **tutorial** — `h`
- list Info commands — `?`
- **quit** Info — `q`
- search nodes for regexp — `s`

Keyboard Macros

start defining a keyboard macro	`C-x (`
end keyboard macro definition	`C-x )`
execute last-defined keyboard macro	`C-x e`
append to last keyboard macro	`C-u C-x (`
name last keyboard macro	`M-x name-last-kbd-macro`
insert Lisp definition in buffer	`M-x insert-kbd-macro`

Commands Dealing with Emacs Lisp

eval **sexp** before point	`C-x C-e`
eval current **defun**	`C-M-x`
eval **region**	`M-x eval-region`
eval entire **buffer**	`M-x eval-current-buffer`
read and eval minibuffer	`M-ESC`
re-execute last minibuffer command	`C-x ESC ESC`
read and eval Emacs Lisp file	`M-x load-file`
load from standard system directory	`M-x load-library`

Simple Customization

Here are some examples of binding global keys in Emacs Lisp. Note that you cannot say `"\M-#"`; you must say `"\e#"`.

```
(global-set-key "\C-cg" 'goto-line)
(global-set-key "\C-x\C-k" 'kill-region)
(global-set-key "\e#" 'query-replace-regexp)
```

An example of setting a variable in Emacs Lisp:

```
(setq backup-by-copying-when-linked t)
```

Writing Commands

```
(defun command-name (args)
  "documentation"
  (interactive "template")
  body)
```

An example:

```
(defun this-line-to-top-of-window (line)
  "Reposition line point is on to top of window.
With ARG, put point on line ARG.
Negative counts from bottom."
  (interactive "P")
  (recenter (if (null line)
                0
              (prefix-numeric-value line))))
```

The argument to `interactive` is a string specifying how to get the arguments when the function is called interactively. Type `C-h f interactive` for more information.

designed by Stephen Gildea, May 1993 v2.0
for GNU Emacs version 19 on Unix systems

For copies of the GNU Emacs manual, write to the Free Software Foundation, Inc., 675 Massachusetts Ave, Cambridge MA 02139.

.emacs

Here is a sample .emacs file:

```
(setq text-mode-hook 'turn-on-auto-fill)
(setq make-backup-files nil)
(put 'eval-expression 'disabled nil)

;; S=shift, C=control
(global-set-key [f1] 'help-for-help)
(global-set-key [S-f1] 'apropos)
(global-set-key [C-f1] 'describe-key)
(global-set-key [C-S-f1] 'describe-function)
(global-set-key [f2] 'scroll-other-window)
(global-set-key [S-f2] 'center-line)
(global-set-key [f3] 'isearch-forward)
(global-set-key [S-f3] 're-search-forward)
(global-set-key [f4] 'query-replace)
(global-set-key [S-f4] 'query-replace-regexp)
(global-set-key [f5] 'global-set-key)
(global-set-key [S-f5] 'font-lock-mode)
(global-set-key [f6] 'calc)
(global-set-key [f7] 'mtex-insert-typewriter)
(global-set-key [S-f7] 'mtex-insert-roman)
(global-set-key [f8] 'mtex-insert-italic)
(global-set-key [S-f8] 'mtex-insert-bold)
(global-set-key [C-f8] 'mtex-insert-brackets)
(global-set-key [f9] 'save-buffer)
(global-set-key [S-f9] 'latex-mode)
(global-set-key [C-f9] 'plain-tex-mode)
(global-set-key [f10] 'buffer-menu)
(global-set-key [f11] 'eval-print-last-sexp)
(global-set-key [f12] 'goto-line)
(global-set-key [C-up] 'backward-paragraph)
(global-set-key [C-down] 'forward-paragraph)
(global-set-key [C-left] 'backward-word)
(global-set-key [C-right] 'forward-word)
(global-set-key [C-prior] 'beginning-of-buffer)
(global-set-key [C-next] 'end-of-buffer)
(global-set-key [home] 'beginning-of-line)
(global-set-key [end] 'end-of-line)
(global-set-key [C-insert] 'insert-file)
(global-set-key [mouse-3] 'mouse-save-then-kill)
(global-set-key [mode-line mouse-2] 'mouse-tear-off-window)
```

```
(setq auto-mode-alist (append '(("\\.texi$" . tex-mode)) auto-mode-alist))
(setq auto-mode-alist (append '(("\\.mf$" . tex-mode)) auto-mode-alist))

;;; Commands added by calc-public-autoloads on Tue Jun  1 10:52:18 1993.
(autoload 'calc-dispatch          "calc" "Calculator Options" t)
(autoload 'full-calc                  "calc" "Full-screen Calculator" t)
(autoload 'full-calc-keypad          "calc" "Full-screen X Calculator" t)
(autoload 'calc-eval                  "calc" "Use Calculator from Lisp")
(autoload 'defmath                  "calc" nil t t)
(autoload 'calc                           "calc" "Calculator Mode" t)
(autoload 'quick-calc                   "calc" "Quick Calculator" t)
(autoload 'calc-keypad                   "calc" "X windows Calculator" t)
(autoload 'calc-embedded          "calc" "Use Calc inside any buffer" t)
(autoload 'calc-embedded-activate  "calc" "Activate =>'s in buffer" t)
(autoload 'calc-grab-region          "calc" "Grab region of Calc data" t)
(autoload 'calc-grab-rectangle          "calc" "Grab rectangle of data" t)
(autoload 'edit-kbd-macro          "macedit" "Edit Keyboard Macro" t)
(autoload 'edit-last-kbd-macro          "macedit" "Edit Keyboard Macro" t)
(autoload 'read-kbd-macro          "macedit" "Read Keyboard Macro" t)
(setq load-path (append load-path (list "/usr/lib/emacs/site-lisp/calc")))
(global-set-key "\e#" 'calc-dispatch)
;;; End of Calc autoloads.


;;; Mode settings stored by Calc on Wed Jun  2 11:56:42 1993
(setq calc-complex-format 'i)
(setq calc-angle-mode 'rad)
(setq calc-language 'tex)
;;; End of mode settings

;; if you like colors:
;  (load-file "/usr/lib/emacs/site-lisp/hilit19.el")
;  (add-hook 'find-file-hooks 'hilit-highlight-after-find t)
;  (setq hilit-auto-highlight-maxout 200000) ; set a higher autohighlight max

;what shell to use:
    (setq shell-file-name "/bin/bash")

;not needed with auc-tex

    (defun mft-buffer ()
    (interactive)
    (start-process "mft" "out" "mft" (buffer-name) "-s" "/book/mplain.mft"))
```

```
(defun mf-buffer ()
"run METAFONT on the buffer"
(interactive)
(start-process "mf" "METAFONT" "mf" "screenstrokes; input " (buffer-name)))

(defun mtex-buffer ()
(interactive)
(start-process "tex" "out" "tex" (buffer-name)))

;    (defun mtex-insert-bold (s) "insert bold text."
;    (interactive "sBold text:  ")
;    (insert "{\\bf ") (insert s) (insert "}"))

(defun mtex-insert-bold (s) "insert bold text."
(interactive "sBold text:  ")
(insert "<b> ") (insert s) (insert "</b>"))

;    (defun mtex-insert-italic (s) "insert italic text."
;    (interactive "sItalic text:  ")
;    (insert "{\\it ") (insert s) (insert "}"))
;
;    (defun mtex-insert-typewriter (s) "insert typewriter text."
;    (interactive "sTypewriter text:  ")
;    (insert "{\\tt ") (insert s) (insert "}"))
;
;    (defun mtex-insert-roman (s) "insert roman text."
;    (interactive "sRoman text:  ")
;    (insert "\\rm ") (insert s) (insert "}"))

(defun mtex-insert-italic (s) "insert italic text."
(interactive "sItalic text:  ")
(insert "<i> ") (insert s) (insert "</i>"))

(defun mtex-insert-typewriter (s) "insert typewriter text."
(interactive "sTypewriter text:  ")
(insert "<tt> ") (insert s) (insert "</tt>"))

(defun mtex-insert-roman (s) "insert roman text."
(interactive "sRoman text:  ")
(insert "<pre> ") (insert s) (insert "</pre>"))

(defun mtex-insert-brackets (s) "insert command."
(interactive "sCommand:  ")
(insert "<") (insert s) (insert ">"))
```

```
;; default size of torn-off frames
    (setq default-frame-alist
          (append (list
                   '(width . 81)
                   (cond
                    ((> (x-display-pixel-height) 1023) '(height . 32))
                    ((> (x-display-pixel-height)  899) '(height . 27))
                    (t                                 '(height . 26)))
                   )
                  default-frame-alist))



(defun space-then-fill (fillp)
  "Insert SPACE then fill-paragraph"
  (interactive "P")
  (insert " ")
  (fill-paragraph fillp)
  (if (eolp)
                (insert " ")
        )
)

;then use:
;  (local-set-key " " 'space-then-fill)
; to use it, ie for auto filling of paragraphs as
; one goes along


; (load-file "/usr/lib/emacs/site-lisp/auctex/tex-site.elc")
;   (load "auc-tex") ;for old keymappings
;   (setq-default TeX-master t)
;   (setq TeX-auto-global nil)
;     (setq-default TeX-parse-self nil) ; Disable parse on load.
;     (setq-default TeX-auto-save nil) ; Disable parse on save.

;;colors for point and pointer
(if (equal window-system 'x)
          (progn
            (transient-mark-mode 1) ;highlighting on
            (display-time) ;display time and Mail
            ;(setq baud-rate 19200) ; used to be a bug under X
```

```
  ;(set-default-font "10x20")
  (set-border-color "blue")
  (set-foreground-color "Black")
  (set-background-color "bisque")
  (set-cursor-color "blue")
  ;;needs set-mouse-color to be effective
  (setq x-pointer-shape x-pointer-left-ptr)
  (set-mouse-color "green3")
  ;(menu-bar-mode -1) ;no menus
  ;(toggle-scroll-bar -1)  ;no scroll bars
))
```

Appendix 4

The main difficulty in configuring X is in adjusting for one's monitor and video card. The file `/usr/X386/lib/X11/VideoModes.Doc` gives detailed instructions on setting the video timings. X allows far more flexibility than DOS configurations. The downside of this is that one has to do the configuration oneself. A crucial point is to ensure that one doesn't try to drive one's monitor past its ability–**this can damage the monitor!** To check the horizontal refresh rates needed by the below, start emacs, press `ESC x calc` and then enter: '[25/800 28/800 36/1024 40/1056 44/1264] 1000 *. This will give a vector giving the horizontal refresh in kilohertz. I've commented out the fast refreshs; still, you should definitely read (and understand) the VideoModes.Doc file and modify the Xconfig appropriately before starting X.

```
#mach8 config (with commented out stuff for vga256,S3,Mach32)
FontPath        "/usr/lib/X11/fonts/misc/"
FontPath        "/usr/lib/X11/fonts/75dpi/"
FontPath        "/usr/andrew/X11fonts/"
Keyboard
  AutoRepeat 500 5
  ServerNumLock
ps/2            "/dev/ps2aux"
#MouseSystems       "/dev/ttys1"
#Microsoft      "/dev/ttys2"
#Busmouse       "/dev/mouse"
Emulate3Buttons
#vga256
vga2
  Virtual       1024 768
  ViewPort      0 0
  Modes         "1024x768" "640x480h"
vga16
  Virtual       640 480
  ViewPort      0 0
  Modes         "640x480h" "640x480"
#  Clocks      25 28 65 36 0 78 38 48   #vga2
accel
  Virtual       1024 768
#  Virtual       640 480
  ViewPort      0 0
  Modes         "1024x768" "1024x664" "800x600" "640x480"
#  Modes         "640x480h" "640x480hh"
  Clocks 43.00  49.00  92.50  36.30  50.50  56.60   0.00  44.90
         30.40  32.20 110.90  80.50  40.00  44.80  75.50  65.20
         21.50  24.50  46.25  18.15  25.25  28.30   0.00  22.45
         15.20  16.10  55.45  40.25  20.00  22.40  37.75  32.60
#  Clocks 43 49 92 36 50 56 0 45 30 32 110 80 40 44 75 65 #Mach8
```

```
#Mach32
#Clocks  100 126  92  36  51  57   0  44 135  32 110  80  39  45  75  65
#           50  63  46  18  25  28   0  22  67  16  55  40  19  23  37  33
#Clocks 25 28 39 71 49 93 37 46 127 120 77 31 110 64 74 94 #S3 801

ModeDB
# clock  horzontal timing      vertical timing
 "640x480"     25      640  664  760  800    480  491  493  525
 "640x480h"    28      640  664  760  800    480  489  492  520
 "640x480hh"  31.5    640  664  704  832    480  489  492  520
 "800x600"     36      800  824  896 1024    600  601  603  625
#              48       800  880  944 1080    600  639  643  690  -hsync -vsync
 "1024x664"    65     1024 1032 1176 1304    664 664 672 680
# "1024x768"    65     1024 1048 1184 1344    768  771  777  806
 "1024x768"    65     1024 1088 1176 1304    768  768  774  790
              75     1024 1048 1184 1288    768  771  777  806  -hsync -vsync
              78     1024 1056 1336 1368    768  768  774  790 -hsync -vsync
               80     1024 1056 1336 1368    768  776  780  807
#              80     1024 1088 1176 1304    768  776  780  807
 "1024x868"    80     1024 1088 1176 1304    868  870  876  890
 "1280x1024"  110     1280 1320 1480 1728   1024 1029 1036 1077
```

.Xdefaults

Here is a sample .Xdefaults (the olvwm and Openwindows resources are just for olvwm):

```
aMosaic*TitleFont: -adobe-times-bold-r-normal-*-14-*-100-100-*-*-iso8859-1
aMosaic*Font: -adobe-times-medium-r-normal-*-12-*-100-100-*-*-iso8859-1
aMosaic*ItalicFont: -adobe-times-medium-i-normal-*-12-*-100-100-*-*-iso8859-1
aMosaic*BoldFont: -adobe-times-bold-r-normal-*-12-*-100-100-*-*-iso8859-1
aMosaic*FixedFont: -adobe-courier-medium-r-normal-*-12-*-100-100-*-*-iso8859-1
aMosaic*Header1Font: -adobe-times-bold-r-normal-*-14-*-100-100-*-*-iso8859-1
aMosaic*Header2Font: -adobe-times-bold-r-normal-*-12-*-100-100-*-*-iso8859-1
aMosaic*Header3Font: -adobe-times-bold-r-normal-*-12-*-100-100-*-*-iso8859-1
aMosaic*Header4Font: -adobe-times-bold-r-normal-*-10-*-100-100-*-*-iso8859-1
aMosaic*Header5Font: -adobe-times-bold-r-normal-*-10-*-100-100-*-*-iso8859-1
aMosaic*Header6Font: -adobe-times-bold-r-normal-*-08-*-100-100-*-*-iso8859-1
aMosaic*AddressFont: -adobe-times-medium-i-normal-*-12-*-100-100-*-*-iso8859-1
aMosaic*PlainFont: -adobe-courier-medium-r-normal-*-10-*-100-100-*-*-iso8859-1
aMosaic*ListingFont: -adobe-courier-medium-r-normal-*-10-*-100-100-*-*-iso8859-1
Mosaic*homeDocument: http://128.173.6.137/public.html
Mosaic*catchPriorAndNext: true
*StickyIconScreen: true
emacs.font:        -adobe-courier-medium-r-normal--14-*
Emacs&geometry:        81x26-0+0
emacs.title:        emacs
Scrollbar.JumpCursor:        True
```

```
*Scrollbar*background:        Gray80
xdvi*Offset:        .2cm
xdvi*expert:        true
xdvi*geometry:        +0+0
xdvi*shrinkFactor:        3
XTerm*JumpScroll:        true
XTerm*font:        10x20
XTerm*noScroll:        true
XTerm*VT100.geometry: 81x26-0+0
mf*geometry:        -0+0
mf*height:        600
mf*width:        800
rxvt*font:        7x14bold
Xfm*file window*dir_icon.foreground:        green3
Xfm*file window*exe_icon.foreground:        red
Xfm*file window*file_icon.foreground:        blue
Xfm*file window*other_icon.foreground:        black
Xfm*file window.Geometry:        550x230+0+200
Xfm.defaultDisplayType:        Text
Xfm.applicationDataFile:        ~/.xfmrc
Xfm.showDate:        false
Xfm.defaultEditor:        exec xterm -e jove
Xfm.confirmCopies:        false
Xfm.showLength:        false
Xfm.defaultSortType:        SortByDate
Xfm.confirmDeletes:        false
Xfm.showOwner:        false
Xfm.initialDisplayType:        Text
Xfm.Geometry:        450x170+5+5
Xfm.confirmMoves:        false
Xfm.showPermissions:        false
*multiScroll:        true
*bitmapFilePath:        /usr/X386/include/X11/bitmaps
*timeFormat:        C
*numeric:        C
*background:        linen
*bottomShadowcontrast:        40
*topShadowContrast:        20
*displayLang:        C
*basicLocale:        C
*shadowWidth:        2
*inputLang:        C
*beNiceToColormap:        false
*shapeStyle:        Rectangle
```

.fvwmrc

Here's the rc file for the feeble window manager I'm using:

```
SmartPlacement
StdForeColor            Black
StdBackColor            #ffe4c4
HiForeColor             Black
HiBackColor             sandybrown
Font                    7x14
WindowFont                -adobe-helvetica-bold-r-*-*-*-100-*-*-*-*-*-*
IconFont                -adobe-helvetica-medium-r-*-*-*-100-*-*-*-*-*-*
PagerFont    5x7
DeskTopSize 2x2
DeskTopScale 16
Pager -1 +1
IconBox 0 0 400 150
IconBox -200 100 -1 500
StaysOnTop Fvwm Pager
EdgeScroll 0 0
NoTitle rclock
NoTitle GoodStuff
Sticky rclock
Sticky xclock
Sticky xdaliclock
Sticky xcalendar
Sticky xbiff
StaysOnTop xclock
WindowListSkip xclock
WindowListSkip rclock
WindowListSkip xbiff
Icon "rxvt" /usr/X386/include/X11/bitmaps/term.xpm
Icon "GoodStuff" /usr/X386/include/X11/bitmaps/toolbox.xpm
Icon "xterm" /usr/X386/include/X11/bitmaps/xterm.xpm
Icon "x3270-4" /usr/X386/include/X11/bitmaps/term.xpm
Icon "rclock" /usr/X386/include/X11/bitmaps/clock.xpm
Icon "xcalendar" /usr/X386/include/X11/bitmaps/datebook.xpm
Icon "mfgterm" /usr/X386/include/X11/bitmaps/sm-mf.xbm
Icon "mfterm" /usr/X386/include/X11/bitmaps/metafont.xpm
Icon "tex" /usr/X386/include/X11/bitmaps/sm-tex.xbm
Icon "texterm" /usr/X386/include/X11/bitmaps/tex.xpm
Icon "vtaix" /usr/X386/include/X11/bitmaps/kterm.xpm
Icon "pixmap" /usr/X386/include/X11/bitmaps/pixmap.xpm
Icon "Fvwm Pager" /usr/X386/include/X11/bitmaps/porsche.xpm
Icon "xman" /usr/X386/include/X11/bitmaps/xman.xpm
Icon "emacs" /usr/X386/include/X11/bitmaps/emacs.xpm
```

```
Icon "xfm" /usr/X386/include/X11/bitmaps/next.xpm
Icon "xcalc" /usr/X386/include/X11/bitmaps/xcalc.xpm
Icon "cdrom" /usr/X386/include/X11/bitmaps/cdrom.xpm
Icon "clipboard" /usr/X386/include/X11/bitmaps/clipboard.xpm
Icon "linux" /usr/X386/include/X11/bitmaps/linux.xpm
Icon "mail" /usr/X386/include/X11/bitmaps/mail.xpm
Icon "postit" /usr/X386/include/X11/bitmaps/postit.xpm
Icon "sharks" /usr/X386/include/X11/bitmaps/sharks.xpm
Icon "xedit" /usr/X386/include/X11/bitmaps/xedit.xpm
Icon "xrn" /usr/X386/include/X11/bitmaps/xrn.xpm
Icon "xgrab" /usr/X386/include/X11/bitmaps/xgrab.xpm
Icon "xarchie" /usr/X386/include/X11/bitmaps/xarchie.xpm
Icon "unknown" /usr/X386/include/X11/bitmaps/unknown.xpm
Icon "xcalc" /usr/X386/include/X11/bitmaps/xcalc.xpm
#now define the menus - defer bindings until later
Popup "Quit-Verify"
        Title   "Really Quit Fvwm?"
        Quit    "Yes, Really Quit"
        Nop     "No, Don't Quit"
EndPopup
# This defines the most common window operations
Popup "Window Ops"
        Title                "Window Ops"
        Move                "Move"
        Resize                 "Resize"
        Raise           "Raise"
        Lower           "Lower"
        Iconify         "(De)Iconify"
        Stick                "(Un)Stick"
        Maximize         "(Un)Maximize"
        Maximize         "(Un)Maximize Vertical"        0 100
        Nop             ""
        Destroy         "Destroy"
        Delete          "Delete"
EndPopup

Popup "Applications" Applications
        Exec    "XTerm"         exec xterm -title Terminal -sb -bg linen -e bash&
        Exec    "TeX"           exec texterm -e jove &
        Exec    "METAFONT"      exec mfterm -sb -fn 7x14bold -e mf &
        Exec    "Mosaic"        exec Mosaic +0+30 &
        Exec    "Xdu"           exec xdu </mdirs &
        Exec        "Xv"            exec xv -vsperfect&
        Exec    "phonenrs"      exec xterm -e jove /root/phone.nrs &
```

```
    EndPopup
    Popup "Games" Games
            Exec "Tetris"                exec tetris &
            Exec "FreeCell"         exec xpat -rules FreeCell &
            Exec "Klondike"         exec xpat -rules Klondike &
            Exec    "x3270"         exec x3270 -geometry +0-0 &
    EndPopup
    Popup "Utilities"
            Title   "Utilities"
            Exec        "XTerm-vtaix"   exec xrlogin -telnet -sb -geometry 81x26+0-0 -fn
10x20 vtaix &
            EXEC        "X3270"         exec x3270 vtvm1.cc.vt.edu &
            Exec    "Xterm"         exec xterm -sb -fn 10x20 -bg linen -geometry 81x26+0-
0 -sb -e bash&
            Exec    "XTeX"                exec xtex &
            Exec    "Emacs" exec emacs -geometry 81x26+0-0 &
            Exec    "Mosaic"        exec Mosaic -geometry +0+30&
            Exec    "EZ"            exec ez /usr/andrew/doc/AtkTour/Media &
            Exec    "jove" exec xterm -fn 6x12 -geometry 80x14+0+0 &
            Exec    "Andrew Help"   exec ahelp ez &
            Exec    "Calendar"      exec xcalendar -geometry 270x220+530+0 &
    EndPopup

    ############################################################################
    # Now define some handy complex functions

    # This one moves and then raises the window if you drag the mouse,
    # only raises the window if you click,  or does a RaiseLower if you double
    # click
    Function "Move-or-Raise"
            Move                "Motion"
            Raise               "Motion"
            Raise               "Click"
            RaiseLower       "DoubleClick"
    EndFunction

    # This one moves and then lowers the window if you drag the mouse,
    # only lowers the window if you click,  or does a RaiseLower if you double
    # click
    Function "Move-or-Lower"
            Move                "Motion"
            Lower               "Motion"
            Lower               "Click"
            RaiseLower       "DoubleClick"
```

```
EndFunction

# This one resizes and then raises the window if you drag the mouse,
# only raises the window if you click,  or does a RaiseLower if you double
# click
Function "Resize-or-Raise"
        Resize                 "Motion"
        Raise                "Motion"
        Raise                "Click"
        RaiseLower        "DoubleClick"
EndFunction



# define the mouse bindings
#     Button    Context Modifi  Function
Mouse 1         R       N       PopUp "Window Ops"
Mouse 2         R       N       PopUp "Applications"
Mouse 3         R       N       PopUp "Utilities"

Mouse 1         1       N       PopUp "Window Ops"
Mouse 2         1       N       PopUp "Window Ops"
Mouse 3         1       N       PopUp "Window Ops"
Mouse 1         2       N       Iconify
Mouse 2         2I      N       Iconify
Mouse 3         I2      N       Iconify

Mouse 1         F       N       Resize
Mouse 1                TSI        A        Function "Move-or-Raise"
Mouse 2                I        A        Iconify
Mouse 2         I       N       Iconify
Mouse 2         FST     N       PopUp "Window Ops"
Mouse 3         TSF     N       RaiseLower

Key Left        A       C       Scroll -600 +0
Key Right       A       C       Scroll +600 +0
Key Up          A       C       Scroll +0   -600
Key Down        A       C       Scroll +0   +600

Key Left        R       N       Scroll -600 +0
Key Right       R       N       Scroll +600 +0
Key Up          R       N       Scroll +0   -600
Key Down        R       N       Scroll +0   +600
```

```
Key F1  A M Exec "help" exec xterm -e more /usr/man/cat7/fvwm.7 &
Key F2         A       M       Popup "Window Ops"
Key F3         A       M       Popup "Utilities"
Key F4         A       M       Move
Key F5         A       M       Resize

Key Next       A       M       CirculateUp
Key Prior      A       M       CirculateDown
Key n          A       M       CirculateUp
Key p          A       M       CirculateDown
Key r          A       M       Refresh
Key m          A       M       Move
Key s          A       M       Resize
Key l          A       M       RaiseLower
Key x          A       M       Destroy
Key i          A       M       Iconify

DecorateTransients
RandomPlacement
```

Appendix 5

Sample file structure

>

>

>

M

>

>

AMAKEDE

H

H

J>
J>

J>

>
>

>

>

>

>

E

E

E

E

E

E

E

E

DC
E
IAL

MK
>

E

>

>

>

D

F

F

F

>FGA
FGA

>

B

B

B

B

B
B

M
EDB
KDB

B

>

>

>E