Typesetting with TEX and LATEX

Alan Hoenig
17 Bay Avenue
Huntington, NY 11743
ajhjj@cunyvm.cuny.edu

(This presentation appears in a considerably expanded form as chapter 1 of my book *TEX Unbound: LATEX and TEX Strategies for Fonts, Graphics, and More* published just this year by the Oxford University Press.)

By *typesetting*, we mean the ability to place elements of a document on a page according to generally accepted principles which most people seem to agree look best and make it easiest to read and comprehend the document. It's surprisingly difficult to do that — a typesetter has to decide how best to break paragraphs into lines, how to hyphenate words, how to leave space for footnotes, how to prepare indexes and the other detritus of scholarly publishing, and provide the optimum space between elements on the page (among many other things). The spacing issue is particularly critical for technical documents. Formulas make extensive use of arcane symbols which have different appearances and spacing depending on context. Consider, for example, how the placement and spacing of the ordinary numeral '2' changes in

$$2x \qquad x^2 \qquad e^{-x^2}$$

and how the spacing surrounding minus sign changes in

$$x-y \quad \text{and} \quad -x+y$$

Other symbols may change depending upon whether the equation appears in text '$\int x\,dx$' or display mode:

$$\int x\,dx.$$

Furthermore, if a computer system is going to control the typesetting, we expect more of it than from a mere human. We may expect, for example, to be able to label an equation in some logical way and then refer to it later by this label in our source document. It would be up to the typesetter to resolve these labels and references and replace the labels by properly formatted label numbers.

The TEX system has been freely available since the mid-80s or so and accomplishes all of the above tasks (and more) in a particularly effective manner. TEX is the creation of Donald E. Knuth of Stanford University, who has placed all the source code for TEX in the public domain. The logo 'TEX' is related to the Greek root '$\tau\epsilon\chi$' from whence come words like 'technology'. If pronounced properly, the face of your listener may become slightly moist (but no one complains if you say 'tek').

The purpose of this survey is to acquaint readers with the aspects of the TEX cycle necessary to produce handsome papers and books. This presentation should *not* be regarded as a be-all-and-end-all tutorial, since (like many other mature and sophisticated software systems) lengthy books are not enough to do full justice to it.

The TEX production cycle

Why is 'typesetting' not the same thing as 'word processing'? Typically, a word processor allows editing of the document, but in its impatience to display the results immediately onscreen (most word processors are aggressively WYSIWYG in behavior), certain niceties are sacrificed. These niceties — fine control of spacing, word placement, hyphenation, and so forth — are never ignored in TEX.

It's useful to consider the TEX production cycle by comparing it with that of word processors. In a word processor, the program assists you in preparing the document, after which it is printed. TEX relies on three steps.

1. We use a *text editor* to prepare the *source document* — the document file which consists of the text and data of your document together with the TEX formatting commands. Let's suppose this file is called `myfile.tex`.
2. We run `myfile.tex` through the TEX program. If all goes well, this generates a file in which the typesetting commands are made explicit using a generic printer description language independent of any particular printer; it is *device independent*. TEX names this file `myfile.dvi`. Just like a computer program source file with syntax errors, if there are any errors, we return to step 1 and correct them before continuing.
3. Finally, we need the assistance of a special *device driver* customized to the printer. It's the

driver's task to translate the generic `dvi` commands into the form the printer understands.

The advantages of creating a `.dvi` file are that we can print the document on any printer (at least, any printer for which device drivers exist) and rest assured that the output is identical on each device (except for raster resolution).

Macros; logical document design

TEX has been called an assembly language for typesetting. This means that there are plenty of primitive commands to control fine points, but these commands may not be entirely appropriate for creating a new section head or aligned equation. As a result, TEX has a rich and powerful macro creation facility. It's possible (as we will see later) to string primitive commands and pre-existing macro commands together to create new, custom typesetting commands.

Remember that the TEX production cycle means that we prepare a source file which is fed into TEX at a later point. This plus the nature of the macros means that TEX supports the notion of *logical document design*. We can embed components of the document by means of tags which can be defined or redefined depending upon context. One example suffices. Here's a theorem.

Theorem There is no royal road to typesetting. Computer typesetting is a surprisingly complex task.

This was typeset by means of inserting

```
\theorem There ...
```

in the source document. It may happen that it is more appropriate to display that theorem as

THEOREM *There is no royal road to typesetting. Computer typesetting is a surprisingly complex task.*

The same command string will accomplish this *provided* that only the macro definition of `\theorem` needs be changed. The implications are enormous—we can design our document so that it will properly printed for any set of particular formatting requirements provided only that we change the particular definitions of the macros. Many strategies exist for facilitating this use of definitions.

A first TEX document

The "steps" for generating a TEX document are well-defined, but there are sufficiently idiosyncratic implementations of TEX floating around so that it may be necessary to adapt these procedures to a local adaptation. By the way, some readers may be interested in the TEX dialect called 'LATEX'; as we will see, LATEX is the same as TEX, so these procedures follow for a LATEX document as well.

1. Use a *text editor* to create the *source document* for subsequent processing by TEX. The source document is the document file—text and typesetting commands. Take care not to use a word processor. These programs aim to do the formatting themselves, and tend to do so by inserting non-Ascii characters into the document file. Quite apart from the fact that TEX (or LATEX) needs no help with the typesetting, these binary characters will only confuse TEX. (If it is necessary to use a word processor, make sure to save the document in some way so as not to include the word processing formatting information.)

2. Run this source file through the TEX program. the simplest form of the command to do that is

   ```
   tex myfile
   ```

   where the source file has the name `myfile.tex`. LATEX users will use the command

   ```
   tex &lplain myfile
   ```

   (Unix users may have to enter the ampersand as `\&`.)

   As in any compilation process, TEX may uncover errors. (Warnings may be ignored, at least at this stage.) Return to step 1 to correct these errors, and re-run it through TEX. Repeat this process until all errors have been dealt with (or until there is enough of a document to print.)

   The result of a successful TEX compilation is a new file with a `dvi` extension. In this example, we would have a new file `myfile.dvi`.

3. With the document in hand, it can be printed or previewed on screen. In each case, appropriate *device drivers* are necessary to properly render the document on screen or on paper.

As we see, the TEX process is actually a concerted action between several programs in addition to TEX—a text editor, a device driver, and a screen previewer. Many implementations of TEX may merge several or all of these into one integrated module.

The TEX document: input conventions

Although it is not practical or possible to deal with all or even a completely useful subset of all TEX (or LATEX) commands in this article, it is possible to summarize the keyboard conventions that any TEX typesetting must adhere to.

White space. TEX normally regards all white space as equivalent, where we include carriage returns, tabs, and of course spaces in this category. Furthermore, multiple spaces are generally equivalent to a single space. **Important exception:** we signal the end of one paragraph and the beginning of another by skipping a line in the source file; that is, we enter two hard carriage returns in a row. (But three or more consecutive carriage returns is still equivalent to a pair of carriage returns.)

Once in a while, spaces are special in that we don't want a line broken between two words or word groups. For example, in a discussion of World War I, it would look silly if a line broke between 'World War' and 'I'; it would be too confusing to the reader. To guarantee that the line break won't happen at that point, we replace the space with the tilde character `~`. If this *conditional space* is typeset in the middle of a line, it appears as a regular space. Aspiring TEX typists should develop the habit of typing things like `King Henry~VIII`, `Dr.~Knuth`, and `pages~44--55` to protect the manuscript from unwarranted line breaks. Note that whereas `Henry~VIII` will work as advertised, `Henry ~ VIII` will not. Here is one instance where users need to be careful.

Characters. We generate most characters by simply entering the character in the source document. That is, we type

```
Oh!  What a beautiful morning.
```

to get

Oh! What a beautiful morning.

See below for exceptions to this; certain special characters need be entered in a special way.

But TEX is smarter than that. Certain character pairs are replaced by special glyphs. For example, if we type ` `` ` or `''` we get true "quotes." With its special attention to details, TEX will replace certain character combinations such as `fi`, `fl`, and `ff` by the *ligatures* fi, fl, and ff (provided these ligatures are present in the current font).

TEX does a similar thing with hyphens and dashes. We can type `-`, `--`, or `---` to put -, –, or — in our documents. (And we will see later that the mathematical minus sign — yet a different dash — can be gotten using the hyphen character in mathematics mode.)

By the way, no user should *ever* put an explicit line-break hyphen in a word which has to be split at the end of a line. TEX's hyphenation algorithm takes care of such should a word break be necessary.

In summary, we type

```
``Oh, the selfish shell-fish---that
lobster mobster---tasted
best when basted west,'' quoth Aaron
while reading
pages~12--33 of his cookbook.
```

to typeset

"Oh, the selfish shell-fish — that lobster mobster — tasted best when basted west," quoth Aaron while reading pages 12 – 33 of his cookbook.

TEX formatting instructions and commands. TEX is very good about applying default typesetting parameters to text, but there will be many times when you wish to actively control the printed appearance of your document by issuing commands to TEX. Since the entire file must contain Ascii characters only, TEX has decided to reserve the meanings of certain characters to itself. The tilde `~` is one such special character. To typeset an actual tilde ˜ in the document, you must enter a short command to do so.

These characters

```
\  #  $  %  ^  &  ~  {  }
```

have special meanings to TEX. The backslash `\` is TEX's *escape character* — it escapes the normal meaning of the following bit of text. This character generally begins all of TEX's commands. For example, to typeset an ampersand &, you would type `\&`. Typesetting commands for some of these symbols are formed in the same way. (All the symbols can be typeset, but for some, addtional TEX expertise is needed.)

TEX can do àçc̈éñṫṣ as well. If you need them, check your main manual. We can get the Spanish punctuation marks ¿ and ¡ by typing `?‘` and `!‘`.

We will discuss the special TEX characters bit by bit, but commands generally begin with the escape character. The escape character can be followed one single non-letter, or by an arbitrary sequence of letters, terminated by a space. Examples of commands from the first category include `\&`, `\1`, `\$`, and `\"`. Examples of the second category include `\TeX`, `\L`, `\noindent`, `\vskip`, and `\futurelet`. Note that TEX is case sensitive, so the command `\TeX` (which typesets the TEX logo) is different from the (nonsense) commands `\tex` and `\TEX`.

These rules lead to our first piece of TEXarcana. As part of TEX's digestive process, it is smart enough to know that when a non-letter follows an escape character (the backslash), the command name consists only of a single letter. Hence, anything following that command, such as a space, is typeset as you expect. To get the sequence, '& &', type `\& \&`.

The situation is subtly different when a command name follows the backslash. For now, TEX has no way *a priori* to know the length of the command name. It reads your file, and terminates the command name when it encounters a space or a new command. Consequently the space following a command is 'eaten up' by TEX — it serves not introduce a space into the document but rather to delimit the command. But since at other times, spaces typed in the document file *do* generate a space, it's easy to see why newcomers are easily confused.

Anyway, to illustrate the point, suppose you wanted to typeset 'TEX TEX'. The way *not* to do it would be by entering `\TeX \TeX` into the source, for the interior space terminates the initial `\TeX` command and is therefore eaten alive. (`\TeX \TeX` typesets as TEXTEX.) Since multiple spaces count as a single space to TEX, the solution is *not* to insert addtional spaces. The following list, which suggests several ways out of the impasse, also hones your beginning TEX skills.

1. Use the TEX command which explicitly generates an interword gobber of space. This *control space* command is `\␣`, and so your source should like `\TeX\␣\TeX` if you use this method.
2. Terminate the command by inserting some command which does not print anything. The empty group `{}` is one such; thus, we could type `\TeX{} \TeX`.
3. Simply surround the command in its own group: `{\TeX} \TeX` is one appropriate way to do this.

A working TEX system

A complete TEX system is actually a concert between several different component pieces of hardware and software. There are at least three different but necessary pieces of software.

First is a version of TEX for a particular computer and operating system. At this time, there are versions of TEX available for every reasonable computer. In the unlikely event that there isn't, it's possible to customize TEX by doing a reasonable amount of spade work yourself. The TEX program is in the public domain (and in electronic form), and all you need to do is make whatever changes (if any) are called for and recompile the TEX source code in a robust Pascal compiler that works on your system. (Most likely, you will translate the original Pascal WEB source to a C source program using the freely available `web2c` utility, and then use a robust C compiler to compile TEX.) TEX was originally written in Pascal, and depending on how you "pretty print" the listing, it amounts to between 20,000 and 30,000 lines of code. TEX exercises all the dark corners of any compiler, so you need a compiler that has itself been thoroughly debugged. (There is a white lie of omission in this account. All this source code is written in WEB, so some mastery of this WEB system must be acquired.)

Do we need a special version of LATEX to match our hardware? The core LATEX files, which "sit" on top of TEX, are ASCII files, and we can easily transfer ASCII files from one computer to another. However, proper behavior of LATEX will require us to install LATEX and in that process to create a special binary format file for use by our computer. In general, binary files may not be transferred from one type of hardware to another (but format files are easy to construct).

Next is a *text editor*. This was discussed earlier and is necessary for preparing the document source file.

Finally there are the *device driver* and *screen previewer*. TEX's output is a file containing commands to typeset all the letters, rules, and special symbols in the document. Unfortunately, different printers obey distinctly different sets of such commands. Therefore, TEX employs a generic, no-frills, *device-independent* language in which to express these commands. That's why the output file from TEX has the extension `dvi`, to suggest *device-independence*. In this way, the TEX program is relevant to virtually any hardware setup, but it does mean that we need yet another program, a so-called device driver. The purpose of this program is simply to translate TEX's generic, device independent typesetting commands into commands that our particular printer understands.

Everybody will want to arm themselves with a *screen previewer*, a special-purpose device driver. Remember, TEX is not WYSIWYG, and we frequently want to see what the TEX document will look like without going to the bother of printing it out. (This may be because we share printing facilities in some computer center, or because your printer takes a long time to deliver a single page. Anyone who has tried generating TEX output on a dot matrix printer knows that feeling.) A video monitor is just a special purpose printing device, and it is usually a straightforward matter to write a device driver to paint the image of the page on a monitor screen.

It makes sense to choose hardware on the basis of TEX software. For example, you'll want to make *really* sure that the printer is one for which a device driver exists. (Or else make sure that the printer is one that will *emulate* — imitate — a supported printer. For example, there are many laser printers for sale, each with its own protocol for generating printed images. There are relatively few laser printer device drivers available. Among these few with support are the Hewlett-Packard laser printers, which many other laser printers emulate well. Since there are several Hewlett-Packard laser printer drivers available, an HP-like laser printer may be a safe bet.)

TEX also runs well on printers that understand the PostScript page description language. This PostScript language is another means for creating device-independent files, because the mechanism for rendering the PostScript document resides in the printer itself. Consequently, we need a special PostScript printer in order to take advantage of the PostScript technology. (That there are hundreds of beautiful digital PostScript fonts is another inducement to use PostScript.) Special dvi-to-PostScript postprocessors translate a dvi file to a PostScript equivalent. Many such programs are available from any number of vendors. Fortunately, one of the best, dvips by Tomas Rokicki, is freely available.

Getting TEX

Although the TEX software is "free" — within the public domain — it often takes work to port it to a particular computer. This is true of implementations for the original IBM PC and for the Apple Macintosh, for example. In any case, device drivers and screen previewers were never part of the original TEX package. Consequently, some firms sell their implementations for TEX.

An interesting recent phenomenon is the availability of several public domain TEX implementations for microcomputers. One or more such implementations exist for all kinds of personal computer, including IBM-type computers, Apple Macintosh, Amiga, Atari, and Acorn. Some of them may even be as good or better than commercial products. Of course, when we use a public domain version, we are on our own. Companies have "helplines" for users who find themselves in trouble. With rare exceptions, no such lifelines exist for users of public domain TEXs. Public domain implementations are available from user groups (TUG, Dante, GUT, and so on), from Internet archives, and from special TEX CDROMs.

When acquiring a TEX "package," make sure it's complete. In addition to the TEX executable, the associated ancillary files TEX needs, various important input files, and the latest version of important macro packages, we must make sure additional utilities are part of the suite even if your current plans don't include using them. I have in mind here the METAFONT program (and the MP program if possible), various TEXware utilities (of which tftopl, vftovp and their inverses pltotf and vptovf are probably the most important), and the METAFONTware utilities. The TEX program should be version 3.1415 or higher, and the METAFONT program should be version 2.71 or higher.

Unique TEXs. Although each implementation of TEX is typographically equivalent to any other, a few are worthy of special notice by virtue of some distinguishing feature. A few of these special TEXs are worthy of mention here.

The em-TEX software collection is especially interesting — it's a complete implementation of TEX, METAFONT, all TEXware and MFware programs, printer drivers, previewers, and documentation (in English and German) for PC-DOS and OS/2 operating systems, and it's all free. Several executables of TEX and METAFONT are provided, from "small" to "huge" versions. (These designations refer to the speed and/or the amount of material these TEXs and METAFONTS can process.) Eberhard Mattes is the man behind this prodigious effort, but there are those who wonder if "Eberhard Mattes" doesn't refer, like the fictional "Nicolas Bourbaki," to a dedicated group of workers. In any case, this material is all available free for downloading from any CTAN site, and some user groups make it available to their members for a nominal fee. Furthermore, there is a special em-TEX Internet list, so this is one important instance where public domain software does have some support. Over the last several years, Mattes has proven to be a conscientious developer, providing bug fixes in a timely way and keeping up with the latest master source files of TEX, METAFONT, and their friends.

Tom Rokicki is well-known in TEX circles for his dvips post-processor (it converts dvi output to a form suitable for rendering on a PostScript printer). Less well known but just as impressive is TEXView, his version of TEX for the NextStep operating system. NextStep, which runs on the Intel-486 architecture among others, is a flavor of Unix (BSD 4.2) onto which has been grafted a very convenient windowing system. NextStep contains a version of Display PostScript, which means that TEX documents that incorporate PostScript graphics and PostScript fonts may be previewed effortlessly (including color). Because Unix is a multitasking system, an author can run a document through TeX, begin the previewing process, and continue editing. One odd feature has proven invaluable — the ability to measure actual distances on a page with clicks of a mouse. It's surprising how often it is possible to fix a bug in a TeX file by simply knowing how much extra or missing space there is. Most of the TeXView enhancements can be found incorporated into the `web2c` TeX kits for Unix platforms. (As a result of various corporate acquisitions, NextStep is now called OpenStep.)

Lightning Textures, by Blue Sky Research, is a version of TeX for the Macintosh. Its distinguishing feature is its ability to show TeX output produced simultaneously as text is keyed in. (And look for its newly-arrived sibling "Synchronicity.") The freely available InstantTeX for NextStep (originally by Dmitri Linde and now maintained by Gregor Hoffleit) provides the same functionality — it's great for debugging macros. InstantTeX is freely available from the `ftp` site `peanuts.leo.org` and others, in the area

`pub/comp/platforms/next/Text/tex/apps`

and its mirrors. The file will have a name like `InstantTeX.3.11d.NIHS.b.tar` and there is an accompanying "`readme`" file as well.

AucTeX is not an implementation, but an editing enhancement available to Unix users of the Emacs editor. With it, Emacs becomes highly TeX-aware, making available a large number of shortcuts, command completion, automatic indentation, special outlining, online documentation, the ability to customize (provided you can program in Lisp, the language in which it is programmed), and a good bit more. Kresten Krab Thorup is its author, and it is available from any CTAN site.

Inking the page

Neither TeX nor LaTeX (nor any other macro package) actually paints the page. TeX simply creates a file, the `dvi` file, which precisely records the position of each character, rule, and other graphic element in the document. Other programs take responsibility for using this information to place ink on the page. Remember, the TeX program needs only to know how much space is required for each character, rule, and typographic element on the page.

The characters of a font. It's the province of the device driver (or previewer, just another type of device driver) to deal with the characters of a digital font. Information in the `dvi` file tells it where to position each character, and then the driver paints each character where it is supposed to be.

Bitmap fonts. How does the driver know these shapes? One way to store shape information is within a collection of *pixel files*. Computer printers generate their shapes by putting lots of tiny dots next to each other in such a way that they form patterns which our eyes resolve into letters and graphs. Office laser printers, for example, are capable of placing as many as 600 dots per inch (or more) to the left or right and up or down. Only the dots needed to create a character are printed, and the human eye smoothes out any jaggedness as it perceives the image. Pixel files, or *bitmap fonts*, provide instructions as to which dots to blacken and which to leave blank.

There are lots of pixel files because TeX needs a different file for each font of "type" at each size and at each magnification. What's the difference between the *size* of a font and the *magnification* of a font? Type designers of old took care to slightly redesign each font of type at different sizes. Subtle issues of spacing require that the proportions between thick and thin strokes, the size of the white areas within some letters, and so on be readjusted at each size. *The TeXbook* makes this point early on, on page 16, which shows the difference between 10-point type and 5-point type magnified two hundred percent. That and the following demonstrations have members from the Computer Modern Roman families. Computer Modern Roman type at a 10-pt design size (type size or just *size*) is referred to as `cmr10`.

Although TeX has been designed to appreciate this subtle difference in type sizing, many computer typesetting systems do *not*. Therefore, prevailing electronic fonts of type construct different type sizes by taking a single font and magnifying it by whatever amount necessary to get the size you want. That is, there is generally no difference between *type size* and *magnification* unless you work with Computer Modern types. (Recently, there are some indications that the non-TeX world may be beginning to perceive the importance of this distinction. Adobe's Multiple Master technology is one sign of this trend.)

The many different font files that normally come with TeX often confuse users, but now we know why they are necessary. TeX *does* distinguish between size and magnification, and TeX's ability to make this distinction calls this diversity into being.

On DOS systems (and any others that impose rigid lengths on possible file names), the necessity for having many different fonts at different sizes and magnifications calls an intricate directory structure into being. All `cmr10` fonts at any magnification have the name `cmr10.pk`. The magnification is distinguished by creating different directories to hold pixel files with like magnifications. For fonts rendered to print on a 300 dpi printer, fonts at magnification 1000 (normal size) appear in directories named something like

```
\tex\fonts\dpi300
```

while for 120% magnification, since $360 = 1.2 \times 300$, the directory will be

```
\tex\fonts\dpi360
```

Scalable fonts and a PostScript postscript. PostScript technology has come to dominate the world of computer typesetting and desktop publishing. Files fully (and carefully) prepared in PostScript are device-independent and are generally ASCII (so they can be freely transferred across computer platforms). "Device independence" means any PostScript device. (With the Level 2 enhancement, PostScript supports some binary encoding, so such files may no longer be pure ASCII. Such files may still be moved across platforms as Adobe took great care to ensure this.)

Since PostScript files are independent of the printer resolution, fonts for PostScript cannot rely on bitmap descriptions, which are inherently tied to a printer's resolution. Each character in a PostScript font is given by a mathematical description of the outline of that character (hence, the appellation 'outline font'); this description is not dependent on any printer resolution. Nevertheless, any raster printing device is, in the final analysis, a set of bitmap images, but it's the job of the special PostScript printer to resolve the outline descriptions to their bitmap equivalents.

PostScript fonts are called *scalable fonts* because the technology scales these outline descriptions up or down to get different sizes.

TEX and PostScript technology coexist quite companionably. To render a `dvi` file on a PostScript printer, the `dvi` file must be translated to the PostScript language. A variety of dvi-to-PostScript converters exist for this purpose; one such, and one of the most highly regarded, is the freely available `dvips` by Tomas Rokicki. (It has been compiled for virtually every computer platform.) The end product of the translation is a series of statements in the PostScript language, which are either transmitted immediately to the printer or saved to a special file with a `ps` extension.

Beware — these `dvi` postprocessors are savvy enough to embed bitmap descriptions into the output `.ps` file in the proper format for printing on a PostScript printer. Only now the file has become device-dependent — it will print properly only on the printer for which the bitmaps were created. PostScript does try to scale the bitmap image, but an inevitable loss of quality accompanies this operation. Thus, if you plan to print the document later via phototypesetter, you'll have to regenerate the `.dvi` and `.ps` files with the proper bitmaps.

Document files

What do TEX commands look like? TEX reserve ten characters for their own use. Otherwise, when TEX encounters a letter or symbol such as an "A" or "9" it interprets the symbol as a command to typeset that symbol (to typeset an uppercase A or the numeral 9). We call the character that alerts TEX to an immediately following command the *escape character*, but this character bears no relation to the key marked "escape" or "esc" on many keyboards.

We issue explicit commands to TEX by means of one of several hundred commands beginning with the backslash, the usual escape character for TEX. Immediately following the backslash are one or more characters, which may be followed by additional information that the command needs. For example, the command

```
\noindent
```

suppresses indentation at the beginning of a paragraph while

```
\vspace{1in}
```

instructs LATEX to skip one inch of vertical space.

It's important that special reserved characters begin or introduce special formatting instructions, because you never know when you might want to include the word "noindent" or "vspace" within the document. By the way, there are special commands to typeset any of the reserved symbols, so it is possible (and easy) to typeset a backslash or a dollar sign within the document.

TEX contains a rich set of several hundred or so commands. Although by themselves they do just about anything we might wish, their real strength flows from the ability to string commands together to form new commands for special typesetting purposes.

We might create our own personal `\newchapter` command to begin a new page, skip down a third of the page, center the chapter title, skip a quarter of an inch, suppress indentation on the first paragraph, and set the first two words of the chapter in a small caps font. These new commands are called *macros*.

Creating a macro is a lot like writing a computer program. TEX possesses several commands to test conditions and perform action on the basis of the test, to perform looping, and to handle input/output operations. Tricky macros can take a long time to write and test, but more often than not, it's possible to write simple macros on the fly that make typesetting much easier.

TEX lets us place all our personal macro definitions into a separate *style file*. The document would contain special instructions for TEX to read and assimilate those macros before typesetting. Publishers, for example, could exploit this by making generic style files available to their authors while commissioning a designer to prepare a style file to implement a specialized book layout. The typeset document file remains the same, and only the macro definitions change. LATEX exploits this philosophy to the hilt.

Learning and joining

Although the numbers of users of TEX are (as of yet) dwarfed in magnitude by users of famous commercial products, the growth in use of TEX is astonishing. For consider this: an idiosyncratic product, passed along by word of mouth, has no public-relations dollars behind it. The reaction of users learning it was "free" was often the same — if it is so good, why isn't anyone selling it?

But as these words are written, it has become clear that there will always be a prominent place for TEX. Although there is widespread perception that TEX may be difficult to use well, no desktop publishing package is particularly easy. Furthermore, TEX has been able to perform certain nice things from the beginning that some famous commercial programs are still struggling over, and this has won the hearts of several mainstream publishers. Scientists the world over will never relinquish TEX — how else will they *unambiguously* capture on the page their technical expressions? Furthermore, the many millions of pages of LATEX and TEX electronic manuscript already in existence require TEX's continued presence.

Joining the TEX community. TEX, METAFONT, and their friends form a rich set of tools. Many workers have spent many happy hours adapting TEX to various specialized tasks or to creating special front ends that might be just what you were looking for. Although the many computer networks form a platform for the communication of this news, the various user groups provide a more formal forum for the exchange of important news. Not only do the larger of these groups publish their own newsletters, but they often sponsor annual meetings.

The TEX Users Group — TUG — is the original user group organization. Originally an offshoot of the American Mathematical Society, it is now an independent organization. TUG serves as a clearinghouse for all information on TEX, and members receive the journal *TUGboat*, the transactions of the TEX User Group.

Other user group organizations have since arisen. New user groups, particularly behind the former Iron Curtain, are constantly being formed. (The attention of English-speaking readers must certainly be drawn to UK TUG, whose journal *Baskerville* is well worthwhile.)