

**Beyond Word Processors
Text Processing with UNIX**

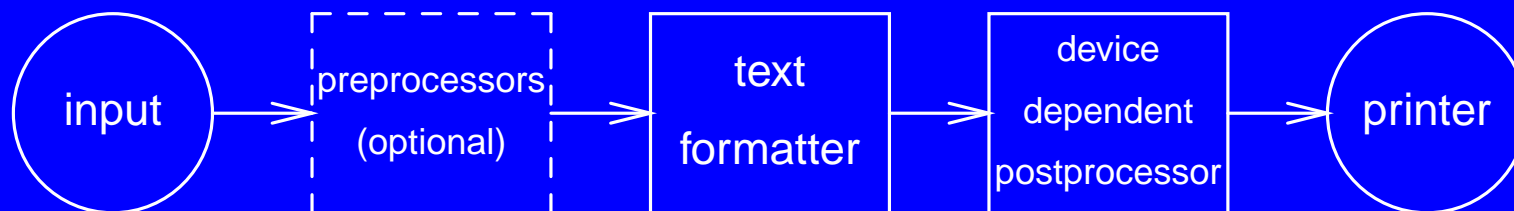
Brian Mays

March 2000

Note: This presentation was prepared using the *seminar* document class for \LaTeX with some preprocessing provided by GNU `pic`. Several examples were typeset using `groff`, `tbl`, `eqn`, and `grap`.

What is Text-Processing?

- The traditional publishing process is author (manuscript) → typographic designer (format) → typesetter (matrix of metal type)
- Text-processors (also called “text formatters”) act as the typesetter
- The input must include *both* the text to be typeset and the instructions explicitly describing the format
- Fortunately, with the power of a computer, macros can be used to automate and facilitate the formatting commands
- Ultimately, these macros can be used to divorce the logical structure of the document from the details of its appearance (*logical design*)



Word-Processing vs. Text-Processing

Word-Processors

- Typewriter like
- Use special (often proprietary) file formats
- Monolithic
- WYSIWYG
- Hidden control commands

Text-Processors

- Batch oriented
- Use plain text input
- Large collection of tools
- Visible control commands
- Macros
- Greater ability to decouple context from formatting details

Advantages of Text-Processing

- Professionally crafted layouts are available
- The user only needs to learn a few easy-to-understand commands which specify the logical structure of a document
- Even complex structures such as footnotes, references, table of contents, and bibliographies can be generated easily
- Macro packages encourage authors to write well-structured texts
- Requires smaller resources (larger documents)
- Able to tackle complex typographical problems
- More “typographically correct” (quotes, hyphenation, ligatures)
- Portable and Free

Disadvantages

- Higher learning curve (increased complexity)
- Sometimes difficult to coerce into non-standard behavior
- It is very hard to write unstructured and disorganized documents
- Feedback delay
- Requires debugging

Text-Processing in UNIX

TROFF (pronounced “tee-roff”) The traditional UNIX text formatting tools developed at Bell Labs

- Still essential to modern UNIX-like OSes, since it provides the text processing for manual pages
- On free operating systems, encountered as GNU `groff`

TEX (rhymes with “bleach”) The public domain text formatting system developed by Donald Knuth at Stanford University

- Particularly good for typesetting mathematics
- Used by the GNU project’s `texinfo` documentation system
- Has become the *de facto* standard text-processing system in many academic departments and research laboratories
- Adopted and heavily supported by the American Mathematical Society (AMS), who provide their own extensions ($\mathcal{A}\mathcal{M}\mathcal{S}\text{-TEX}$ and $\mathcal{A}\mathcal{M}\mathcal{S}\text{-LATEX}$)

A Comparison of TROFF and T_EX

Both . . .

- Accept text in free form
- Produce device independent output
- Allow for comments in the document source
- Allow for the definition of macros
- Use macro packages to provide a level of abstraction
- Can accept input that has been preprocessed

However . . .

- T_EX is better at typesetting mathematics
- T_EX has better device support for including figures, postscript images, etc.
- T_ro_ff (via n_ro_ff) is better at producing output for typewriter-like devices (terminals, line printers, etc.)
- T_ro_ff works better with UNIX pipes

The History of TROFF

- Troff began with UNIX
 - The PDP-11 on which UNIX was developed was acquired for a “text preparation project”
 - The first UNIX user was the Patent Department at BTL
- Troff developed at AT&T
 - First came `nroff` (“en-roff”) based on the earlier programs such as RUNOFF from the early 60’s
 - Written by Joe Ossanna in 1973, in assembly language for the PDP-11, to drive the Graphic Systems CAT-4 typesetter
 - Rewritten in C around 1975
 - Ossanna died late in 1977
 - Modified to make it device independent by Brian Kernighan in 1979

- Troff left AT&T with UNIX
 - Bell Labs developed several macro packages, including `ms` and `mm` and `man`
 - Berkeley adopted `troff` along with UNIX, and developed its own macro package: `me`
- Groff provides Troff for GNU systems
 - In the late 80's, James Clark began work on a GNU version of `troff`
 - Provides good compatibility with UNIX `troff` and adds several extensions

An Introduction to TROFF

- Ordinary text is interspersed with formatting commands
- Two types of commands:
 - On a line by itself (includes macros)

```
.PP  
.ft B  
This is a little bold font paragraph.
```

- Proceeded by a backslash

```
\s8UNIX\s10
```

- Input between the commands is free form
- Through macro packages, a document is described in terms of its logical parts—title, sections, paragraphs—not by details of spacing, fonts, and size.

The ms Macro Package

.TL

Title of document (one or more lines)

.AU

Author names, one per line

.AB

Abstract, terminated by .AE

.AE

.NH

Numbered heading (automatic numbering)

.PP

Paragraph . . .

.PP

Another paragraph . . .

.SH

Sub-heading (not numbered)

.PP

...

Provides macros for

- Title and author
- Abstract
- Sections
- Paragraphs
- Indented paragraphs
- Text size
- Fonts
- Displays, keeps, and floats
- Footnotes

Paragraphs: There are three ways to typeset paragraphs.

.PP

This is a normal indented paragraph.

.LP

This is paragraph without indentation.

.IP (1)

This is an indented paragraph with a number or mark in the margin.

This is a normal indented paragraph.

This is paragraph without indentation.

(1) This is an indented paragraph with a number or mark in the margin.

Size: It is possible to change the size of the text.

Typeset smaller
or larger text,
either `\s-2in-line\s0`
.LG
or with macros.

Typeset smaller or larger text,
either in-line or with macros.

Fonts: There are three ways to typeset "text" in a different font.

<code>.ft XX</code>	<code>\fX</code> or <code>\f(XX</code>	<code>.B</code> , <code>.I</code> , and <code>.R</code>
<code>.ft CW</code> <code>text</code> <code>.ft R</code>	<code>\fItext\fP</code>	<code>.B text</code>

```
This is roman, but
.I this
is italic, and
.B "these words"
are bold.
(parenthetical
.I "italic words" )
.B
The rest is bold.
```

```
This is roman, but this is italic,
and these words are bold.
(parenthetical italic words)
The rest is bold.
```

Keeps and Displays

.DS

This text is printed indented but without rearrangement.

.DE

.KS

This text is kept together. It will be forced to a new page if it won't all fit on the current page.

.KE

.KF

This text is kept together, but it will float past text. This is useful for tables.

.KE

Footnotes

.FS

\(dg Footnote text goes here. You are responsible for any mark like an asterisk or a dagger.[†]

.FE

Miscellaneous

.br causes a break

.bp begins a new page

.nf/.fi turns filling off/on

.ce centers the next line

.sp *n* skips *n* lines

.sp *ni* skips *n* inches

Running TROFF

Troff is executed on the command line as follows:

```
nroff -ms file
```

or

```
troff -ms file | postprocessor
```

Troff's output is device independent and therefore needs to be sent to a postprocessing program. Such programs include `grotty` which converts GNU `troff` output to a form suitable for typewriter-like devices, `grops` which converts GNU `troff` output to Postscript, and `grodvī` which converts the output to the DVI format used by $\text{T}_{\text{E}}\text{X}$.

Preprocessors

The UNIX Way

`Troff` can do almost anything, but its language is complicated and can be difficult to use. Therefore, preprocessors make it easy to do certain tasks. They implement their own “languages” that compile into `troff`. Several preprocessors are available:

`eqn` Typesets equations

`tbl` Typesets tables

`pic` Typesets pictures

`grap` A preprocessor for `pic` to make graphs

The power of UNIX pipes works very well with preprocessors. For example, *The C Programming Language* was typeset with

```
pic|tbl|eqn|troff -ms
```

EQN

History

- Written by Brian Kernighan and Lorinda Cherry at Bell Labs in 1974
- It was the first preprocessor for `troff` and the first non-compiler program built with `yacc`

Example

```
sum from i=0 to inf x sub i = pi over 2
```

produces

$$\sum_{i=0}^{\infty} x_i = \frac{\pi}{2}$$

The Language

- `.EQ/ .EN` turn on/off the preprocessor
- Two characters can serve as left and right delimiters of expressions

$$@E = m c \sup 2@$$

- The language is small and context-free
- Input is free form
- Spaces (or tildes) delimit the input
- Braces `{ }` are used to group objects together

$$x = \{-b \pm \sqrt{b \sup 2 - 4ac}\} \text{ over } 2a$$

- Quotes `" "` turn off grammatical significance

$$\lim \sim \text{roman "sup"} \sim x \text{ sub } n = 0$$

More Examples

```
.PP
A displayed equation is marked
with an equation number at the right margin
by adding an argument to the EQ line:
.EQ (1.3)
x sup 2 over a sup 2 ~ = ~ sqrt {p z sup 2 +qz+r}
.EN
```

A displayed equation is marked with an equation number at the right margin by adding an argument to the EQ line:

$$\frac{x^2}{a^2} = \sqrt{pz^2 + qz + r} \quad (1.3)$$

This equation is indented, rather than centered.
 This is accomplished by adding the I argument.

```
.EQ I (2.2a)
V bar sub nu~ = ~left [ pile {a above b above
c } right ] + left [ matrix { col { A(11) above .
above . } col { . above . above . } col { . above .
above A(33) } } right ] cdot left [ pile { alpha
above beta above gamma } right ]
.EN
```

This equation is indented, rather than centered. This is accomplished by adding the I argument.

$$\bar{\mathbf{V}}_v = \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} A(11) & . & . \\ . & . & . \\ . & . & A(33) \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \quad (2.2a)$$

Equations can be aligned by using mark in one equation and lineup in the other.

```
.EQ L
F hat ( chi ) ~ mark = ~ | del V | sup 2
.EN C
.EQ L
lineup =~
{left ( {partial V} over {partial x} right ) } sup 2
+ { left ( {partial V} over {partial y} right ) }
sup 2 ~~~~~ lambda -> inf
.EN
```

Equations can be aligned by using mark in one equation and lineup in the other.

$$\hat{F}(\chi) = |\nabla V|^2$$

$$= \left(\frac{\partial V}{\partial x}\right)^2 + \left(\frac{\partial V}{\partial y}\right)^2 \quad \lambda \rightarrow \infty$$

TBL

History Written by M. E. Lesk at Bell Labs shortly after eqn

Input Format

- .TS/ .TE turn on/off the preprocessor

- Format of each table is

```
options ;  
format .  
data
```

- Format is l for left, c for center, r for right, etc.
- Rows are separated by newlines
- Columns are separated by tabs (by default)
- .T& is used to specify a new format

Examples

```
.TS
box;
c c c; l l l.
Language      Authors      Runs on
Fortran       Many        Almost anything
PL/1          IBM          360/370
C             BTL          11/45,H6000,370
BLISS         Carnegie-Mellon PDP-10,11
IDS           Honeywell    H6000
Pascal        Stanford     370
.TE
```

Language	Authors	Runs on
Fortran	Many	Almost anything
PL/1	IBM	360/370
C	BTL	11/45,H6000,370
BLISS	Carnegie-Mellon	PDP-10,11
IDS	Honeywell	H6000
Pascal	Stanford	370

```
.TS
box;
L L L
LL _
LL|L B
LL _
LLL .
january february      march
april    may
june     july           Months
august   september
october  november       december
.TE
```

january	february	march
april	may	
june	july	Months
august	september	
october	november	december

More complex examples

Name	Definition
Gamma	$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$
Sine	$\sin(x) = \frac{1}{2i} (e^{ix} - e^{-ix})$
Error	$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$
Bessel	$J_0(z) = \frac{1}{\pi} \int_0^{\pi} \cos(z \sin \theta) d\theta$
Zeta	$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \quad (\operatorname{Re} s > 1)$

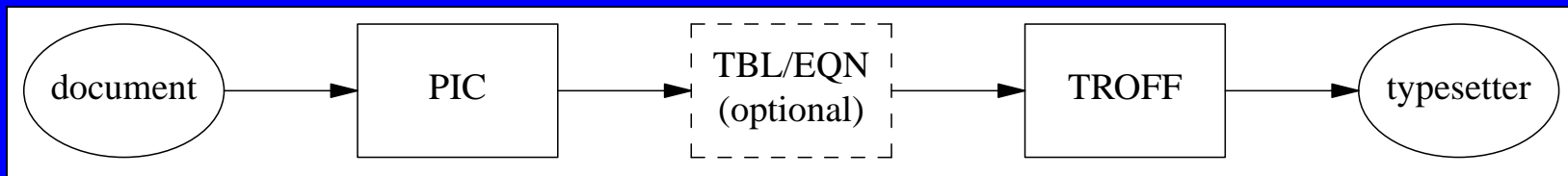
PIC

History Written by Brian Kernighan at Bell Labs

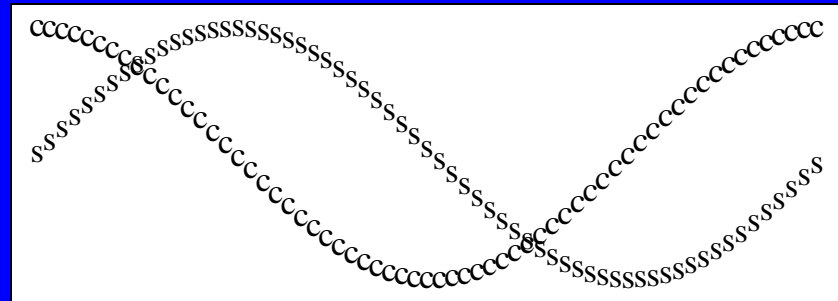
- A language for drawing simple figures
- `.PS/ .PE` turn on/off the preprocessor
- Basic objects are boxes, circles, ellipses, lines, arrows, arcs, spline curves, and text.
- These may be placed any-where, at positions specified absolutely or in terms of previous objects
- Braces `{ }` and brackets `[]` are used to group objects together

Examples

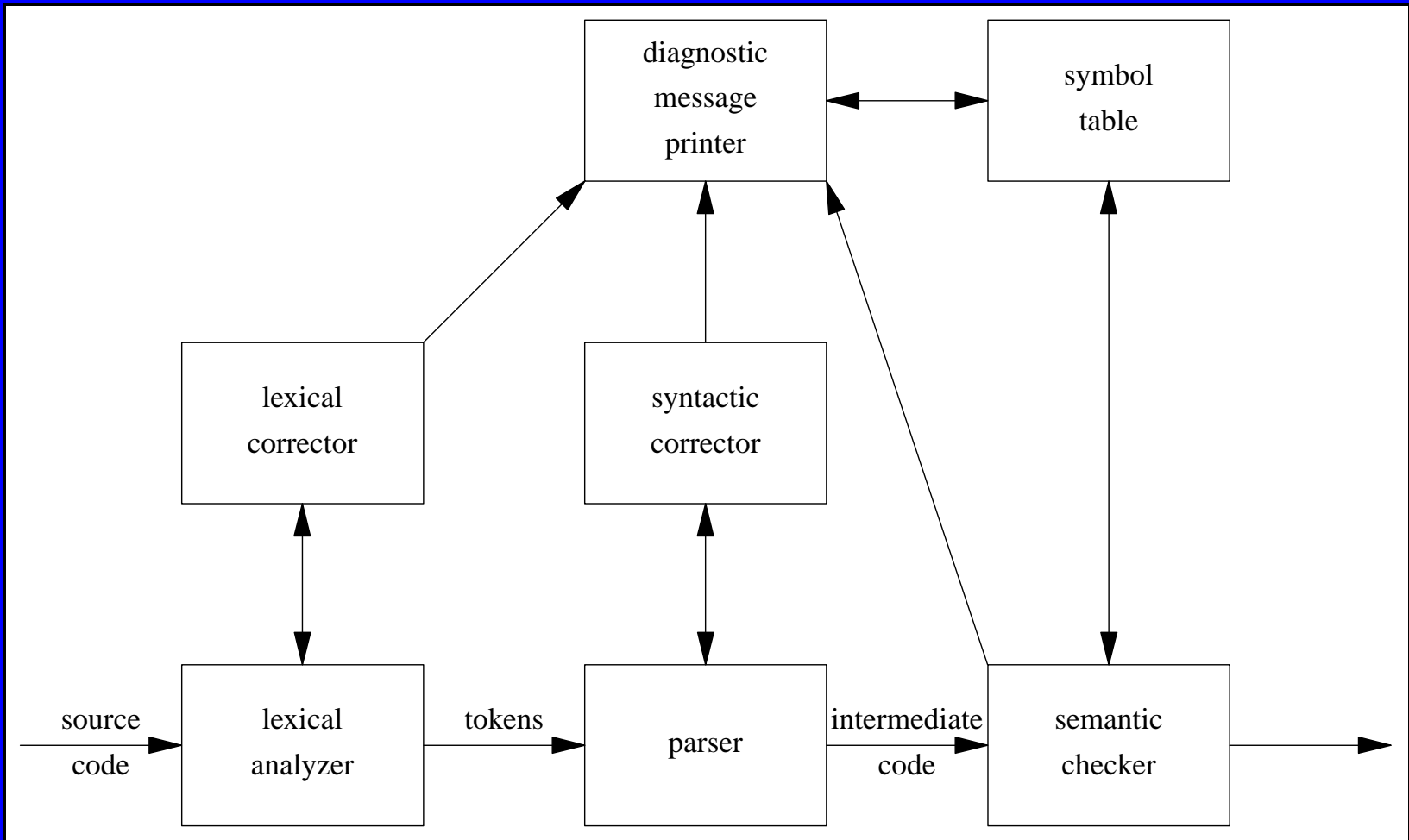
```
.PS
ellipse "document"; arrow
box "PIC"; arrow
box "TBL/EQN" "(optional)" dashed; arrow
box "TROFF"; arrow
ellipse "typesetter"
.PE
```



```
pi = atan2(0,-1)
for i = 0 to 2 * pi \
  by 0.1 do {
  " " at i/2, 0
  "s" at i/2, sin(i)/2
  "c" at i/2, cos(i)/2
}
```



```
.PS 5
.ps 8
arrow "source" "code"
LA: box "lexical" "analyzer"
arrow "tokens" above
P: box "parser"
arrow "intermediate" "code"
Sem: box "semantic" "checker"
arrow
arrow <-> up from top of LA
LC: box "lexical" "corrector"
arrow <-> up from top of P
Syn: box "syntactic" "corrector"
arrow up
DMP: box "diagnostic" "message" "printer"
arrow <-> right from right of DMP
ST: box "symbol" "table"
arrow from LC.ne to DMP.sw
arrow from Sem.nw to DMP.se
arrow <-> from Sem.top to ST.bot
.PE
```



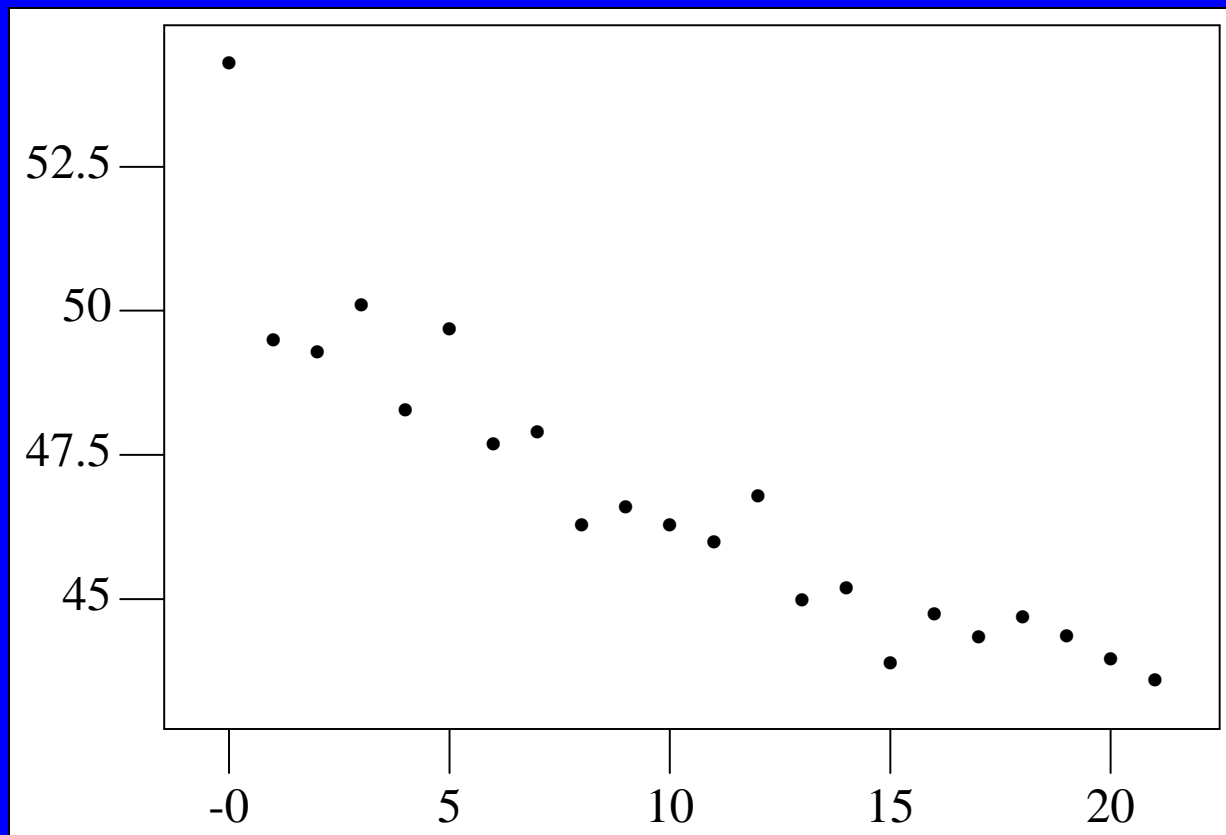
GRAP

History Written by Jon Bentley and Brian Kernighan at Bell Labs

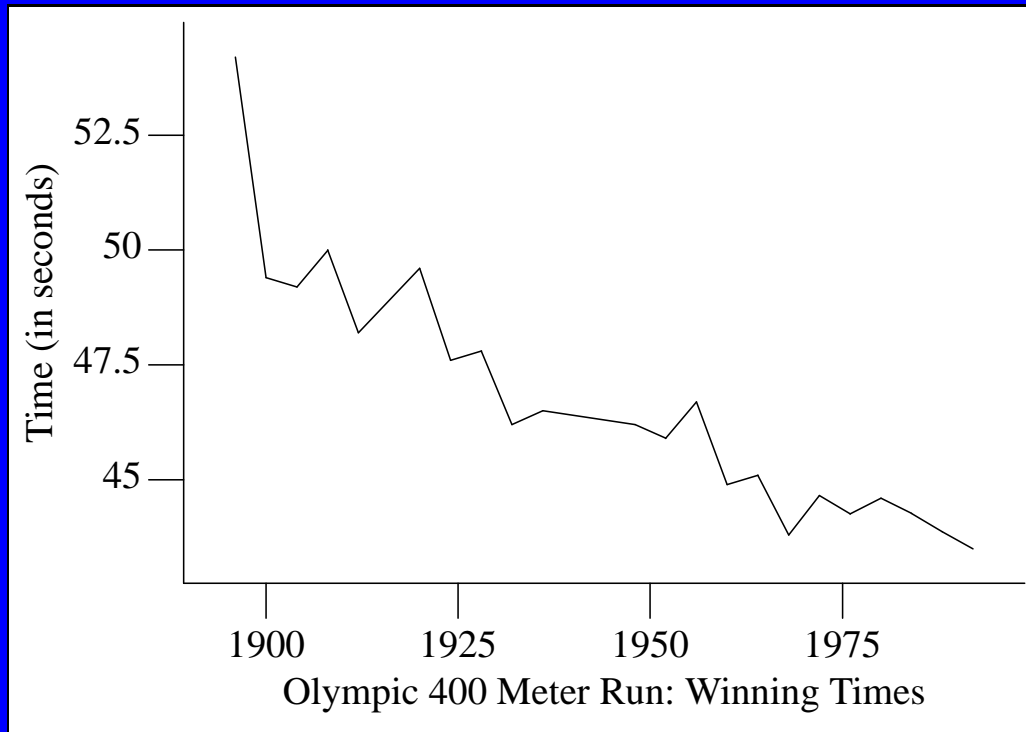
- A language for describing plots of data
- A `pic` preprocessor
- `.G1/.G2` turn on/off the preprocessor
- Data can be entered directly or included via a `copy` command
- Provides such services as automatic scaling and labeling of axes

Examples

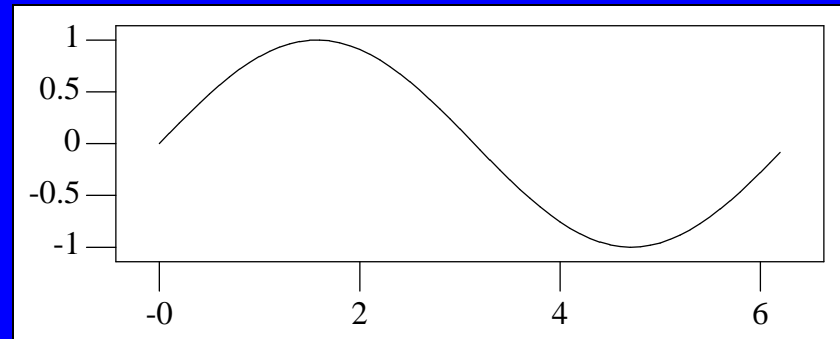
```
.G1  
copy "400mtimes.d"  
.G2
```



```
frame invis ht 2 wid 3 left solid bot solid
label left "Time (in seconds)" left 0.1
label bot "Olympic 400 Meter Run: Winning Times"
draw solid; copy "400mpairs.d"
```



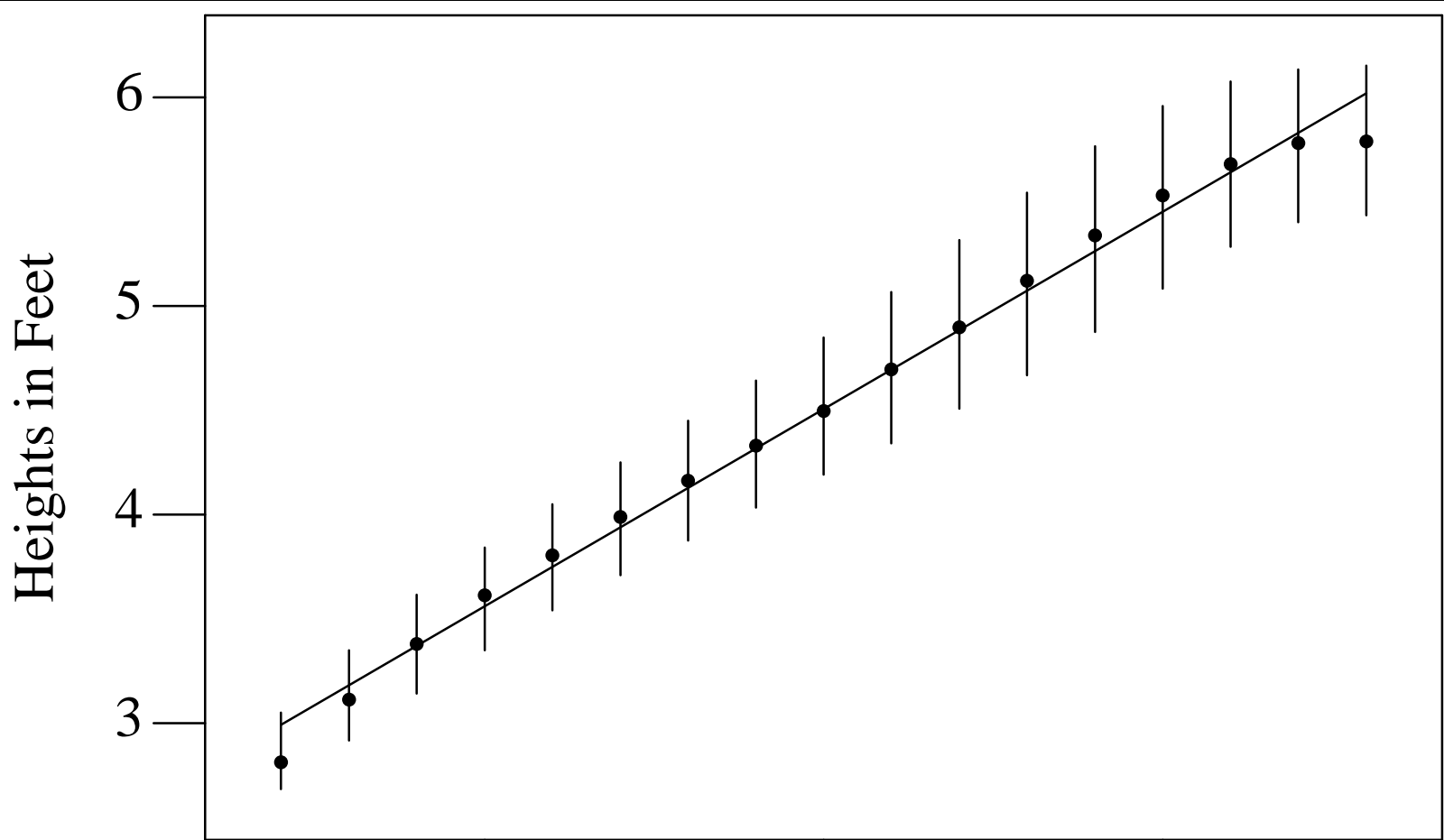
```
.G1
frame ht 1 wid 3
draw solid
pi = atan2(0,-1)
for i from 0 to 2* pi \
    by .1 do {
    next at i, sin(i)
}
.G2
```



```

label left "Heights in Feet"
label bot "Heights of Boys in the US Ages 2-18"
cmpft = 12
minx = 1e12; maxx = -1e12
n = sigx = sigx2 = sigy = sigxy = 0;
copy "boyhts.d" thru {
    line from $1, $2/cmpft to $1, $4/cmpft
    ty = $3 / cmpft
    bullet at $1, ty
    n = n+1
    sigx = sigx + $1; sigx2 = sigx2 + $1 * $1
    sigy = sigy + ty; sigxy = sigxy + $1*ty
    minx = min(minx,$1); maxx = max(maxx,$1);
}
slope = ( n*sigxy - sigx* sigy) / \
        (n*sigx2 - sigx * sigx)
inter = ( sigy - slope * sigx) / n
line from minx, slope * minx+inter to \
        maxx, slope * maxx + inter

```



Heights of Boys in the US Ages 2-18

The History of T_EX

- First there was T_EX . . .
 - Created by Donald E. Knuth at Stanford in 1977
 - Knuth hoped that he could reverse the trend of deteriorating typographical quality (especially for mathematics) that he saw affecting his own books and articles
 - Today's T_EX was released in 1982, with some slight enhancements added in 1989 to better support 8-bit characters and multiple languages
 - T_EX has been frozen, and all new releases are bug fixes
 - T_EX's version number is slowly converging to π (3.14159 . . .)

- Then came \LaTeX . . .
 - Originally written by Leslie Lamport from DEC to “convert a highly-tuned racing car into a comfortable family sedan”
 - Evolved over the years until version 2.09
 - In 1994 the \LaTeX package was updated by the \LaTeX 3 team, led by Frank Mittelbach, to include some long-requested improvements, and to reunify all the patched versions
 - This new version of \TeX was called $\LaTeX 2_{\epsilon}$ to distinguish it from \LaTeX 2.09
- And others . . .
 - Numerous developers have written add-on packages for \LaTeX
 - Special tools have been developed, such as $\text{Bib}\TeX$ (by Oren Patashnik) for automatically generating bibliographies in numerous formats from a database, and *MakeIndex* for generating indices

An Introduction to T_EX

Text:

- Ordinary text is interspersed with formatting commands
- Words are separated by one or more spaces
- Paragraphs are separated by one or more blank lines
- Double quotes are typed like this: ```quoted text''`
- Long dashes (—) are typed as three dash characters: `---`
- These must have a backslash in front of them: `$ & # % _ {` and `}`

Commands:

- All \LaTeX commands consist of a “\” followed by one or more characters
- \backslash BEGIN and \backslash Begin are not the same as \backslash begin
- There are other commands that look like $\text{\backslash command}\{text\}$ — sometimes square brackets $[]$ are used
- You must make sure that the end of the command is marked by something that isn't a letter, usually an opening bracket or a space
- Such spaces are always ignored by \LaTeX

Math mode:

- Math mode commands are surrounded by $\langle \dots \rangle$ or by $\$ \dots \$$

$$\$E = m c^2\$ \quad \Rightarrow \quad E = mc^2$$

- Spaces in the input file are ignored in math mode
- Displayed equations are surrounded by $\langle \dots \rangle$ or by $\$\$ \dots \$\$$
- Subscripts and superscripts are produced with $_$ and $^$
- Fractions are produced by the `\frac` command:
 $\langle \backslash\text{frac}\{a + b\}\{c\}\rangle$ gives $\frac{a+b}{c}$
- Various commands give names to mathematical symbols
- Arrays are produced by an `array` environment
- The commands `\left` and `\right` produce delimiters that grow as large as needed

Document Classes: There are four standard document classes available in \LaTeX :

article intended for short documents and articles for publication.

report intended for longer technical documents—similar to article, except that it contains chapters and the title appears on a page of its own

book intended as a basis for book publication—assumes that the output will eventually be used to print on both sides of the paper

letter intended for producing personal letters

This is a sample LaTeX input file. (Version of 11 April 1994.)

A '%' character causes TeX to ignore all remaining text on the line, and is used for comments like this one.

```
\documentclass{article}      % Specifies the document class

                               % The preamble begins here.
\title{An Example Document} % Declares the document's title.
\author{Leslie Lamport}     % Declares the author's name.
\date{January 21, 1994}    % Deleting this command produces today's date.

\newcommand{\ip}[2]{(#1, #2)}
                               % Defines \ip{arg1}{arg2} to mean
                               % (arg1, arg2).

\newcommand{\ip}[2]{\langle #1 | #2\rangle}
                               % This is an alternative definition of
                               % \ip that is commented out.

\begin{document}            % End of preamble and beginning of text.
```

```
\maketitle % Produces the title.
```

This is an example input file. Comparing it with the output it generates can show you how to produce a simple document of your own.

```
\section{Ordinary Text} % Produces section heading. Lower-level  
% sections are begun with similar  
% \subsection and \subsubsection commands.
```

The ends of words and sentences are marked by spaces. It doesn't matter how many spaces you type; one is as good as 100. The end of a line counts as a space.

One or more blank lines denote the end of a paragraph.

Since any number of consecutive spaces are treated like a single one, the formatting of the input file makes no difference to

```
\LaTeX, % The \LaTeX command generates the LaTeX logo.
```

An Example Document

Leslie Lamport

January 21, 1994

This is an example input file. Comparing it with the output it generates can show you how to produce a simple document of your own.

1 Ordinary Text

The ends of words and sentences are marked by spaces. It doesn't matter how many spaces you type; one is as good as 100. The end of a line counts as a space.

One or more blank lines denote the end of a paragraph.

Since any number of consecutive spaces are treated like a single one, the formatting of the input file makes no difference to \LaTeX , ...

but it makes a difference to you. When you use `\LaTeX`, making your input file as easy to read as possible will be a great help as you write your document and when you change it. This sample file shows how you can add comments to your own input file.

Because printing is different from typewriting, there are a number of things that you have to do differently when preparing an input file than if you were just typing the document directly.

Quotation marks like

```
``this''
```

have to be handled specially, as do quotes within quotes:

```
``\,'this'          % \, separates the double and single quote.  
is what I just  
wrote, not `that'\,'''.
```

...but it makes a difference to you. When you use \LaTeX , making your input file as easy to read as possible will be a great help as you write your document and when you change it. This sample file shows how you can add comments to your own input file.

Because printing is different from typewriting, there are a number of things that you have to do differently when preparing an input file than if you were just typing the document directly. Quotation marks like “this” have to be handled specially, as do quotes within quotes: “‘this’ is what I just wrote, not ‘that’”.

...

Dashes come in three sizes: an
intra-word
dash, a medium dash for number ranges like
1--2,
and a punctuation
dash---like
this.

A sentence-ending space should be larger than the
space between words within a sentence. You
sometimes have to type special commands in
conjunction with punctuation characters to get
this right, as in the following sentence.

Gnats, gnus, etc.\ all % '\ ' makes an inter-word space.
begin with G\@. % \@ marks end-of-sentence punctuation.

You should check the spaces after periods when
reading your output to make sure you haven't
forgotten any special cases.

...Dashes come in three sizes: an intra-word dash, a medium dash for number ranges like 1–2, and a punctuation dash—like this.

A sentence-ending space should be larger than the space between words within a sentence. You sometimes have to type special commands in conjunction with punctuation characters to get this right, as in the following sentence. Gnats, gnus, etc. all begin with G. You should check the spaces after periods when reading your output to make sure you haven't forgotten any special cases.

...

Generating an
ellipsis

```
\ldots\ % '\ ' is needed after '\ldots' because TeX
% ignores spaces after command names like \ldots
% made from \ + letters.
%
% Note how a '%' character causes TeX to ignore
% the end of the input line, so these blank lines
% do not start a new paragraph.
%
```

with the right spacing around the periods requires
a special command.

`\LaTeX\` interprets some common characters as
commands, so you must type special commands to
generate them. These characters include the
following:

```
\$ \& \% \# \{ and \}.
```

...Generating an ellipsis ... with the right spacing around the periods requires a special command.

L^AT_EX interprets some common characters as commands, so you must type special commands to generate them. These characters include the following: \$ & % # { and }. ...

In printing, text is usually emphasized with an
`\emph{italic}`
type style.

```
\begin{em}
```

A long segment of text can also be emphasized
in this way. Text within such a segment can be
given `\emph{additional}` emphasis.

```
\end{em}
```

It is sometimes necessary to prevent `\LaTeX` from
breaking a line where it might otherwise do so.
This may be at a space, as between the `''Mr.''` and
`''Jones''` in

```
''Mr.~Jones'',           % ~ produces an unbreakable interword space.
```

or within a word---especially when the word is a
symbol like

```
\mbox{\emph{itemnum}}
```

that makes little sense when hyphenated across
lines.

...In printing, text is usually emphasized with an *italic* type style.

A long segment of text can also be emphasized in this way. Text within such a segment can be given additional emphasis.

It is sometimes necessary to prevent L^AT_EX from breaking a line where it might otherwise do so. This may be at a space, as between the “Mr.” and “Jones” in “Mr. Jones”, or within a word—especially when the word is a symbol like *itemnum* that makes little sense when hyphenated across lines. ...

`\LaTeX\` is good at typesetting mathematical formulas like

```
\( x-3y + z = 7 \)
```

or

```
\( a_{1} > x^{2n} + y^{2n} > x' \)
```

or

```
\( \int_A^B = \sum_i a_i b_i \)
```

The spaces you type in a formula are ignored. Remember that a letter like

```
 $x$   $\int \dots$  and  $\int \dots$  are equivalent
```

is a formula when it denotes a mathematical symbol, and it should be typed as one.

Mathematical formulas may also be displayed. A displayed formula is

one-line long; multiline

formulas require special formatting instructions.

```
\[ \int \Gamma \psi' = x'' + y^2 + z_i^n \]
```

Don't start a paragraph with a displayed equation, nor make one a paragraph by itself.

```
\end{document}
```

```
% End of document.
```

... \LaTeX is good at typesetting mathematical formulas like $x - 3y + z = 7$ or $a_1 > x^{2n} + y^{2n} > x'$ or $(A, B) = \sum_i a_i b_i$. The spaces you type in a formula are ignored. Remember that a letter like x is a formula when it denotes a mathematical symbol, and it should be typed as one.

Mathematical formulas may also be displayed. A displayed formula is one-line long; multiline formulas require special formatting instructions.

$$(\Gamma, \psi') = x'' + y^2 + z_i^n$$

Don't start a paragraph with a displayed equation, nor make one a paragraph by itself.