TEX

Josh Baldwin

Programming Languages Fall 2008

History & Background of TEX

- TEX was developed by Donald Knuth in 1977/78.
- Knuth is a renowned computer scientist and Professor Emeritus (retired professor) at Stanford University.
- He is the author of the seminal multi-volume work The Art of Computer Programming, which is on programming algorithms and their analysis.
- Knuth is considered the 'father'of the analysis of algorithms.
- Knuth developed WEB which is what TEX and METAFONT is programmed in.

History & Background of TEX

- TEX was frozen at version 3.0 in 1989 (no more updates, only bug fixes).
- The TEX code is open source so anyone can modify and create a new version.
- Since version 3.0 it appends one more number onto the number π. The current version of TEX is 3.1415926 and was last updated in March 2008.
- TEX is a typesetting programming language. This means it is designed for typesetting tasks (duh?).

History & Background of TEX

- TEX has about 300 primitive commands that all the other commands use, 600 standard from TEX and also user defined commands. Primitive commands cannot be broken down any further.
- TEX commands are case sensitive. The command \tex is not equivalent to \TeX.
- The primitive commands in TEX use tail recursion, making them highly effective at memory management.
- TEX originally used floating point calculations, but as of TEX82 it uses only fixed point arithmetic. This means it will reproduce the same outputs on any system.
- TEX normally outputs to .dvi file, which stands for DeVice Independent. It can also output to PostScript and pdf.
- This presentation was written completely in TEX . If you want a copy of the source code then just ask.

WEB

WEB Programming Language: *Was developed by Donald Knuth as the first implementation of what he called "literate programming".*

Literal Programming: *The idea that one could create software as works of literature by embedding source code inside descriptive text so that it is easy for human readers, rather than easy for a compiler.*

WEB consists of two primary programs, TANGLE and WEAVE. TANGLE produces compilable Pascal code from the source texts. WEAVE produces nicely-formatted and printable documentation using TEX.

Hello World!

Source code:

```
Hello World!  % Prints ‘‘Hello World!’’  to the screen.
\end           % Marks the end of the .tex file.
```

Output:

Hello World!

Special Characters

Name	Character	How to print explicitly in TEX
Escape character	\	`$\backslash$`
Beginning of block	{	`${$`
End of block	}	`$}$`
Start and end math mode	$	`\$`
Alignment tab	&	`\&`
Parameter	#	`\#`
Superscript	^	`\^{␣}`
Subscript	_	`\_`
Active space	~	`\~{␣}`
Comment	%	`\%`
Space	␣	`\␣`

Uses ASCII internally, so instead you can simply use `\char` ⟨`number`⟩.

- `$_\sqcup$` for the bucket that represents a space in this character chart.

Control Sequence

Control Sequence: *consists of a ‘ \ ’ backslash followed by a command.*

Control Word: *Consists of “ \⟨one or more letters⟩ ”*

TEX needs to know where the control sequence ends so a space is required after the word, unless followed immediately by another control word.

Example: \TeX for TEX

Control Symbol: *Consists of “ \⟨non-letter⟩ ”*

TEX does not require a space after a control symbol because it consists only of the escape character and a non-letter.

Example: `\'a` for an accent on a letter like á. The ’ is the non-letter and the a is an argument to the control sequence (more on arguments later).

Letter: *Consists of A-Z and a-z.*

Non-letter: *Anything that isn’t a letter.*

Boxes

Everything in TEX is a box: *EVERYTHING!, Well ok just about everything.*

TEX will complain if it cannot fit these boxes in a "good"way:

Underfull box: *You have an area that TEX is complaining about because space on a page is being wasted, it is not filled up completely.*

Overfull box: *You have to much on one line(or page) and TEX cannot fit it without running off the page.*

Overfull boxes can leave giant black boxes (to show where the overfull box is) or can run text off the page. In paragraphs you can use descretionary hyphens (\-) in words like des\-cre\-tion\-ary to let TEX know it can line break at these points. Yes it makes your text look very ugly.

Simple Input & Output

Whitespace: *TEX ignores whitespace, (spaces, tabs, and newlines), except for one or more blank lines, then it treats it as a new paragraph.*

```
\TeX␣␣␣␣␣␣␣will ignore all this whitespace.  % Input
```

TEX will ignore all this whitespace.

```
This is the first paragraph.  ... This is the first paragraph.
% Blank line here.
This is the second paragraph.  ... This is the second paragraph.
```

This is the first paragraph. This is the first paragraph. This is the first paragraph. This is the first paragraph. This is the first paragraph. This is the first paragraph.

This is the second paragraph. This is the second paragraph. This is the second paragraph. This is the second paragraph. This is the second paragraph.

Periods & Active Spaces

Periods: *TEX automatically inserts the correct amount of spaces after a period. A capital letter and then a period results in a single space, for names really. If you still want the normal amount, usually two spaces, put a \null right before the period.*

```
This period is correcT\null.  This period is noT. Blank.
```

This period is correcT. This period is noT. Blank.

Active Spaces ~: *TEX will treat an active space as a ␣ that cannot be broken by a line break or page break. It makes it so that two words will be considered one.*

Active spaces are useful for keeping names of people together, so they don't get split up at the end of a line.

Dashes

Four types of dashes:

- a hyphen (-)	Used in compound words (in-law)
-- an en-dash (–)	Used for number ranges (13–14)
--- an em-dash (—)	Used for punctuation in sentences—they are simply dashes
$-$ a minus sign (−)	Math $5 - 4$

Dimensions and Units of Measure

Dimension:

⟨number⟩⟨unit of measure⟩

```
5in % A dimension of 5 inches.
```

List of most of the units of measure TEX supports:

Name:		Conversion:
point	pt	12pt = 1em
scaled point	sp	65536sp = 1pt
m width	em	6em = 1in
half m width	en	2en = 1em
pica	pc	1pc = 12pt
inch	in	1in = 72.72pt
math unit	mu	18mu = 1 em
centimeter	cm	2.54cm = 1 in
millimeter	mm	10mm = 1cm

Magnification

Magnification: *TEX has some predefined magnification commands that step the magnification by 1.2 times. 100% magnification is equal to 1000 mag scale. magstep5 is the last magstep TEX provides. Why are these all predefined at 1.2? According to Knuth it is "convenient"to supply fonts at magnifications that grow in geometic ratios, think of an instrument and its scales.*

This is Roman 10pt at magstep0. 1000×1.2^0

This is Roman 10pt at magstephalf. $1000 \times \sqrt{1.2}$

This is Roman 10pt at magstep1. 1000×1.2^1

This is Roman 10pt at magstep2. 1000×1.2^2

This is Roman 10pt at magstep3. 1000×1.2^3

This is Roman 10pt at magstep4. 1000×1.2^4

This is Roman 10pt at magstep5. 1000×1.2^5

Fonts & Sizes

Define a TEX Font: *\font\Name = Font ⟨scale⟩*

```
\font\FontTwoStep = cmr10 scaled\magstep2
{\FontTwoStep This font is Roman 10pt scaled to two step.}
It is in a block so that the font after the block will return to the
previous font that was specified.  Otherwise this font would go until a
new font is specified.
```

This font is Roman 10pt scaled to two step. It is in a block so that the font after the block will return to the previous font that was specified. Otherwise this font would go until a new font is specified.

```
\font\FontEighteen = cmr10 at 18pt
{\FontEighteen This font is Roman 10pt at 18pt.}
```

This font is Roman 10pt at 18pt.

Fonts

Styles:

{\rm roman}
roman
{\sl slanted}
slanted
{\bf boldface}
boldface
{\it italic\/}
italic

Why are they all in their own scope?
Why does italic have a command at the end?

Glue

Glue: *It is the whitespace inserted in between all objects. Glue can stretch depending on what you specify in the "plus"and "minus"dimensions. A dimension of* $\geq$ *10000 is considered infinite. Some glue is breakable and some glue is not breakable. This means a line break or page break could happen or not depending on what kind of glue is used.*

Glue template:

⟨dimen⟩ plus⟨dimen⟩ minus⟨dimen⟩

```
\hskip 10pt plus 30pt minus 10pt
```

This glue will go hskip 10pt and then be able to stretch another 30pt or contract 10pt. This means it can stretch anywhere from 0pt to 40pt. The plus or minus arguments can be left out, and if both are left out the glue cannot stretch. Glue stretch can be negative, in this sense you can overlap things.

TEX provides the commands \llap and \rlap to left overlap and right overlap so it is easier to do than specifying the negative glue.

Definitions

Definitions (or Macros): *are control sequences that the user defines. They can call other control sequences in them, and can also have parameters that can also be control sequences, words, letters, numbers, whatever you want. Parameters are optional. To define a definition:* *def*\⟨*Name*⟩ ⟨*Parameters*⟩ {*Expansion*}

The brackets (scope) associated in the definition is only to show the formal definition starting and ending points. It is not actually putting a new scope in when it replaces it in the code.

```
\def\Bold#1{{\bf #1}}
```

The above code defines a macro called \Bold that takes one argument and then makes that argument to be bold. Since the \bf will make everything after it bold, we limit the scope by putting brackets around it. To call this macro we would write something like

`\Bold{This text will be bold.}` This text will not be bold.

This text will be bold. This text will not be bold.

Arguments

Arguments:

So we saw a simple example on how to pass a single argument. Here we pass three arguments, and use a different delimiter (a period followed by a space) and just standard brackets for the first argument. You can use anything you like as a delimeter in this way.

```
\def\Payment#1#2.␣#3.␣{Mr. #1, you owe \$#2 #3.}
```

To call this macro we would write something like (with important spaces):

```
\Payment {Baldwin}10.00.␣Please Pay up.␣
```

Mr. Baldwin, you owe $10.00 Please Pay up.

Arguments

Runaway arguments: *Forgetting a {or }would normally produce a really bad error, but TEX deals with this by limiting the scope of arguments to within the current paragraph. This means that the token \par cannot appear as part of an argument unless one actually specifies its ok by putting a \long before the definition.*

```
\long\def\Bold#1{{\bf #1}}
```

This will allow you to have multiple paragraphs within this macro. Why is this not really all that desiriable though? How come TEX defines it as {\bf (text)}? The simple reason is this way there are no arguments to pass around, which makes it more efficient, and it restricts the scope just like the macro we defined, plus it allows for multiple paragraphs.

Outer & Global

Outer: *Putting a \outer in-front of a definition will make it unable to become an argument to another macro.*

Global: *Putting a \global means that the definition is valid after the current scope that it is currently in.*

Any combination of these can be added, and they can appear more than once.

Registers

Registers: *TEX has registers called \count0 to \count255 and they are 32 bit signed integers. TEX also has registers called \dimen0 to \dimen255 to hold dimension. Same goes for \skip0-255 which contains glue.*

To manipulate these registers we can use the following commands:

\advance\Register by (number | dimen | glue)
\multiply\Register by (number | dimen | glue)
\divide\Register by (number | dimen | glue)

A dimension register can be used just like a number register, it assumes the dimen is of scaled point no matter what its actual unit of measure is. So if \dimen1 holds 1em, and you do \count1=\dimen1 then \count1 will hold 65536.

A glue register can be converted to a dimension by omitting the plus and minus and maintaining the unit of measure.

If, Loop and Let

If:

`\if`⟨`condition`⟩␣⟨`true statement`⟩`\else` ⟨`false statement`⟩`\fi`

The space is incredibly important, without it TEX would expand the 'true statement' while evaluating the condition.

Loop:

`\loop` α `\if`... β `\repeat`

Loops α until β is false. Its like a do while.

Let: *sets a new control sequence equal to the token.*

`\let\Name=`⟨`token`⟩

The token can be another control sequence or simply a string of letters and/or numbers. If it is a number, then it is still just a character, not actually like the count registers.

Items

Items: *List of items can be achieved with the \item command. The amount that it indents is based on the \parindent control word.*

```
\item {$\bullet$}This is an item.
\item {}This is another item.
```

- This is an item.
  This is another item.

```
\parindent = 50pt \item {$\bullet$}This is an item.
\item {}This is another item.
```

- This is an item.
  This is another item.

Double item: *Gets twice the indentation.*

```
\item {$\bullet$}This is an item.
\itemitem {$\bullet$}This is an itemitem.
```

- This is an item.
  - This is an itemitem.

Items

So how could we nest items more than item and itemitem provide? TEX defines the hard part for us:

```
\def\textindent#1{\indent\llap{#1\enspace}\ignorespace}
```

Now we can use the above control sequence to generate the nice bullet next to our item, and then generate the correct amount of indents using the parindent control sequence:

```
\def\myitem#1#2#3{{\parindent = #2\parindent \par\hang\textindent{#1}#3}}

\myitem{$\bullet$}{1}{This is the text.}
\myitem{$\bullet$}{2}{This is the text.}
\myitem{this is what llap does}{3}{This is the text.}
\myitem{$\bullet$}{1}{This is the text.}
```

- This is the text.
    - This is the text.

this is what llap does This is the text.

- This is the text.

Spacing

Horizontal spacing:

Breakable Horizontal Spacing:	
\enskip	1en
\quad	1em
\qquad	2em
\hskip ⟨glue⟩	Glue
Unbreakable Horizontal Spacing:	
~	Active space, as seen before
\enspace	1en
\thinspace	this much
\kern	Glue

Spacing

Vertical spacing:

\vskip ⟨glue⟩	Skips the specified glue.
\smallskip	Skips a small space between lines (is a line break).
\medskip	Skips a medium space between lines (is a line break).
\bigskip	Skips a big space between lines (is a line break).
\smallbreak	Encourages a line break with a smallskip.
\medbreak	Encourages a line break with a medskip.
\bigbreak	Encourages a line break with a bigskip.

hfill

Center an object:

```
\hfil This text is centerlined.\hfil
```

This text is centerlined.

```
\hfill This text is not centerlined.\hfil
```

This text is not centerlined.

```
This is normal text!\hfil\par
This is normal text!\hfill\par
```

This is normal text!
This is normal text!

TEX also has a predefined \centerline{} control word to center text.

Page Breaks

Page break:

```
Here is some text and now we are going to page break.
\vfil\eject % vfil fills the rest of the page with whitespace.
% Without it you will get an underfull box because TeX could not fill the
% entire page. eject ends the page.
```

Here is some text and now we are going to page break.

Badness

Badness: *Another way to deal with underfull and overfull boxes. The default is a tolerance of 200. Tolerance $\geq$ 10000 is infinite (just like glue).*

TEX assigns a value called "badness"to each line that it creates out of these boxes. This is to assess the quality of the spacing. Perfect lines have a badness of 0, anything higher and either the line is too tight or it is too spaced out. Sometimes you won't care but TEX will give you an overfull or underfull box with a badness of say 200 or 500. You can have TEX ignore this by simply saying \tolerance=500 within that block.

Tables

Set tabs: *Consists of equally spaced columns. & denotes the end of a column and \cr denotes a new line. \+ denotes the beginning of a line. Since they are equally spaced if the entry is to large it will overlap*

```
\settabs 7 \columns
\+ Monday& Tuesday& Wednesday& Thursday& Friday& Saturday& Sunday\cr
```

Monday Tuesday Wednesday Thursday Friday Saturday Sunday

```
\settabs 7 \columns
\+ Overlap Overlap& Overlap& Overlap& Overlap& Overlap& Overlap& Overlap\cr
```

Overlap Overlap Overlap Overlap Overlap Overlap Overlap Overlap

Tables

halign: *Another more durable way to set tables. Use a sample line to designate how much spacing should be between the columns. The # designates the parameter for that part in the column.*

```
\halign{#\hfil\qquad&#\hfil\qquad#\hfil\cr % Sample spacing line.
Color& Wight& Build&\cr
Blue& 50kg& Old\cr
Red& 40kg& New\cr
Green& 22kg& Used\cr}
```

Color	Weight	Build
Blue	50kg	Old
Red	40kg	New
Green	22kg	Used

Math Mode

Math Mode: *Use $ to enter math mode and $ to end math mode. Math mode allows for an easy to way write in equations. There are many math mode specific functions that require you to be in math mode to use them, for instance the greek letters, or perhaps the square root function.*

```
$-b\pm \sqrt{b^2 - 4ac}\over 2a$
```

$-b\pm \sqrt{b^2 - 4ac}\over 2a$

Displayed Math Mode: *Use $$ to enter displayed math mode and $$ to end displayed math mode. Displayed math mode centers the equation to the middle of the screen.*

```
$$-b\pm \sqrt{b^2 - 4ac}\over 2a$$
```

$$-b\pm \sqrt{b^2 - 4ac}\over 2a$$

Math Mode

Elementary Math Control Sequences:

$\overline{x+y}$	`$\overline{x + y}$`
$\underline{x+y}$	`$\underline{x + y}$`
$\sqrt{x+y}$	`$\sqrt{x + y}$`
$\sqrt[n]{x+y}$	`$\root n\of{x + y}$`
$\frac{a+b}{x+y}$	`${a + b \over x + y}$`
$\begin{smallmatrix} a+b \\ x+y \end{smallmatrix}$	`${a + b \atop x + y}$`
$\binom{n+1}{5}$	`${n + 1 \choose 5}$`
$\left\{{n+1 \atop 5}\right\}$	`${n + 1 \brace 5}$`
$\left[{n+1 \atop 5}\right]$	`${n + 1 \brack 5}$`

Questions?