The cjw-latex Macro Collection[*]

Colin J. Wynne[†]

1998/09/01

Contents

---

[*] This file has version 0.13 as of 1998/09/01.

[†] E-Mail at: `cwynne@mts.jhu.edu`, `cwynne@jhu.edu` .

Introduction

I have been a TEX user for quite a long time now. It was in my junior year in college, in 1992, that one of my friends and one of my math professors decided to warp my perception of reality and introduce me to Dr. Knuth's wonderful creation. In those days, Plain-TEX was the tool of choice, we coded everything ourselves from primitives on up, logical markup was unknown, and LATEX was known as Lame-TEX. During my senior year I wrote an honors thesis in mathematics which required quite a lot of things not present in standard Plain-TEX. My macro files grew, and grew, and. . .

In the year following my graduation, I converted myself to LATEX, mostly because of the convergence my own personal modified ePlain/NFSS1/Plain format was having towards LATEX in terms of logical markup. Most of macros were easily converted into LATEX-ese. They got more complicated from there.

So now I regularly write up papers, letters, mathematical problem sets, and just about anything else that uses the English language, in LATEX. The macros have evolved quite a bit. More has been added. I took to using the `dtx` format for most of my input files, and finally decided to wrap my big three up into a single documented source file.

I hope that these macros will prove useful to somebody out there, and if they do, feel free to buy me a beer next time you see me. I have other `dtx` files available, including a modified `letter` class which also does German formal letters, and a package for doing outlines. Any package of mine should be identifiable on your friendly neighborhood CTAN site with a name like `cjw*.(dtx|ins)`.

1 General macros

I tend to organize my package files as follows—first come flow control structures for the package itself, usually in the form of new conditionals; then come the options; then comes the meat of the package in some sort of vaguely thought out order. This package is no exception.

1.1 Package initialization

Conditionals are usually used in conjunction with package options to provide conditional inclusion of certain code, either *via* an `\if`...`\endif` block or using class options. For this package, the subsystem in question is the inclusion of verbatim typeset files.

```
\newif\if@verbext       \@verbextfalse
```

Not surprisingly, this conditional is used directly by an option.

```
\DeclareOption{verbext}{\@verbexttrue}
```

I used to use options for the loading of additional packages. In particular, when I used psfig.new which could not be handled as a LaTeX 2$_\epsilon$ package, I did this. Now that I use the epsfig package that comes with LaTeX 2$_\epsilon$, I simply issue a warning to include the package separately.

```
\DeclareOption{psfig}{%
  \PackageWarning{cjwmacro}%
    {Obsolete option \CurrentOption.  Use package ‘epsfig’ instead.}}
```

Since, however, using pstricks requires several files, I still use a package option to take care of all of that. This option checks for the existence of *both* files before including either, hence the nested calls to `\InputIfFileExists`. A similar option is used for pst-plot.tex, since is not implemented as a package file.

```
\DeclareOption{pstricks}{%
  \InputIfFileExists{pstricks.sty}{%
  \InputIfFileExists{pst-node.tex}{}{%
    \PackageError{cjwmacro}{File ‘pst-node.sty’ not found.}{}}}%
   {\PackageError{cjwmacro}{File ‘pstricks.sty’ not found.}{}}}

\DeclareOption{psplot}{\InputIfFileExists{pst-plot.tex}{}{%
  \PackageError{cjwmacro}{File ‘pst-plot.tex’ not found.}{}}}
```

The next two options are used to change behavior of some macros on draft as opposed to final copies. Currently, only `\ssbreak` has such a dependency, and in final form it uses the PS-Tricks package to typeset a nice section delimiter. Note the use of `\ExecuteOption` by the final option to make sure that PS-Tricks is, indeed, available.

```
% What to do for draft vs. final copy.
\DeclareOption{draft}{%
  \def\ssbreakbar{\hbox to 2in{\hrulefill}}}
\DeclareOption{final}{%
  \ExecuteOptions{pstricks}
  \def\ssbreakbar{%
    \psset{linewidth=0.4pt,unit=1in}%
    \pspicture(-2.5,-0.15)(2.5,0.15)%
      \qdisk(0,0){0.04}%
      \qdisk(0.33,0){0.02}%
      \qdisk(-0.33,0){0.02}%
      \pspolygon*(0.33,-0.02)(0.33,0.02)(1.75,0)%
      \pspolygon*(-0.33,-0.02)(-0.33,0.02)(-1.75,0)%
    \endpspicture}}
```

To finish off option handling, we declare a default (warn about unknown options), execute defaults, and process the passed option list.

```
\DeclareOption*{%
  \PackageWarning{cjwmacro}{Unknown option ‘\CurrentOption’}}
\ExecuteOptions{draft}
\ProcessOptions
```

1.2 General definitions

These general definitions set up some ‘meta-macros’, to be used by other commands.

One of the things I liked a lot about TeX was the use of `\let` to make aliases for existing commands without sacrificing much of the control sequence space. In the spirit of LaTeX 2$_\epsilon$, however, I have implemented this thrice with name checking. Analogously to the `\newcommand`-like macros, we offer the following three:

```
\newcommand{\alias}       [2]{\@ifdefinable #1{\let #1 #2}}
\alias\realias\let
\newcommand{\providealias}[2]{\@ifundefined #1{\let #1 #2}}
```

Usage is, for example,

```
\alias\foo\bar
```

which makes `\foo` an alias for `\bar`. As expected, `\alias` only works if the new name is currently undefined and `\providealias` does nothing if its first argument is already defined. Somewhat more lax than its counterpart `\renewcommand`, `\realias` does not care if its first argument is defined or not. In essence, that command is used to unconditionally alias a command. This is why, oddly enough, `\realias` is itself just an alias of `\let`. Is this getting confusing yet?

Next we input wholesale a few useful packages. These are still in the spirit of meta-macros which define this section. The first package, amstext, provides the `\text` command, which basically puts its argument in text mode inside a box, but in the current style (textstyle, scriptstyle, etc.). This is used later on. The xspace package is used for control sequences which would encounter ‘the space problem’ when expanded as is.

```
\RequirePackage{amstext}
\RequirePackage{xspace}
```

The command `\intertext` from the amsmath package is quite useful, but I do not want to include that entire package unless it is necessary. Therefore, I make sure that command is defined one way or another. I also give it the alias `\rem` since I am somewhat nostalgic about ancient forms of BASIC…

```
\providecommand{\intertext}[1]{\noalign{%
  \penalty\postdisplaypenalty\addvspace{ 0.5\belowdisplayskip}
  \vbox{\normalbaselines\noindent#1}%
  \penalty\predisplaypenalty\addvspace{0.5\abovedisplayskip}}}
\alias\rem\intertext
```

Next we define some font style names which will be used in some contexts later. This is done to avoid hard-coding of certain styles and to allow as much customization as possible.

```
\providecommand{\pagenofont}    {\normalfont}
\providecommand{\declarefont}   {\normalfont\bfseries\mathversion{bold}}
\providecommand{\altdeclarefont}{\normalfont\itshape}
\providecommand{\captionfont}   {\normalfont\itshape}
\providecommand{\examplefont}   {\normalfont}
\providecommand{\altexamplefont}{\normalfont\itshape}
\providecommand{\labelfont}     {\normalfont\bfseries\mathversion{bold}}
\providecommand{\timelinefont}  {\normalfont}
\providecommand{\titlefont}    {\normalfont\bfseries\Large\mathversion{bold}}
\providecommand{\verbatimfont}  {\normalfont\ttfamily}
```

The next few commands are for programming convenience. First we want to be able to swap the definitions of two control sequences.

```
\newcommand{\swapdef}[2]{{%
        \let \@tempa #1\relax
  \global\let #1      #2\relax
  \global\let #2      \@tempa}}
```

We also want to be able to do the same for lengths (or glue or whathaveyou).

```
\newcommand{\swapdim}[2]{{%
         \@tempdima #1\relax
  \global #1        #2\relax
  \global #2        \@tempdima}}
```

Next is a macro constructed from an exercise in *The TEXbook*, which takes three control sequences and expands them in reverse order.

```
\newcommand{\expandthree}[2]{%
  \expandafter\expandafter\expandafter #1\expandafter #2}
```

This next macro is modified from code I received in the `comp.text.tex` newsgroup. According to the e-mail in which I received it, the original source is a set of macros for *TUGboat*. It turns a number into an ordinal, finding the correct ordinal label which is set as a superscript.

```
\newcommand{\nth}[1]{{%
  \@tempcnta = #1\relax
  \ifnum \@tempcnta < 0\relax           % Make sure our number is
    \@tempcnta = -\@tempcnta            %   non-negative.
  \fi
  \ifnum \@tempcnta < 14\relax          % Deal first with the
    \ifnum \@tempcnta > 10\relax        %   exceptions for
      \def\@tempa{th}                   %   11, 12, and 13.
    \fi
  \else
    \loop \ifnum\@tempcnta > 9\relax    % Loop until the recursive
      \@tempcntb = \@tempcnta           %   remainder (mod 10) is
      \divide  \@tempcntb by  10\relax  %   a single digit in order
      \multiply\@tempcntb by  10\relax  %   to successfully satisfy
      \advance \@tempcnta by -\@tempcntb%   the ordinality test.
    \repeat
    \def\@tempa{\ifcase\@tempcnta       % Figure the proper label:
              th%                       %   0th
        \or   st%                       %   1st
        \or   nd%                       %   2nd
        \or   rd%                       %   3rd
      \else th%                         %   nth
    \fi}
  \fi
  #1\ensuremath{^{\text{\@tempa}}}}}    % Superscript the label in
                                        %   math mode.
```

Continuing in the vein of superscripts, we define two macros which put their arguments as sub- and superscripts in script-script style. This was motivated by such things as derivative indices which look just plain ugly in script style.

```
\alias\sst\scriptscriptstyle
\newcommand{\ssp}[1]{^{\sst#1}}
\newcommand{\ssb}[1]{_{\sst#1}}
```

We now come to some very important and necessary macros, namely the creation of typeset sideways ASCII smiley faces. :-) Since I like to be as general as possible, I have also written an `\emote` macro for indicating emotions. ⟨*smirk*⟩

```
\newcommand{\smiley}[1][\@smiley]{%
  \edef\@sf{\spacefactor=\the\spacefactor}%
  \unskip\spacefactor=1000\relax\space #1\@sf\xspace}
\newcommand{\@smiley}{%
  {\ttfamily\raise 0.078em\hbox{:}\kern-0.1em{-}\kern-0.1em{)}}}
\newcommand{\emote}[1]{%
  \smiley[\ensuremath{\langle}\emph{#1}\ensuremath{\rangle}]}
```

Since I learned the good habit of doing so at Washington and Lee, I often append pledges to my assignments. The generic pledge is implemented as an environment. It formerly took an argument, the date, but I decided that was superfluous, seeing as how the assignment headers set the date once. Why risk inconsistency? Much to my surprise, I found out that LaTeX 2ε's `\maketitle` command unsets not only the date holder, but also the command which is used to set the date in the first place. Anyhow, this means that the date *does* need to be set, but I have left that to be done by the headers. The pledge environment issues a warning if the date is not set.

```
\newenvironment{pledge}%
  {\ifx\@empty\@date
      \PackageWarning{cjwmacro}{Date is not set.}
   \fi
   \parskip=2pt \parindent=0pt\relax
   \null\vfill\begin{flushright}
     \itshape\small}
  {\\[5ex]\normalfont\footnotesize
     \makebox[2in]{\hrulefill}\quad\@date\\
     \makebox[2in]{Colin J.~Wynne}\quad{\hphantom{\@date}}\\
   \end{flushright}}
```

The old Washington and Lee pledge lives on in my macros… It requires one argument, namely the type of assignment being pledged. The argument is optional, though, and a paper is assumed by default.

```
\newcommand{\wnlpledge}[1][paper]{%
  \ifx\@empty\@date
    \PackageWarning{cjwmacro}{Date is not set.}
  \fi
  \parskip=2pt \parindent=0pt\relax
  \null\vfill\begin{flushright}
    \itshape\small
    On my honour, I have neither given nor received\\
    any unacknowledged aid on this #1.\\[5ex]
    \normalfont\footnotesize
    \makebox[2in]{\hrulefill}\quad\@date\\
    \makebox[2in]{Colin J.~Wynne,~'94}\quad{\hphantom{\@date}}\\
  \end{flushright}}
```

As mentioned in the option section, there is a macro used to put fancy section delimiters into, say, a story. The `\ssbreak` command expects the type of delimiter, the `\ssbreakbar`, to be defined. Since either draft or final must be chosen as an option, this should be fine, but I have put a hopefully redundant command in just in case.

```
\newcommand{\ssbreak}{\bigskip
  \centerline{\ssbreakbar}\bigbreak}
\providecommand{\ssbreakbar}{}
```

1.3 Box formatting

I have written some of my own commands for handling boxes. The first thing I wanted was an analog of `\mbox` or `\hbox` for math mode. The simple version—`\mathbox` puts its argument into an `\hbox`, in math mode, in the current style. The second version is `\Mathbox`, which takes two arguments, the first of which is put in the box and evaluated *before* math mode is entered. This was done for a specific application where I needed to get the contents of the `\mathbox` itself into boldface. Of course, `\boldmath` cannot be evaluated within math mode. Note that the style is chosen by the `\mathpalette` macro, and that the command `\@mathbox` is essentially just a dummy to allow the proper expansion of `\mathpalette`.

```
% \mathbox puts its argument into an \hbox, in math mode, with the
% current \...style.
\def\mathbox #1{\hbox{$\mathpalette\@mathbox{#1}$}}
\def\Mathbox #1#2{\hbox{#1$\mathpalette\@mathbox{#2}$}}
\def\@mathbox#1#2{#1#2}
```

Now, there is a reason why these are defined with `\def` and not `\newcommand`. You see, what I really wanted to do was something like

```
\newcommand{\mathbox}[2][]{%
  \hbox{#1$\mathpalette\@mathbox{#1}$}}
\newcommand{\@mathbox}[2]{#1#2}
```

in order to get optional arguments to my `\mathbox`es. The problem, though, is that I want to use this command in the context of `\box`N`=\mathbox{`…`}`, and for that to work, the first token in the expansion of `\mathbox` *must* be a `\?box` command. The overhead imposed by `\newcommand` precludes this. So, I use the cheap hack until I figure out a more workable way of implementing what I really want.

A more generically applicable box command is one which does unto width what `\smash` does to height. Hence `\smush`:

```
\newcommand{\smush}{\relax
  \ifmmode
    \def\next{\mathpalette\math@smush}
  \else
    \let\next\make@smush
  \fi \next}
\newcommand{\make@smush}[1]{\setbox0=\hbox{#1}\fin@smush}
\newcommand{\math@smush}[2]{\setbox0=\hbox{$\m@th#1{#2}$}\fin@smush}
\newcommand{\fin@smush}{\wd0=0pt \box0 }
```

And finally, vaguely in the realm of boxes, we have struts. Here I have defined some math struts of various sizes (corresponding to the various delimiter sizes on which they are based).

```
\newcommand{\bigmathstrut} {\vphantom{\big()}}
\newcommand{\biggmathstrut}{\vphantom{\bigg()}}
\newcommand{\Bigmathstrut} {\vphantom{\Big()}}
\newcommand{\Biggmathstrut}{\vphantom{\Bigg()}}
```

1.4 Abbreviations, etc.

I have found myself using particular types of abbreviations quite often—often enough that I wanted control sequences for them, whence these first few specimens.

```
\newcommand{\ie}     {\emph{i.e.}\xspace}
\newcommand{\eg}     {\emph{e.g.}\xspace}
\newcommand{\heisst}{d.h\null.\xspace}            % \dh is taken.
```

Note the use of `\xspace` so that explicit space need not be given afterward. I did this mostly because I have never decided whether or not I want to use a comma after either of this.

The second type of abbreviation is the initial, or should I say initials. I finally settled on a style I like—two initials should be separated by a thinspace, and will of course need to have the spacefactor adjusted if at the end of a sentence (followed by a period, in particular). Here is the implementation:

```
\newcommand{\initials}[2]{%
  \break@init #2
  \@ifdefinable #1{%
    \global\edef#1{%
      \noexpand\hbox{\@tempa.\noexpand\,\@tempb}%
      \noexpand\@ifnextchar.{\noexpand\@}{.\noexpand\xspace}}}}
\def\break@init #1.#2.{%
  \def\@tempa{#1}\def\@tempb{#2}}
```

What happens is this. The `\initials` command is given a control sequence name and the initials to be used. The initials are broken on the periods and returned in the specified tokens. Then, if the control sequence is available for definition, it is defined in such a way to make all the spacing and punctuation work out. Since the tokens need to be expanded back to the separate initials, an `\edef` is required—at the same time, use of `\noexpand` is made to keep things from going kablooie at definition time. Here are some standard initials by way of usage example.

```
\initials{\UN}{U.N.}
\initials{\US}{U.S.}
\initials{\AI}{A.I.}
```

1.5 Dates

I have had call to do a fair amount of TeX in both English and German. Therefore, in implementing the examples of date macros from *The TeXbook*, I have provided for both languages.

```
% LaTeX style commands for date-parts, both English and German.
\providecommand{\theday}{\number\day\relax}
\providecommand{\themonth}{%
  \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}
\providecommand{\themonat}{%
  \ifcase\month\or Januar\or Februar\or%
  M\"arz\or April\or Mai\or Juni\or Juli\or August\or%
  September\or Oktober\or November\or Dezember\fi}
\providecommand{\theyear}{\number\year\relax}
```

Note that `\today` is unconditionally defined by the following underhandedness.

```
\providecommand{\today}{}
\renewcommand{\today}{\theday~\themonth, \theyear\xspace}
\providecommand{\heute}{}
\renewcommand{\heute}{den~\theday.\ \themonat\ \theyear\xspace}
  \alias\gdate\heute
```

1.6 Page styles and titles

We are finally into the realm of more traditional package macros, namely creating some general page appearances. Here I have redefined the plain pagestyle to take advantage of the `\pagenofont` defined above.

```
\renewcommand{\ps@plain}{%
  \let\@mkboth  \@gobbletwo
  \let\@oddhead \@empty
  \let\@evenhead\@empty
  \def\@oddfoot{\pagenofont\hfil\thepage\hfil}
  \let\@evenfoot\@oddfoot}
```

The topright pagestyle has page numbers (strangely enough) at the top right of the page.

```
\newcommand{\ps@topright}{%
  \let\@mkboth  \@gobbletwo
  \def\@oddhead{\pagenofont\hfil\thepage}
  \let\@evenhead\@oddhead
  \let\@oddfoot \@empty
  \let\@evenfoot\@empty}
```

1.7 Text formatting

1.7.1 Timelines

A timeline is a long, running, two-column format used, for example, to do résumés or vitæ(or, if you are in Germany, a *Lebenslauf*). The idea is that a date (or some identifying information) appears at the left, and the content is given in the righthand column. The usage is

```
\timeline[⟨pos⟩]{⟨date⟩}
```

where ⟨*pos*⟩ is exactly the argument to `\makebox`, the justification of the ⟨*date*⟩ entry within the lefthand column. That column has length `\timelineskip`, which can off course be set as desired.

```
\newlength{\timelineskip}
\setlength{\timelineskip}{1.75in}
```

The actual entries are not considered to be two separate columns. Rather, the first line is padded out to `\timelineskip` with makebox, and following lines use a hanging indentation. The control sequence `\endtimeline` is defined trivially so that a timeline entry may be used as a timeline environment.

```
\newcommand{\timeline}[2][l]{%
  \noindent\hangindent=\timelineskip
  \makebox[\timelineskip][#1]{\timelinefont{#2}}\ignorespaces}
\let\endtimeline\relax
```

1.7.2 Mathematical declarations

In writing up mathematics, one often wishes to declare definitions, theorems, and so forth. I have written generic declaration macros which can be customized for these uses. Since I prefer to have all such things numbered seuqentially, they use a common counter, called `\declare`, strangely enough. They are numbered within sections if sections are being numbered.

```
\@ifundefined{c@section}
  {\newcounter{declare}}
  {\newcounter{declare}[section]
   \renewcommand{\thedeclare}{\thesection.\arabic{declare}}}
```

When declarations are numbered, it is sometime nice to have the declaration type and number appear uniformly wide throughout. This is done by forcing the declaration to appear in a box of width `\declareindent`.

```
\newlength{\declareindent}
  \setlength{\declareindent}{0pt}
```

We use some internal commands to specify exactly how the declaration is typeset. In fact, we will be defining not only declaration, but an alternate declaration form so that two different methods may be used simultaneously in a document—*e.g.*, when theorems and major results are to be italicized, but definitions and so forth are not.

```
\newcommand{\@declare}    [1]{{\declarefont#1:}\quad}
\newcommand{\@altdeclare}[1]{{\altdeclarefont#1:}\quad}
```

The generic declarations are environments, and are provided in both numbered (normal) and unnumbered (starred) forms. The latter are more simple.

```
\newenvironment{declaration*}[1]%
  {\medbreak\noindent\ignorespaces
     \@declare{#1}\ignorespaces}%
  {\kern0pt\nobreak\smallskip}
```

```
\newenvironment{altdeclaration*}[1]%
  {\medbreak\noindent\ignorespaces
     \@altdeclare{#1}\ignorespaces}%
  {\kern0pt\nobreak\smallskip}
```

The numbered versions introduce nothing surprising, but are a tad more involved.

```
\newenvironment{declaration}[1]%
  {\medbreak\refstepcounter{declare}
     \noindent\ignorespaces
     \ifnum\declareindent = 0\relax%
       \@declare{\thedeclare\quad #1}
     \else
       \makebox[\declareindent]{\@declare{\thedeclare\hss #1}}
     \fi\ignorespaces}
  {\kern0pt\nobreak\smallskip}
\newenvironment{altdeclaration}[1]%
  {\medbreak\noindent\ignorespaces
     \refstepcounter{declare}
     \ifnum\declareindent = 0\relax
       \@altdeclare{\thedeclare\quad #1}
     \else
       \makebox[\declareindent]{\@altdeclare{\thedeclare\hss #1}}
     \fi\ignorespaces}
  {\kern0pt\nobreak\smallskip}
```

Now, because I am essentially lazy and do not want the extra typing needed for an environment, I have shortcuts, `\declare` and `\altdeclare`, as well as numbered versions `\ndeclare` and `\altndeclare`. The first argument is passed to the corresponding environment, and the following paragraph is the body of the environment.

```
\def\declare     #1#2\par{%
  \begin{declaration*}{#1}#2\end{declaration*}\par}
\def\altdeclare #1#2\par{%
  \begin{altdeclaration*}{#1}#2\end{altdeclaration*}\par}
\def\ndeclare    #1#2\par{%
  \begin{declaration}{#1}#2\end{declaration}\par}
\def\altndeclare#1#2\par{%
  \begin{altdeclaration}{#1}#2\end{altdeclaration}\par}
```

Genreality is all well and good, but there are some stock declarations, given here in both numbered and unnumbered versions.

```
\providecommand{\corollary}   {\declare{Corollary}}
\providecommand{\definition}  {\declare{Definition}}
\providecommand{\lemma}       {\declare{Lemma}}
\providecommand{\proposition} {\declare{Proposition}}
\providecommand{\theorem}     {\declare{Theorem}}
\providecommand{\note}        {\altdeclare{Note}}

\providecommand{\ncorollary}  {\ndeclare{Corollary}}
\providecommand{\ndefinition} {\ndeclare{Definition}}
\providecommand{\nlemma}      {\ndeclare{Lemma}}
\providecommand{\nproposition}{\ndeclare{Proposition}}
\providecommand{\ntheorem}    {\ndeclare{Theorem}}
\providecommand{\nnote}       {\altndeclare{Note}}
```

In addition, the following German declaration (meaning a claim) is also defined.

```
\providecommand{\behaupt}    {\declare{Behauptung}}
\providecommand{\nbehaupt}    {\ndeclare{Behauptung}}
```

Finally, since one is not likely to mix numbered and unnumbered, here is a control sequence that will make sure everything is numbered.

```
\newcommand{\allndeclares}{%
  \let\declare     \ndeclare
  \let\altdeclare  \altndeclare}
```

Now that we have propositions, claims, and theorems (oh my!), we want to be able to prove them. The first step is to define a proof environment. The environment simply sets up a label and some spacing. The label is in `\altdeclarefont`. The label is an optional argument which defaults to 'Proof', oddly enough.

```
\newenvironment{proof}[1][Proof]%
  {\smallbreak\noindent{\altdeclarefont#1:}%
     \quad\ignorespaces}%
  {\qed}
```

Just in case you are prooving in German, we also have the following:

```
\newenvironment{beweis}[1][Beweis]%
  {\smallbreak\noindent{\altdeclarefont#1:}%
     \quad\ignorespaces}%
  {\qed}
```

Note that the end of a proof has the command `\qed`, which we will unconditionally define here. This is adapted from *The TeXbook*. The idea is to right justify `\qedsymbol` on the line where `\qed` is invoked, unless there is not a comfortable amount of room. That amount is given as 2 em. When this happens, the line is broken and the `\qedsymbol` appears flush right on the following line.

```
\providecommand{\qed}{}
  \renewcommand{\qed}{%
    {\unskip\nobreak\hfil\penalty 50%
     \hskip 2em\hbox{}\nobreak\hfil\qedsymbol%
     \parfillskip=0pt \finalhyphendemerits=0 \par}}
```

The standard end-of-proof symbol is a box, but I prefer somewhat less ink. I use TeX's hollow diamond suit symbol (again unconditionally defined).

```
\providecommand{\qedsymbol}{}
  \renewcommand{\qedsymbol}{\lower 0.35ex\hbox{$\diamondsuit$}}
```

In case it is desired, the box symbol is defined as `\qedbox`, and then with a simple alias this can be used to end all proofs.

```
  \newcommand{\qedbox}{\vrule height4pt width3pt depth2pt}
```

Now we wish to define control sequences for some constructions commonly found in proofs. I often find myself writing out proofs which require cases. There are two types of cases. Often I will want to have two cases, one for each definition of an equivalence for example. In this case, the case delimiters will be something like `\then` and `\when` commands, and should be set in parentheses to mark them clearly. The other type is the more general 'Case n:' (where n, one hopes, will not be *too* large). To cover both of these, we use a fairly typical LaTeX conceit:

```
\Case*{⟨case⟩}
```

where the unstarred argument sets ⟨*case*⟩ inside parentheses and the star supresses the parentheses. A German alias is given.

```
\newcommand{\Case}{\@ifstar{\@starCase}{\@Case}}
\newcommand{\@starCase}[1]{\@@Case{#1}}
\newcommand{\@Case}[1]{\@@Case{(#1)}}
\newcommand{\@@Case}[1]{%
  \noindent{\declarefont#1}\quad\ignorespaces}
\alias\Fall\Case
```

And finally, just in case the proof is by contradiction, we have the following.

```
\newcommand{\contra}{\ensuremath{\Rightarrow\Leftarrow}}
```

1.7.3 Problems and examples

In addition to declarations, and for subjects other than mathematics, one might want to provide examples and worked problems. I implement examples as a separate environment (well, *four* separate environments) so that they may have fonts distinct from declarations.

```
\newenvironment{example*}%
  {\@nameuse{declaration*}{Example}\examplefont}
  {\medbreak}
\newenvironment{altexample*}%
  {\@nameuse{declaration*}{Example}\examplefont}
  {\medbreak}
\newenvironment{example}%
  {\declaration{Example}\examplefont}
  {\medbreak}
\newenvironment{altexample}%
  {\declaration{Example}\examplefont}
  {\medbreak}
```

Problems are handled differently. In my experience, problems do not appear in longwinded documents using sectioning, and so the counter need not be embedded.

```
\newcounter{problem}
  \setcounter{problem}{0}
\renewcommand{\theproblem}{\arabic{problem}}
\renewcommand{\p@problem}{}
```

Often when working a problem, one wishes to include a reference to the source. This is accomplished via the `\Page` macro. Usage is

`\Page*[`⟨*author*⟩`]{`⟨*pp*⟩`}{`⟨*problem*⟩`}`

In the LaTeX tradition, the standard command tries to outguess you by prepending a hash '#' to the problem number, whereas the starred version omits this. Thus, the standard version produces

([⟨*author*⟩,]p. ⟨*pp*⟩, #⟨*problem*⟩)

and `\Page*` leaves off the hash. Notice that the space between author and page number is inserted by the macro. The macro is defined robustly so that it can be used in moving arguments.

```
\DeclareRobustCommand{\Page}{%
  \@ifstar{\@Page{}}{\@Page{\#}}}
\def\@Page#1{%
  \@ifnextchar [{\@@Page{#1}}{\@@Page{#1}[]}}
\def\@@Page#1[#2]#3#4{%
  \def\@tempa{#2}%
  \ifx\@empty\@tempa%
    \let\@tempb\@tempa%
  \else%
    \edef\@tempb{\@tempa,~}%
  \fi%
  (\@tempb p.\,#3, #1{#4})}
```

The statement of a problem is given in an environment, again so that font customization can be easily done. The statement environment takes a single optional argument, which is typeset at the beginning of the statement in `\altdeclarefont`. The rest of the statement is in `\declarefont`. I use the optional argument to pass a `\Page` reference most often—note, though, that if the optional argument to `\Page` is used, the whole `\Page` command should be put in braces sp that the square brackets of optional arguments do not get confused.

```
\newenvironment{statement}[1][\null]%
  {\def\@tempa{#1}\def\@tempb{\null}%
    \ifx\@tempa\@tempb%
      \def\@tempc{\null}%
    \else%
      \def\@tempc{\altdeclarefont\@tempa\quad}%
    \fi%
    \declarefont{\@tempc}\ignorespaces}
  {\removelastskip\nopagebreak\smallskip}
```

A problem, now, is basically just a wrapper for the statement. It generates and sets the problem number, does some spacing, and the body of the problem environment becomes the body of the statement. In essence, a problem is a numbered statement. There is a starred version which does not do numbers and references—this just does the correct spacing and calls statement. Note that the problem number is set using a macro from the cjw-outl package.

```
\newenvironment{problem}%
  {\setcounter{equation}{0}%
    \gdef\theequation{\theproblem.\arabic{equation}}%
    \removelastskip\medbreak%
    \refstepcounter{problem}%
    \noindent\theoutlabel{\theproblem.}%
    \statement}
  {\endstatement}
\newenvironment{problem*}%
  {\removelastskip\medbreak%
   \noindent\statement}
  {\endstatement}
```

Since I have been known to write assignments in German, we provide the aliases to make an aufgabe environment.

```
\alias  \aufgabe   \problem
\realias\endaufgabe\endproblem
```

For parts and subparts of problems, the aliases are English and the main definitions are German because I wrote these while I was in Germany. So, parts (*Teile*) are numbered within problems and subparts (*Subteile*) within parts. The defaults for cross-referencing (the `\p@` forms) are set, too.

```
\newcounter{teil} [problem]
\newcounter{steil}[teil]
\renewcommand{\theteil}  {(\alph{teil})}
  \renewcommand{\p@teil} {\theproblem}
\renewcommand{\thesteil} {(\roman{steil})}
  \renewcommand{\p@steil}{\p@teil\theteil}
```

The environment part/teil used to be implemented entirely in terms of the cjw-outl package, but that was a little bit of overkill, and made the numbering more difficult to implement. I ought to one day implement problems/parts/subparts entirely in terms of the outline macros. We'll see.

Anyway, the current implementations still rely on some of the definitions from the cjw-outl package. The part/teil environments take a single optional argument, namely the outline depth. Level one starts the text flush left at the margin, which is usually where the problem is, hence a part should be at level two, and this is the default. A more detailed explanation of the goings-on here can be found in cjw-outl.dtx.

```
\newenvironment{teil}[1][2]%
  {\@tempcnta=#1\advance\@tempcnta by -1\relax
   \ifnum\@tempcnta < 1\relax
     \leftskip=0pt\relax
   \else
     \leftskip=\@tempcnta\outlindent
   \fi
   \refstepcounter{teil}
   \addvspace{\medskipamount}%
   \noindent\theoutlabel{\theteil}%
   \ignorespaces}
  {\par\smallbreak}
```

The default level for ppart or steil, the subpart environments, is three.

```
\newenvironment{steil}[1][3]%
  {\@tempcnta=#1\advance\@tempcnta by -1\relax
   \ifnum\@tempcnta < 1\relax
     \leftskip=0pt\relax
   \else
     \leftskip=\@tempcnta\outlindent
   \fi
   \refstepcounter{steil}
   \addvspace{\medskipamount}%
   \noindent\theoutlabel{\thesteil}%
   \ignorespaces}
  {\par\smallbreak}
```

English aliases are, of course, given.

```
\realias\part    \teil
\realias\endpart \endteil
\alias  \ppart   \steil
\realias\endppart\endsteil
```

1.7.4 Footnotes

This is a simple modification to a standard LaTeX 2ε internal macro, because I prefer hanging indentation on my footnote text.

```
\long\def\@makefntext#1{%
  \parindent 1em\noindent\hangindent=\parindent%
  \hb@xt@    1em{\hss \llap{\@makefnmark} }#1}
```

1.7.5 Text displays

I have turned the `\begindisplay` and `\enddisplay` macro pair from *The TeXbook* (page 421) into a LaTeX environment. As with Knuth's macros, local definitions for use within the display can be given, in this case via the display's optional argument. Since the environment is implemented through the standard tabular environment, there is a mandatory argument specifying column layout. Overall, usage is

`\begin{display}[`⟨*local*⟩`]{`⟨*cols*⟩`}`

with local definitions ⟨*local*⟩ and column descriptol ⟨*col*⟩. (By the way, the previous display was created with the display environment. . .)

The display's offset from the left margin is specified by `\textdisplay indent`. Default value is equal to `\parindent`, and is therefore set below, after `\parindent`.

```
\newlength{\textdisplayindent}
```

The actual display environment uses the spacing and penalties of mathematical displays.

```
\newenvironment{display}[2][]
  {\vadjust{\penalty\predisplaypenalty}
   \@newline[\abovedisplayskip]%
   \begingroup%
     #1%
     \begin{tabular}{@{\null\hspace{\textdisplayindent}\null}#2}}
  {\end{tabular}\endgroup
   \vadjust{\penalty\postdisplaypenalty}
   \@newline[\belowdisplayskip]\ignorespaces}
```

1.8 Verbatim inclusions

Since I only occassionally need to include verbatim files, the following macros need to be specifically included by a package option, as was mentioned above.

```
\if@verbext
```

For numbered inclusions, we need a line number counter.

```
\newcounter{vfline}
\renewcommand{\thevfline}{\arabic{vfline}}
```

We need the following command from *The TeXbook.*

```
\providecommand{\uncatcodespecials}{%
  \def\do##1{\catcode`##1=12 }\dospecials}
```

The basic command is `\verbfile` which takes an optional argument specifying the starting line number (default is one), and a mandatory argument which is, of course, the name of the file to include. There is also `\verbfilenolines` which does not number lines, and needs only the one mandatory argument.

```
\providecommand{\verbfile}[2][1]{%
  \par\begingroup\@vf@lines{#1}\input{#2}\relax\endgroup}
\providecommand{\verbfilenolines}[1]{%
  \par\begingroup\@vf@nolines\input{#1}\relax\endgroup}
```

In the manner of command which need to do `\catcode` trickery, the above are primarily wrappers for the real commands. The one problem with these as currently implemented is that they do not handle leading space in the included file. Oh, well.

```
\newcommand{\@vf@lines}[1]{%
  \verbatimfont
  \setcounter{vfline}{#1}
    \addtocounter{vfline}{-1}
  \setlength{\parindent}{0pt}
  \setlength{\parskip}{0pt}
  \def\par{\leavevmode\endgraf}
  \obeylines \uncatcodespecials \obeyspaces
  \everypar{\null\stepcounter{vfline}%
    \llap{\scriptsize\thevfline\quad}\null}}
\newcommand{\@vf@nolines}{%
  \verbatimfont
  \setlength{\parindent}{0pt}
  \setlength{\parskip}{0pt}
  \def\par{\leavevmode\endgraf}
  \obeylines \uncatcodespecials \obeyspaces
  \everypar{\null}}
```

Now we end the inclusion conditional.

```
\fi
```

1.9 Initialization

We use the `\AtBeginDocument` command to set up some default values when the document is actually started.

```
\AtBeginDocument{%
  \setlength{\parindent}        {20pt}
  \setlength{\parskip}          { 2pt plus 1pt}
  \setlength{\textdisplayindent}{\parindent}}
```

2 Math macros

While some macros useful for typesetting mathematics have already been covered, none of them had to do with mathematical equations or symbols—they were macros for logical flow and delineation. In this package, cjwmath, I have written macros which are specifically for typesetting the actual math—the vast majority of these macros, if not actually all of them, are meant to be used in math mode.

2.1 Package initialization

Since different papers require different types of math, I have again used the introduction of conditionals and package options to control what code is actually loaded. The important one concerns use of the AMS math packages in AMS-LaTeX. This is included as an option to my package because some of my definitions depend upon whether the AMS macros are being used. There are conditionals for including code for calculus (both derivatives and integrals) and some code for physics.

```
\newif \if@amsmath
\newif \if@derivatives
\newif \if@integrals
\newif \if@physics
```

There are options corresponding to each conditional.

```
\DeclareOption{amsmath}  {\@amsmathtrue}
\DeclareOption{derivs}   {\@derivativestrue}
\DeclareOption{integrals}{\@integralstrue}
\DeclareOption{physics}  {\@physicstrue}
```

There used to be another option for typesetting units. While I originally included that code in this package directly, I found several occasions where I wanted units but not the rest of the math code. Therefore, units are in a separate package, and the option now just reminds the user to input that package by itself.

```
\DeclareOption{units}{%
  \PackageWarning{cjwmath}%
    {Obsolete option \CurrentOption.  Use package ‘cjwunits’ instead.}}
```

Finally, there is a default option, to warn about unknown options, and the passed option list is processed.

```
\DeclareOption*{%
  \PackageWarning{cjwmath}{Unknown option ‘\CurrentOption’}}
\ProcessOptions
```

This package depends upon the previous one.

```
\RequirePackage{cjwmacro}
```

It also uses the AMS fonts, for which we require amssymb, which itself requires amsfonts.

```
\RequirePackage{amssymb}
```

Just in case things get screwy in cjwmacro—which they shouldn't, we explicitly require amstext here, too, for the `\text` command.

```
\RequirePackage{amstext}
```

I much prefer the following package to AMS's own blackboard bold font.

```
\RequirePackage{bbm}
```

If the amsmath option is specified, we load the package (which itself brings in a lot of other stuff).

```
\if@amsmath
  \RequirePackage{amsmath}
\fi
```

2.2 Miscellaneous macros

Here is a package command which I have written to cover the shortcomings of AMS-LaTeX's `\DeclareNewMathOperator` command. In particular, I would like to be able to set different fonts for some operators. The syntax is

`\NewMathOp*[`⟨*font*⟩`]{\cs}{`⟨*text*⟩`}`

The optional star makes an operator with limits. The ⟨*font*⟩ is, by default, `\operator@font`. `\cs` is the name of the new mathop. ⟨*text*⟩ should be the printed version of the operator, but may also include, for example, extra kerning information, as in

`\NewMathOp[\mathfrak]{\so}{o\kern 0pt}`

The command should produce something robust.

```
\DeclareRobustCommand{\NewMathOp}{%
  \@ifstar{\@makenewop{\displaylimits}}
          {\@makenewop{\nolimits}}}
```

The first iteration applies a font if the optional argument is not given.

```
\def\@makenewop#1{%
  \@ifnextchar [{\@@makenewop{#1}}
    {\@@makenewop{#1}[\operator@font]}}
```

Finally, the net operator itself is declared robustly. The arguments are, in order, either `\displaylimits` or `\nolimits`, the font, the control sequence, and the operator text.

```
\def\@@makenewop#1[#2]#3#4{%
  \DeclareRobustCommand{#3}{%
    \mathop{\kern\z@{#2{#4}}}#1}}
```

The next few macros have to do with things not specific to any particular flavor of mathematics. For example, I like some of the alternate Greek characters more than the originals—notice how we cleverly required the cjwmacro package which gives us the `\swapdef` command.

```
\swapdef{\epsilon}{\varepsilon}
% \swapdef{\theta}{\vartheta}
\swapdef{\rho}{\varrho}
```

I also like the empty set symbol from AMS, to which I also assign a German alias.

```
\swapdef{\nothing}{\varnothing}
\alias\leer\nothing
```

The standard symbols ‘∃’ and ‘∀’ do not have satisfactory spacing, in my opinion, so I redefine them as relations. Notice the aliasing so that the symbols' redefinition can be carried out regardless of current math fonts.

```
\alias\@@exists\exists
  \renewcommand{\exists}{\mathrel{\@@exists}}
\alias\@@forall\forall
  \renewcommand{\forall}{\mathrel{\@@forall}}
```

What LaTeX cleverly calls `\ni` (a backwards ‘$\in$’) really ought to mean ‘such that,’ hence I rename it:

```
\newcommand{\st}{\mathrel{\ni}}
```

Being essentially lazy, I also prefer to make a nice control sequence for some standard abbreviations. One happens to be German (since in German it is more acceptable to use abbreviations of long phrases even in a more formal setting).

```
\newcommand{\WLOG}{Without loss of generality\xspace}
\newcommand{\Wlog}{without loss of generality\xspace}
\newcommand{\obda}{o.B.d.A.\xspace}
\newcommand{\fp}{floating-point\xspace}
```

The following two commands are simply for phantoms I often find myself using, for example to make alignments in arrays and matrices come out right. The mnemonic is ‘phantom negative’ or ‘phantom equals’.

```
\newcommand{\pneg}{\phantom{-}}
\newcommand{\peq}{\phantom{=}}
```

Going probably too far into the realm of generalization, here are some macros to set their arguments inside matching scaled delimiters of various sorts.

```
\newcommand{\anglebrackets}[1]{%
  \left\langle #1 \right\rangle}
\newcommand{\curlybrackets}[1]{%
  \left\{ #1 \right\}}
\newcommand{\squarebrackets}[1]{%
  \left[ #1 \right]}
\newcommand{\vertbrackets}[1]{%
  \left| #1 \right|}
\newcommand{\Vertbrackets}[1]{%
  \left\| #1 \right\|}
```

And now for something completely different—sometimes an operand should be left generic, but not in terms of a variable. The usual way of accomplishing this is to place a small dot where the argument would otherwise go. As I consider this to imply ‘no argument’, the command is `\noarg`.

```
\newcommand{\noarg}{\,\cdot\,}
```

We end with a few things that should be fairly obvious.

```
\newcommand{\ee}[1]{\times10^{#1}}
\newcommand{\half}{\sfrac12}
\newcommand{\ninfty}{-\infty}
```

This is shorthand for function definitions, including an extra control sequence for some backwards compatibility and an alias to German.

```
\newcommand{\fcn}[2]{\colon{#1}\rightarrow{#2}}
  \newcommand{\mapping}[3]{{#1}\fkt{#2}{#3}}
\alias\fkt\fcn
```

Restrictions of functions:

```
\newcommand{\restr}[2][\big]{\kern -.1em #1|_{#2}}
```

2.3 Combinatorics

The binomial coefficient $\binom{n}{k}$ is defined, depending on whether or not AMS-LaTeX is being used.

```
\if@amsmath
  \realias\choose\binom
\else
  \renewcommand{\choose}[2]{{{#1}\atopwithdelims(){#2}}}
\fi
```

And lastly, we have the combinatorial doohickie which is read as 'n multichoose k', using doubled parentheses as delimiters. I think this comes out looking right.

```
\newcommand{\mchoose}[2]{%
  \mathchoice%
   {\left(\kern-0.48em\choose{#1}{#2}\kern-0.48em\right)}
   {\left(\kern-0.30em
        \choose{\smash{#1}}{\smash{#2}}\kern-0.30em\right)}
   {\left(\kern-0.30em
        \choose{\smash{#1}}{\smash{#2}}\kern-0.30em\right)}
   {\left(\kern-0.30em
        \choose{\smash{#1}}{\smash{#2}}\kern-0.30em\right)}
  }
```

There is also the old-fashioned $_n\mathrm{C}_k$-type notation, as used both in English and in German.

```
\newcommand{\Comb}[2]{%                         %   C
  {}_{#1}{\operator@font C}_{#2}}               % #1 #2
\newcommand{\Komb}[2]{%                         %         #2
  {\operator@font Ko}_{#1}^{#2}}                %       Ko
\newcommand{\Kombun}[2]{\Komb{#1,\neq}{#2}}     %         #1
\newcommand{\Perm}[2]{%                         %               #2
  {\operator@font Pe}_{#1}^{#2}}                %             Pe
\newcommand{\Permun}[2]{\Perm{#1,\neq}{#2}}     %               #1
```

2.4 Sets

The most important macro in this section is named, of course, `\set`. The idea is to make sets which look like

$$\left\{x \in \mathbb{R}^2 \,\middle|\, \|x\|_p = 1 \;\forall\, p = 1, 2, 3, \ldots\right\}.$$

That is, there should be scaled braces around two halves separated by a scaled logical delimiter, the vertical bar. The problem with this is getting everything the same height, since the $|$ specifier does not scale and there is no middle-counterpart to `\left`...`\right`.

So, the command has the form:

`\set[`⟨*mid*⟩`]{`⟨*left*⟩`}{`⟨*right*⟩`}`

The optional ⟨*mid*⟩ specifies an alternate delimiter to use between the two definition halves of the set. If it is left *empty*, the null delimiter '.' will be assumed. If anything at all appears in the optional argument, though, the first token *must* be a delimiter, as it will immediately be preceded by a sizing macro. For example, if you wish to use a colon to separate the definitions, use '`[.:]`' as the optional argument. The mandatory ⟨*left*⟩ and ⟨*right*⟩ are simply the halves of the set definition.

```
\newcommand{\set}[3][|]{{%
  \newdimen\@tempdimd%
```

Each half is set in its own box, then the larger of the respective heights and depths are determined.

```
  \setbox0=\mathbox{#2}\@tempdima=\ht0 \@tempdimb=\dp0%
  \setbox0=\mathbox{#3}\@tempdimc=\ht0 \@tempdimd=\dp0%
  \ifdim\@tempdimc > \@tempdima
    \@tempdima=\@tempdimc
  \fi
  \ifdim\@tempdimd > \@tempdimb
    \@tempdimb=\@tempdimb
  \fi
```

We create an invisible rule with that height and depth, and make sure we have a valid delimiter if the optional argument is empty.

```
  \def\@tempa{\vrule width0pt height\@tempdima depth\@tempdimb}
  \def\@tempb{#1}
  \ifx\@empty\@tempb
    \def\@tempb{.}
  \fi
```

Finally, we use a null left delimiter to balance the middle delimiter, and then a left brace to balance the right brace. The rule is set inside both pairs so that they scale identically. Note the use of `\expandafter` so that when the first `\right` is expanded, it can grab the delimiter in `\@tempb`.

```
  \left.\left\{ \@tempa{#2} \,\expandafter\right\@tempb\,{#3} \right\} }}
```

For backwards compatibility, I make two aliases for `\set`; the old commands required the user to specify the larger side of the set definition in order to get sizing correct.

```
\alias\setl\set
\alias\setr\set
```

Here are some macros for typesetting sets symbolically. First off, we might want to know how to typeset a level set.

```
\newcommand{\lvl}[2][\alpha]{\Gamma\ssb{#2}\ssp{(#1)}}
```

There are also fuzzy sets, and their corresponding level sets.

```
\if@amsmath
  \newcommand{\fset}[1]{\Tilde{#1}}
\else
    \newcommand{\fset}[1]{\tilde{#1}}
\fi
\newcommand{\flvl}[2][\alpha]{\lvl[#1]{\fset{#2}}}
```

For want of a better font, I will typeset set collections in `\mathcal`.

```
\alias\coll\mathcal
```

Finally, we deal with some set operators. *The TEXbook* points out the difference between `\setminus` and `\backslash`. I prefer to think of them as ‘set complementation’ and ‘coset’, respectively.

```
\alias\scomp\setminus
\alias\coset\backslash
```

The next macro attempts to create a symmetric difference operator. I don’t like it, but I probably won’t do better until I learn to make my own METAFONT characters...

```
\newcommand{\symmdiff}{%
  \mathbin{\text{\footnotesize$\bigtriangleup$}}}
```

2.5 Sequences and series

Just a few macros are required for various sequences and series, mostly for indexing. The best explanation is simply an example. The code

```
$y \in \seq{x_{ij}}$, where $i\inset{n}$, $j\inrange[0]{m}$
```

produces

$y \in \{x_{ij}\}$, where $i \in \{1, \ldots, n\}$, $j = 0, \ldots, m$.

```
\newcommand{\seq}     [1]   {\curlybrackets{#1}}
\newcommand{\inset}   [2][1]{\in\{ #1,\ldots,#2 \}}
\newcommand{\inrange}[2][1]{ = #1,\ldots,#2}
```

2.6 Calculus

The calculus macros are relegated to auxiliary files, as I rarely need them.

2.6.1 Derivatives

We load the derivatives in if they are requested.

```
\if@derivatives
  \InputIfFileExists{cjwderiv.tex}{}{%
    \PackageWarning{cjwmath}{Option ‘cjwderiv.tex’ not found.}
    \@@derivativesfalse}
\fi
```

This loads both simple and partial derivative macros.

The derivatives are all variations on the basic `\dd` macro, which should be fairly self explanatory.

```
\newcommand{\dd} [2]{\frac{d#1}{d#2}}
\newcommand{\ddt}[1]{\dd{#1}{t}}
\newcommand{\ddu}[1]{\dd{#1}{u}}
\newcommand{\ddv}[1]{\dd{#1}{v}}
\newcommand{\ddx}[1]{\dd{#1}{x}}
\newcommand{\ddy}[1]{\dd{#1}{y}}
```

```

\newcommand{\sdd} [2]{\frac{d^2#1}{d#2^2}}
\newcommand{\sddx}[1]{\sdd{#1}{x}}
\newcommand{\sddy}[1]{\sdd{#1}{y}}
\newcommand{\sddt}[1]{\sdd{#1}{t}}
\newcommand{\sddu}[1]{\sdd{#1}{u}}
\newcommand{\sddv}[1]{\sdd{#1}{v}}
```

2.6.2 Partial derivatives

The partial derivatives are all variations on the theme of `\pard`, which is as `\dd`, replacing the d with ∂.

```
\newcommand{\pard} [2]{\frac{\partial#1}{\partial#2}}
\newcommand{\pardx}[1]{\pard{#1}{x}}
\newcommand{\pardy}[1]{\pard{#1}{y}}
\newcommand{\pardz}[1]{\pard{#1}{z}}
\newcommand{\pardu}[1]{\pard{#1}{u}}
\newcommand{\pardv}[1]{\pard{#1}{v}}
\newcommand{\pardt}[1]{\pard{#1}{t}}

\newcommand{\spard} [2]{\frac{\partial^2#1}{\partial#2^2}}
\newcommand{\spardx}[1]{\spard{#1}{x}}
\newcommand{\spardy}[1]{\spard{#1}{y}}
\newcommand{\spardz}[1]{\spard{#1}{z}}
\newcommand{\spardu}[1]{\spard{#1}{u}}
\newcommand{\spardv}[1]{\spard{#1}{v}}
\newcommand{\spardt}[1]{\spard{#1}{t}}

\newcommand{\spardxy}[1]{\frac{\partial^2#1}{\partial x\partial y}}
\newcommand{\spardyx}[1]{\frac{\partial^2#1}{\partial y\partial x}}
\newcommand{\spardxz}[1]{\frac{\partial^2#1}{\partial x\partial z}}
\newcommand{\spardzx}[1]{\frac{\partial^2#1}{\partial z\partial x}}
\newcommand{\spardyz}[1]{\frac{\partial^2#1}{\partial y\partial z}}
\newcommand{\spardzy}[1]{\frac{\partial^2#1}{\partial z\partial y}}
```

2.6.3 Integrals

We load the integrals in if they are requested.

```
\if@integrals
  \InputIfFileExists{cjwinteg.tex}{}{%
    \PackageWarning{cjwmath}{Option ‘cjwinteg.tex’ not found.}
    \@@integralsfalse}
\fi
```

The first macro is simply a variation of `\int` using the `\limits` macro.

```
\def\integ{\mathop{\int}\limits}
```

Next we have a small shortcut for the differential at the end of an integral. We work around LaTeX’s font encoding macro.

```
\alias\latex@d\d
  \renewcommand{\d}{\,d}
```

We have macros for double and triple integrals, with and without `\limits`.

```
\newcommand{\dint}{\int\!\!\!\int}
\newcommand{\dinteg}{\mathop{\int\!\!\!\int}\limits}
\newcommand{\tint}{\int\!\!\!\int\!\!\!\int}
\newcommand{\tinteg}{\mathop{\int\!\!\!\int\!\!\!\int}\limits}
```

To be honest, I have no idea why I wrote this one. It is probably buried in a homework file of mine somewhere, but I'll be a fiddler crab if I can remember where. . .

```
\newcommand{\flushintlim}[1]{{\phantom{#1} #1}}
```

2.7 Algebra

We define how to typeset an algebra.

```
\alias\alg\mathbbm
```

2.7.1 Fields

A field will also be done in blackboard bold.

```
\alias\field\mathbbm
```

The following fields are defined.

```
\newcommand{\C}{\field{C}}      % Complex
\newcommand{\E}{\field{E}}      % Euclidean (also Evens)
```

Note that $\mathbb{H}$ is a LaTeX accent, so we save it away before redefining it as a field.

```
\alias\latex@H\H                % Quaternions
  \renewcommand{\H}{\field{H}} %   (Hamiltonian field)
\newcommand{\N}{\field{N}} % Natural numbers
\newcommand{\Q}{\field{Q}} % Rationals
\newcommand{\R}{\field{R}} % Reals
% \newcommand{\Rn}[1][n]{\R^{#1}}
\newcommand{\Z}{\field{Z}} % Integers
\newcommand{\pr}{\field{P}} % Primes
```

2.7.2 Groups

Remember `\NewMathOp`? One use for it is in defining mathematical groups. Here are a bunch.

```
% Groups are typeset as operators.
\NewMathOp           {\Aut}{Aut} % Automorphisms
\NewMathOp           {\End}{End} % Endomorphisms
\NewMathOp           {\GL}{GL} % General Linear
\NewMathOp           {\Inn}{Inn} % Inner products
\NewMathOp           {\Pin}{Pin} % Pin
\NewMathOp           {\SL}{SL} % Special Linear
\NewMathOp           {\SO}{SO} % Special Orthogonal
\NewMathOp           {\SU}{SU} % Special Unitary
\NewMathOp[\mathfrak]{\Sn}{S} % Symmetric
\NewMathOp           {\Spin}{Spin} % Spin
\NewMathOp           {\Sp}{Sp} % Symplectic
```

```
\NewMathOp           {\Unit}{U\kern 0pt}% Unitary
\NewMathOp           {\Orth}{O\kern 0pt}% Orthogonal
\NewMathOp[\mathfrak]{\slin}{sl}        % Tangent group to SL
\NewMathOp[\mathfrak]{\so}{o\kern 0pt}  % skew orthogonal
\NewMathOp[\mathfrak]{\sp}{sp}          % skew symplectic
\NewMathOp[\mathfrak]{\su}{u\kern 0pt}  % skew hermitian
```

2.7.3 Linear algebra

If matrices are to be typeset specially, we will use the `\mathcal` font.

```
\alias\mtx\mathcal
```

I have often seen the letter Θ used for the matrix of zeros. I like it that way.

```
\newcommand{\nullmtx}{\mtx\Theta}
```

Taken from Horn and Johnson, a matrix norm can be represented with a triple-bar delimiter.

```
\newcommand{\mnorm}[1]{%
  \left\vert\kern-0.9pt\left\vert\kern-0.9pt\left\vert #1
    \right\vert\kern-0.9pt\right\vert\kern-0.9pt\right\vert}
```

We define the Lie product of two matrices.

```
\newcommand{\lie}[1]{\squarebrackets{#1}}
```

There is an for the trace (Spur, auf Deutsch) of a matrix…

```
\NewMathOp{\Spur}{Spur}
\NewMathOp{\Tr}{Tr}
```

…as well as for diagonal matrices.

```
\NewMathOp{\Diag}{Diag}
```

This is a shortcut for putting delimiters around matrices. With AMS-LaTeX, we use an environment, taking as its two mandatory arguments the left and right delimiters, respectively.

```
\if@amsmath
  \newenvironment{arbmatrix}[2]%
    {\def\@tempa{#2}\left#1 \matrix}{\endmatrix \right\@tempa}
```

Without AMS, we use a command as does standard LaTeX, and define some basic types.

```
\else
  \newcommand{\arbmatrix}[3]{\left#1 \matrix{#2} \right#3}
  \providecommand{\bmatrix}[1]{\arbmatrix[{#1}]}
  \providecommand{\vmatrix}[1]{\arbmatrix|{#1}|}
\fi
```

The next bit of code is used to enter sparse matrices which are often represented in the literature with an oversized zero marking the region of zeros. This takes more than a bit of trickery in LaTeX. The oversized digit will be put in a box, which in most cases needs horizontal and/or vertical adjustment from the position where it is placed in the matrix by default. The default vertical offset will be called `\numoffset`, and is set by default to the height of a `\Bigmathstrut`.

```
\newlength{\numoffset}
```

```
{\setbox0=\hbox{$\Bigmathstrut$}
 \@tempdima=0.8\ht0\relax
 \global\numoffset\@tempdima}
```

Occasionally, one might wish to use something other than a zero as the oversized digit. We define a generic `\Number` macro to be used. The usage is

> `\Number`[⟨*raise*⟩]{⟨*num*⟩}

where ⟨*raise*⟩ is the amount by which the number is raised and ⟨*num*⟩ is the number to be used.

```
\newcommand{\Number}[2][-\numoffset]{%
 \@tempdima=#1\relax
 \smash{\hbox{\raise\@tempdima\@bignumber{#2}}}}
```

The macros `\@bignumber` is called to do the dirty work of typesetting the number.

```
\newcommand{\@bignumber}[1]{\hbox{\LARGE$#1$}}
```

Now, the horizontal adjustment mentioned earlier usually refers to the need to have the large digit straddle two columns in a matrix. This is easily accomplished with `\multicolumn`. Here things start to get ugly, though; `\multicolumn` must be the first thing after the `&` in the array. But the reasonable way to include an optional argument to be passed through to `\Number` is to use the `\newcommand` feature, which ends up putting junk in the way—this is exactly the same problem which arose in writing `\mathbox` earlier. Therefore, there are currently two separate commands, one which takes the optional argument, and one which doesn't. I would really like to remedy this if I ever figure out how.

```
\def\bignumber    #1{\multicolumn{2}{c}{\Number{#1}}}
\def\Bignumber[#1]#2{\multicolumn{2}{c}{\Number[#1]{#2}}}
```

Here are some specific cases for using zero, as is most commonly the case.

```
\newcommand{\Zero}[1][-\numoffset]{\Number[#1]{0}}
\def\bigzero    {\bignumber{0}}
\def\Bigzero[#1]{\Bignumber[#1]{0}}
```

From this we wish to construct various types of matrices. Each variation will have two versions, depending on whether or not AMS-LaTeX has been invoked. Each version, though, has an optional argument which is what will be passed through as ⟨*raise*⟩ to `\Bigzero`. The names of each matrix are an attempt to indicate the layout. For example, the command `\iidiagi` takes three mandatory arguments which will be the diagonal entries, the first two in the upper left, and the third in the lower right with `\ddots` separating them. Likewise, we have `\idiagii` and `\idiagi`.

```
\if@amsmath
  \newcommand{\iidiagi}[4][-\numoffset]{%       % 2     0
    \begin{bmatrix}                             %   3
       #2 &        &    \Bigzero[#1] \\         %     .
          &     #3 &        &        \\         %       .
      \Bigzero[#1] & \ddots &        \\         %   0     4
          &        &        &     #4
    \end{bmatrix}}
  \newcommand{\idiagii}[4][-\numoffset]{%       % 2     0
    \begin{bmatrix}                             %   .
```

```
       #2 &        & \Bigzero[#1] \\            %     .
          & \ddots &     &        \\            %       3
      \Bigzero[#1] &  #3 &        \\            %   0     4
          &        &     &    #4
    \end{bmatrix}}
  \newcommand{\idiagi}[3][-1.2pt]{%             % 2     0
    \begin{bmatrix}                             %   .
       #2 &         \Bigzero[#1] \\             %     .
          & \ddots  &            \\             %       .
      \Bigzero[#1]  &    #3                     %   0     3
    \end{bmatrix}}
\else
  \newcommand{\iidiagi}[4][-\numoffset]{%       % 2     0
    \matrix{%                                   %   3
       #2 &        &    \Bigzero[#1] \\         %     .
          &     #3 &        &        \\         %       .
     \Bigzero[#1] & \ddots &        \\          %   0     4
          &        &        &    #4}}
  \newcommand{\idiagii}[4][-\numoffset]{%
    \pmatrix{%                                  % 2     0
       #2 &        & \Bigzero[#1] \\            %   .
          & \ddots &     &        \\            %     .
      \Bigzero[#1] &  #3 &        \\            %       3
          &        &     &    #4}}              %   0     4
                                  %
  \newcommand{\idiagi}[3][-1.2pt]{%             % 2     0
    \pmatrix{%                                  %   .
       #2 &         \Bigzero[#1] \\             %     .
          & \ddots  &            \\             %       .
      \Bigzero[#1]  &    #3}}                   %   0     3
\fi
```

We now wish to typeset the transpose of a matrix. The most general form is

`\@trans[`⟨*pre*⟩`]{`⟨*post*⟩`}`

which expands to '`^{`⟨*pre*⟩`t`⟨*post*⟩`}`'. Next is `\trans` which takes a single optional argument for ⟨*post*⟩ (no ⟨*pre*⟩).

```
\newcommand{\@trans}[2][]{^{#1\text{\normalfont\textsf{t}}#2}}
\newcommand{\trans} [1][]{\@trans[]{#1}}
```

Why? I don't know—I needed `\trinv`, below, and decided to generalize.

```
\newcommand{\trinv}      {\@trans[-]{}}
```

For backwards compatibility, I have `\ct{`⟨*mtx*⟩`}` to represent matrix ⟨*mtx*⟩ as conjugated and transposed.

```
\newcommand{\ct}[1]{\conj{#1}\trans}
```

The next topic is vectors. I prefer `\vec` to be logical markup as opposed to a specific accent. Thus, I create an alias `\sarvec` (short arrow vector) for the original `\vec`, and another alias `\arvec` for a long arrow, which is just `\overrightarrow`.

```
\alias\sarvec\vec
\alias\arvec \overrightarrow
```

The actual typesetting I prefer for vectors (when I use anything at all) is boldface. This requires the `\Mathbox` command defined earlier so that the argument can be set in a bold math version.

```
\renewcommand{\vec}[1]{\Mathbox{\boldmath}{#1}}
```

To typeset vectors in long form simply uses the `\matrix` command—but this depends, again, on whether AMS-LaTeX is being used. In either case, a single argument—the contents of the vector—is required. Simply delimit with `\\` for column vectors and `&` for rows.

```
\if@amsmath
  \newcommand{\bvec}[1]{%
    \begin{bmatrix}#1\end{bmatrix}}
  \newcommand{\pvec}[1]{%
    \begin{pmatrix}#1\end{pmatrix}}
```

There are also two aliases for row vectors for backwards compatibility.

```
  \alias\brvec\bvec
  \alias\prvec\pvec
\else
  \newcommand{\bvec}[1]{\bmatrix{#1}}
  \newcommand{\pvec}[1]{\pmatrix{#1}}
% \newcommand{\bvec}[2][r]{%
%   \left[ \begin{array}{#1}#2\end{array} \right]}
% \newcommand{\pvec}[2][r]{%
%   \left( \begin{array}{#1}#2\end{array} \right)}
\fi
```

The null vector, like the null matrix, is given with θ.

```
\newcommand{\nullvec}{\vec{0}}
```

Once we have vectors, we need dot products. The simple version just puts its argument inside angled brackets.

```
\newcommand{\dotp}[1]{\anglebrackets{#1}}
```

If special vector notation is required, the two arguments should be separated by a comma (which should be the case anyway), and then each half is passed to `\vec`.

```
\newcommand{\vdotp}[1]{\@vdotp#1@@@}
\def\@vdotp #1,#2@@@{\dotp{\vec #1,\vec #2}}
```

We define some other vector operators to go with the dot product. These are the curl, the divergence, and the Laplacian. These are usually read as 'del dot', 'del cross', and 'del squared', so the first thing to do is rename the `\nabla`(?).

```
\newcommand{\del}  {\vec\nabla}
```

LaTeX defines `\div`, so we rename that and renew the command.

```
\alias\@@div\div
\renewcommand{\div}{\del\dot}
```

Finally, we have the other two.

```
\newcommand{\curl} {\del\cross}
\newcommand{\lapl} {\del^2}
```

In German texts, the linear hull is usually represented as a list of vectors in square brackets.

```
\alias\huelle\squarebrackets
```

2.8 Operators

This section is simply a gathering point for all sorts of mathematical operators—not in the `\NewMathOp` sense, like various groups defined above, but in the sense of mathematical doohickies which take one or two operands and give you something new.

2.8.1 Binary operators

What we have first is a whole bunch of names for the large, 'x'-like times symbol. In the case of specifying dimension (as in, a 4-by-3 matrix), we declare it a `\mathord` so as not to invoke binary spacing. Then `\mal` is simply German for 'times', and `\cross` is the vector space (or group) product using the same symbol.

```
\newcommand{\by}{\mathord{\times}}
\alias\mal \times
\alias\cross \times
```

To indicate isomorphism, I have most often used an equals sign with a tilde above it.

```
\alias\iso \simeq
```

This next one stretches the usefulness of aliasing by defining an operator for normal subgroups.

```
\alias\nsubgrp \trianglelefteq
```

Next we specify a congruence symbol.

```
\realias\cong   \equiv
```

In graph theory, we want a symbol for adjacency of nodes.

```
\alias\adj\leftrightarrow
```

We unconditionally define `\Box` to be a square operator symbol.

```
\providecommand{\Box}{}
\renewcommand{\Box}{\mathbin{\square}}
```

We give a number theoretic division relation, in English and German.

```
\newcommand{\teilt}{\mathbin{|}}
  \alias\divides\teilt
```

Once upon a time I wanted a 'big dot' operator.

```
% \newcommand{\bdot}{\mathop{\lower0.33ex\hbox{\LARGE$\cdot$}}}
```

We can use LaTeX's built-in `\stackrel` to create a definition relation.

```
\newcommand{\defeq}{\stackrel{\text{def}}{=}}
```

The symbol I first saw used to indicate disjoint union was a union ‘cup’ with a bar through the middle. Using a naming convention similar to AMS-LaTeX’s `\Uplus` we get text and display versions.

```
\newcommand{\uminus}{%
  \,\,{\mathbin{\cup\kern-.6em{\raise.05em%
  \hbox{-\negthinspace-\kern-.25em-}}}}\,\,}
\newcommand{\Uminus}{%
  \mathop{\bigcup\kern-0.9em{\raise.05em%
  \hbox{-\negthinspace-\kern-.25em-}}}}
```

We define some handy names for various arrows.

```
\providecommand{\implies}{\;\Longrightarrow\;}
  \alias\then\implies
\if@amsmath
  \newcommand{\when}{\DOTSB \;\Longleftarrow \;}
\else
  \newcommand{\when}{\;\Longleftarrow \;}
\fi
```

The following can be used when tracing a series of implications through a multiline equation environment, for example.

```
\newcommand{\limplies}{\llap{$\implies$}\quad}
```

Based on *The TeXbook*, I have written a ‘skewed’ fraction, which uses a diagonal separator.

```
\newcommand{\sfrac}[2]{%
  \hbox{\kern 0.1em%
  \raise 0.5ex\hbox {\scriptsize$#1$}%
  \kern -0.1em $/$%
  \kern -0.15em%
  \lower 0.25ex\hbox {\scriptsize$#2$}}%
  \kern  0.2em}
```

If we are not using AMS-LaTeX, the following well not yet be defined.

```
\providecommand{\tfrac}{\sfrac}
\providecommand{\dfrac}[2]{{{#1}\over{#2}}}
```

2.8.2 Unary operators

I most often use the longer versions of various math accents, so I redefine them by default to be long. The short ones are saved in a macro with identical name save for a prepended ‘s’ (as we have already seen for `\arvec` and `\sarvec`). The simple ones are bars, tildes, and hats.

```
\alias\sbar\bar
  \renewcommand{\bar}[1]{\overline{#1}}
\alias\stilde\tilde
  \alias\retilde\widetilde
\alias\shat\hat
  \realias\hat\widehat
```

We need something better than the default real- and imaginary-part macros.

```
\renewcommand{\Im}{%
```

```
  \mathop{\mathfrak{Im}}}
\renewcommand{\Re}{%
  \mathop{\mathfrak{Re}}}
```

Complex conjugation is usually denoted by a bar.

```
\alias\conj  \bar
```

Inversion of various types is used often enough to warrant a control sequence.

```
\newcommand{\inv}{^{-1}}
```

To denote power sets, I have used to different alternatives. The default is the standard `\wp` symbol—I don't know what it is supposed to be used for, but it can be modified for the task. On the other hand, if we have a real math script font (as one might have if using my callig style...) we will certainly use that.

```
\@ifundefined{mathscript}
  {\newcommand{\Pow}{\raise 0.4ex\Mathbox{\Large}{\wp}}}
  {\NewMathOp[\mathscript]{\Pow}{P}}
```

And what macro would be complete without a German alias? (Auf Deutsch, die Potenzmenge.)

```
\alias\Pot\Pow
```

We can leave the letter 'I' to computer scientists who don't know how to write an indicator function. For our purposes, we want the neat-o-keen blackboard bold font.

```
\newcommand{\1}{\mathbbm{1}}
```

Next we have some trigonometric shortcuts.

```
\alias\acos\arccos
\alias\asin\arcsin
\alias\atan\arctan
```

Using our generic bracketing macros from earlier, we have absolute value, cardinality (order of a set), cyclic generators, and norms.

```
\alias\abs \vertbrackets
\alias\ord \abs
\alias\cyc \anglebrackets
\alias\norm\Vertbrackets
```

Multiple sums are recurrent enough (no pun intended `:-)`) to get macros. We have double sums, triple sums, and n-fold sums.

```
\newcommand{\dsum}{\mathop{\sum\sum}\limits}
\newcommand{\tsum}{\mathop{\sum\sum\sum}\limits}
\newcommand{\nsum}{\mathop{\sum\sum\cdots\sum}\limits}
```

The next two macros indicate monotone limits, respectively ascending (mnemonic 'limit up') and descending (you guessed it—'limit down').

```
\NewMathOp*{\ulim}{lim\raise0.4ex\mathbox{\mathord{\smash{\uparrow}}}}
\NewMathOp*{\dlim}{lim\raise0.4ex\mathbox{\mathord{\smash{\downarrow}}}}
```

Finally we have the miscellany, where we can really go to town with `\NewMathOp`! These should be pretty clear, unless you don't do math in German, in which case some will be pretty odd.

```
\NewMathOp*{\argmax}{arg\,max}          % arg min
\NewMathOp*{\argmin}{arg\,min}          % arg min
\NewMathOp {\Aff}   {Aff}               % Affine hull
\NewMathOp {\Bild}  {Bild}              % Bild
\NewMathOp {\Cone}  {Cone}              % Cone
\NewMathOp {\Conv}  {Conv}              % Convex hull
\NewMathOp {\Core}  {Core}              % Fuzzy set core
\NewMathOp {\diam}  {diam}              % diameter
\NewMathOp {\dom}   {dom}               % Domain
\NewMathOp {\Epi}   {Epi}               % Epigraph
\NewMathOp*{\esssup}{ess\,sup}          % Essential supremum
\NewMathOp {\fl}    {fl}                % float-point
\NewMathOp {\ggT}   {ggT}               % ggT
\NewMathOp {\Grad}  {Grad}              % Grad
\NewMathOp {\Hypo}  {Hypo}              % Hypograph
\NewMathOp {\Int}   {Int}               % Interior
\NewMathOp {\Kern}  {Kern}              % Kernel
\NewMathOp {\kgV}   {kgV}               % kgV
\NewMathOp {\Lin}   {Lin}               % Linear hull
\NewMathOp {\lcm}   {lcm}               % LCM
\NewMathOp {\Ord}   {Ord}               % order
\NewMathOp {\proj}  {proj}              % Projection
\NewMathOp {\Rang}  {Rang}              % Rang
\NewMathOp {\range} {range}             % Range
\NewMathOp {\Rank}  {Rank}              % Rank
\NewMathOp {\rot}   {rot}               % Rotation
\NewMathOp {\Span}  {Span}              % Span
\NewMathOp {\val}   {val}               % value
```

2.9 Physics

Now, since physics is a subset of mathematics, physics macros are invoked from within the math macro file.

```
\if@physics
  \InputIfFileExists{cjwphys.tex}{}{%
    \PackageWarning{cjwmath}{Option ‘cjwphys.tex’ not found.}
    \@@physicsfalse}
\fi
```

The only thing really specific to physics that I ever used—meaning not applicable in any more general mathematical setting—was the bra/ket notation. Here we have bras (bræ?), kets, and brakets, the latter of which bear uncoincidental resemblance to the `\set` macro.

```
\newcommand{\bra}[1]{\left\langle #1 \right|\,}
\newcommand{\ket}[1]{\,\left| #1 \right\rangle}

\newcommand{\braket}[2]{%
  \newdimen\@tempdimd%
  \setbox0=\mathbox{#1}\@tempdima=\ht0 \@tempdimb=\dp0%
  \setbox0=\mathbox{#2}\@tempdimc=\ht0 \@tempdimd=\dp0%
  \ifdim\@tempdimc > \@tempdima
    \@tempdima=\@tempdimc
  \fi
  \ifdim\@tempdimd > \@tempdimb
    \@tempdimb=\@tempdimb
  \fi
  \def\@tempa{\vrule width0pt height\@tempdima depth\@tempdimb}
  \left.\left\langle \@tempa{#1} \,\right|\,{#2} \right\rangle }
```

2.10 Probability

Here comes that `\NewMathOp` thingie again. First thing is to define some standard probabilistic operators, which really need to be typeset in an operator font in order not to look horrible.

```
\NewMathOp{\Prob} {P}                  % Probability operator
\NewMathOp{\Corr} {Corr}               % Correlation
\NewMathOp{\Cov}  {Cov}                % Covariance
\NewMathOp{\Expct}{E}                  % Expectation
\NewMathOp{\SD}   {SD}                 % Standard Deviation.
\NewMathOp{\Var}  {Var}                % Variance
```

Here is a macro to put inside of some of those operators where conditional events are being considered.

```
\newcommand{\given}{\,|\,}
```

Usually a single tilde, which as an operator bears the name of `\sim` in LaTeX, indicates a distribution. Hence, we make an alias.

```
\alias\distrib\sim
```

And coming back to `\NewMathOp`, we define some standard distributions…

```
\NewMathOp{\Bin} {Bin}                 % Binary dist.
  \newcommand{\Nbin}{-\!\Bin}          % Negative Binom.
\NewMathOp{\Exp} {Exp}                 % Exponential dist.
\NewMathOp{\Geom}{Geom}                % Geometric dist.
\NewMathOp{\Norm}{Norm}                % Normal dist.
\NewMathOp{\Poi} {Poi}                 % Poisson dist.
\NewMathOp{\Unif}{Unif}                % Uniform dist.
```

…and the normal density and distribution functions (which might also be represented as Φ and ϕ).

```
\NewMathOp[\mathfrak]{\Ndens}{N}
\NewMathOp[\mathfrak]{\Ndist}{n}
```

The last thing to do is create some macros for probabilistic modes of convergence. We go, again, from the general to the specific.

```
\NewMathOp*{\@mapsto}{\mapstochar\rightarrow}
\newcommand{\@probconv}[1]{\mathrel{\@mapsto\limits^{1}}}
  \newcommand{\asconv}{\@probconv{a.s.}}       % Almost sure conv.
  \newcommand{\inprob}{\@probconv{P}}          % Conv. in probability
  \newcommand{\inlaw} {\@probconv{L}}          % Conv. in law
  \newcommand{\vague} {\@probconv{v}}          % Vague conv.
```

3 Units

The package cjwunits simply standardizes how to typeset units (or dimensions—things such as seconds, meters, and so forth). Most of the work is done by defining a pretty simple macro, `\unit`. The rest is just a collection of standard units which may be invoked (meaning ones I have used, and therefore stuck into the file).

3.1 Package initialization

The initialization does most everything. We first specify a font for the unit types.

```
\newcommand{\unitfont}{\operator@font}
```

Now the workhorse of the package is defined. It is pretty foolproof, in that it can be invoked in or out of math, may or may not be followed by explicit space, and the font as defined above is pretty customizable.

```
\newcommand{\unit}[1]{\ensuremath{\,{\unitfont{#1}\kern\z@}}\xspace}
```

Now come the examples.

3.2 Distance

```
\newcommand{\ang} {\unit{\AA}}                  % angstroms
  \alias\Ao\ang
\newcommand{\cm}  {\unit{cm}}                   % centimetres
\newcommand{\inch}{\unit{in}}                   % inches
\newcommand{\km}  {\unit{km}}                   % kilometres
\newcommand{\mi}  {\unit{mi}}                   % miles
\newcommand{\m}   {\unit{m}}                    % metres
```

3.3 Electricity and magnetism

```
\newcommand{\Hz}  {\unit{Hz}}                   % herz
\newcommand{\J}   {\unit{J}}                    % joules
\newcommand{\V}   {\unit{V}}                    % volts
\newcommand{\eV}  {\unit{eV}}                   % electron volts
\newcommand{\erg} {\unit{erg}}                  % ergs
```

3.4 Mass

```
\newcommand{\amu} {\unit{amu}}                  % atomic mass units
\newcommand{\gram}{\unit{g}}                    % grams
\newcommand{\kg}  {\unit{kg}}                   % kilograms
\newcommand{\Ton} {\unit{T}}                    % tons
\newcommand{\kT}  {\unit{kT}}                   % kilotons
\newcommand{\MT}  {\unit{MT}}                   % megatons
```

3.5 Thermodynamics

```
\newcommand{\kelv}{\unit{K}}                    % kelvins
```

3.6 Time

It seems that `\sec` already means secant, so we need a preservation and renewal here.

```
\alias\secant\sec
\renewcommand{\sec} {\unit{s}}                  % seconds
```

3.7 Velocity

```
\newcommand{\cee} {\unit{c}}                    % speed o' light
```