BIJLAGE V

Program text generation with TEX/LATEX [1]

Piet van Oostrum
piet@cs.ruu.nl

March, 1991

Inhoud:

1 Web

1.1 Cweb

1.1.1 ctangle, cweave - translate CWEB to C and/or TEX

The ctangle program converts a CWEB source document into a C program that may be compiled in the usual way. The output file includes #line specifications so that debugging can be done in terms of the CWEB source file.

The cweave program converts the same CWEB file into a TEX file that may be formatted and printed in the usual way. It takes appropriate care of typographic details like page layout and the use of indentation, italics, boldface, etc.,and it supplies extensive cross-index information that it gathers automatically.

CWEB allows you to prepare a single document containing all the information that is needed both to produce a compilable C program and to produce a well-formatted document describing the program in as much detail as the writer may desire. The user of CWEB ought to be familiar with TEX as well as C.

Don Knuth wrote WEB for TEX and Pascal. Silvio Levy designed and developed CWEB by adapting the WEB conventions to C and by recoding everything in CWEB.

1.1.2 Levy's Cweb system ported to MS-DOS and VAX/VMS

On the UK TEX Archive at Aston University, directory [tex-archive.web.cweb] contains the sources for building Silvio Levy's CWEB system (a WEB suite which uses C as the programming language, and TEX as the typesetting language) under Unix; this includes a bootstrapping version of Ctangle which permits one to get Cweave, etc., working.

Originally this was accompanied by a file VMS.CH which purported to port Cweb onto VAX/VMS: however, Vax-C has been revised since Levy released this, and further changes were necessary to get the system working under VMS. The original VMS.CH has threfore been deleted, and a new subdirectory [tex-archive.web.cweb.vms] added to hold the files for

[1]Deze NTG bijlage beschrijft diverse methoden om programma-tekst in een TEX/LATEX document op te nemen. Voor het grootste gedeelte is deze beschrijving overgenomen uit de documentatie van de betreffende programma's resp. macro-pakketten. De simpelste manier is natuurlijk het gebruik van 'verbatim' en soortgelijke faciliteiten. Het nadeel van deze methode (en misschien tegelijk ook weer een voordeel) is dat er slechts gebruik gemaakt wordt van een vast font zonder variabele spatiering. De hierna genoemde methoden proberen de programmatekst te formatteren op een wat meer flexibele manier.

bootstrapping CWEB onto VMS, and the change files to make this Ctangle and Cweave.

In addition, I have ported CWEB to MS-DOS (under Borland's Turbo-C V1.5). This was a *major* undertaking, because of clashes between identifiers used in ANSI-standard function prototypes in Borland's libraries and constants defined in the .web files. Other very extensive changes were required to handle the large data structures, which exceed the 64kB segment limit of the PC architecture. However, it's all working eventually, and the files required for bootstrapping it, and the change files, will be found in [tex-archive.web.cweb.ms-dos].

For both of the above ports, the majority of the files of the Unix distribution will also be required.

To get started, fetch the file
`[tex-archive.web.cweb]00readme.txt`
from Aston, either by NIFTP (username PUBLIC, password PUBLIC), or by sending mail to
`<TeXserver@Uk.Ac.Aston.TeX>`,
with the body of the message consisting of the two lines:
FILES
[tex-archive.web.cweb]00readme.txt

This file will tell you which other files will be required to be fetched to have a working version of CWEB under Unix, VMS or MS-DOS.

Brian Hamilton Kelly

1.2 Fweb, A Fortran/Ratfor/C version of WEB

Since the advent of Knuth's famous WEB system for documenting Pascal and, later, Levy's C version thereof, various Fortran users have inquired about the possibility of a WEB system for Fortran. I have developed a version of WEB that supports Fortran, Ratfor, and C (which can be mixed in the same WEB run). The new version is called FWEB. I am releasing v. 0.99 for beta-testing. This release is a bit premature (there are a few known bugs), but professional responsibilities force me to put FWEB on the back burner for a few months.

This project was driven by necessity. I was developing a large scientific code that I planned to write mostly in C. I was eager to document it as well as I could (so my graduate students could understand it), so I was interested in Silvio Levy's CWEB. Levy graciously gave me v. 0.5 of CWEB, which I used successfully for a few years.

However, part of my code could not be efficiently written in C; Fortran was necessary in some form. Straight Fortran is terrible, but RATional FORtran (Ratfor) removes many of the deficiencies and makes Fortran look something like C. Thus, part of the code was written in Ratfor, with a preliminary pass through the macro preprocessor m4 for good measure. Of course, I wanted to document that code as well. Now although CWEB was never intended for Ratfor, because of Ratfor's C-like syntax CWEB was able in many cases to format Ratfor code satisfactorily. Of course, sometimes the documentation fell apart completely because keywords like DIMENSION aren't known to C, and certain syntax is different. Thus, after living with usable but somewhat marginal woven Ratfor output from CWEAVE for a while, I decided to do things right and teach CWEAVE the appropriate rules for Fortran. I though that would take a couple of weeks...

The first project was to endow FWEB with the concept of a current language. Since I routinely mix C and Ratfor code and I wanted all the documentation to be all in one place, it was important to be able to switch between syntaxes as necessary. After some MONTHS (FWEB was developed in my spare time), I had taught the WEAVE processor to produce reasonable quality typeset Fortran and Ratfor, and to switch back and forth between the various languages. Little had been done to TANGLE at that point. However, I then decided that it was pointless to make separate passes through m4 and Ratfor; why not make TANGLE do it all. Levy had eliminated the macro preprocessor from CTANGLE since C has its own. I reinstated one, patterned after ANSI C. I also added a statement translator, so Ratfor keywords could be expanded.

Thus, the present version of FWEB has these significant enhancements over CWEB:
1. The concept of a current language (Fortran, Ratfor, and/or C);
2. A C-like macro preprocessor;
3. Ratfor to Fortran-77 translation.

My students and I use FWEB every day; we find it to be a tool of great utility. For writing in Fortran, the macro processor is indispensible. As far as Ratfor is concerned, I find it to be a great step forward; I hope never again to have to write a straight Fortran program.

The goal for FWEB v. 1.0 was to achieve functionality. It would be pointless to defend the elegance of all of the internal code at this point: some should be optimized for speed; some should be rewritten. Eventually, these projects will be undertaken. (Note that since I'm a major user I have a powerful motivation.) But now it's most important to get some Fortran users involved so I get some feedback and bug reports.

The relevant files are available via anonymous FTP from Internet host CCC.NMFECC.GOV in directory (VMS syntax) `TEX$ROOT:[DISTR.FWEB]`. The files are described in READ.ME. As this explains, for VAX/VMS users only a subset of all files is necessary, since the executable binaries FTANGLE.EXE and FWEAVE.EXE are provided. If you have the courage to try to bring things up on another machine, you should also read INSTALL.FWEB, and transfer all files except *.EXE, *.HLB, and *.HLP to your machine.

If you are bootstrapping onto another machine, please note that you may have to make a few operating system-dependent changes in the source code. Feel free to contact me for help.

I will appreciate bug reports, suggestions, etc. My addresses are

MFEnet: krommes@ppc.mfenet
Internet: Krommes%ppc.mfenet@ccc.nmfecc.gov
Bitnet: krommes%ppc.mfenet@lbl.bitnet

John Krommes

1.3 Spiderweb

A note about a new implementation of WEB Norman Ramsey, Odyssey Research Associates, July 4, 1988 [2]

1.3.1 abstract

Literate programming has received recent attention in the *Communications of the ACM* [Bentley 86, Van Wyk 87]. WEB is a tool intended for literate programming, but until recently it was useful only for writing PASCAL programs. The author has developed a new tool, SPIDER, which reads a description of a programming language and writes a WEB system that can be used to write programs in that language. SPIDER has been used in the author's organization to build WEB systems for Ada, C, AWK, and other languages. The author hopes that SPIDER will enable people to write literate programs in many more languages than they could before.

1.3.2 Introduction

Donald Knuth developed the WEB system of structured documentation as part of the TEX project [Knuth 84]. His implementation of WEB combined PASCAL and TEX. The WEB idea suggests a way of combining *any* programming language with *any* document formatting language, but until recently there was no software support for writing in WEB anything but PASCAL programs. In 1987, Silvio Levy rewrote the WEB system in C for C, while retaining TEX as the formatting language [Levy 87]. I have has modified Levy's implementation by removing the parts that make C the target programming language, and I have added a third tool, SPIDER, which complements WEAVE and TANGLE. SPIDER reads a description of a programming language, and writes source code for a WEAVE and TANGLE which support that language. Using SPIDER, a C compiler, and an AWK interpreter, an experienced systems programmer can generate in a few hours a WEB system for an Algol-like language.

1.3.3 Features of Spidery WEB

An exhaustive list of Spidery WEB's features would interest only WEB experts, but I do want to mention some features that I hope will encourage people to use Spidery WEB.

- TANGLE and WEAVE can read from multiple files (this feature is present in Levy's CWEB), and TANGLE can write to multiple files. Included files will be searched for on a path if not found in the current directory. These features make Spidery WEB more usable on systems that have make.
- TANGLE can expand macros with multiple parameters.
- The starred sections in Spidery WEB can be organized hierarchically (in three levels). We have a UNIX tool that can extract different pieces of the hierarchy from the output of WEAVE, so that it is possible to take excerpts from WEB documents.
- TANGLE writes #line directives, so you can debug at the WEB source level if your compiler respects the C conventions for #line.
- Many features of WEB seem to exist only to compensate for deficiencies in PASCAL, and most of those were dropped in CWEB. I have changed much of CWEB in order to avoid being bound too much by C conventions. As a result, there are dozens of minor differences between Spidery WEB and original WEB. To give just one example, Spidery WEB supports octal and hexadecimal constants using WEB-style notation, not the C notation used in CWEB.

1.3.4 Scope of SPIDER

SPIDER can generate WEB systems for a variety of languages. The author has written SPIDER description files for C, AWK, Ada, SSL (a language that describes attribute grammars to the Cornell Synthesizer Generator), the Larch Shared Language (a language for describing equational theories), and Dijkstra's language of guarded commands. Debugging the grammar that WEAVE uses to prettyprint the language is the most time-consuming part of creating a WEB system for a new target language, and SPIDER makes it trivial to change that grammar. To make a SPIDER description file for an Algol-like language that uses infix expression notation, an experienced systems programmer should be able to adapt an existing SPIDER description file very quickly.

SPIDER's major limitations are lexical. All Spidery WEBs assume that spaces and tabs in the input are not significant, except as separators; this makes it impossible to construct Spidery WEBs for languages like Fortran and Miranda, where the position of text on a line is significant. The lexical structures of identifiers, string literals, and numeric literals are fixed.

[2]Copyright 1989 by Norman Ramsey, Odyssey Research Associates. To be used for research purposes only. For more information, see file COPYRIGHT in the parent directory.

Conclusions

SPIDER is a modest piece of engineering; it does not introduce new ideas. SPIDER does make it possible to create a new WEB quickly, and to tinker with it easily. The author's group routinely uses Spidery WEB to write programs in Ada, C, and SSL, and has been pleased with the result. We have written in WEB an application of eighteen thousand lines, and we are very pleased at how easy it has been to review and maintain this code. The author hopes that the availability of Spidery WEB will encourage other groups to try literate programming, and that they, too, will be pleased with the results.

References

[Bentley 86] Jon L. Bentley, "Programming Pearls," *Communications of the ACM* **29:5** (May 1986), 364–368, and **29:6** (June 1986), 471–483.

[Knuth 84] Donald E. Knuth, "Literate Programming," *The Computer Journal* **27:2** (1984), 97–111.

[Levy 87] Silvio Levy, "WEB Adapted to C, Another Approach," *TUGBoat* **8:1** (1987), 12–13.

[Van Wyk 87] Christopher J. Van Wyk, "Literate Programming," *Communications of the ACM* **30:7** (July 1987), 593–599, and **30:12** (December 1987), 1000–1010.

2 Tgrind

Several people have asked about including program source in TEX documents. I believe the most general approach is to use a utility like tgrind (TeX analog of vgrind). One specifies the language to be processed (so that tgrind can detect keywords). Tgrind converts tabs into the appropriate spacing (generates things like `\tab{24}` for 3 tabs), boldens keywords, prints quoted strings in typewriter font, prints comments in italics, and other nice things. All these goodies are customizable. This requires an extra pass, but the preprocessing is quite fast.

Tgrind is on the Unix TEX tape, in the directory TeX-contrib/van. It was written by Van Jacobson of LBL. Unfortunately it is Unix specific, but I think equivalents for other OS are no problem. Heck, just take your favourite pretty-printer and generate TEX instead of prettified output.

Tgrind formats program sources in a nice style using TEX. Comments are placed in italics, keywords in bold face and strings in typewriter font. Source file line numbers appear in the right margin (every 10 lines). The start of a function is indicated by the function name in large type in the right margin.

In regular mode tgrind processes its input file(s) and passes them to TEX for formating and output.

In format mode (i.e., when the flag is used), tgrind processes its input file(s) and writes the result to standard output. This output can be saved for later editting, inclusion in a larger document, etc.

Currently known are PASCAL, RATFOR, Modula-2, MODEL, C, ISP, Yacc, Prolog, Icon, TEX, CSH, and Bourne Shell .

Author of Tgrind is Van Jacobson, Lawrence Berkeley Laboratory (based on "vgrind" by Dave Presotto & William Joy of UC Berkeley).

Ken Yap
<ken@rochester.arpa>

3 C2latex

C2latex provides simple support for literate programming in C. Given a C source file in which the comments have been written in LATEX, c2latex converts the C source file into a LATEX source file. It can be used to produce typeset listings of C programs and/or documentation associated with the program.

The C source given to c2latex usually has the following form. It starts with a large comment containing LATEX commands that start a document along with any initial text. Then there is a sequence of comment and code pairs, with the comment explaining the code to follow. The source file is ended by a comment containing LATEX commands that finish the document.

C2latex produces LATEX source by implementing a small number of rules. A C comment that starts at the beginning of a line is copied unmodified into the LATEX source file. Otherwise, non-blank lines are surrounded by a pair of formatting commands (`\begin{flushleft}` and `\end{flushleft}`), and the lines are separated by `\\*`. Each non-blank line is formatted using LATEX's `\verb` command, except comments within the line are formatted in an `\mbox`.

The c2latex program is written in ANSI C and can be processed by c2latex to produce LATEX source containing a typeset listing of itself. It has a copyright similar to those distributed with GNU software. c2latex is available from me as a shar file via electronic mail. If there is enough interest, I will request that the sources be placed on a public server.

John D. Ramsdell
ramsdell@celebes.mitre.org

4 C++2latex

The program c++2latex converts ANSI-C/C++ programs into LATEX source.

It requires flex which can be found on various ftp sites, e.g. prep.ai.mit.edu. For those without flex and without the possibility to get one, I can email the flex'ed program.

Please notice that this program is under GNU Copyleft.

Description

c++2latex is a tool for generating LaTeX source from ANSI-C or C++ programs. It recogizes all keywords, strings, and comments. These recognized parts can be set in different fonts. c++2latex can generate complete LaTeX files which can directly passed through LaTeX or parts of LaTeX files which can be included in other files (either direct or by the `\input` or `\include commands)`. The output filename is searched in various steps. First, if the `{-o,+output}` flag is given, the output is written to the file pointed to by the value of this flag. If the `{-t,+pipe}` option is given, the output is written to stdout. (It is an error to specify both options together.) If none of this options but an input pathname is given, the output is written to a file who's name is the last component of the input pathname with the substituted or else added suffix '.tex'. If the input is read from stdin and none of the above options is given, the output is written to '<program-name>.tex' with <program-name> being the name of this program.

5 Cprog/Csty macros

The cprog macros allow programs in C, C++, Pascal, and Modula-2 to be included directly into TeX documents. Program text is set in a Roman font, comments in slanted, and strings in typewriter. Operators such as <= are optionally combined into single symbols like `$\le$`. Keywords are *not* emphasised—I find this ugly and distracting. (By purest coincidence it would also be very hard to do.)

These macros can be `\input` in plain TeX or used as a style file in LaTeX. They provide a convenient alternative to tgrind, particularly for program fragments embedded in documents. Full instructions for use appear in the macro package itself.

This allows C programs to be formatted directly by TeX. It can be invoked by `\cprogfile{filename}` or (in LaTeX) `\begin{cprog}` ... `\end{cprog}` or (in plain TeX) `\cprog ... \end{cprog}`. In LaTeX, the alternative form `\begin{cprog*}` is allowed, where spaces in C strings are printed using the 'square u' character (like LaTeX verbatim*). In plain TeX, you have to use `\csname cprog*\endcsname` for this (sorry). If you are using `\cprogfile`, say `\cprogttspacetrue` beforehand if you want this effect.

The formatting is (necessarily) simple. C text is set in a normal Roman font, comments in a slanted font, and strings in a typewriter font, with spaces optionally made visible as the 'square u' symbol. Tabs are expanded to four spaces (this does not look good when comments are aligned to the right of program text). Some pairs of input characters appear as single output characters: `<< <= >> >= != ->` are respectively TeX's `\ll \le \gg \ge \ne \rightarrow`. Say `\cprogpairsfalse`to disable this.

You can escape to TeX within cprog text by defining an escape character. The character @ is suitable for C and Pascal. I have not tested other characters so they may interact badly with their existing definitions here. To define @ as the escape character, do `\cprogescape@`. Then within text you can do @ followed by TeX commands. These commands will be in a TeX group with the `\catcodes` of `\{}%` as normal. The commands are terminated by a newline, which is not considered part of the program text.

The fonts below can be changed to alter the setting of the various parts of the program. The `\cprogbaselineskip` parameter can be altered to change the line spacing. LaTeX's `\baselinestretch` is taken into account too. The indentation applied to the whole program is `\cprogindent`, initially 0. Before and after the program there are skips of `\beforecprogskip` and `\aftercprogskip`; the default values are `\parskip` and 0 respectively (since there will often be a `\parskip` after the program anyway).

If the source text is Pascal or Modula-2, say `\pascaltrue` or `\modulatrue` (respectively) before formatting it. This makes (* *) be recognised for comments instead of /* */. Braces {} are also recognised for Pascal. `\pascalfalse` or `\modulafalse` as appropriate restores the default of C.

This package works by making a large number of characters active. Since even spaces are active, it is possible to examine the next character in a macro by making it a parameter, rather than using `\futurelet` as one would normally do. This is more convenient, but the coding does mean that if the next character itself wants to examine a character it may look at a token from the macro rather than the input text. I think that all cases that occur in practice have been looked after.

The macros could still do with some work. For example, the big macro defined with [] taking the place of {} could be recoded to use {} and so be more legible. The internal macros etc should have @ in their names, and should be checked against LaTeX macros for clashes.

Eamonn McManus
<emcmanus@cs.tcd.ie>

6 Program environment

In TEX or LATEX is it possible, but difficult, to create a nice layout for programs. The easiest way is to use the `verbatim` environment. The layout is then copied from the input file. In most books a program (e.g., a PASCAL-program) is displayed using boldface reserved words, using math-italic for statements and using teletype fonts for string representations. Without extra equipment this can be done in TEX as well as in LATEX using a `tabbing`-environment. Of course, each font choice should be made explicitly, e.g., you must say `{\bf begin}` to create a boldface begin-symbol and `$x:=x+1$` to denote an assignment. Furthermore, the user is responsible for setting the tabs and jumping to the right ones. This is far from being user-friendly.

The `program`-environment tries to be of some help while displaying program-texts. It contains a number of macros of the form `\BEGIN`, `\PROCEDURE` and alike that not only put down a boldface begin- or procedure-symbol but also sets and jumps to the right tabs. Using the `program` environment from the `program2`-style file also automatically puts the statements in math-mode (you do no have to use `$`-signs anymore).

Rein Smedinga
Department of computing science,
P.O.box 800,
9700 AV Groningen
`rein@cs.rug.nl`
November 9, 1990

7 Schemetex

SchemeTeX provides simple support for literate programming in any dialect of Lisp on Unix. Originally created for use with Scheme, it defines a new source file format which may be used to produce LATEX code or Lisp code. Roughly speaking, LATEX formats Lisp code in a verbatum-like environment, and it formats Lisp comments in an ordinary environment.

SchemeTeX is available via anonymous FTP from linus (192.12.120.51) in the shar file named "pub/schemeTeX.sh". Included is an operating system independent version for the T dialect of Lisp.

John D. Ramsdell
`<ramsdell%linus@mitre-bedford.ARPA>`

8 Ada

Zie TUGboat 10#1, April 1989
APE – A set of TEX macros to format Ada programs

I have developed a set of macros to do exactly this for Ada programs. To get more details on these macros, read the report in the April issue of TUGBoat, or the Nov/Dec issue of Ada Letters (both in 1989).

The macros are available by anonymous ftp to anna.stanford.edu in the pub directory (I think). I can send them to you if you are willing to cover the costs.

Sriram Sankar

Zie ook:
GUTenberg'90
May 15-18, 1990
University Paul Sabatier, Toulouse, FRANCE

Typesetting ADA programs (P. Naudin, C. Quitte)

9 Miscellaneous

9.1 Typesetting programming languages in LATEX

About two years ago I wrote a program that converts programs to TEX. The program sets keywords in boldface (or any font you select) using a data file to find out which strings are keywords and how to skip comments. I have data files for Modula-2, C, C++, Pascal, Occam, Beta, and some more. The program is written in C (under VMS) and could well be improved. However it does what I want it to do, it makes programs 'look' nice in listing. As said it translates to TEX and not LATEX but that is no big deal to change I'd guess.

If you need this program send a mail to rosenber@ra.abo.fi. mail

Robin Rosenberg

9.2 Typesetting PASCAL in LATEX

I have an SED script and Pascal environment for LATEX that follows the standard Algol 60 style for setting Pascal text. The SED script translates everything between `\begin{pascal}` and `\end{pascal}` in various ways that the pascal environment understands. The only problem with the thing is that indenting must be in multiples of 4 spaces, but I've used it for a number of publications without hearing any complaints about my awkward indenting style.

In any case, I strongly recommend the notion of environments for language types, as opposed to the various grind programs. It would be nice to have a standard set of environment parameters to control things like keyword font so language environments from different sources could be at least somewhat interchangable.

Doug Jones
`jones@herky.cs.uiowa.edu`

9.3 manpage.sty

This style option is designed to work with the report document style of LATEX version 2.09. Use `\documentstyle[11pt,manpage]{report}`

This LATEX style file is similer to the UNIX troff man macros in format and is specially tuned for documenting the C++ library that the author wrote.

The commands that are created in the style file are:

```
\begin{manpage}{Title}{Module}{Version}  % see an example, all will be clear
\end{manpage}                            % end of manpage environment
\variable{#1}  (e.g., \variable{int foo}) % with \medskip added
\variable*{#1} (e.g., \variable*{int bar})% no extra spacing
\function{#1}  (e.g., \function{void demo(int dummy)}) % with \medskip added
\function*{#1} (e.g., \function*{void demo(int dummy)})% no extra spacing
\subtitle{#1}  (e.g., \subtitle{AUTHOR}) % fit in the same line if possible
\subtitle*{#1} (e.g., \subtitle*{AUTHOR})% always break a newline
"#1"           (e.g., "dummy_variable")  % argument is in italic&unbreakable
\separator           % draw a thin line to seperate suntitle from the text
\header{#1}{#2}{#3}  % in case you want to have a header and
\footer{#1}{#2}{#3}  % a footer outside of the manpage environment
\dq                  % print double quote character (")
```

In the `\function` macro, data types and their dummy arguments are separated by a space. So if you have a function like "int f(const int n)", you should document it as `\function{int f(const~int n)}`. The argument n is optional. In the `\subtitle` macro, two lines of text may be devided by "\\".

Rong Chen (rchen@cs.uiuc.edu)
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

9.4 Typesetting Scheme code

I wrote something called SLATEX that allows listings of Scheme code in LATEX without restricting it to the usual monospace typewriter font supported by other code typesetters.

I've placed the current version in the titan.rice.edu's anonymous ftp area: get public/slatex.sh.

SLATEX decides which tokens should be, say, boldfaced, italicized, or sansserifed, pretty much along the style of the Little Lisper [1]. (The user can completely control this default decision process, so much so that he can flip the fonts around, add new fonts, or even do something silly like make everything come out in typewriter – i.e., turn the program into a no-op.)

I'd been leery of distributing SLATEX before because of the frequent updating that it's undergone following the Rice Scheme-and-similar- language users' lively demands for bells and whistles, e.g.,

- allowing arbitrarily positionable displays, boxed code, in-text code, and directly inputing actual Scheme files;
- getting little pockets of LATEX text or mathmode into the Scheme code, for readable Schemelike pseudocode (useful for expository papers and class handouts);
- making it learn automatically that a macro definition implies that keyword should henceforth be boldfaced, etc.

At any time, the ftp site will contain the most recent code. The shar file contains the Scheme source (Ch*z, but should carry over to other Schemes with minor changes), a shellscript that piggybacks the codesetter on to LATEX, the requisite LATEX style file, installation instructions, a manual in LATEX, a man page, and a copyleft. The first 2 sections in the manual suffice for most uses, with fine tuning being described later. (The code, as of now, contains more fine tuning than documented – I'll update the docs when I next get time.)

This is free(ly distributable) software, and hence no warranty, though I'll be glad to field bug and other reports.

[1] D.P. Friedman and M. Felleisen, 'The Little Lisper', Science Research Associates (3e), 1989, and The MIT Press, 1987.

Dorai Sitaram
dorai@tone.rice.edu

BIJLAGE W

LaTeX for engineers and scientists

(book review) [1]

Nico Poppelier

n.poppelier@elsevier.nl

Februari, 1991

Although the LaTeX manual is a useful book, it is not suitable as an introduction, as a book for beginning users. 'LaTeX for engineers and scientists' by David J. Buerger, which was published this year, at first sight appears to be a good introduction. In the preface the author writes: '[this book] was written to provide a fast and easy way to learn how to produce technical documents with LaTeX.' And indeed, 'LaTeX for engineers and scientists' is a book that doesn't frighten readers by its length and is easy to read. It describes BIBTeX and MakeIndex, it gives exercises – with answers – that are really not bad, and it contains an index – although I find that a bit short – and a glossary.

Unfortunately my general opinion about this book is not positive: both the contents of the book and the quality of the book as a printed product leave a lot to be desired.

My overall impression of the contents of the book can be summarized in a few points.

- The author has not quite grasped the concept of a document style and the separation between logical and visual structure, two fundamental concepts of LaTeX.
- The author does not distinguish between LaTeX proper and LaTeX *plus* the standard document styles. There are many document styles beside the standard ones, so this distinction is essential.
- In several examples LaTeX and TeX commands are mixed. My opinion is that in examples only LaTeX commands should be used. If the author insists on mentioning the TeX equivalents, he should explain what sort of functionality LaTeX adds.
- Some functions of LaTeX, among which at least one important function, are not explained in the book.
- Explanations in the book are sometimes confusing or sloppy. In a few cases they are even incorrect.

I will give some examples:

- In chapter 4, Formatting environments, the author starts with the `center`, `flushleft` and `flushright` environments, and then goes on to treat the list and quotation environments. The main purpose of LaTeX's markup instructions is describing the logical structure of the text. In a book on LaTeX, descriptions of logical design should come before descriptions of visual design.
- The custom description list on p. 28 refers to layout parameters of the `list` environment that cannot be found in the index or the glossary. Although the author includes in his book instructive page-layout diagrams[2] that are unfortunately absent in the LaTeX manual, he forgets to include the equally useful list-layout diagram that is printed on p. 113 of the LaTeX manual.
- The custom description list given as an example on p. 28 is a variation on the `description` environment described in the LaTeX manual. In this example the items are typed as `\item[{\bf Fox}]`; as a result there is no clear separation between form and contents. A better way would be to define the layout of the items in the definition of the customised list. That way, one only has to type `\item[Fox]`.
- On p. 39 the author gives a table of the typeface sizes that correspond to LaTeX commands such as `\small`, `\normalsize` and `\large`. The correspondence given in the table is valid only for the standard document styles and not for *every* document style. By failing to make this distinction, the author suggests that the table is universally valid, which it isn't.
- In chapter 6 the author treats only the `$ ... $` and not the `\( ... \)` construction for in-line mathematical formulae. `$ ... $` and `$$ ... $$` give formulae in a more or less fixed layout. If one uses LaTeX's `\( ... \)` and `equation` environment instead, the user lets the document style control the formula layout. Furthermore, the LaTeX notation for formulae has opening and closing tags that are not identical, which results in fewer errors.
- In chapter 7, on p. 52, the author introduces the `\lefteqn` command without any explanation. This is a command that a lot of users find confusing: they often think that `\lefteqn` puts an equation flush with the left margin of the text.

[1] To be submitted to TUGboat, © 1991, TeX Users Group.
[2] Similar diagrams have appeared in *TUGboat*.

- In chapter 8 the author gives a confusing description of the two environments `table` and `tabular`. The `tabular` environment produces a table, i.e. an arrangement of cells in rows and columns, possibly with horizontal and vertical rules[3]. The `table` environment creates a floating object, i.e. a part of the document for which LaTeX tries to find a good place to print it. In most cases, the `table` environment contains a caption that starts with the word 'Table'[4] and a `tabular` environment for the actual table contents. However, Buerger writes (italics mine):

  > Tables *created* with the `tabbing` or `tabular` environments— . . .
  > The `\begin{table[ ]` or `\begin{figure[ ]` command will *create* a table or figure.

- On p. 64 the author explains the use of `\label` and `\ref`. He instructs the reader to put the `\label` command after sectional-unit commands and after the `\caption` command of a `figure` or `table` environment. However, there is no information on where to put the label in `equation` and `eqnarray` environments.
- In chapter 10, Organizing a document, the author uses in an example
  ```
  \topmargin 0mm
  \def\BibTeX{ ... }
  ```
  instead of the LaTeX equivalents
  ```
  \setlength{\topmargin}{0mm}
  \newcommand{\BibTeX}{ ... }
  ```
- In chapter 10 the author fails to distinguish between LaTeX proper and the standard document styles. On p. 68 the author writes:

  > Title information is automatically centered.

  and (italics by the author):

  > You can produce an abstract placed below the title information . . . by typing the following command *before* the `\maketitle` command.

  In both cases the behaviour the author describes is that of the standard document styles: in other document styles a title could be left-justified and emphasized phrases could be printed in a boldface font. In the second case, the author is also definitely wrong since the `\maketitle` command defined in the standard document styles does not print the abstract, but only the title, author and date.

The author is also inconsistent with notation: for example, in pages vii-xiii, the table of contents, list of figures and list of tables, I found 'LaTeX', 'LaTeX' and 'LaTeX'! I sometimes got the feeling that the book was written or at least finished in some haste.

Some examples of features of LaTeX that are missing in 'LaTeX for engineers and scientists'.

- The author writes that the `\include` command is similar to the `\input` command, except that it starts on a clean page. He doesn't mention one of the nicest mechanisms in LaTeX: cross-referencing between sub-documents if some of the sub-documents are excluded from the current formatting run by means of `\includeonly`.
- The only information on TeX's units was the sentence 'There are 72.27 points to an inch', and I found it in the chapter on error messages!
- One of the sample input files contains the `\;` command, without explanation and without treating other, similar commands.

So far, I have only criticized the author. However, I think the publisher of this book, McGraw–Hill, can be blamed for a few things as well. Concerning the quality of the book as a printed product: the book was produced from camera-ready pages prepared by the author on a laser printer. Computer Modern is a good typeface, if only you use it on a printing device of sufficiently high quality. Laser printer quality is, I'm afraid, not good enough and I hope this book is one of the last books on TeX-related matters produced in such a way. As for the contents of the book: it seems likely that McGraw–Hill did not ask an expert to review the book, otherwise they would have asked the author to rewrite parts of it.

'LaTeX for engineers and scientists' is not a bad book, but it is not a good book either. It can be used, but I can't really commend it.

[3] An imprecise definition of a table, I know!
[4] To be precise: this is specified by the document style, but it should be 'Table' or something equivalent.