Literate plain source is available!

Włodek Bzyl
matwb@halina.univ.gda.pl

Abstract

When introduced, literate programming was synonymous with WEB, a system for writing literate Pascal programs. Since then many different WEB's, aimed on particular programming language, were created. Each WEB is constructed of two separate parts. One is called TANGLE, the other WEAVE. Typically each part consists of just one program performing many tasks—expands macros, prettyprints code, generates and sorts an index etc. This makes adaptation of the existing WEB to another language extremely difficult.

Other approach to literate programming is presented by Norman Ramsey, the author of noweb. He designed and realized TANGLE/WEAVE pair as UNIX pipes. By extending and/or replacing parts of pipes with programs, written in AWK, ICON, FLEX, PERL, C, TEX, METAFONT, new tool could be created with relatively small effort. For example, with noweb, it was possible to create simple TEX-WEB system by writing AWK script and new TEX format. New system was applied for creation of literate plain source. Although the resulted file is principally plain.tex code interleaved with documentation, borrowed mainly from *The TEXbook*, it presents the whole code from the different perspective. The documentation is organized around the macros as they appear in the plain.tex rather than around the topics as in *The TEXbook*. This means that WEB source is *not* a user manual, even though many notions are explained there.

WEB for everyone?

WEB is a powerful tool.
The strength of literate programs lies in their ability to produce high-quality typeset documentation. The strength of literate programming lies in allowing you to write code where you are telling to humans what computer should do, instead of telling to computer what should be done. Obviously we are more efficient and precise when communicate with humans than computers. Thus literate programs are more easily written and maintained than ordinary ones.

WEB is a complex tool.
A literate program consists of pieces of documentation and named chunks containing code and references to other chunks. The pieces are arranged in order which helps to explain (and understand) the program as a whole. WEB consists of two processors: TANGLE and WEAVE. TANGLE is used for extracting a program by replacing one named chunk by its definition. The process of replacement is recursive. It continues until no named chunks remain. From one WEB source many programs could be extracted (this is achieved by presenting TANGLE with different chunks). WEAVE is used for converting WEB markup into TEX markup as described and coded in a separate format file which handles numerous typographical details of typeset documentation and provide support for typical tasks as cross-referencing, preparation of indexes, bibliography. Formats for long and short documents will be different. To typeset converted file you will need TEX running on your system. Errors can creep in TEX code. To get TEX code working with other formats could end with a short trip into TEX language (this will be needed if you plan your literate program to form a part of article, report, or book).

We learn by reading. Why not to read 'literate books'. There are a few such books already and more will appear. We learn by writing too. Why not to try one of finished tools. C/C++/FORTRAN programmer could try CWEB or FWEB. Programmers which write in other languages could check CTAN:/tex-archive/web directory. If your language is not on the list, or you are not able to express yourself with the style offered then you are welcome in the province of those who build their own tools. This territory is growing fast due to efforts of Norman Ramsey who established a base for creating simple and extensible literate tools.

Presenting a new tool TEX-WEB

N. Ramsey was the first who attempted to create generic — not aimed for particular language — literate tool. Such a tool would be useless because its generality. Key to usefulness of `noweb` lies in extendibility. The tasks for `TANGLE`/`WEAVE` were divided among stand-alone programs. To simplify tangling and weaving a front end was introduced. It performs a kind of lexical analysis of the source, task previously performed by both processors separately. The front end provided with `noweb` is called `markup` because it marks each line of source as line of text, as beginning/end of code/documentation, as definition/use of named chunks, etc.[1]

`WEAVE`

```
markup foo.tw | web2tex > foo.tex
markup foo.tw | awk -f web2tex > foo.tex
```

where the second command line is used on MSDOS systems. With `markup` as front end, `WEAVE` was build as AWK script `web2tex`[2] which reads marked source line by line and performs actions attached to line type. Most of the time it inserts a bunch of TEX macros.

The format `tweb.sty` provides support for cross references, indexes, and multicolumn output. There you find macros `\chapter`, `\[sub[sub]]section`, `\paragraph`,[3] `\printcontents`, `\title`.

Index macros are inserted by `web2tex`.

`TANGLE`

```
markup foo.tw | nt > foo.sty
markup foo.tw | nt -R'Chunk B' > foo.sty
markup foo.tw | mnt 'Chunk B' 'Chunk A'
```

Here we have several possibilities. We can extract code beginning from the chunk named '`<<*>>`', or from the '`Chunk A`' (see template file below). Finally, '`Chunk A`' and '`Chunk B`' could be simultaneously extracted to the files with the same names.

TEX

```
tex foo.tex
makeindex -s dnd.ist -o foo.dnd foo.ddx
makeindex -s und.ist -o foo.und foo.udx
makeindex -s chn.ist -o foo.chn foo.chk
tex foo.tex
```

[1] There is `unmarkup` which works in opposite way. I borrowed from `noweb` two more programs: `nt` (tangle) and `mnt` (multiple tangle).

[2] To allow tangling many files at once `web2tex` is actually shell wrapper for the AWK script.

[3] These macros should not be overused. Usually the chunk name alone is a better choice.

Indexes are sorted by `makeindex`. Three very short style files are provided due to different formatting of indexes. MSDOS `makeindx` breaks on large indexes.

Sample `Makefile`

To ease work with tools a simple `Makefile` is provided. Write `make` on the command line, press `Enter` key, and, assuming that only one TEX-WEB file named `foo.tw` resides in the current directory, the following lines will appear on a terminal

```
        Weaving:  make foo.tex
      Tangling:  make foo.sty
         Texing:  make foo.dvi
Making archive:  make archive
       Cleaning:  make clean or veryclean
```

Many different conventions are used where to store files in a file system. In the `Makefile` three variables are defined: `SCRIPTDIR` – place for `web2awk` and other scripts (defaults to `BIN`), `INDEXDIR` – place for index styles (defaults to `IDXSTY`), `MAKEINDEX` – the name of makeindex program (defaults to `makeindex`).

Template of TEX-WEB source

The structure of TEX-WEB file will be shown on the example.

File name: `foo.tw`

```
\title{foo.tw -- template file}
\printcontents % if you want TOC
@
The skeleton of the file foo.tw
<<*>>=
<<Chunk A>>
<<Chunk B>>
@
Documentation for Chunk A.
<<Chunk A>>=
TEX code / references to other chunks
@
Documentation for Chunk B.
<<Chunk B>>=
TEX code / references to other chunks
```

Documentation chunks begin with the line that starts with `@` followed by space or newline. Code chunks begin with `<<`Chunk name`>>=` on a line by itself. Chunks are terminated by the beginning of another chunk or end of file.

Making changes/updates

The change file mechanism is not needed in case of TEX language. Change files are used to incorporate system dependent code into source file, but

TEX code is already system independent. TEX code could be only 'format dependent'. Another feature of format file is that it evolves with time, yet some intermediate versions are used for preparation of books, articles etc. All these versions and configurations must be kept well organized, otherwise you are bound to be lost. The Revision Control System is the tool that assists with these tasks. With the RCS it is possible, with small overhead, to preserve *all revisions* which evolved from given text document, merge changes made by others, compare different versions, keep log of changes.

RCS

```
ci foo.tw                          check-in last version
co foo.tw                          check-out last version
co -rrev foo.tw
rlog foo.tw
rcsdiff -rrev foo.tw
rcsmerge -rlater_rev -rearlier_rev foo.tw
```

When the first command is executed `foo.tw` is stored in a *group file* (with default name `foo.tw,v` on UNIX machines, or `foo.tw%` on MSDOS) as new revision. For each deposited revision `ci` prompts for log message. The file `foo.tw` is deleted unless you say `ci -l foo.tw`. The message `ci error: no lock set by` *login* means that RCS was configured with 'strict locking feature' enabled. Locking prevents overlapping modifications if several users are working on the same file. This feature is disabled with `rcs -U foo.tw`; it is unnecessary if only owner of the file is expected to deposit revisions into it.

The next two commands are used to extract the latest, or the specified revision from the group file. `rlog` is used to print log messages. With `rcsdiff` different revisions are compared. `rcs foo.tw` compares the latest revision with the contents of the working file. Differences between files are found by the program `diff`; if you do not like the `diff` default output, change it by passing appropriate switches to `rscdiff`. The last command undoes the changes between revisions; the file `foo.tw` will be overwritten. `rcsmerge` incorporates changes between two revisions into the working file. Similar effect could be achieved with a stand-alone program called `merge`. If files to compare are `mine`, `older`, `yours` then with the command

```
merge mine older yours
```

`merge` tries to add to `main` the result of subtracting `older` from `yours`; if overlap occurs i.e. both files `mine` and `yours` have changes to the same segment of lines in `older` then `merge` delimits the alternatives with

```
<<<<<<< mine
lines in mine
=======
lines in yours
>>>>>>> yours
```

and writes above to `mine`. Now is up to you wich set of changes you adopt. `merge -p ...` sends the result of merging to the standard output.

To keep working directory uncluttered, all RCS files are usually stored in the subdirectory with the name `RCS`. RCS commands look first into this directory when searching for files.

Concluding remarks

It seems that TEX language constitutes a good starting point for exploring the idea of literate programming. The system is simple, because many features present in other `WEB`'s are not needed. The system is extensible, which means that it possible to try different styles and features. And finally, programs written in TEX are not too long — `plain.tex` is about 1000 lines of code — which means that you can print the documentation of real programs yourself and share it with others.

For those who are convinced by the analysis above I included literate source of `plain.tex` – a basic set of macros for TEX. Please read and enjoy.