Nuweb
A Simple Literate Programming Tool

Preston Briggs

preston@cs.rice.edu

In 1984, Knuth introduced the idea of *literate programming* and described a pair of tools to support the practise [1]. His approach was to combine Pascal code with TEX documentation to produce a new language, `WEB`, that offered programmers a superior approach to programming. He wrote several programs in `WEB`, including `weave` and `tangle`, the programs used to support literate programming. The idea was that a programmer wrote one document, the web file, that combined documentation (written in TEX [4]) with code (written in Pascal).

Running `tangle` on the web file would produce a complete Pascal program, ready for compilation by an ordinary Pascal compiler. The primary function of `tangle` is to allow the programmer to present elements of the program in any desired order, regardless of the restrictions imposed by the programming language. Thus, the programmer is free to present his program in a top-down fashion, bottom-up fashion, or whatever seems best in terms of promoting understanding and maintenance.

Running `weave` on the web file would produce a TEX file, ready to be processed by TEX. The resulting document included a variety of automatically generated indices and cross-references that made it much easier to navigate the code. Additionally, all of the code sections were automatically pretty printed, resulting in a quite impressive document.

Knuth also wrote the programs for TEX and METAFONT entirely in `WEB`, eventually publishing them in book form [3, 2]. These are probably the largest programs ever published in a readable form.

Inspired by Knuth's example, many people have experimented with `WEB`. Some people have even built web-like tools for their own favorite combinations of programming language and typesetting language. For example, `CWEB`, Knuth's current system of choice, works with a combination of C and TEX [6]. Another system, FunnelWeb, is independent of any programming language and only mildly dependent on TEX [8]. Inspired by the versatility of FunnelWeb and by the daunting size of its documentation, I decided to write my own, very simple, tool for literate programming.[1]

1 Nuweb

Nuweb works with any programming language and LaTEX [5]. I wanted to use LaTEX because it supports a multi-level sectioning scheme and has facilities for drawing figures. I wanted to be able to work with arbitrary programming languages because my friends and I write programs in many languages (and sometimes combinations of several languages), *e.g.,* C, Fortran, C++, yacc, lex, Scheme, assembly, Postscript, and so forth. The need to support arbitrary programming languages has many consequences:

No pretty printing Both `WEB` and `CWEB` are able to pretty print the code sections of their documents because they understand the language well enough to parse it. Since we want to use *any* language, we've got to abandon this feature.

No index of identifiers Because `WEB` knows about Pascal, it is able to construct an index of all the identifiers occurring in the code sections (filtering out keywords and the standard type identifiers). Unfor-

[1] There is another system similar to mine, written by Norman Ramsey, called *noweb* [7]. It perhaps suffers from being overly Unix-dependent and requiring several programs to use. On the other hand, its command syntax is very nice. In any case, nuweb certainly owes its name to his inspiration.

tunately, this isn't as easy in our case. We don't know what an identifiers looks like in each language and we certainly don't know all the keywords. (On the other hand, see the end of Section 1.3)

Of course, we've got to have some compensation for our losses or the whole idea would be a waste. Here are the advantages I can see:

Simplicity The majority of the commands in `WEB` are concerned with control of the automatic pretty printing. Since we don't pretty print, many commands are eliminated. A further set of commands is subsumed by LaTeX and may also be eliminated. As a result, our set of commands is reduced to only four members (explained in the next section). This simplicity is also reflected in the size of this tool, which is quite a bit smaller than the tools used with other approaches.

No pretty printing Everyone disagrees about how their code should look, so automatic formatting annoys many people. One approach is to provide ways to control the formatting. Our approach is simpler – we perform no automatic formatting and therefore allow the programmer complete control of code layout.

Control We also offer the programmer complete control of the layout of his output files (the files generated during tangling). Of course, this is essential for languages that are sensitive to layout; but it is also important in many practical situations, *e.g.,* debugging.

Speed Since nuweb doesn't do to much, the nuweb tool runs quickly. I combine the functions of `tangle` and `weave` into a single program that performs both functions at once.

Multiple file output The programmer may specify more than one output file in a single nuweb file. This is required when constructing programs in a combination of languages (say, Fortran and C). It's also an advantage when constructing very large programs that would require a lot of compile time.

This last point is very important. By allowing the creation of multiple output files, we avoid the need for monolithic programs. Thus we support the creation of very large programs by groups of people.

A further reduction in compilation time is achieved by first writing each output file to a temporary location, then comparing the temporary file with the old version of the file. If there is no difference, the temporary file can be deleted. If the files differ, the old version is deleted and the temporary file renamed. This approach works well in combination with `make` (or similar tools), since `make` will avoid recompiling untouched output files.

2 Writing Nuweb

The bulk of a nuweb file will be ordinary LaTeX. In fact, any LaTeX file can serve as input to nuweb and will be simply copied through unchanged to the `.tex` file – unless a nuweb command is discovered. All nuweb commands begin with an "at-sign" (`@`). Therefore, a file without at-signs will be copied unchanged. Nuweb commands are used to specify *output files,* define *macros,* and delimit *scraps.* These are the basic features of interest to the nuweb tool – all else is simply text to be copied to the `.tex` file.

2.1 The Major Commands

Files and macros are defined with the following commands:

`@o` *file-name flags scrap* Output a file. The file name is terminated by whitespace.

`@d` *macro-name scrap* Define a macro. The macro name is terminated by a return or the beginning of a scrap.

A specific file may be specified several times, with each definition being written out, one after the other, in the order they appear. The definitions of macros may be similarly divided.

2.1.1 Scraps

Scraps have specific begin markers and end markers to allow precise control over the contents and layout. Note that any amount of whitespace (including carriage returns) may appear between a name and the beginning of a scrap.

`@{`*anything*`@}` where the scrap body includes every character in *anything* – all the blanks, all the tabs, all the carriage returns.

Inside a scrap, we may invoke a macro.

`@<`*macro-name*`@>` Causes the macro *macro-name* to be expanded inline as the code is written out to a file. It is an error to specify recursive macro invocations.

Note that macro names may be abbreviated, either during invocation or definition. For example, it would be very tedious to have to repeatedly type the macro name

```
@d Check for terminating at-sequence and return name if found
```

Therefore, we provide a mechanism (stolen from Knuth) of indicating abbreviated names.

```
@d Check for terminating...
```

Basically, the programmer need only type enough characters to uniquely identify the macro name, followed by three periods. An abbreviation may even occur before the full version; nuweb simply preserves the longest version of a macro name. Note also that blanks and tabs are insignificant in a macro name; any string of them are replaced by a single blank.

When scraps are written to an output or `.tex` file, tabs are expanded into spaces by default. Currently, I assume tab stops are set every eight characters. Furthermore, when a macro is expanded in a scrap, the body of the macro is indented to match the indentation of the macro invocation. Therefore, care must be taken with languages (*e.g.,* Fortran) that are sensitive to indentation. These default behaviors may be changed for each output file (see below).

2.1.2 Flags

When defining an output file, the programmer has the option of using flags to control output of a particular file. The flags are intended to make life a little easier for programmers using certain languages. They introduce little language dependences; however, they do so only for a particular file. Thus it is still easy to mix languages within a single document. There are three "per-file" flags:

`-d` Forces the creation of `#line` directives in the output file. These are useful with C (and sometimes C++ and Fortran) on many Unix systems since they cause the compiler's error messages to refer to the web file rather than the output file. Similarly, they allow source debugging in terms of the web file.

`-i` Suppresses the indentation of macros. That is, when a macro is expanded in a scrap, it will *not* be indented to match the indentation of the macro invocation. This flag would seem most useful for Fortran programmers.

`-t` Suppresses expansion of tabs in the output file. This feature seems important when generating `make` files.

2.2 The Minor Commands

We have two very low-level utility commands that may appear anywhere in the web file.

`@@` Causes a single "at sign" to be copied into the output.

`@i` *file-name* Includes a file. Includes may be nested, though there is currently a limit of 10 levels. The file name should be complete (no extension will be appended) and should be terminated by a carriage return.

Finally, there are three commands used to create indices to the macro names, file definitions, and user-specified identifiers.

`@f` Create an index of file names.

`@m` Create an index of macro name.

`@u` Create an index of user-specified identifiers.

I usually put these in their own section in the LaTeX document.

Identifiers must be explicitely specified for inclusion in the `@u` index. By convention, each identifier is marked at the point of its definition; all references to each identifier (inside scraps) will be discovered automatically. To "mark" an identifier for inclusion in the index, we must mention it at the end of a scrap. For example,

```
@d a scrap @{
Let's pretend we're declaring the variables FOO and BAR
inside this scrap.
@| FOO BAR @}
```

I've used alphabetic identifiers in this example, but any string of characters (not including whitespace) will do. Therefore, it's possible to add index entries for things like `<<=` if desired. An identifier may be declared in more than one scrap.

In the generated index, each identifier appears with a list of all the scraps using and defining it, where the defining scraps are distinguished by underlining. Note that the identifier doesn't actually have to appear in the defining scrap; it just has to be in the list of definitions at the end of a scrap.

3 Running Nuweb

Nuweb is invoked using the following command:

`nuweb` *flags file-name...*

One or more files may be processed at a time. If a file name has no extension, `.w` will be appended. While a file name may specify a file in another directory, the resulting `.tex` file will always be created in the current directory. For example,

```
nuweb /foo/bar/quux
```

will take as input the file `/foo/bar/quux.w` and will create the file `quux.tex` in the current directory.

By default, nuweb performs both tangling and weaving at the same time. Normally, this is not a bottleneck in the compilation process; however, it's possible to achieve slightly faster throughput by avoiding one or another of the default functions using command-line flags. There are currently three possible flags:

`-t` Suppress generation of the `.tex` file.

`-o` Suppress generation of the output files.

`-c` Avoid testing output files for change before updating them.

Thus, the command

```
nuweb -to /foo/bar/quux
```

would simply scan the input and produce no output at all.

4 Restrictions and Extensions

Because nuweb is intended to be a simple tool, I've established a few restrictions. Over time, some of these may be eliminated; others seem fundamental.

- The scraps don't print well with older versions of TEX (pre 3.0). I'm not sure why and I'm not sure it's always true (I don't have a version to experiment with).
- The handling of errors is not completely ideal. In some cases, I simply warn of a problem and continue; in other cases I halt immediately. This behavior should be regularized.
- I warn about references to macros that haven't been defined, but don't halt. This seems most convenient for development, but may change in the future.
- File names and index entries should not contain any `@` signs.
- Macro names may be (almost) any well-formed TEX string. It makes sense to change fonts or use math mode; however, care should be taken to ensure matching braces, brackets, and dollar signs.
- Anything is allowed in the body of a scrap; however, very long scraps (horizontally or vertically) may not typeset well.

Very long scraps may be allowed to break across a page if declared with `@O` or `@D` (instead of `@o` or `@d`). This doesn't work very well as a default, since far too many short scraps will be broken across pages; however, as a user-controlled option, it seems very useful.

5 Obtaining a Copy

If you'd like a copy of nuweb, write to

preston@cs.rice.edu

and I'll be glad to e-mail the necessary pieces. In my (limited) experience, it seems to compile and run without change on the various Unix-based workstations.

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984.

[2] Donald E. Knuth. METAFONT: *The Program*. Computers & Typesetting. Addison-Wesley, 1986.

[3] Donald E. Knuth. *TEX: The Program*. Computers & Typesetting. Addison-Wesley, 1986.

[4] Donald E. Knuth. *The TEXbook*. Computers & Typesetting. Addison-Wesley, 1986.

[5] Leslie Lamport. LATEX*: A Document Preparation System*. Addison-Wesley, 1986.

[6] Silvio Levy and Donald E. Knuth. `CWEB` user manual: The CWEB system of structured documentation. Technical Report STAN-CS-83-977, Stanford University, October 1990. Available for anonymous ftp from `labrea.stanford.edu` in directory `pub/cweb`.

[7] Norman Ramsey. Literate-programming tools need not be complex. Submitted to IEEE Software, August 1992.

[8] Ross N. Williams. FunnelWeb user's manual, May 1992. Available for anonymous ftp from `sirius.itd.adelaide.edu.au` in directory `pub/funnelweb`.