Rnoweb: Literate Programming with and for R

Ross Ihaka

December, 15, 2011

1 Introduction

In a 1984 paper [4], Donald Knuth introduced the concept of *literate programming*. He argued that there needed to be a change in the way that programmers view the way they work.

> "Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do."

A number of literate programming systems have been developed over the years. This document describes *Rnoweb*, an R implementation of (a subset of) the *noweb* system developed by Norman Ramsey [5]. Introductory information on *noweb* can be found in Ramsey [5] and Johnson and Johnson [3].

Noweb has a number of advantages over other literate programming systems. First, it is extremely light-weight; there is almost no effort required to learn to use *noweb*. Second, noweb is not tied to any particular progamming language. This is important when tools are created from a mix of languages.

Rnoweb is implemented entirely in R and targetted mainly, but not exclusively, at writing code in R. The original version of *noweb* can be hard to get running on some platforms but, because it depends only on R, *Rnoweb* is easy to install and use whenever R is available.

2 Document Structure

The goal of literate programming is to write a single source document that can be used to produce (i) a printed guide to how the program works and is implemented and (ii) the source code for the actual program. The source document is written in plain text in a special markup format that indicates what is description and what is program code. The description is extracted and turned into a printable document in a process called *weaving* and the actual program source code is extracted in a process called *tangling*.

Because *Noweb* documents are written in plain text, almost any text editor can be used to create them. The *Emacs* editor provides particularly good support for working with *noweb* (and hence *Rnoweb*).

A *noweb* document consists of a sequence of code and documentation "chunks." These provide the code for a single software component, or for several such components, together with a full description of what that code does.[1]

A *code* chunk consists of computer code preceeded by "`<<...>>=`", where the ... provides a name for the chunk. The characters must occur on a line by themselves, with no leading space. As an example, consider the following chunk.

```
<<initialize>>=
count = 0
sum = 0
```

This chunk is called "initialise" and contains the two code lines "`count = 0`" and "`sum = 0`".

A documentation chunk consists of a block of text that starts with a line which has "`@ `" as its first two characters. The rest of the chunk provides a description of the code that follows it. The description is written in standard LaTeX but may include code fragments quoted with the delimiters "`[[`" and "`]]`".[2]

Chunks of either type are terminated by the start of a following chunk. (It is permissible to have a code chunk terminated by a code chunk and a documentation chunk terminated by a documentation chunk.)

When a code chunk is terminated by the start of a documentation chunk, the first line of the documentation chunk can be of the form

`@ %def` var_1 var_2 ...

where var_1, var_2 etc. are the names of variables defined by the preceding code chunk. These variable specifications are processed by *Rnoweb* and used to annotate where variables are defined and used.

The contents of a named code chunk can be included within another code chunk as indicated by the following example.

```
<<sum of a and b>>=
a + b
@

<<summation function>>=
sum.ab =
    function(a, b)
    <<sum of a and b>>
@ %def sum.ab
```

Here, the fragment `<<sum of a and b>>` within the `sub.ab` function is replaced by the contents of the "`sum of a and b`" fragment defined above it. Such inclusions can occur on a line by themselves or as part of a statement.

```
<<create c>>=
c = <<sum of a and b>>
@
```

[1] Any initial text which is not a code or documentation chunk is treated as plain LaTeX source.

[2] In *Rnoweb*, the these code fragments are processed with regular expressions. Becaue of this, the fragments themselve cannot contain the delimiters.

The ability to decompose complex blocks of code into smaller fragments makes it possible to write clear descriptions of the thinking behind the code. For an example of this, see Ihaka [2], or the literate source to *Rnoweb* itself [1].

When a *noweb* document is tangled, the dependencies between code chunks are computed and those chunks not included in others are output, *provided that their chunk names are suitable as file names.* When a document is woven, the result is a LaTeX document in a file whose name is the same as the original but with its extension changed from "`.Rnw`" to "`.tex`".

3 Creating and Processing Documents

Typically, *Rnoweb* documents are created in files with the extension "`.Rnw`". If the *Emacs ESS* package is installed this will mean that LaTeX editing support will be available in documentation chunks and R ediiting support will be available in code chunks.

To process the document you will need to install the Rnoweb package [1]. You can then use the `noweb` function in the package to carry out the weaving and tangling processess.

```
> noweb("mycode.Rnw")
```

Alternatively, you can use one of the following two scripts to do the job. The first script uses the *Rscript* front-end to R.

```
#!/usr/bin/env Rscript
library(Rnoweb)
for(arg in commandArgs(TRUE))
  noweb(arg)
```

The second uses a Unix/Linux shell script.

```
#!/bin/sh
for i in $*
do
  (echo "library(Rnoweb.R)"
   echo "noweb(\""$i"\")") | R --no-save --quiet --slave
done
```

Assuming that one of these scripts has been renamed *Rnoweb* and placed on the command path, a document can be both tangled and woven with the following command.

```
$ Rnoweb mycode.Rnw
```

The advantage of using these scripts is that the processing can then be included as part of a larger process managed by a Makefile.

However the file `mycode.Rnw` is processed, the result is to produce a LaTeX file called called `mycode.tex` together with files corresponding to unreferenced code chunks. The former can be processed with LaTeX (or pdfLaTeX) and the latter files read into R.

When LaTeX is run on a file, it is necessary to have the correct set of LaTeX macros available. In the case of *noweb* files, the correct set of macros is contained in the file `noweb.sty`, available from the `CTAN` archive site [6].

4 Cross-Referencing and Indices

In woven *noweb* files, code chunks are labelled by the page number and, if there are multiple chunks that appear on the same page, a lower case roman letter (a, b, ..., z). The chunk labelled *3a* is the first chunk on page 3, that labelled *3b* is the second on page 3 etc.

As noted in section 2, *noweb* makes it possible to specify the locations where variables are defined.[3] The locations where variables are defined and used are indicated underneath code chunks in the woven document.

It is possible to produce indices for both chunks and variables by using the following LaTeX macros.

```
\nowebchunks
\nowebindex
```

The first of these produces the chunk index and the second the variable index. For larger pieces of software, these indices can be invaluable.

5 A Small Example

The following lines show a very simple *Rnoweb* file that describes a function that computes the factorial function. The function is defined in *closure* form so that renaming will not interfer with its ability to call itself.

```
\documentclass{article}
\usepackage{noweb}
\begin{document}
\title{A Recursive Factorial Function}
\author{Ross Ihaka}
\date{}

\section{Introduction}
\label{sec:introduction}

This document describes an R function for computing the
factorial function recursively.  The function is
defined in \emph{closure} form so that renaming will
not interfer with its ability to call itself.

The function is defined as the value defined with a
local environment created by a call to the [[local]]
function.
<<factorial.R>>=
factorial =
    local({
        <<factorial function>>
    })
@
```

[3] *Noweb* can automatically detect variable definitions for some languages. In *Rnoweb*, the annotation *must* be done manually.

```
Within this local environment the factorial function is
defined in a recursive fashion.  It refers to itself
using its name in this local environment. This means
that if the top-level function is renamed, it will have
no effect on how the function works.
<<factorial function>>=
fact =
    function(n)
    if (n <= 1) 1 else n * fact(n - 1)
@ %def fact

The use of local environments like this is a
\emph{very} useful and very-much under-utilised R
programming technique.

\end{document}
```

And that's really all there is to it. The rest is really just a matter of being creative with the mechanism that *noweb* provides.

References

[1] Ihaka, R. (2011). "Literate Programming with and for R." `http://www.stat.auckland.ac.nz/~ihaka/software/Rnoweb`.

[2] Ihaka, R. (2011). "R Session Scripting." `http://www.stat.auckland.ac.nz/~ihaka/software/R-script`.

[3] Johnson, A. L. and Johnson, B. C. (October 1997). "Literate programming using `noweb`." *Linux Journal*, pages 64–69.

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

[5] Ramsey, N. (1994). "Literate Programming Simplified." *IEEE Software*, **11**(5): 97–105.

[6] Ramsey, N. (2006). "LaTeX macro support for `noweb` 2.11b." `http://mirrors.ctan.org/web/noweb/src/tex/noweb.sty`.