Literate Programming

Jan van Eijck

The virtues of literate programming are extolled in [Knu92]. This instruction explains the concept by example.

This LaTeX file is called `LP.tex`. It contains parts like this:

```
module LP where

import Data.List
```

The frame is made by means of the environment `\bc` ... `\ec`. The program text is rendered verbatim.

To turn the file `LP.tex` into a new file `LP.lhs` with the same number of lines, where each line containing `\bc\begin{verbatim}` is turned into a line containing `\begin{code}`, and each line containing `\end{verbatim}\ec` is turned into a line containing `\end{code}`, we use the following script, called `lhs`.

The proper way to call the script is with `lhs LP`. This will create the file `LP.lhs`.

```
cat < $1'.tex' | sed -f /home/jve/bin/lhsfilter.sed > $1'.lhs'
```

The script uses a `sed` filter called `lhsfilter.sed`. Here it is:

```
s/\%\#/\#/
s/\\bc\\begin[{]verbatim[}]/\\begin\{code\}/g
s/\\end[{]verbatim[}]\\ec/\\end\{code\}/g
```

Please consult the `sed` manual if you want to understand the details of this. (But it is also OK if you decide not to bother.)

Now suppose we want to define a piece of program code. Here is a definition of the general form of a while loop with a single parameter:

```
while1 :: (a -> Bool) -> (a -> a) -> a -> a
while1 p f x
  | p x       = while1 p f (f x)
  | otherwise = x
```

Another way to express this is in terms of the built-in Haskell function `until`:

```
neg :: (a -> Bool) -> (a -> Bool)
neg p = \x -> not (p x)

while1 = until . neg
```

An example of the use of this:

```
g = while1 (\x -> even x) (\x -> x ‘div‘ 2)
```

This can be written more compactly as:

```
g’ = while1 even (‘div‘ 2)
```

Note that the second definition of `while1` should not end up in the Haskell code base, for this would lead to a "repeated definition" error. Therefore, this piece of code is wrapped up differently, in an `\bp` ... `\ep` environment with a slightly different LaTeX definition.

References

[Knu92] D.E. Knuth. *Literate Programming.* CSLI Lecture Notes, no. 27. CSLI, Stanford, 1992.