xfor v1.03: Reimplementation of `\@for` to allow premature termination of the loop

Nicola L.C. Talbot

School of Computing Sciences
University of East Anglia
Norwich. Norfolk
NR4 7TJ. United Kingdom.
http://theoval.cmp.uea.ac.uk/~nlct/

21st August 2007

Contents

1 Introduction

The xfor package redefines `\@for` so that the loop can be prematurely terminated, akin to C/Java's `break` statement except that the loop will terminate at the *end*

`\@for` of the current iteration. The syntax for `\@for` remains the same:

$$\texttt{\textbackslash@for}\langle cmd\rangle\texttt{:=}\langle list\rangle\texttt{\textbackslash do\{}\langle body\rangle\texttt{\}}$$

where $\langle cmd\rangle$ is a command name that is assigned to the current element of the list given by $\langle list\rangle$ at each iteration.

To terminate the loop at the end of the current iteration, use the command

`\@endfortrue` `\@endfortrue`. This command may be used anywhere in $\langle body\rangle$, but will only take effect at the end of the current iteration. The remainder of the list is stored

`\@forremainder` in `\@forremainder`. You can test whether the loop was prematurely terminated

`\if@endfor` using the conditional `\if@endfor`.

As from version 1.02, there is also provision for looking ahead. At each iteration

`\@xfor@nextelement` in the loop, the next element is stored in `\@xfor@nextelement`. On the last

iteration, this value will be `\@nil`, and so can be checked using

```
\ifx\@xfor@nextelement\@nnil
% last iteration
\else
% not last iteration
\fi
```

1.1   Example (ordered insertion)

Suppose you have list of sorted numbers stored in the command `\mylist`, e.g.:

```
\def\mylist{1,3,5,7,8,12,15,20}
```

and you want to insert a new value given by the command `\newval`, e.g.

```
\def\newval{11}
```

in the correct order. You can use `\@for` to iterate through each element in the sorted list, testing the value against the new value to be inserted. Once the new value has been inserted, the loop can be terminated, and any remaining elements can be appended to the new list. The following defines the command `\insertinto{`⟨*new val*⟩`}{`⟨*list*⟩`}` which uses this method:

```
\newcommand{\insertinto}[2]{%
\def\nlst{}%
\@for\n:=#2\do{%
% store new list in \toks@
\expandafter\toks@\expandafter{\nlst}%
% test current value against new value
\ifnum\n>#1\relax
  \edef\newstuff{\number#1,\n}%
  % end for loop at the end of this iteration
  \@endfortrue
\else
  \edef\newstuff{\n}%
\fi
% append new stuff to new list
\ifx\nlst\@empty
  \edef\nlst{\newstuff}%
\else
  \edef\nlst{\the\toks@,\newstuff}%
\fi
}%
% check to see if for loop was prematurely terminated
\if@endfor
 % loop may have been terminated during final iteration, in
 % which case \@forremainder is empty.
 \ifx\@forremainder\@empty
   % do nothing
 \else
   % loop prematurely ended, append remainder of original list
   % to new list
   \expandafter\toks@\expandafter{\nlst}%
   \edef\nlst{\the\toks@,\@forremainder}%
```

```
  \fi
 \else
  % wasn't prematurely terminated, so new value hasn't been added
  % add now.
   \expandafter\toks@\expandafter{\nlst}%
   \ifx\nlst\@empty
     \edef\nlst{\number#1}%
   \else
     \edef\nlst{\the\toks@,\number#1}%
   \fi
 \fi
 \let#2=\nlst
 }
```

The \insertinto macro can then be used as follows:

```
\def\mylist{1,2,5,9,12,15,18,20}%
\def\newval{11}%
Original list: \mylist. New value: \newval.

\insertinto{\newval}{\mylist}
New list: \mylist.
```

1.2   Example (numerical insertion sort)

Care needs to be taken when nesting \@for-loops. Suppose you have a list of unsorted numbers, say

```
\def\mylist{4,2,7,1,10,11,20,15}
```

and you want to sort the list in numerical order using an insertion sort method. To do this, a macro needs to be defined which iterates through each element in the unordered list, and the element is then inserted into an ordered list. The previous example described the macro \insertinto which does this, but this results in nested \@for commands. The \insertinto command will need to be grouped to avoid errors:

```
\newcommand*{\insertionsort}[1]{%
\def\sortedlist{}%
\@for\val:=#1\do{{\insertinto{\val}{\sortedlist}}}%
\let#1=\sortedlist
}
```

This won't work with the definition of \insertinto as given in the previous section, as the grouping causes the definition of the sorted list to be localised to that group. Replacing

```
\let#2=\nlst
```

with

```
\global\let#2=\nlst
```

at the end of the definition of \insertinto will fix that.

1.3 Example (looking ahead)

This example checks the next value to determine if the loop is on the last iteration, if it is, it does nothing, otherwise it does a semi-colon:

```
\makeatletter
\def\mylist{1,2,3,4,5}%
\@for\val:=\mylist\do{\val
\ifx\@xfor@nextelement\@nnil \else ;\fi}
\makeatother
```

which produces: 1;2;3;4;5

2 The Code

Declare package:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{xfor}[2007/08/21 v1.03 (NLCT)]
```

Define a switch to determine if the for loop should be terminated:

`\if@endfor`

```
\newif\if@endfor
```

Redefine `\@for`, so that it resets `\if@endfor` and `\@forremainder`:

`\@for`

```
\long\def\@for#1:=#2\do#3{%
\@endforfalse
\def\@forremainder{}%
\expandafter\def\expandafter\@fortmp\expandafter{#2}%
\ifx\@fortmp\@empty
\else
  \expandafter\@forloop#2,\@nil,\@nil\@@#1{#3}%
\fi
}
```

Redefine `\@fornoop` to be a long command:

`\@fornoop`

```
\long\def\@fornoop#1\@@#2#3{}
```

Redefine `\@forloop` to check for `\if@endfor`:

`\@forloop`

```
\long\def\@forloop#1,#2,#3\@@#4#5{%
\def#4{#1}%
\ifx#4\@nnil
\else
 \@xfor@storenext#2,#3\@@%
 #5%
 \if@endfor
   \@iforgatherrest#2,#3%
 \else
   \def#4{#2}%
```

4

```
        \ifx#4\@nnil
        \else
          \@xfor@storenext#3,\@nil\@@%
          #5%
          \if@endfor
            \@iforgatherrest#3%
          \else
            \@iforloop#3\@@#4{#5}%
          \fi
        \fi
     \fi
   \fi
   }
```

\@xfor@storenext Stores first argument in **\@xfor@nextelement**.

```
\def\@xfor@storenext#1,#2\@@{%
\def\@xfor@nextelement{#1}%
}
```

\@forgatherrest Get remainder of list (stores in **\@forremainder**):

```
\long\def\@forgatherrest#1,\@nil,\@nil{\def\@forremainder{#1}}
```

\@iforgatherrest As above, but there may not be anything before **\@nil**:

```
\long\def\@iforgatherrest#1\@nil,\@nil{%
\def\@fortmp{#1}%
\ifx\@fortmp\@empty
 \def\@forremainder{}%
\else
 \@forgatherrest#1\@nil,\@nil
\fi
}
```

Redefine **\@iforloop** to check **\if@endfor**:

\@iforloop

```
\long\def\@iforloop#1,#2\@@#3#4{%
\def#3{#1}%
\ifx#3\@nnil
 \let\@ifornext\@fornoop
\else
 \@xfor@storenext#2,\@nil\@@%
 #4\relax
 \if@endfor
  \@iforgatherrest#2\relax
  \let\@ifornext\@fornoop
 \else
  \let\@ifornext\@iforloop
 \fi
\fi
\@ifornext#2\@@#3{#4}%
}
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the definition; numbers in roman refer to the pages where the entry is used.