Algorithms

Peter Williams
Peter.Williams@dsto.defence.gov.au

7 April 1996

Contents

List of Algorithms

1 Introduction

This package provides two environments, `algorithmic` and `algorithm`, which are designed to be used together but may be used separately. The `algorithmic` environment provides an environment for describing algorithms and the `algorithm` environment provides a "float" wrapper for algorithms (implemented using `algorithmic` or some other method at the author's option). The reason that two environments are provided is to allow the author maximum flexibility.

2 The `algorithmic` Environment

Within an `algorithmic` a number of commands for typesetting popular algorithmic constructs are available. In general, the commands provided can be arbitrarily nested to describe quite complex algorithms. An optional argument to the `\begin{algorithmic}` statement can be used to turn on line numbering by giving a positive integer indicating the required frequency of line numbering. For example, `\begin{algorithmic}[5]` would cause every fifth line to be numbered.

2.1 The Simple Statement

The simple statement takes the form

```
\STATE <text>
```

and is used for simple statements, e.g.

```
\begin{algorithmic}
\STATE $S \leftarrow 0$
\end{algorithmic}
```

would produce

$S \leftarrow 0$

and with line numbering selected for every line using

```
\begin{algorithmic}[1]
\STATE $S \leftarrow 0$
\end{algorithmic}
```

would produce

1: $S \leftarrow 0$

For users of earlier versions of `algorithmic` this construct is a cause of an incompatibility. In the earlier version, instead of starting simple statements with the `\STATE` command, simple statements were entered as free text and terminated with `\\` command. Unfortunately, this simpler method failed to survive the modifications necessary for statement numbering. However, the `\\` command can still be used to force a line break within a simple statement.

2.2 The *if-then-else* Construct

The *if-then-else* construct takes the forms.

```
\IF{<condition>} <text> \ENDIF
\IF{<condition>} <text1> \ELSE <text2> \ENDIF
\IF{<condition1>} <text1> \ELSIF{<condition2>} <text2> \ELSE <text3> \ENDIF
```

In the third of these forms there is no limit placed on the number of `\ELSIF{<C>}` that may be used. For example,

```
\begin{algorithmic}
\IF{some condition is true}
\STATE do some processing
\ELSIF{some other condition is true}
\STATE do some different processing
\ELSIF{some even more bizarre condition is met}
\STATE do something else
\ELSE
\STATE do the default actions
\ENDIF
\end{algorithmic}
```

would produce

if some condition is true **then**
  do some processing
else if some other condition is true **then**
  do some different processing
else if some even more bizarre condition is met **then**
  do something else
else
  do the default actions
end if

with appropriate indentations.

2.3 The *for* Loop

The *for* loop takes the forms.

```
\FOR{<condition>} <text> \ENDFOR
\FORALL{<condition>} <text> \ENDFOR
```

For example,

```
\begin{algorithmic}
\FOR{$i=0$ to $10$}
\STATE carry out some processing
\ENDFOR
\end{algorithmic}
```

produces

for $i = 0$ to 10 **do**

carry out some processing
end for

and

```
\begin{algorithmic}[1]
\FORALL{$i$ such that $0\leq i\leq 10$}
\STATE carry out some processing
\ENDFOR
\end{algorithmic}
```

produces

1: **for all** i such that $0 \leq i \leq 10$ **do**
2: carry out some processing
3: **end for**

2.4 The *while* Loop

The *while* loop takes the form.

```
\WHILE{<condition>} <text> \ENDWHILE
```

For example,

```
\begin{algorithmic}
\WHILE{some condition holds}
\STATE carry out some processing
\ENDWHILE
\end{algorithmic}
```

produces

while some condition holds **do**
carry out some processing
end while

2.5 The *repeat-until* Loop

The *repeat-until* loop takes the form.

```
\REPEAT <text> \UNTIL{<condition>}
```

For example,

```
\begin{algorithmic}
\REPEAT
\STATE carry out some processing
\UNTIL{some condition is met}
\end{algorithmic}
```

produces

repeat
carry out some processing
until some condition is met

2.6 The Infinite Loop

The infinite loop takes the form.

```
\LOOP <text> \ENDLOOP
```

For example,

```
\begin{algorithmic}
\LOOP
\STATE this processing will be repeated forever
\ENDLOOP
\end{algorithmic}
```

produces

loop
  this processing will be repeated forever
end loop

2.7 The Precondition

The precondition (that must be met if an algorithm is to correctly execute) takes the form.

```
\REQUIRE <text>
```

For example,

```
\begin{algorithmic}
\REQUIRE $x \neq 0$ and $n \geq 0$
\end{algorithmic}
```

produces

Require: $x \neq 0$ and $n \geq 0$

2.8 The Postcondition

The postcondition (that must be met after an algorithm has correctly executed) takes the form.

```
\ENSURE <text>
```

For example,

```
\begin{algorithmic}
\ENSURE $x \neq 0$ and $n \geq 0$
\end{algorithmic}
```

produces

Ensure: $x \neq 0$ and $n \geq 0$

2.9 Comments

Comments may be inserted at most points in an algorithm using the form.

```
\COMMENT{<text>}
```

For example,

```
\begin{algorithmic}
\STATE do something \COMMENT{this is a comment}
\end{algorithmic}
```

produces

do something {this is a comment}

Because the mechanisms used to build the various algorithmic structures make it difficult to use the above mechanism for placing comments at the end of the first line of a construct, the commands `\IF`, `\ELSIF`, `\ELSE`, `\WHILE`, `\FOR`, `\FORALL`, `\REPEAT` and `\LOOP` all take an optional argument which will be treated as a comment to be placed at the end of the line on which they appear. For example,

repeat {this is comment number one}
  if condition one is met **then** {this is comment number two}
    do something
  else if condition two is met **then** {this is comment number three}
    do something else
  else {this is comment number four}
    do nothing
  end if
until hell freezes over

2.10 An Example

The following example demonstrates the use of the `algorithmic` environment to describe a complete algorithm. The following input

```
\begin{algorithmic}
\REQUIRE $n \geq 0$
\ENSURE $y = x^n$
\STATE $y \Leftarrow 1$
\STATE $X \Leftarrow x$
\STATE $N \Leftarrow n$
\WHILE{$N \neq 0$}
\IF{$N$ is even}
\STATE $X \Leftarrow X \times X$
\STATE $N \Leftarrow N / 2$
\ELSE[$N$ is odd]
\STATE $y \Leftarrow y \times X$
\STATE $N \Leftarrow N - 1$
```

```
\ENDIF
\ENDWHILE
\end{algorithmic}
```

will produce

Require: $n \geq 0$
Ensure: $y = x^n$
  $y \Leftarrow 1$
  $X \Leftarrow x$
  $N \Leftarrow n$
  while $N \neq 0$ **do**
    if N is even **then**
      $X \Leftarrow X \times X$
      $N \Leftarrow N/2$
    else $\{N$ is odd$\}$
      $y \Leftarrow y \times X$
      $N \Leftarrow N - 1$
    end if
  end while

which is an algorithm for finding the value of a number taken to a non-negative power.

2.11 Options

There is a single option, `noend` that may be invoked when the `algorithmic` package is loaded. With this option invoked the *end* statements are omitted in the output. This allows space to be saved in the output document when this is an issue.

2.12 Customization

In order to facilitate the use of this package with foreign languages, all of the words in the output are produced via redefinable macro commands. The default definitions of these macros are:

```
\newcommand{\algorithmicrequire}{\textbf{Require:}}
\newcommand{\algorithmicensure}{\textbf{Ensure:}}
\newcommand{\algorithmicend}{\textbf{end}}
\newcommand{\algorithmicif}{\textbf{if}}
\newcommand{\algorithmicthen}{\textbf{then}}
\newcommand{\algorithmicelse}{\textbf{else}}
\newcommand{\algorithmicelsif}{\algorithmicelse\ \algorithmicif}
\newcommand{\algorithmicendif}{\algorithmicend\ \algorithmicif}
\newcommand{\algorithmicfor}{\textbf{for}}
\newcommand{\algorithmicforall}{\textbf{for all}}
\newcommand{\algorithmicdo}{\textbf{do}}
```

```
\newcommand{\algorithmicendfor}{\algorithmicend\ \algorithmicfor}
\newcommand{\algorithmicwhile}{\textbf{while}}
\newcommand{\algorithmicendwhile}{\algorithmicend\ \algorithmicwhile}
\newcommand{\algorithmicloop}{\textbf{loop}}
\newcommand{\algorithmicendloop}{\algorithmicend\ \algorithmicloop}
\newcommand{\algorithmicrepeat}{\textbf{repeat}}
\newcommand{\algorithmicuntil}{\textbf{until}}
```

In addition, the formatting of comments is implemented via a single argument command macro which may also be redefined. The default definition is

```
\newcommand{\algorithmiccomment}[1]{\{#1\}}
```

3 The `algorithm` Environment

3.1 General

Algorithm 1 Calculate $y = x^n$

Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$
  $y \Leftarrow 1$
  if $n < 0$ **then**
    $X \Leftarrow 1/x$
    $N \Leftarrow -n$
  else
    $X \Leftarrow x$
    $N \Leftarrow n$
  end if
  while $N \neq 0$ **do**
    if N is even **then**
      $X \Leftarrow X \times X$
      $N \Leftarrow N/2$
    else $\{N$ is odd$\}$
      $y \Leftarrow y \times X$
      $N \Leftarrow N - 1$
    end if
  end while

When placed within the text without being encapsulated in a floating environment `algorithmic` environments may be split over a page boundary greatly detracting from their appearance. In addition, it is useful to have algorithms numbered for reference and for lists of algorithms to be appended to the list of contents. The `algorithm` environment is meant to address these concerns by providing a floating environment for algorithms. For example, the input text

```
\begin{algorithm}
\caption{Calculate $y = x^n$}
\label{alg1}
\begin{algorithmic}
\REQUIRE $n \geq 0 \vee x \neq 0$
\ENSURE $y = x^n$
\STATE $y \Leftarrow 1$
\IF{$n < 0$}
\STATE $X \Leftarrow 1 / x$
\STATE $N \Leftarrow -n$
\ELSE
\STATE $X \Leftarrow x$
\STATE $N \Leftarrow n$
\ENDIF
\WHILE{$N \neq 0$}
\IF{$N$ is even}
\STATE $X \Leftarrow X \times X$
\STATE $N \Leftarrow N / 2$
\ELSE[$N$ is odd]
\STATE $y \Leftarrow y \times X$
\STATE $N \Leftarrow N - 1$
\ENDIF
\ENDWHILE
\end{algorithmic}
\end{algorithm}
```

produces Algorithm 1 which is a slightly modified version of the earlier algorithm for determining the value of a number taken to an integer power. In this case, provided the power may be negative provided the number is not zero.

The command `\listofalgorithms` may be used to produce a list of algorithms as part of the table contents as shown at the beginning of this document. An auxiliary file with a suffix of `.loa` is produced when this feature is used.

3.2 Options

The appearance of the typeset algorithm may be changed by use of the options: `plain`, `boxed` or `ruled` during the loading of the `algorithm` package. The default option is `ruled`.

The numbering of algorithms can be influenced by providing the name of the document component within which numbering should be recommenced. The legal values for this option are: `part`, `chapter`, `section`, `subsection`, `subsubsection` or `nothing`. The default value is `nothing` which causes algorithms to be numbered sequentially throughout the document.

3.3 Customization

In order to facilitate the use of this package with foreign languages, methods have been provided to facilitate the necessary modifications.

The title used in the caption within `algorithm` environment can be set by use of the standard `\floatname` command which is provided as part of the `float` package which was used to implement this package. For example,

```
\floatname{algorithm}{Procedure}
```

would cause **Procedure** to be used instead of **Algorithm** within the caption of algorithms.

In a manner analogous to that available for the built in floating environments, the heading used for the list of algorithms may be changed by redefining the command `listalgorithmname`. The default definition for this command is

```
\newcommand{\listalgorithmname}{List of Algorithms}
```