Pseudocode: A \LaTeX Style File for Displaying Algorithms

D.L. Kreher
Department of Mathematical Sciences
Michigan Technological University
Houghton, MI 49931
kreher@mtu.edu

and

D.R. Stinson
Department of Combinatorics and Optimization
University of Waterloo
Waterloo ON, N2L 3G1
dstinson@uwaterloo.ca

September 5, 1999

1 Introduction

This paper describes a \LaTeX environment named pseudocode that can be used for describing algorithms in pseudocode form. This is the style used in our textbook Combinatorial Algorithms: Generation, Enumeration and Search [2]. The style file pseudocode.sty is available for free downloading from the web page http://www.math.mtu.edu/~kreher/cages.html.

This package is quite easy to use, and allows algorithms to be described in a \LaTeX document using a natural Pascal-like syntax. In the remaining sections of this note, we describe how to use the pseudocode environment and present some examples. Readers familiar with \LaTeX (see [3]) should be able to easily customize the style file to include additional desired features.

The pseudocode environment requires the fancybox package by Timothy Van Zandt. This package is described in Section 10.1.3 of [1]. Other environments for describing algorithms include alg, algorithmic, newalg
and program. These style files, as well as fancybox, are all available from the CTAN web site http://www.ctan.org/

2 The pseudocode Environment

Within the pseudocode environment, a number of commands for popular algorithmic constructs are available. In general, the commands provided can be nested to describe quite complex algorithms.

The pseudocode environment is invoked as follows:

\begin{pseudocode}{<Name>}{<Parameters>}
  pseudocode constructs
\end{pseudocode}

The argument <Name> is the name of the algorithm, and <Parameters> is a list of parameters for the algorithm. For example, the commands

\begin{pseudocode}{CelsiusToFahrenheit}{c}
  f \texttt{ \GETS (9c/5) + 32} \\
  \RETURN f
\end{pseudocode}

produce the following output when included in a \LaTeX{} document:

\textbf{Algorithm 2.1: CelsiusToFahrenheit}(c)

\begin{verbatim}
  f \leftarrow 9c/5 + 32
  \RETURN (f)
\end{verbatim}

Notice that the command \texttt{\GETS} produces a left arrow, which we use to indicate an assignment of a variable. The user could use instead some other symbol, if desired. For example, \texttt{\GETS} could be replaced by $\triangleq$, as is done in the “C” programming language.

2.1 The \texttt{begin-end} Construct

To form compound statements from simple statements, the \texttt{begin-end} construct is used as follows:

\begin{verbatim}
\BEGIN
  some statement\\
  another statement\\
  yet another statement
\END
\end{verbatim}
This generates the following:

\begin{verbatim}
\{ some statement
another statement
yet another statement
\end{verbatim}

The effect of this construct is to group a collection of statements using a left brace bracket of the appropriate size.

In the sections that follow we will use the notation <stmt> to indicate a simple statement or a compound statement. Note that the contents of statements are typeset in math mode. Therefore, non-math mode text must be enclosed in an \texttt{\textbackslash{bbox}}.

Observe that the double backslash \ plays the same role as the semicolon in Pascal, i.e., it is used to separate statements, and should never appear before \texttt{\textend{verbatim}.}

\section{The if-then-else Construct}

The if-then-else construct takes various forms, such as the following:

\begin{verbatim}
\texttt{\textbackslash{IF} <condition> \textbackslash{THEN} <stmt>}
\texttt{\textbackslash{ELSE} <stmt>}
\texttt{\textbackslash{ELSEIF} <stmt> \textbackslash{THEN} <stmt>}
\end{verbatim}

Note that there is no limit placed on the number of ELSEIFs that may be used in an if-then-else construct. For example, the commands:

\begin{verbatim}
\texttt{\textbackslash{IF} some condition is true}
\texttt{\textbackslash{THEN}}
\texttt{\textbackslash{BEGIN}}
\texttt{\textbackslash{some statement}}\texttt{\textbackslash{\}}
\texttt{\textbackslash{another statement}}\texttt{\}}
\texttt{\textbackslash{yet another statement}}\texttt{\}}
\texttt{\textbackslash{END}}
\texttt{\textbackslash{ELSEIF} some other condition is true}
\texttt{\textbackslash{THEN}}
\texttt{\textbackslash{BEGIN}}
\texttt{\textbackslash{some statement}}\texttt{\textbackslash{\}}
\texttt{\textbackslash{another statement}}\texttt{\}}
\texttt{\textbackslash{yet another statement}}\texttt{\}}
\texttt{\textbackslash{END}}
\texttt{\textbackslash{ELSEIF} some even more bizarre condition is met}
\texttt{\textbackslash{THEN}}
\texttt{\textbackslash{do something else}}
\texttt{\textbackslash{ELSE}}
\texttt{\textbackslash{do the default actions}}
\end{verbatim}
would produce the following output:

```plaintext
if some condition is true
    some statement
then
    another statement
    yet another statement
else if some other condition is true
    some statement
then
    another statement
    yet another statement
else if some even more bizarre condition is met
then do something else
else do the default actions
```

### 2.3 The for Loop

The `for` loop takes the following forms:

\`
\FOR <var> \GETS <lower> \TO <upper> \DO <stmt>
\FOR <var> \GETS <upper> \DOWNTO <lower> \DO <stmt>
\FOREACH <condition> \DO <stmt>
```

For example,

\`
\FOR i \GETS 0 \TO 10 \DO
    some processing
```

produces

for \( i \leftarrow 0 \text{ to } 10 \)
    do some processing

and

\`
\FOREACH x \in \mathcal{S} \DO
    some processing
```

produces

for each \( x \in S \)
    do some processing

### 2.4 The while Loop

The `while` loop takes the following form:

\`
\WHILE <condition> \DO <stmt>
```

For example,
\WHILE some condition holds \DO
   some processing
\ENDWHILE

produces

\textbf{while} some condition holds
\textbf{do} some processing

\subsection{The repeat-until Loop}

The \textit{repeat-until} loop takes the following form:

\begin{verbatim}
\REPEAT \textit{stmt} \UNTIL \textit{condition}
\end{verbatim}

For example,

\begin{verbatim}
\REPEAT
   some processing
\UNTIL some condition is met
\end{verbatim}

produces

\begin{verbatim}
\textbf{repeat}
   some processing
\textbf{until} some condition is met
\end{verbatim}

\subsection{Main Programs and Procedures}

We can describe a main program that calls one (or more) procedures as follows:

\begin{verbatim}
\begin{pseudocode}<Name>{<Parameters>}
\PROCEDURE{<ProcedureName>{<ProcedureParameters>}
   some stuff
\ENDPROCEDURE
\MAIN
   some stuff\\
\CALL{<ProcedureName>{<ActualParameters>}\\
   more stuff
\ENDMAIN
\end{pseudocode}
\end{verbatim}

Here is a simple example to illustrate the use of a main program calling a procedure. The commands
\begin{pseudocode}\{TemperatureTable\}\{lower, upper\}
\PROCEDURE\{CelsiusToFahrenheit\}\{c\}
\quad f \ \GETS \ 9c/5 + 32
\quad \RETURN\{f\}
\ENDPRECEDURE
\MAIN
x \ \GETS \ \text{lower} \ \\
\WHILE \ x \ \leq \ \text{upper} \ \DO
\BEGIN
\OUTPUT\{x, \ \CALL\{CelsiusToFahrenheit\}\{x\}\}\\\
x \ \GETS \ x + 1
\END
\ENDDO
\ENDMAIN
\end{pseudocode}

produce the following output:

**Algorithm 2.8:** \textsc{TemperatureTable}(lower, upper)

\begin{verbatim}
procedure CelsiusToFahrenheit(c)
    f := 9c/5 + 32
return \{f\}
end
main
x := lower
while x <= upper
    do \{output \{x, CelsiusToFahrenheit(x)\}
        x := x + 1
\end
\end\end{verbatim}

2.7 Comments

A comment statement may be inserted in an algorithm using the following command:

\COMMENT{<stmt>}

For example, the commands

\begin{verbatim}
A \ \GETS \ B\\
\COMMENT{Now increment the value of A}\\\nA \ \GETS \ A+1
\end{verbatim}

produce the output
\( A \leftarrow B \)

**comment:** Now increment the value of \( A \)

\( A \leftarrow A + 1 \)

Note that comments are assumed to be text. Thus, in order to include mathematical expressions in a comment, math mode must be used explicitly.

### 2.8 Other Predefined Keywords

Several other predefined keywords are available. We summarize their usage in Table 1.

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{{list of variables}}</td>
<td>local list of variables</td>
</tr>
<tr>
<td>\texttt{GLOBAL{list of variables}}</td>
<td>global list of variables</td>
</tr>
<tr>
<td>\texttt{EXTERNAL{list of procedures}}</td>
<td>external list of procedures</td>
</tr>
<tr>
<td>\texttt{RETURN{list of values}}</td>
<td>return (list of values)</td>
</tr>
<tr>
<td>\texttt{OUTPUT{list of values}}</td>
<td>output (list of values)</td>
</tr>
<tr>
<td>\texttt{EXIT}</td>
<td>exit</td>
</tr>
<tr>
<td>\texttt{AND}</td>
<td>and</td>
</tr>
<tr>
<td>\texttt{OR}</td>
<td>or</td>
</tr>
<tr>
<td>\texttt{NOT}</td>
<td>not</td>
</tr>
<tr>
<td>\texttt{TRUE}</td>
<td>true</td>
</tr>
<tr>
<td>\texttt{FALSE}</td>
<td>false</td>
</tr>
<tr>
<td>\texttt{GETS}</td>
<td>\leftarrow</td>
</tr>
</tbody>
</table>

Also note that all of the keywords \texttt{IF}, \texttt{WHILE}, \texttt{CALL\{}{}, \texttt{NOT}, etc. are available for use outside of the pseudocode environment, but they must be input in math mode. For example,

The \$\texttt{\{WHILE\}}$ loop is our friend.

generates

The \texttt{while} loop is our friend.

### 2.9 Statement Numbering

Statements can be numbered and given a reference key so that they can be referenced in a \LaTeX\ document using a \texttt{ref} command (see section 4.2 of [3]). This is done as follows:
The argument `<space>` is the amount of space to be left between the text and the statement number. This is a length that is specified by the user, and generally will require some experimentation in order for it to look nice. The argument `<key>` is the reference key used in the LaTeX \texttt{ref{}} command to refer to the given statement.

The default numbering for statements is arabic. However, it can be changed by a suitable \texttt{renewcommand{}}. An example is provided in the next section.

3 An example

The following example demonstrates the use of the pseudocode environment to describe a complete algorithm, the familiar “mergesort” algorithm. The \LaTeX input

\texttt{
\renewcommand{\thealgorithm}{\roman{algorithm}}
\begin{algorithm}{\label{MergeSort} \begin{algorithmic}{n, X}
\Comment{Sort the array $X$ of length $n$}
\IF{n = 2 \text{ \texttt{THEN}}}
\BEGIN
  \IF{X[0] > X[1] \text{ \texttt{THEN}}}
  \BEGIN
    T \GETS X[0]\texttt{;}
    X[0] \GETS X[1]\texttt{;}
    X[1] \GETS T
  \END
\END
\ELSEIF{n > 2 \text{ \texttt{THEN}}}
\BEGIN
  m \GETS \lfloor n/2 \rfloor
  \FOR{i \GETS 0 \TO m-1 \DO A[i] \GETS X[i]\texttt{;}}
  \FOR{i \GETS m \TO n-1 \DO B[i] \GETS X[i]\texttt{;}}
  \Comment{Now sort the subarrays $A$ and $B$}
  \CALL{MergeSort}{m, A}\texttt{;}
  \CALL{MergeSort}{n-m, B}\texttt{;}
  i \GETS 0\texttt{;}
  j \GETS 0\texttt{;}
  \FOR{k \GETS 0 \TO n-1 \DO}
  \BEGIN
    \IF{A[i] \leq B[j] \text{ \texttt{THEN}}}
  \END
\END
\\fi
\end{algorithmic}
}}
\BEGIN
\STATE $X[k] \gets \text{GETS } A[i] \ \text{STMTNUM}(i+1)\}\GRAY
$\STATE $i \gets \text{GETS } i+1$
\END
\ELSE
\BEGIN
\STATE $X[k] \gets \text{GETS } B[j] \ \text{STMTNUM}(1.03)\}\GRAY
\STATE $j \gets \text{GETS } j+1$
\END
\END
\END
\end{pseudocode}

produces the following output:

**Algorithm 3.1:** \textsc{MergeSort}(n, X)

\textbf{comment:} Sort the array $X$ of length $n$

\begin{itemize}
\item \textbf{if} $n = 2$
\begin{itemize}
\item \textbf{if} $X[0] > X[1]$
\begin{itemize}
\item \textbf{then} \quad $T \leftleftarrow X[0]$
\item \textbf{then} \quad $X[0] \leftleftarrow X[1]$
\item \textbf{then} \quad $X[1] \leftleftarrow T$
\end{itemize}
\end{itemize}
\item \textbf{else if} $n > 2$
\begin{itemize}
\item $m \leftleftarrow \lfloor n/2 \rfloor$
\item \textbf{for} $i \leftleftarrow 0$ \textbf{to} $m - 1$
\item \textbf{do} $A[i] \leftleftarrow X[i]$
\item \textbf{for} $i \leftleftarrow m$ \textbf{to} $n - 1$
\item \textbf{do} $B[i] \leftleftarrow X[i]$
\item \textbf{comment:} Now sort the subarrays $A$ and $B$
\item \textbf{MERGESort}(m, A)
\item \textbf{MERGESort}(n - m, B)
\item \textbf{then}
\item \textbf{i} \leftleftarrow 0
\item \textbf{j} \leftleftarrow 0
\item \textbf{for} $k \leftleftarrow 0$ \textbf{to} $n - 1$
\begin{itemize}
\item \textbf{if} $A[i] \leq B[j]$
\begin{itemize}
\item \textbf{then} \quad $X[k] \leftleftarrow A[i]$
\item \textbf{then} \quad $i \leftleftarrow i + 1$
\end{itemize}
\item \textbf{else} \quad $X[k] \leftleftarrow B[j]$
\item \textbf{else} \quad $j \leftleftarrow j + 1$
\end{itemize}
\end{itemize}
\end{itemize}
\end{itemize}

The counter \texttt{pseudonum} keeps track of the statement numbers. The style of the counter values can be changed using the method described in Section 6.3 of [3]. For example, we used the command
in our example so that statements were numbered with lowercase Roman numerals. We also assigned a label to the algorithm using the `\label{}` command that is described in Section 4.2 of [3]. Finally, by trial and error, we determined spacing so that the statement numbers would be vertically aligned.

We now give an example of how the numbered statements in the above algorithm can be referenced in a \LaTeX document. The commands

On lines (`\ref{st.1}`) and (`\ref{st.2}`) of Algorithm `\ref{MergeSort}`, we determine the $k$th element of the sorted array.

produce the following output:

On lines (i) and (ii) of Algorithm 3.1, we determine the $k$th element of the sorted array.

4 Framing

The pseudocode environment also has an optional parameter, `<frame>`. The complete form of the pseudocode environment is

```latex
\begin{pseudocode}[<frame>]{<Name>}{<Parameters>}
\end{pseudocode}
```

The possible values of `<frame>` are:

```
shadowbox  doublebox  ovalbox  ovalbox
framebox   plain     ruled    display
```

The values ending with “box” draw various types of frames around the algorithm. The value plain is the default and adds no frame to the algorithm. The value display is used for displaying sections of code without the algorithm name or parameters. Here are some examples with input:

```latex
\begin{pseudocode}[SquareAndMultiply] {x,b,n}
\COMMENT{ Compute $x^b \mod(n)$}
\GETS z 1\\
\WHILE b > 0 \DO
BEGIN
\GETS z^2 \mod(n) \\
\IF b \{ is odd
```

10
\[ \text{THEN } z \ \text{GETS } z \ \text{cdot } x \ \text{mdot } \{ n \} \ \text{END} \]
\[ \text{RETURN } \{ z \} \]
\end{pseudocode}

where we give <frame> each of the values described above.

When <frame> is shadowbox we obtain:

\begin{algorithm}
\textbf{Algorithm 4.1: SQUAREANDMULTIPLY}(x, b, n)
\begin{algorithmic}
\STATE \textbf{comment:} Compute \( x^b \pmod{n} \)
\STATE \( z \leftarrow 1 \)
\WHILE {\( b > 0 \)}
\DO \STATE \( z \leftarrow z^2 \pmod{n} \)
\IF {\( b \) is odd}
\THEN \STATE \( z \leftarrow z \cdot x \pmod{n} \)
\STATE \( b \leftarrow \text{SHIFTRIGHT}(b) \)
\RETURN \{ \( z \) \}
\ENDWHILE
\end{algorithmic}
\end{algorithm}

When <frame> is doublebox we obtain:

\begin{algorithm}
\textbf{Algorithm 4.2: SQUAREANDMULTIPLY}(x, b, n)
\begin{algorithmic}
\STATE \textbf{comment:} Compute \( x^b \pmod{n} \)
\STATE \( z \leftarrow 1 \)
\WHILE {\( b > 0 \)}
\DO \STATE \( z \leftarrow z^2 \pmod{n} \)
\IF {\( b \) is odd}
\THEN \STATE \( z \leftarrow z \cdot x \pmod{n} \)
\STATE \( b \leftarrow \text{SHIFTRIGHT}(b) \)
\RETURN \{ \( z \) \}
\ENDWHILE
\end{algorithmic}
\end{algorithm}

When <frame> is ovalbox we obtain:
Algorithm 4.3: \textsc{SquareAndMultiply}(x, b, n)
\begin{verbatim}
comment: Compute $x^b \pmod{n}$
z \leftarrow 1
while $b > 0$
do 
\begin{cases} 
z \leftarrow z^2 \pmod{n} 
\text{if $b$ is odd} 
\text{then} \quad z \leftarrow z \cdot x \pmod{n} 
b \leftarrow \text{ShiftRight}(b) 
\end{cases}
return (z)
\end{verbatim}

When <frame> is 0valbox we obtain:

Algorithm 4.4: \textsc{SquareAndMultiply}(x, b, n)
\begin{verbatim}
comment: Compute $x^b \pmod{n}$
z \leftarrow 1
while $b > 0$
do 
\begin{cases} 
z \leftarrow z^2 \pmod{n} 
\text{if $b$ is odd} 
\text{then} \quad z \leftarrow z \cdot x \pmod{n} 
b \leftarrow \text{ShiftRight}(b) 
\end{cases}
return (z)
\end{verbatim}

When <frame> is framebox we obtain:

Algorithm 4.5: \textsc{SquareAndMultiply}(x, b, n)
\begin{verbatim}
comment: Compute $x^b \pmod{n}$
z \leftarrow 1
while $b > 0$
do 
\begin{cases} 
z \leftarrow z^2 \pmod{n} 
\text{if $b$ is odd} 
\text{then} \quad z \leftarrow z \cdot x \pmod{n} 
b \leftarrow \text{ShiftRight}(b) 
\end{cases}
return (z)
\end{verbatim}

When <frame> is plain or if [frame] is omitted we obtain:
Algorithm 4.6: \texttt{SQUAREANDMULTIPLY}(x, b, n)

\textbf{comment:} Compute $x^b \pmod{n}$

$z \leftarrow 1$

\textbf{while} $b > 0$
\begin{align*}
&\quad \begin{cases} 
&\quad z \leftarrow z^2 \pmod{n} \\
&\quad \text{if } b \text{ is odd} \\
&\quad \text{then } z \leftarrow z \cdot x \pmod{n} \\
&\quad b \leftarrow \text{SHIFTRIGHT}(b)
\end{cases}
\end{align*}
\textbf{return} (z)

When <frame> is ruled we obtain:

Algorithm 4.7: \texttt{SQUAREANDMULTIPLY}(x, b, n)

\textbf{comment:} Compute $x^b \pmod{n}$

$z \leftarrow 1$

\textbf{while} $b > 0$
\begin{align*}
&\quad \begin{cases} 
&\quad z \leftarrow z^2 \pmod{n} \\
&\quad \text{if } b \text{ is odd} \\
&\quad \text{then } z \leftarrow z \cdot x \pmod{n} \\
&\quad b \leftarrow \text{SHIFTRIGHT}(b)
\end{cases}
\end{align*}
\textbf{return} (z)

The purpose of the value \texttt{display} is to allow portions of algorithms to be displayed without the algorithm header. Thus for example to display the section of code in the \texttt{while} loop of the \texttt{SQUAREANDMULTIPLY()} algorithm one could write

\begin{verbatim}
\begin{center}
\begin{minipage}{2in}
\begin{pseudocode}[display]{}
z \GETS z^2 \pmod{n}\ \\
\IF \text{b is odd} \\
\THEN z \GETS z \cdot x \pmod{n}\ \\
\GETS \text{CALL}(\text{SHIFTRIGHT})\{b\}
\end{pseudocode}
\end{minipage}
\end{center}
\end{verbatim}

which would produce the following output:
\[ z \leftarrow z^2 \pmod{n} \]
if \( b \) is odd
\[ \text{then } z \leftarrow z \cdot x \pmod{n} \]
\[ b \leftarrow \text{SHIFTRIGHT}(b) \]

References

