# 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 we 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 \GETS {9c/5} + 32\\
  \RETURN{f}
\end{pseudocode}
```

produce the following output when included in a LaTeX document:

**Algorithm 2.1:** CelsiusToFahrenheit($c$)

```
$f \leftarrow 9c/5 + 32$
**return** $(f)$
```

Notice that the command `\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, `\GETS` could be replaced by `=`, as is done in the "C" programming language.

### 2.1 The *begin-end* Construct

To form compound statements from simple statements, the *begin-end* construct is used as follows:

```
\BEGIN
   some statement\\
   another statement\\
   yet another statement
\END
```

This generates the following:

$$\left\{\begin{array}{l}\mbox{some statement}\\ \mbox{another statement}\\ \mbox{yet another statement}\end{array}\right.$$

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 `\mbox{}`.

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 `\END`.

## 2.2 The *if-then-else* Construct

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

```
\IF <condition> \THEN <stmt>
\IF <condition> \THEN <stmt> \ELSE <stmt>
\IF <condition> \THEN <stmt> \ELSEIF <stmt> \THEN <stmt>
```

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

```
\IF some condition is true
\THEN
 \BEGIN
    some statement\\
    another statement\\
    yet another statement
 \END
\ELSEIF some other condition is true
\THEN
 \BEGIN
    some statement\\
    another statement\\
    yet another statement
 \END
\ELSEIF some even more bizarre condition is met
\THEN
  do something else
\ELSE
  do the default actions
```

would produce the following output:

```
if some condition is true
  then { some statement
       { another statement
       { yet another statement
  else if some other condition is true
  then { some statement
       { 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 $i \leftarrow 0$ to 10
  do some processing
```

and

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

produces

```
for each $x \in \mathcal{S}$
  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
```

produces

**while** some condition holds
 **do** some processing

## 2.5 The *repeat-until* Loop

The *repeat-until* loop takes the following form:

```
\REPEAT <stmt> \UNTIL <condition>
```

For example,

```
\REPEAT
  some processing
\UNTIL some condition is met
```

produces

**repeat**
 some processing
**until** some condition is met

## 2.6 Main Programs and Procedures

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

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

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}
  f \GETS {9c/5} + 32\\
  \RETURN{f}
\ENDPROCEDURE
\MAIN
x \GETS lower \\
\WHILE x \leq upper \DO
\BEGIN
 \OUTPUT{x, \CALL{CelsiusToFahrenheit}{x}}\\
 x \GETS x+1
\END
\ENDMAIN
\end{pseudocode}
```

produce the following output:

**Algorithm 2.8:** TEMPERATURETABLE(*lower*, *upper*)

```
procedure CELSIUSTOFAHRENHEIT($c$)
  $f \leftarrow 9c/5 + 32$
  return ($f$)

main
  $x \leftarrow lower$
  while $x \leq upper$
    do $\begin{cases} \textbf{output } (x, \textsc{CelsiusToFahrenheit}(x)) \\ x \leftarrow x + 1 \end{cases}$
```

## 2.7 Comments

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

```
\COMMENT{<stmt>}
```

For example, the commands

```
A \GETS B\\
\COMMENT{Now increment the value of $A$}\\
A \GETS A+1
```

produce the output

```
$A \gets B$
**comment:** Now increment the value of $A$
$A \gets A + 1$
```

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 1: Other Predefined Keywords

| command | output |
|---|---|
| `\LOCAL{list of variables}` | **local** list of variables |
| `\GLOBAL{list of variables}` | **global** list of variables |
| `\EXTERNAL{list of procedures}` | **external** list of procedures |
| `\RETURN{list of values}` | **return** (list of values) |
| `\OUTPUT{list of values}` | **output** (list of values) |
| `\EXIT` | **exit** |
| `\AND` | **and** |
| `\OR` | **or** |
| `\NOT` | **not** |
| `\TRUE` | **true** |
| `\FALSE` | **false** |
| `\GETS` | $\gets$ |

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

```
The  $\WHILE$ loop is our friend.
```

generates

The **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 `\ref{}` command (see section 4.2 of [3]). This is done as follows:

```
\STMTNUM{<space>}{<key>}
```

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 `\ref{}` command to refer to the given statement.

The default numbering for statements is arabic. However, it can be changed by a suitable `\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

```
\renewcommand{\thepseudonum}{\roman{pseudonum}}
\begin{pseudocode}{MergeSort}{n,X}
\label{MergeSort}
\COMMENT{Sort the array $X$ of length $n$}\\
\IF n=2 \THEN
\BEGIN
  \IF X[0]>X[1] \THEN
  \BEGIN
    T \GETS X[0]\\
    X[0]\GETS X[1]\\
    X[1]\GETS T
  \END
\END
\ELSEIF n>2 \THEN
\BEGIN
  m\GETS \lfloor n/2 \rfloor\\
  \FOR i\GETS 0 \TO m-1 \DO A[i] \GETS X[i]\\
  \FOR i\GETS m \TO n-1 \DO B[i] \GETS X[i]\\
  \COMMENT{Now sort the subarrays $A$ and $B$}\\
  \CALL{MergeSort}{m,A}\\
  \CALL{MergeSort}{n-m,B}\\
  i\GETS 0\\
  j\GETS 0\\
  \FOR k \GETS 0 \TO n-1 \DO
  \BEGIN
    \IF A[i] \leq B[j] \THEN
    \BEGIN
       X[k]\GETS A[i] \STMTNUM{1in}{st.1}\\
       i\GETS i+1
    \END
    \ELSE
    \BEGIN
       X[k]\GETS B[j] \STMTNUM{1.03in}{st.2}\\
       j\GETS j+1
    \END
  \END
\END
\end{pseudocode}
```

produces the following output:

**Algorithm 3.1:** MERGESORT$(n, X)$

```
comment: Sort the array $X$ of length $n$
if $n = 2$
  then { if $X[0] > X[1]$
           then { $T \leftarrow X[0]$
                  $X[0] \leftarrow X[1]$
                  $X[1] \leftarrow T$
  else if $n > 2$
  then { $m \leftarrow \lfloor n/2 \rfloor$
         for $i \leftarrow 0$ to $m - 1$
           do $A[i] \leftarrow X[i]$
         for $i \leftarrow m$ to $n - 1$
           do $B[i] \leftarrow X[i]$
         comment: Now sort the subarrays $A$ and $B$
         MERGESORT$(m, A)$
         MERGESORT$(n - m, B)$
         $i \leftarrow 0$
         $j \leftarrow 0$
         for $k \leftarrow 0$ to $n - 1$
           do { if $A[i] \leq B[j]$
                  then { $X[k] \leftarrow A[i]$          (i)
                         $i \leftarrow i + 1$
                  else { $X[k] \leftarrow B[j]$          (ii)
                         $j \leftarrow j + 1$
```

The counter `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

```
\renewcommand{\thepseudonum}{\roman{pseudonum}}
```

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 LaTeXdocument. 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

```
\begin{pseudocode}[<frame>]{<Name>}{<Parameters>}
  pseudocode constructs
\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:

```
\begin{pseudocode}[<frame>]{SquareAndMultiply}{x,b,n}
 \COMMENT{ Compute $x^b \pmod{n}$}\\
 z\GETS 1\\
 \WHILE b > 0 \DO
 \BEGIN
  z \GETS z^2 \pmod{n} \\
  \IF b\mbox{ is odd}
```

```
    \THEN z \GETS z \cdot x  \pmod{n} \\
    b \GETS \CALL{ShiftRight}{b}
 \END\\
 \RETURN{z}
\end{pseudocode}
```

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

When `<frame>` is `shadowbox` we obtain:

```
Algorithm 4.1: SquareAndMultiply($x, b, n$)

 comment: Compute $x^b \pmod{n}$
 $z \leftarrow 1$
 while $b > 0$
   do { $z \leftarrow z^2 \pmod{n}$
        if $b$ is odd
          then $z \leftarrow z \cdot x \pmod{n}$
        $b \leftarrow$ ShiftRight($b$)
 return ($z$)
```

When `<frame>` is `doublebox` we obtain:

```
Algorithm 4.2: SquareAndMultiply($x, b, n$)

 comment: Compute $x^b \pmod{n}$
 $z \leftarrow 1$
 while $b > 0$
   do { $z \leftarrow z^2 \pmod{n}$
        if $b$ is odd
          then $z \leftarrow z \cdot x \pmod{n}$
        $b \leftarrow$ ShiftRight($b$)
 return ($z$)
```

When `<frame>` is `ovalbox` we obtain:

```
Algorithm 4.3: SquareAndMultiply($x, b, n$)

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

  $z \leftarrow 1$
  while $b > 0$
    do {
      $z \leftarrow z^2 \pmod{n}$
      if $b$ is odd
        then $z \leftarrow z \cdot x \pmod{n}$
      $b \leftarrow$ ShiftRight($b$)
    }
  return ($z$)
```

When `<frame>` is `Ovalbox` we obtain:

```
Algorithm 4.4: SquareAndMultiply($x, b, n$)

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

  $z \leftarrow 1$
  while $b > 0$
    do {
      $z \leftarrow z^2 \pmod{n}$
      if $b$ is odd
        then $z \leftarrow z \cdot x \pmod{n}$
      $b \leftarrow$ ShiftRight($b$)
    }
  return ($z$)
```

When `<frame>` is `framebox` we obtain:

```
Algorithm 4.5: SquareAndMultiply($x, b, n$)

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

  $z \leftarrow 1$
  while $b > 0$
    do {
      $z \leftarrow z^2 \pmod{n}$
      if $b$ is odd
        then $z \leftarrow z \cdot x \pmod{n}$
      $b \leftarrow$ ShiftRight($b$)
    }
  return ($z$)
```

When `<frame>` is `plain` or if `[<frame>]` is omitted we obtain:

**Algorithm 4.6:** SQUAREANDMULTIPLY$(x, b, n)$

```
comment: Compute $x^b \pmod{n}$
$z \leftarrow 1$
while $b > 0$
  do ⎧ $z \leftarrow z^2 \pmod{n}$
     ⎨ if $b$ is odd
     ⎪   then $z \leftarrow z \cdot x \pmod{n}$
     ⎩ $b \leftarrow$ SHIFTRIGHT$(b)$
return $(z)$
```

When `<frame>` is `ruled` we obtain:

**Algorithm 4.7:** SQUAREANDMULTIPLY$(x, b, n)$

```
comment: Compute $x^b \pmod{n}$
$z \leftarrow 1$
while $b > 0$
  do ⎧ $z \leftarrow z^2 \pmod{n}$
     ⎨ if $b$ is odd
     ⎪   then $z \leftarrow z \cdot x \pmod{n}$
     ⎩ $b \leftarrow$ SHIFTRIGHT$(b)$
return $(z)$
```

The purpose of the value `display` is to allow portions of algorithms to be displayed with out the algorithm header. Thus for example to display the section of code in the **while** loop of the SQUAREANDMULTIPLY() algorithm one could write

```
\begin{center}
\begin{minipage}{2in}
\begin{pseudocode}[display]{}{}
  z \GETS z^2 \pmod{n} \\
  \IF b\mbox{ is odd}
    \THEN z \GETS z \cdot x  \pmod{n}\\
    b \GETS \CALL{ShiftRight}{b}
\end{pseudocode}
\end{minipage}
\end{center}
```

which would produce the following output:

```
$z \leftarrow z^2 \pmod{n}$
**if** $b$ is odd
  **then** $z \leftarrow z \cdot x \pmod{n}$
$b \leftarrow \textsc{ShiftRight}(b)$
```

## References

[1] M. Goossens, F. Mittelbach and A. Samarin, *The LaTeXCompanion*, Addison-Wesley, 1994.

[2] D.L. Kreher and D.R. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search*, CRC Press, 1999.

[3] L. Lamport, LaTeX, *A Document Preparation System*, Addison-Wesley, 1994.