The Algorithm Environment

Otfried Schwarzkopf

The LaTeX 2$_\varepsilon$ package `algo.sty` provides an `algorithm` environment that makes it easier to write algorithms in pseudo code in a uniform style. We use this package to typeset the algorithms in our book.[1]

The environment comes in two flavors, one where algorithms are numbered like theorems, and one where algorithms are named explicitely. You can select the flavor using an option to the `algo` package as follows:

```
\usepackage[named]{algo}
or
\usepackage[numbered]{algo}
```

The default are named algorithm environments.

The environment is invoked as follows.

```
\begin{algorithm}{algorithm name}[parameters]{algorithm header}
    algorithm body, lines terminated by \\
\end{algorithm}
```

If you are using named algorithms, the *algorithm name* and *parameters* arguments are used for the algorithm heading. (The *parameters* argument is optional. If present, it will be enclosed in parentheses and will by typeset in math mode.)

For numbered algorithms, the *name* and *parameters* arguments are ignored, and the algorithm heading contains the algorithm's number. You can share numbers with another theorem-like environment like `definition`, using the following command in the preamble.

```
\algorithmcounter{definition}
```

The *algorithm header* is used to add the following headings to the algorithm (the `\qprocedure` command is ignored for named algorithms):

```
\qinput{description of algorithm input}
\qoutput{description of algorithm output}
\qcomment{description of procedure}
\qprocedure[parameters]{procedure name}
```

You can also put a `\label` command in the algorithm header. The label will be set to the algorithm's number (for numbered algorithms) or to the algorithm name (for named algorithms). This means that you can refer to an algorithm by "`Algorithm~\ref{alg:pipapo}`" regardless of whether you prefer named or numbered algorithms.

The body of the algorithm consists of several lines, separated by `\\`. The lines are numbered automatically. To reference a line number, put a `\label` command in the text of a line. There are macros for all keywords that should be used.

[1]Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf. *Computational Geometry by Example.* To appear.

To properly format keywords and indentation of the algorithm, you have two sets of macros. The macros `\qif`, `\qthen`, `\qelse`, `\qfi`, `\qfor`, `\qdo`, `\qrof`, `\qwhile`, `\qelihw`, `\qrepeat`, `\quntil` serve to describe the control structure, and are typeset as keywords. The algorithm environment will automatically indent subsequent lines, and break statements into multiple lines if necessary. All control structures can be nested, but you have to obey a certain format: control structure keywords should be the first commands on the line, and you have to break the structures into multiple lines. Note that `\qendif`, `\qendfor`, and `\qend` are synonyms for `\qfi`, `\qrof`, and `\qelihw` (like in the C-shell). Algorithm *IfForWhile* presents the four different control structures.

Algorithm *IfForWhile*
(∗ demonstrates control structures ∗)

```
1.  for $i \leftarrow 1$ to $n$
2.      do $x_i \leftarrow x_i^2$;
3.         $y_i \leftarrow x_i - y_i$
4.  if $A = B$
5.     then do whatever is necessary if $A$ equals $B$
6.     else do something else
7.          and wait for better times
8.  while $Q \neq \emptyset$
9.      do let $q$ be the first element of $Q$ and remove it from $Q$
10.        do something with $q$
11. repeat
12.     do something really weird
13. until you get sufficiently tired of it
14. return 42
```

You *must* use a `\qthen` statement on the line after a `\qif` command, and you *must* use a `\qdo` statement on the line following a `\qfor` or `\qwhile` command. The three control structures are terminated by a `\qfi`, `\qrof`, or `\qend` command, which should come on the last line of the control structure, before the `\\` macro. `\quntil` should not be preceeded by the `\\` macro. The source code for this algorithm is shown in Figure 1. Note the use of `\label` commands to refer tot the algorithm itself and to its line 4.

The remaining macros are pretty simple, they are just shorthands for some keywords and have no other side effect.

`\qlet`	$\leftarrow$
`\qto`	**to**
`\qdownto`	**downto**
`\qand`	**and**
`\qor`	**or**
`\qnot`	**not**
`\qreturn`	**return**
`\qqif`	**if**
`\qqthen`	**then**
`\qqelse`	**else**
`\stop`	**stop**
`\qnil`	**nil**
`\qtrue`	**true**
`\qfalse`	**false**
`\qcom{`*comment*`}`	(∗ comment ∗)
`\qproc[`*args*`]{`*proc_name*`}`	*proc_name*(*args*)

```
\begin{algorithm}{IfForWhile}{
    \label{algo:ifforwhile}
    \qcomment{demonstrates control structures}}
  \qfor $i \qlet 1$ \qto $n$ \\
  \qdo $x_{i} \qlet x_{i}^{2}$; \\
  $y_{i} \qlet x_{i} - y_{i}$ \qrof\\
  \qif $A = B$ \label{line:ifAisB}\\
  \qthen do whatever is necessary if $A$ equals $B$\\
  \qelse do something else\\
  and wait for better times \qfi\\
  \qwhile $Q \neq \emptyset$ \\
  \qdo let $q$ be the first element of $Q$ and remove it from $Q$\\
  do something with $q$ \qend \\
  \qrepeat \\
  do something really weird
  \quntil you get sufficiently tired of it\\
  \qreturn $42$
\end{algorithm}
```

Figure 1: LaTeX-source for Algorithm *IfForWhile*

The parameterless macros use the *xspace* package. This means that they will automatically be followed by a space, unless some punctuation character follows the macro. Refer to the *xspace* documentation for details.

The first argument to `\qproc` is optional. If present, *args* is enclosed in parentheses and typeset in math mode. You can also use `\qproc` to refer to named algorithms.

Here are some more examples, demonstrating nested control structures. The source for this document (and therefore all these examples) can be found on the world wide web.[2]

```
Algorithm FuzzyChromatic($T$)
Input: A tree $T$
Output: The Fuzzy Chromatic Number of $T$
1. for $\nu$ is a leaf of $T$
2.       do compute $k = \sigma(S)$ by testing all rectangles in $S$. This is a pretty
            complicated operation, but can be written on a single line within your
            algorithm environment, which will then break it automatically, as you
            can see.
3.          if $\nu$ has property $\mathcal{B}$
4.             then take appropriate action.
5.                  and recursively compute FuzzyChromatic($T$)
6.             else locate $\ell$ in $\Xi(L)$. Let $t$ be the triangle containing it. compute
                    $k(\ell)$.
7. return a random number between 0 and $k(\ell)$.
```

And here is another example.

```
Algorithm NonSense($A$)
(* Computes nothing, but does it fast *)
1. if $\nu$ is a leaf then use naive algorithm
2. if $A = B$
3.    then if $B = C$ then do this
```

[2]at `http://graphics.postech.ac.kr/otfried/tex/algodoc.tex`

```
4.       else  if $A = C$
5.                then (∗ should not arise too often ∗)
6.                       add some more things.
7.                else  do other stuff
8.             Again, there is no limit on how much you can put on a line. The line
               will be broken by the algorithm environment.
9.             for $i \in N$
10.                 do something with $i$
11.                    for $j \in M$
12.                         do compute $k_i$.
13.                    we are still in the outer for-loop.
14.  return $\emptyset$.
```

Remember that if you want to use **if** and **then** on one line like on line 1 or line 4 of Algorithm *NonSense*, then you must use `\qqif` and `\qqthen` instead of `\qif` and `\qthen`. These commands have no influence on the indentation of the algorithm.

Algorithm *Make monotone*(P)
Input: A simple polygon $\mathcal{P}$.
Output: A set of diagonals that partition $\mathcal{P}$ into x-monotone polygons.
(∗ A real algorithm from our book ∗)

```
1.   repeat
2.        Delete the leftmost vertex $v_j$ from $\Delta$.
3.        Advance the sweep-line to contain $v_j$.
4.        if $v_j$ is a start vertex
5.           then if the interior of $\mathcal{P}$ to its left
6.                       then search with $x_j$ in $\mathcal{T}$ to find the edge $e_i$ vertically below
                              $v_j$.
7.                             Choose the diagonal $\overline{v_j v_m}$, where $v_m$ is the vertex asso-
                              ciated to $e_i$
8.                             (∗ Now $v_j$ is not a start vertex in either of the polygons
                              obtained by the new diagonal. ∗)
9.                             Insert $e_j$ into $\mathcal{T}$ and let $v_j$ be the vertex associated to it.
10.                      else  insert $e_{j-1}$ in $\mathcal{T}$ and let $v_j$ be the vertex associated to it.
11.          else  if $v_j$ is an end vertex
12.                      then delete $e_j$ and its associated vertex from $\mathcal{T}$
13.                             if the interior of $\mathcal{P}$ lies to its right
14.                                then delete $e_{j-1}$ from tree.
15.                                       Let $e_i$ be the edge vertically below $v_j$. Replace
                                          its associated vertex with $v_j$.
16.                      else  (∗ when $v_j$ is a bend vertex ∗) if the interior of $\mathcal{P}$ lies
                              above $v_j$
17.                                then delete $e_{j-1}$ and its associated vertex from $\mathcal{T}$
18.                                       Insert $e_{j+1}$ with $v_j$ as its associated vertex.
19.                                else  delete $e_{j+1}$ and its associated vertex from $\mathcal{T}$
20.                                       Insert $e_{j-1}$ with $v_j$ as its associated vertex.
21.  until $\Delta$ is empty.
```