algorithm2e.sty — package for algorithms

release 3.9

(c) 1996–2005 Christophe Fiorio, LIRM Montpellier, France

Report bugs and comments to fiorio@lirmm.fr
algorithm2esty-annonce@lirmm.fr mailing list for announcements
algorithm2esty-discussion@lirmm.fr mailing list for discussion[*,†,‡,§,¶,‖,**,††]

febuary 10 2005

Contents

[*] The author is very grateful to David Carlisle, one of the authors of the LaTeX Companion book, for his advices

[†] Martin Blais (blais@IRO.UMontreal.CA) for his suggestions

[‡] David A. Bader (dbader@eece.unm.edu) for his new option `noend`

[§] Gilles Geeraerts (gigeerae@ulb.ac.be) for his new command `SetKwIfElseIf`

[¶] Ricardo Fukasawa (fukasawa@globo.com) for the portuguese keywords

[‖] Christian Icking (christian.icking@fernuni-hagen.de) for the german translation of keywords

[**] Arnaud Giersch (giersch@icps.u-strasbg.fr) for his suggestions and corrections on SetKwComments

[††] and the many users as Jean-Baptiste Rouquier (jean-baptiste.rouquier@ens-lyon.fr) for their remarks

1 Introduction

Algorithm2e is an environment for writing algorithms in LaTeX2e. An algorithm is defined as a floating object like figures. It provides macros that allow you to create different sorts of key words, thus a set of predefined key words is given. You can also change the typography of the keywords.

You can subscribe to `algorithm2e-announce` mailing list to receive announcements about revisions of the package and to `algorithm2e-discussion` to discuss, send comments, ask questions about the package. In order to subscribe to the mailing lists you have to send an email to `sympa@lirmm.fr` with `subscribe algorithm2e-announce Firstname Name` or `subscribe algorithm2e-discussion Firstname Name` in the body of the message.

2 How to use it: abstract

You must set `\usepackage[`*options*`]{algorithm2e}` before `\begin{document}` command. The available options are described in section 5.

The optional arguments [Hhtbp] works like those of figure environment. The **H** argument forces the algorithm to stay in place. If used, an algorithm is no more a floating object. Caution: algorithms cannot be cut, so if there is not enough place to put an algorithm with H option at a given spot, LaTeX will place a blank and put the algorithm on the following page.

Here is a quick example[1]:

```
\begin{algorithm}[H]
  \SetLine
  \KwData{this text}
  \KwResult{how to write algorithm with \LaTeX2e }

  initialization\;
  \While{not at end of this document}{
    read current\;
    \eIf{understand}{
      go to next section\;
      current section becomes this one\;
      }{
      go back to the beginning of current section\;
      }
    }
  \caption{How to write algorithms}
\end{algorithm}
```

which gives

```
Data: this text
Result: how to write algorithm with LaTeX2e
initialization;
while not at end of this document do
|  read current section;
|  if understand then
|  |  go to next section;
|  |  current section becomes this one;
|  else
|  |  go back to the beginning of current section;
|  end
end
```

Algorithm 1: How to write algorithms

[1] For longer and more complexe examples see Section 8

VERY IMPORTANT : each line **MUST** end with `\;` only those with a macro beginning a block should not end with `\;`. Note then that you can always use the `\;` command in math mode to set a small space.

The caption works as in a figure environment and is used by `\listofalgorithms` as a reference name for the list of algorithms. You can also use the title macro given with the package, but this macro doesn't insert an entry in the list of algorithms.

There are six text types in an algorithm environment:

1. The keywords (**Kw**): Macros which usually indicate words of the language. Some are predefined and given with *the algorithm package.*

   The user can define his own language keywords by using the different macros presented in section 7.1 (see below for a short, non exhaustive list). He can also define simple keywords with the `\SetKw{Kw}{thetext}` macro.

2. The Functions: (**Func**) Macros defined by the user which denote local functions or other algorithms defined in the text.

   They are defined using `\SetKwFunction{KwFn}{Fn}` where \KwFn will be the macro and `Fn` the text printed.

3. The Arguments (**Arg**): The arguments of the *Kw* or *Func* macros.

4. The procedure and function name environment style (**ProcNameSty**): The type style of the caption of *procedure* and *function* environment.

5. The arguments of procedure and function environments style (**ProcArgSty**): the type style of the argument of *procedure* and *function* environments.

6. Data (**Data**): A type of text different from the default. You can use it as you want, and can be useful for example to emphasize a Data structure or denotes some important variables.

   They are defined with the help of the `\SetKwData{KwDat}{data}` macro, where \KwDat will be the macro and `data` the text printed.

7. The text (the default): All the remaining text of the algorithm.

Note that if you define macros outside the algorithm environment they are available in all the document and, in particular, you can use them inside all algorithms without redefining them. Be careful you can't use macros beginning a block outside an algorithm environment.

Here are some language keywords predefined in the package[2]:

- \KwData{input}, \KwResult{output}, \KwIn{input}, \KwOut{output}
- \KwTo, \KwRet{[value]}
- \Begin{block inside}
- \eIf{condition}{then block}{else block}, \If{condition}{then block}
- \Switch{condition}{Switch block}, \Case{a case}{case block}, \Other{otherwise block}
- \For{condition}{text loop}
- \While{condition}{text loop}
- \Repeat{end condition}{text loop}

[2] See Section 7 for a complete list of all predefined macros such as `\uIf`, ...

3 Compatibility issues with other packages

At this time, release 3.9has no known compatibility problem with other packages. Nevertheless, when use with some packages, you need to specify some particular options, either from `algorithm2e` package or from the other packages.

hyperref if you want to compile in LaTeX, you have to use it with `naturalnames` option. You don't need to specify it if you compile with `pdflatex`.

4 Environments defined in the package

This package provides 4 environments :

algorithm : the main environment, the one you will used most of the time.

algorithm* : same as the precedent, but used in a two columns text, puts the algorithm across the two columns.

procedure : This environment works like algorithm environment but:

- the `ruled` (or `algoruled`) style is recommended.
- the caption now writes **Procedure name...**
- the syntax of the `\caption` command is restricted as follow: you MUST put a name followed by 2 braces like this "*Name()*". You can put arguments inside the braces and text after. If no argument is given, the braces will be removed in the title.
- label now puts the name (the text before the braces in the caption) of the procedure or function as reference (not the number like a classic algorithm environment).

procedure* : same as the precedent, but used in a two columns text outs the procedure across the two columns.

function : as the precedent but with **Function** instead of procedure in the title.

function* : same as the precedent, but used in a two columns text outs the function across the two columns.

5 The options of the package

algo2e: changes the name of environment algorithm into algorithm2e and so allows to use the package with some journal style which already define an algorithm environment. Changes also the command name for the list of algorithms, it becomes `\listofalgorithmes`

slide: require package color. Hack for slide class in order to have correct margins.

english: the default.

french: to have for example *algorithme :* instead of *algorithm:*.

german: to have for example *Prozedur :* instead of *procedure:*.

portugues: to have for example *Algoritmo:* instead of *algorithm:*.

czech: to have for example *Algoritmus:* instead of *algorithm:*.

figure: algorithms are put in classical figures and so are numbered as figures and putted in the `\listoffigures`.

algopart: algorithms are numbered within part numbers.

algochapter: algorithms are numbered within chapter numbers.

algosection: (default) algorithms are numbered within section numbers.

boxed: to have algorithms enclosed in a box.

boxruled: to have algorithms enclosed in a box and caption above and boxed to.

ruled: to have algorithms with a line at the top and the bottom. Note that the caption is not centered under the algorithm anymore but is set at the beginning of the algorithm.

algoruled: as above but with extra spaces after the rules.

plain: the default, with no feature.

lined: `\SetLine` becomes the default, see section 6.2 for explanations about the `\SetLine` macros.

vlined: `\SetVline` becomes the default, see section 6.2 for explanations about the `\SetVline` macros.

noline: `\SetNoline` becomes the default, see section 6.2 for explanations about the `\SetNoline` macros.

linesnumbered: lines of the algorithms are numbered except for comments and input/output (KwInput and KwInOut). You must use `\nllabel{label}` to label thoses lines.

linesnumberedhidden: lines of the algorithms are numbered as linesnumbered but numbers are not shown. `\showln` and `\showlnlabel{label}` show the number on line they are put.

commentsnumbered: makes comments be numbered if numbering is active.

inoutnumbered: makes data input/output be numbered if numbering is active.

titlenumbered: `\Titleofalgo{title}` prints *Algorithm n: thetitle* where n is the counter of the algo.
Beware: `\Titleofalgo` don't insert an entry in the list of algorithms. So do not use `\Titleofalgo` with a caption. Both increment the counter of the algorithms.

titlenotnumbered (default) the macro `\Titleofalgo{title}` doesn't number the algorithm.

resetcount the line numbers are reset to 0 at the beginning of each algorithm (by default).

noresetcount the contreverse of the precedent. To reset the line counter to 0 do: `\setcounter{AlgoLine}{0}`

algonl the line numbers will be prefixed with the number of the current algorithm. **Take care** to set the caption of the algorithm at the beginning of the environnement, else you will have the precedent algorithm number as the current one.

longend the end keyword are longer and different for each macro. For example *endif* for a if-then-else macro.

shortend the "end keyword" of the macros is just *end* (default).

noend the “end keyword” of the macros is not printed.

dotocloa adds an entry in the toc for the list of algorithms. This option loads package `tocbibind` if not already done and so list of figures and list of tables are also added in the toc. If you want to control which ones of the lists will be added in the toc, please load package `tocbibind` before package algorithm and give it the options you want.

scright (default) right justified side comments (side comments are flushed to the righr)

scleft left justified side comments (side comments are put right after the code line)

fillcomment (default) end mark of comment is flushed to the right so comments fill all the width of text

nofillcomment end mark of comment is put right after the comment

6 The macros provided with the package

6.1 Typesetting macros

`\;` marks the end of a line. **Don’t forget it !**. By default, it prints a ‘;’. You can change this with `\dontprintsemicolon`.

`\Titleofalgo{thetitle}` prints: “Algorithm n°: thetitle” in the typography and size defined by `\SetTitleSty`. Puts a vertical space below.
Beware: `\Titleofalgo` doesn’t insert an entry in the list of algorithms. So don’t use `\Titleofalgo` with `\caption`. Both increment the counter of the algorithms.
note:*with the french option prints* Algorithme n°:

`\listofalgorithms` inserts the list of all algorithms having a *caption*.

`\BlankLine` prints a blank line. In fact puts a vertical space of one `ex`.

`\Indp` indents plus → the text is shifted to the right.

`\Indm` indents minus → the text is shifted to the left.

`\nllabel{label}` macro for labelling lines when auto-numbering is active.

`\nl` numbers the line: must BEGIN the line.

`\lnl{label}` numbers and labels the line : must BEGIN the line. So you can refer to the number of the line by the following command: `\ref{label}`

`\nlset{}` **and** `\lnlset{}{}` work as `\nl` and `\lnl{}` except that the additional argument is the number (text) to put at the begin of the line.

`\showln` shows number of the line when linesnumberedhidden is activated.

`\showln{label}` same as precedent but with a label.

`\KwSty{<text>}` set <text> in keyword type style.

`\FuncSty{<text>}` set <text> in function type style.

`\ArgSty{<text>}` set <text> in argument type style.

`\DataSty{<text>}` sets <text> in data typography.

`\CommentSty{<text>}` sets <text> in comment typography.

`\AlFnt` is used at the beginning of the body of algorithm in order to define the fonts used for typesetting algorithms. You can use it elsewhere you want to typeset text as algorithm
For example you can do `\SetAlFnt{\small\sf}` to have algorithms typeset in small sf font. Default is nothing so algorithm is typeset as the text of the document.

`\AlCapFnt` is used at the beginning of the caption in order to define the fonts used for typesetting algorithms. You can use it elsewhere you want to typeset text as algorithm
For example you can do `\SetAlCapFnt{\large\em}` to have cpation typeset in large em font. Default is nothing so caption is typeset as the text of the document.

`\AlTitleFnt{<text>}` is used to typeset {Algorithm: } in the caption. You can use it to have text typeset as {Algorithm:} of captions. Default is textbf.

Default can be redefined by `\SetAlTitleFnt{font}`.

6.2 Restyling macros

`\restylealgo{style}` change the layout of the algorithms as do options *boxed*, *boxruled*, *ruled* and *algoruled*.

`\linesnumbered` makes lines of the following algorithms be auto-numbered.

`\linesnumberedhidden` makes lines of the following algorithms be auto-numbered, but numbers stay hidden. You have to use `\showln` and `\showlnlabel` to see them.

`\linesnotnumbered` makes lines of the following algorithms no be auto-numbered.

`\SetAlgoSkip{skip command}` Algorithms puts extra vertical space before and after to avoid having text bumping lines of boxed or ruled algorithms. By default, this is a . You can change this value with this macro. The four possibilities are:

- \SetAlgoSkip{}] for no extra vertical skip
- \SetAlgoSkip{smallskip}] to act as the default behaviour
- \SetAlgoSkip{medskip}] to have a bigger skip
- \SetAlgoSkip{bigskip}] to have the bigger skip

Note that you can apply the skip you want by defining a macro doing it and passing its name (without \) to \SetAlgoSkip

`\SetAlgoInsideSkip{skip command}` Algorithms puts no extra vertical space before and after the core of the algorithm. So text is put right after the lines in boxed or ruled style. To put an extra space, use `\SetAlgoInsideSkip{skip command}`, for example `\SetAlgoInsideSkip{smallskip}`, like for `\SetAlgoSkip{skip command}`.

`\AlCapSkip` is the dimension of the distance between algorithm body and caption in *plain* and *boxed* mode. You can change by hands or by using `\setalcapskip{0ex}`.

`\setalcapskip{length}` set the lenght of `\AlCapSkip`) dimension between algorithm body and caption.

`\setalcaphskip{length}` set the horizontal skip before Algorithm: in caption when used in ruled algorithm.

`\dontprintsemicolon` the ';' are no more printed at the end of each line.

`\printsemicolon` prints a '; ' at the end of each line (by default)

`\SetVline` prints a vertical line followed by a little horizontal line between the start and the end of each block. Looks like that : ⌊

`\SetNoline` Doesn't print vertical lines (by default). The block is marked with keywords such as *begin*, *end*.

`\SetLine` prints vertical lines between bloc start-end keywords as *begin*, *end*.

`\SetKwSty{<font>}` sets the Kw typography to <font> (by default: **textbf**).

`\SetFuncSty{<font>}` sets the function typography (by default: **texttt**).

`\SetArgSty{<font>}` sets the argument typography (by default: **emph**).

`\SetProcNameSty{<font>}` sets caption typography of procedure and function environment (by default the same as `\FuncSty{}`).

`\SetProcArgSty{<font>}` sets argument typography of procedure and function environment (by default the same as `\ArgSty{}`).

`\SetDataSty{<font>}` sets the data typography (by default: **textsf**).

`\SetCommentSty{<font>}` sets the comment text typography (by default: **texttt**).

`\SetAlFnt{<font>}` sets the font used by algorithm text.

`\SetAlCapFnt{<font>}` sets the font used by caption text.

`\SetAlTitleFnt{<font>}` sets the font used for {algorithm: } in caption or tile of algorithm (default is set to `textbf`).

`\Setnlsty{<font>}{<txt before>}{<txt after>}` defines how to print line numbers:
will print {<font> <txt bef> thelinenumber <txt aft>}.
By default `\Setnlsty{textbf}{}{}`.

`\SetTitleSty{type style}{type size}` sets the typography and size of the titles defined with the macro `\Titleofalgo{}` (not with \caption).

`\nocaptionofalgo` doesn't print Algorithm and its number in the caption. This macros is **ONLY** active for *"algoruled"* or *"ruled"* algorithms and for the next algorithm. For example, it is useful when the algorithm just describes a function and you only want to display the name of the function in the caption.

`\restorecaptionofalgo` restores correct captions that was corrupted by a `\nocaptionofalgo` macro.

`\restylealgo{style}` sets the style of the following algorithms to that given by this macro (plain, boxed, ruled, algoruled) unlike those indicated in the options of the package (see options of the package).

`\SetInd{before rule space}{after rule space}` sets the size of the space before the vertical rule and after. In `\NoLine` mode the indentation space is the sum of these two values, by default 0.5em and 1em

`\Setvlineskip{length}` sets the value of the vertical space after the little horizontal line which closes a block in vlined mode.

`\Setnlskip{length}` sets the value of the space between the line numbers and the text, by default 1em.

`\algomargin` this is the value of the margin of all algorithms. You can change it by setting: `\setlength{\algomargin}{2em}` for example. The default value is the sum of the two dimensions `\leftskip` and `\parindent` when the algorithm2e package is loaded. Note that if you change this value, it will take effect with the next algorithm environment. So even if you change it *inside* an algorithm environment, it will not affect the current algorithm.

`\incmargin{length}` increases the size of the `\algomargin` by the length given in argument.

`\decmargin{length}` decreases the size of the `\algomargin` by the length given in argument.

`\decmargin{length}` decreases the size of the `\algomargin` by the length given in argument.

`\SetSideCommentLeft` equivalent to scleft option.

`\SetSideCommentRight` equivalent to scright option.

`\SetFillComment` equivalent to fillcomment option.

`\SetNoFillComment` equivalent to nofillcomment option.

7 The predefined language keywords

Here are the english keywords predefined in the package. There are other language predefined macros provided, such as french keywords, see section 9 for a list of other language keywords. All these keywords are defined using macros provided by the package and described in section 7.1.

1. Input, output macros...
   - \KwData{input}
   - \KwResult{output}
   - \KwIn{input}
   - \KwOut{output}
2. One simple common keyword:
   - \KwTo
3. One keyword requiring an argument:
   - \KwRet{[value]}
   - \Return{[value]}
4. A block:
   - \Begin{block inside}
   - \Begin(*begin comment*){block inside}
5. Comments:
   - \tcc{line(s) of comment}: comment “ la” C
   - \tcc*{right justified side comment}: comment “ la” C
   - \tcc*[r]{right justified side comment}: comment “ la” C
   - \tcc*[l]{left justified side comment}: comment “ la” C
   - \tcc*[h]{left justified comment to be put in place}: comment “ la” C
   - \tcc*[f]{right justified comment to be put in place}: comment “ la” C
   - \tcp{line(s) of comment}: comment “ la” C++
   - \tcp*{right justified side comment}: comment “ la” C++
   - \tcp*[r]{right justified side comment}: comment “ la” C++
   - \tcp*[l]{left justified side comment}: comment “ la” C++
   - \tcp*[h]{left justified comment to be put in place}: comment “ la” C++

   - \tcp*[f]{right justified comment to be put in place}: comment “ la” C++

6. ”if-then-else” macros:

   - \If{condition}{then block}
   - \If(*then comment*){condition}{then block}
   - \uIf{condition}{then block without end}
   - \uIf(*then comment*){condition}{then block without end}
   - \lIf{condition}{then’s line text}
   - \lIf(*if comment*){condition}{then’s line text}
   - \ElseIf{elseif block}
   - \ElseIf(*elseif comment*){elseif block}
   - \uElseIf{elseif block without end}
   - \uElseIf(*elseif comment*){elseif block without end}
   - \lElseIf{elseif’s line text}
   - \lElseIf(*elseif comment*){elseif’s line text}
   - \Else{else block}
   - \Else(*else comment*){else block}
   - \uElse{else block without end}
   - \uElse(*else comment*){else block without end}
   - \lElse{else’s line text}
   - \lElse(*else comment*){else’s line text}
   - \eIf{condition}{then block}{else block}
   - \eIf(*then comment*){condition}{then block}(*else comment*){else block}
   - \eIf(*then comment*){condition}{then block}{else block}
   - \eIf{condition}{then block}(*else comment*){else block}

7. multiple condition selection:

   - \Switch(*switch comment*){condition}{Switch block}
   - \Switch{condition}{Switch block}
   - \Case{a case}{case block}
   - \Case(*case comment*){a case}{case block}
   - \uCase{a case}{case block without end}
   - \uCase(*case comment*){a case}{case block without end}
   - \lCase{a case}{case’s line}
   - \lCase(*case comment*){a case}{case’s line}
   - \Other{otherwise block}
   - \Other(*other comment*){otherwise block}
   - \lOther{otherwise’s line}
   - \lOther(*other comment*){otherwise’s line}

8. loops with ”end condition” test at the beginning:

   - \For{condition}{text loop}
   - \For(*for comment*){condition}{text loop}

- \lFor{condition}{line text loop}
- \lFor(*for comment*){condition}{line text loop}

- \While{condition}{text loop}
- \While(*while comment*){condition}{text loop}
- \lWhile{condition}{line text loop}
- \lWhile(*while comment*){condition}{line text loop}

- \ForEach{condition}{text loop}
- \ForEach(*foreach comment*){condition}{text loop}
- \lForEach{condition}{line text loop}
- \lForEach(*foreach comment*){condition}{line text loop}

- \ForAll{condition}{text loop}
- \ForAll(*forall comment*){condition}{text loop}
- \lForAll{condition}{line text loop}
- \lForAll(*forall comment*){condition}{line text loop}

9. loops with "end condition" test at the end:
   - \Repeat{end condition}{text loop}
   - \Repeat(*repeat comment*){end condition}{text loop}(*until comment*)
   - \Repeat(*repeat comment*){end condition}{text loop}
   - \Repeat{end condition}{text loop}(*until comment*)
   - \lRepeat{end condition}{line text loop}
   - \lRepeat(*repeat comment*){end condition}{line text loop}

Here we describe how they are obtained:

1. `\SetKwInput{KwData}{Data}`
   `\SetKwInput{KwResult}{Result}`
   `\SetKwInput{KwIn}{Input}`
   `\SetKwInput{KwOut}{Output}`
2. `\SetKw{KwTo}{to}`
3. `\SetKw{KwRet}{return}`
   `\SetKw{Return}{return}`
4. `\SetKwBlock{Begin}{begin}{end}`
5. `\SetKwComment{tcc}{/*}{*/}`
   `\SetKwComment{tcp}{//}{}`
6. `\SetKwIF{If}{ElseIf}{Else}{if}{then}{else if}{else}{endif}`
7. `\SetKwSwitch{Switch}{Case}{Other}{switch}{do}{case}{otherwise}{endsw}`
8. `\SetKwFor{For}{for}{do}{endfor}`
   `\SetKwFor{While}{while}{do}{endw}`
   `\SetKwFor{ForEach}{foreach}{do}{endfch}`
   `\SetKwAll{ForEach}{forall}{do}{endfall}`
9. `\SetKwRepeat{Repeat}{repeat}{until}`

7.1 To define your own language keywords

Note that all these macros verify if the keywords are already defined and do a renewcommand if they are. So you can overload the default definitions of this package with your own.

`\SetKw{Kw}{thetext}` defines the macro `\Kw` which defines a keyword *thetext* and prints it in keyword typography. It can take one argument: *backslash*Kw{arg}. If so, *arg* is printed in argument typography.

`\SetKwData{Kw}{thetext}` defines the macro `\Kw{w}`hich defines a data text. Prints *thetext* in data typography. Note that this macros can takes one argument as function macros.

`\SetKwInput{Kw}{input}` defines the macro `\Kw{arg}` which prints *input* followed by ':' in key word typography, and behind the argument *arg*. Typically used to define macros such as `\Input{data}` or `\Output{result}`. Note that *arg* will be shifted so that all the text is vertically aligned and to the right of the ':'.

`\SetKwInOut{Kw}{input}` works as `\SetKwInput{Kw}{input}`. But the position of the ':' is fixed and set by the longest keyword defined by this macro.

`\ResetInOut{input}` resets the position of the ':' for all macros defined previously by `\SetKwInOut{Kw}{input}`. The new position is fixed depending on the size of the text *input* given in argument.

`\SetKwInParam{Kw}{text1}{text2}` defines the macro `\Kw{name}{arg}` which prints *name* followed by *text1* in key word typography, behind the argument *arg*, followed by *text2* in key word typography. Typically used when the algorithm described a function in order to define a macro which allows to give the name of the function and to show its arguments. Note that *arg* will be shifted so that all the text is vertically aligned and to the right of *text1*. Example \SetKwInParam{Func}{(}{)} defines macro \Func and \Func{function}{arg1, arg2, arg3} gives **function(**arg1, arg2, arg3**)**.

`\SetKwFunction{KwFn}{Fn}` defines a macro `\KwFn{arg}` which prints *Fn* in Function typography and its argument *arg* in argument typography, surrounded by a pair of parentheses.

`\SetKwFunction{Dothat}{Do that}` defines the macro `\DoThat{this}`, which is equivalent to `\FuncSty{Do that(}\ArgSty{this}\FuncSty{)}` which gives: `Do that(`*this*`)`.

Note that you can also use it without arguments, it will be printed without '()', example: `\SetKwFunction{Fn}{TheFunction}` use as `\Fn` gives `TheFunction`.

Keywords (with or without arguments) and functions defined previously in normal text (not in an algorithm environment) can be used outside an algorithm environment. You can use it by typing `\DoThat{toto}` (for a function defined by `\SetKwFunction{Dothat}{Do that}`), you will obtain `Do That(`*toto*`)`.

`\SetKwBlock{Begin}{begin}{end}` defines a macro `\Begin{txt}` which denotes a block. The text is surrounded by the words *begin* and *end* in keyword typography and shifted to the right (indented). In `\Vline` *or* `\Line` *mode* a straight vertical line is added.
`\Begin(side text){text}` gives also text in a block surrounded by *begin* and *end*, but *side text* if put after the *begin* keyword. Combined with `\tcc*[f]` macro, it allows you to put comments on the same line as *begin.*

`\SetKwComment{Comment}{start}{end}` defines a macr `\Comment{text comment}` which writes *text comment* between *start* and *end.* Note that *start* or *end* can be empty.
It defines also `\Comment*{side comment text}` macro which allows to put comment on the same line as the code. This macro can take various option to control its behaviour:
`\Comment*[r]{side comment text}` put the end of line mark (';' by default) and side comment text just after and right justified, then end the line. It is the default.
`\Comment*[l]{side comment text}` same thing but side comment text is left justified.

`\Comment*[h]{side comment text}` put side comment right after the text. No end of line mark is put, and line is not terminated (is up to you to put `\;` to end the line).
`\Comment*[f]{side comment text}` same as the previous one but with side comment text right justified.

`\SetKwIF{If}{ElseIf}{Else}{if}{then}{else if}{else}{endif}` defines several macros to give the opportunity to write all if-then-else-elseif-endif possibilities:

- `\If{cond}{Then's text}`
  Then's text is writen in a block (below `then` and on several lines) and terminating by the `endif` given in the last argument.
- `\ElseIf{ElseIf's text}`
  ElseIf's text is writen in a block and terminating by the `endif`.
- `\Else{Else's text}`
  Else's text is writen in a block and terminating by the `endif`.
- `\lIf{cond}{Then's text}`
  Then's text is written on the same line as `then`. No `endif` is printed.
- `\lElseIf{ElseIf's text}`
  ElseIf's text is written on the same line as `else if`. No `endif` is printed.
- `\lElse{Else's text}`
  Else's text is written on the same line as `else`. No `endif` is printed.
- `\uIf{cond}{Then's text}` (for uncomplete if)
  defines a If block unterminated like in a `\eIf` block, i.e. don't print the `endif` or don't put the little horizontal line in *Vline* mode (see examples below).
- `\uElseIf{ElseIf's text}` (for uncomplete elseif)
  Same explanation as for `\uIf` but with `else if`.
- `\uElse{Else's text}` (for uncomplete else)
  Same explanation as for `\uElseIf` but with `else`.
- `\eIf{cond}{Then's text}{Else's text}`
  equivalent to the use of `\uIf` followed by `\Else`.

The macros which begin with a 'l' (l as line) denote that the text passed in argument will be printed on the same line while with the others the text is printed in a block and shifted. You should put `\;` at the end of "l macros".

The macros which begin with a 'u' (u as uncomplete) denote that the text passed in argument will be printed in a block not terminated by endif. They are useful to chain different alternatives.

The keywords *then* and *else* are automatically printed. *cond* is always printed in argument typography just behind the keyword if.

All this macros can be combined with () and `\Comment*` macros to put comments after main keywords as If, Else or ElseIf (see list of predefined keywords above and example below).

Some examples with `\SetKwIF{If}{ElseIf}{Else}{if}{then}{else if}{else}{endif}` the default definition given in the package:

```
\SetVline
\eIf{cond1}{
  a line\;
  a line\;
}{
  another line\;
  another line\;
}
```

⟹

```
if cond1 then
  | a line;
  | a line;
else
  | another line;
  |_ another line;
```

—

```
\SetNoline
\If{cond2}{
  second if\;
  second if\;
}
```

⟹

```
if cond2 then
    second if;
    second if;
end
```

—

```
\lIf{cond4}{ok} \lElse{wrong}\;
```

⟹

```
if cond4 then ok else wrong;
```

—

```
\SetVline
\lIf{cond5}{cond5 true}\;
\uElseIf{cond51}{
  cond 5 false\;
  but cond51 true\;
}
\ElseIf{}{
  all is wrong\;
  \Return result52\;
}
```

⟹

```
if cond5 then cond5 true;
else if cond51 then
  | cond 5 false;
  | but cond51 true;
else if then
  | all is wrong;
  |_ return result52;
```

—

```
\SetLine
\uIf{cond6}{
  cond6 is ok\;
  always ok\;
}
\uElseIf{cond62}{
  choose result62\;
  \Return result62\;
}
\Else{
  all is wrong\;
  do something else\;
}
```

⟹

```
if cond6 then
  | cond6 is ok;
  | always ok;
else if cond62 then
  | choose result62;
  | return result62;
else
  | all is wrong;
  | do something else;
end
```

—

```
Let's have a look at what we can do
with if-then-else and side comments\;
\eIf{if-then-else test}{
  no comment here\;
  neither in then\;
}{
  nor in else\;
}
\eIf(\tcc*[f]{then comment}){test}{
  then with a comment\;
}(\tcc*[f]{comment in else})
{
  here we are in else\;
}
\eIf(\tcc*[f]{then comment}){test}{
  again a comment in then\;
}{
  but not in else\;
}
\eIf{if-then-else test}{
  this time, no comment in then\;
}(\tcc*[f]{else comment})
{
  but one comment in else\;
}
Let's try with other if possibilities\;
\lIf(\tcc*[h]{lif comment}){test}{text}
\uIf(\tcc*[f]{uif comment}){test}{
  then text\;
}
\uElseIf(\tcc*[f]{comment}){test}{
  elseif text\;
}
\lElseIf(\tcc*[h]{comment}){test}{text}
\lElse(\tcc*[f]{comment}){text}
```

⟹

```
Let's have a look at what we can do
with if-then-else and side comments;
if if-then-else test then
    no comment here;
    neither in then;
else
    nor in else;
end
if test then       /* then comment */
    then with a comment;
else             /* comment in else */
    here we are in else;
end
if test then       /* then comment */
    again a comment in then;
else
    but not in else;
end
if if-then-else test then
    this time, no comment in then;
else                /* else comment */
    but one comment in else;
end
Let's try with other if possibilities;
if test then text; /* lif comment */
if test then         /* uif comment */
    then text;
else if test then        /* comment */
    elseif text;
else if test then text; /* comment */
else text;               /* comment */
```

\SetKwSwitch{Switch}{Case}{Other}{switch}{do}{case}{otherwise}{endsw} defines several macros to give a complete Switch-do-case-otherwise environment:

- \Switch{iden}{switch's block}
- \Case{cond}{Case's text}
- \uCase{cond}{Case's text}
- \lCase{cond}{Case's text}
- \Other{Otherwise's text}
- \lOther{Otherwise's text}

The keywords *do* and *endsw* are automatically printed. *iden* and *cond* are always printed in argument typography just behind the keywords Switch, Case and Otherwise. Here is an example with the default keywords:

```
\Switch{the value of T}{
  \uCase{a value}{
    do this\;
    do that\;
  }
  \lCase{another value}{one line}\;
  \Case{last value}{
    do this\;
    break\;
  }
  \Other{
    for the other values\;
    do that\;
  }
}
```

⟹

```
switch the value of T do
    case a value
        do this;
        do that;
    case another value one line;
    case last value
        do this;
        break;
    end
    otherwise
        for the other values;
        do that;
    end
end
```

As for If-then-elseif-else-endif macro, you can use () to put comments after main keywords.

`\SetKwFor{For}{for}{do}{endfor}` defines a loop environment with stop-test done at the beginning of the loop.

- `\For{loop's condition}{For's text}`
- `\lFor{loop's condition}{For's text}`

The keywords *do* and *endfor* are automatically printed. The loop condition is printed in argument typography. For example:

```
\SetLine
\ForAll{elements of $S_1$}{
  remove an element e from $S_1$\;
  put e in the set $S_2$\;
  }
\lFor{i=1 \emph{\KwTo}max}{mark i}\;
\ForEach{$e$ in the set}{
  put $e$ in ${\cal E}$\;
  mark $e$\;
}
```

⟹

```
forall elements of $S_1$ do
  | remove an element e from $S_1$;
  | put e in the set $S_2$;
end
for i=1 to max do mark i;
foreach e in the set do
  | put $e$ in $\mathcal{E}$;
  | mark $e$;
end
```

As for If-then-elseif-else-endif macro, you can use () to put comments after main keywords.

`\SetKwRepeat{Repeat}{repeat}{until}` defines a repeat-until environment (loop with stop-test at the end of the loop):

- `\Repeat{end loop condition}{the loop}`
- `\lRepeat{end loop condition}{only one line}`

It prints the loop condition behind the *until* after the text of the loop.For example:

```
\Repeat{this stop condition}{
  the text of the loop\;
  another line\;
  always in the loop\;
  }
\lRepeat{stop}{a one line loop}
```

⟹

```
repeat
    the text of the loop;
    another line;
    always in the loop;
until this stop condition ;
repeat a one line loop until stop
```

As for If-then-elseif-else-endif macro, you can use () to put comments after main keywords.

8 Two complete examples

The algorithms 2 and 3 are written with this package.

8.1 Algorithm 2 : disjoint decomposition

Here we suppose that we have done:

```
\usepackage[lined,algonl,boxed]{algorithm2e}
```

The algorithm was written in LaTeX2e code as follow:

```
\incmargin{1em}
\restylealgo{boxed}\linesnumbered
\begin{algorithm}
  \SetKwData{Left}{left}
  \SetKwData{This}{this}
  \SetKwData{Up}{up}
  \SetKwFunction{Union}{Union}
  \SetKwFunction{FindCompress}{FindCompress}
  \SetKwInOut{Input}{input}
  \SetKwInOut{Output}{output}
  \caption{disjoint decomposition}

  \Input{A bitmap $Im$ of size $w\times l$}
  \Output{A partition of the bitmap}
  \BlankLine
  \emph{special treatment of the first line}\;
  \For{$i\leftarrow 2$ \KwTo $l$}{
    \emph{special treatment of the first element of line $i$}\;
    \For{$j\leftarrow 2$ \KwTo $w$}{\nllabel{forins}
      \Left$\leftarrow$ \FindCompress{$Im[i,j-1]$}\;
      \Up$\leftarrow$ \FindCompress{$Im[i-1,]$}\;
      \This$\leftarrow$ \FindCompress{$Im[i,j]$}\;
      \If{\Left compatible with \This}{
        \lIf{\Left $<$ \This}{\Union{\Left,\This}}\;
        \lElse{\Union{\This,\Left}}
      }
      \If{\Up compatible with \This}{
        \lIf{\Up $<$ \This}{\Union{\Up,\This}}\;
        \lElse{\Union{\This,\Up}}
      }
    }
  \lForEach{element $e$ of the line $i$}{\FindCompress{p}}
  }
  \label{algo_disjdecomp}
\end{algorithm}
\decmargin{1em}
```

which gives the algorithme 2 on the following page where line 4 denotes the second `For`:

```
input  : a bitmap $im$ of size $w \times l$.
output: A partition of the bitmap.
special treatment of the first line;
for $i \leftarrow 2$ to $l$ do
    special treatment of the first element of line $i$;
    for $j \leftarrow 2$ to $w$ do
        left $\leftarrow$ FindCompress($Im[i, j-1]$);
        up $\leftarrow$ FindCompress($Im[i-1,]$);
        this $\leftarrow$ FindCompress($Im[i, j]$);
        if left compatible with this then
            if left $<$ this then Union(left,this);
            else Union(this,left)
        end
        if up compatible with this then
            if up $<$ this then Union(up,this);
            else Union(this,up)
        end
    end
    foreach element $e$ of the line $i$ do FindCompress($p$)
end
```

Algorithm 2: disjoint decomposition

8.2 Algorithm 3 : IntervalRestriction

Here we suppose we that have done:

`\usepackage[ruled,vlined]{algorithm2e}`

and the LaTeX2e code is:

```
\begin{algorithm}
\dontprintsemicolon
\KwData{$G=(X,U)$ such that $G^{tc}$ is an order.}
\KwResult{$G'=(X,V)$ with $V\subseteq U$ such that $G'^{tc}$ is an
interval order.}
\Begin{
  $V \longleftarrow U$\;
  $S \longleftarrow \emptyset$\;
  \For{$x\in X$}{
    $NbSuccInS(x) \longleftarrow 0$\;
    $NbPredInMin(x) \longleftarrow 0$\;
    $NbPredNotInMin(x) \longleftarrow |ImPred(x)|$\;
    }
  \For{$x \in X$}{
    \If{$NbPredInMin(x) = 0$ {\bf and} $NbPredNotInMin(x) = 0$}{
      $AppendToMin(x)$}
    }
    \lnl{InRes1}\While{$S \neq \emptyset$}{
    \lnlset{InResR}{REM}%
    remove $x$ from the list of $T$ of maximal index\;
    \lnl{InRes2}\While{$|S \cap  ImSucc(x)| \neq |S|$}{
      \For{$ y \in  S-ImSucc(x)$}{
        \{ remove from $V$ all the arcs $zy$ : \}\;
        \For{$z \in  ImPred(y) \cap  Min$}{
          remove the arc $zy$ from $V$\;
          $NbSuccInS(z) \longleftarrow NbSuccInS(z) - 1$\;
          move $z$ in $T$ to the list preceding its present list\;
          \{i.e. If $z \in T[k]$, move $z$ from $T[k]$ to
           $T[k-1]$\}\;
          }
        $NbPredInMin(y) \longleftarrow 0$\;
        $NbPredNotInMin(y) \longleftarrow 0$\;
        $S \longleftarrow S - \{y\}$\;
        $AppendToMin(y)$\;
        }
      }
    $RemoveFromMin(x)$\;
    }
  }
\caption{IntervalRestriction\label{IR}}
\end{algorithm}
```

which give us the algorithm 3 on the next page with line 1 and line REM.

Algorithm 3: IntervalRestriction

```
Data: $G = (X, U)$ such that $G^{tc}$ is an order.
Result: $G' = (X, V)$ with $V \subseteq U$ such that $G'^{tc}$ is an interval order.
begin
    $V \longleftarrow U$
    $S \longleftarrow \emptyset$
    for $x \in X$ do
        $NbSuccInS(x) \longleftarrow 0$
        $NbPredInMin(x) \longleftarrow 0$
        $NbPredNotInMin(x) \longleftarrow |ImPred(x)|$
    for $x \in X$ do
        if $NbPredInMin(x) = 0$ and $NbPredNotInMin(x) = 0$ then
            $AppendToMin(x)$
1   while $S \neq \emptyset$ do
REM     remove $x$ from the list of $T$ of maximal index
2       while $|S \cap ImSucc(x)| \neq |S|$ do
            for $y \in S - ImSucc(x)$ do
                { remove from $V$ all the arcs $zy$ : }
                for $z \in ImPred(y) \cap Min$ do
                    remove the arc $zy$ from $V$
                    $NbSuccInS(z) \longleftarrow NbSuccInS(z) - 1$
                    move $z$ in $T$ to the list preceding its present list
                    {i.e. If $z \in T[k]$, move $z$ from $T[k]$ to $T[k-1]$}
                $NbPredInMin(y) \longleftarrow 0$
                $NbPredNotInMin(y) \longleftarrow 0$
                $S \longleftarrow S - \{y\}$
                $AppendToMin(y)$
        $RemoveFromMin(x)$
end
```

9 Other language predefined keywords

9.1 french keywords

Hey, I am a frenchy , so I have defined the same as in section 7 but in french.

1. \Donnees{données}
   \Res{résultats}
   \Entree{entrées}
   \Sortie{sorties}
2. \KwA
   \Retour{[valeur]}
3. \Deb{intérieur du bloc}
4. \eSi{condition}{bloc du alors}{bloc du sinon}
   \Si{condition}{bloc du alors}
   \uSi{condition}{bloc du alors sans fin}
   \lSi{condition}{ligne du alors}
   \SinonSi{condition}{bloc du sinonsi}
   \uSinonSi{condition}{bloc du sinonsi sans fin}
   \lSinonSi{condition}{ligne du sinonsi sans fin}
   \Sinon{bloc du sinon}
   \uSinon{bloc du sinon sans fin}
   \lSinon{ligne du sinon}
5. \Suivant{condition}{bloc du Suivant-cas-alors} \uCas{cas où}{bloc de ce cas sans fin}
   \Cas{cas où}{bloc de ce cas}
   \lCas{cas où}{ligne de ce cas}
   \Autre{bloc de l'alternative}
   \lAutre{ligne de l'alternative}
6. \Pour{condition}{bloc de la boucle}
   \lPour{condition}{ligne de la boucle}
7. \Tq{condition}{bloc de la boucle}
   \lTq{condition}{ligne de la boucle}
8. \PourCh{condition}{bloc de la boucle}
   \lPourCh{condition}{ligne de la boucle}
9. \PourTous{condition}{bloc de la boucle}
   \lPourTous{condition}{ligne de la boucle}
10. \Repeter{condition d'arrêt}{bloc de la boucle}
    \lRepeter{condition d'arrêt}{ligne de la boucle}

Here we describe how they are obtained:

1. \SetKwInput{Donnes}{Données}
   \SetKwInput{Res}{Résultat}
   \SetKwInput{Entree}{Entrées}
   \SetKwInput{Sortie}{Sorties}
2. \SetKw{KwA}{à}
   \SetKw{Retour}{retourner}
3. \SetKwBlock{Deb}{début}{fin}
4. \SetKwIF{Si}{SinonSi}{Sinon}{si}{alors}{sinon si}{alors}{finsi}
5. \SetKwSwitch{Suivant}{Cas}{Autre}{suivant}{faire}{cas où}{autres cas}{fin d'alternative}
6. \SetKwFor{Pour}{pour}{faire}{finpour}
7. \SetKwFor{Tq}{tant que}{faire}{fintq}
8. \SetKwFor{PourCh}{pour chaque}{faire}{finprch}
9. \SetKwFor{PourTous}{pour tous}{faire}{finprts}
10. \SetKwRepeat{Repeter}{répéter}{jusqu'à}

9.2 German keywords

- \Ein{Eingabe}
  \Aus{Ausgabe}
  \Daten{Daten}
  \Ergebnis{Ergebnis}
- \Bis{bis}
  \KwZurueck{zurück}
  \Zurueck{zurück}
- \Beginn{Beginn}
- \Wiederh{stop condition}{loop}
  \lWiederh{stop condition}{line loop}
- \eWenn{condition}{then text}{else text}
  \Wenn{condition}{then text}
  \uWenn{condition}{then text without end}
  \lWenn{condition}{then line}
  \SonstWenn{condition}{elseif text}
  \uSonstWenn{condition}{elseif text without end}
  \lSonstWenn{condition}{elseif line}
  \Sonst{else text}
  \uSonst{else text without end}
  \lSonst{else line}

- \Unterscheide{conditions}switch-case-default text\Fall{case of}{text}
  \uFall{case of}{text}
  \lFall{case of}{line text}
  \Anderes{default text}
  \lAnderes{default line}
- \Fuer{condition}{loop}
  \lFuer{condition}{line loop}
- \FuerPar{condition}{loop}
  \lFuerPar{condition}{line}
- \FuerJedes{condition}{loop}
  \lFuerJedes{condition}{line}
- \FuerAlle{condition}{loop}
  \lFuerAlle{condition}{line}Ende
- \Solange{condition}{loop}Ende
  \lSolange{condition}{line}

Here we describe how they are obtained:

- `\SetKwInput{Ein}{Eingabe}`
  `\SetKwInput{Aus}{Ausgabe}`
  `\SetKwInput{Daten}{Daten}`
  `\SetKwInput{Ergebnis}{Ergebnis}`
- `\SetKw{Bis}{bis}`
  `\SetKw{KwZurueck}{zurück}`
  `\SetKw{Zurueck}{zurück}`
- `\SetKwBlock{Beginn}{Beginn}{Ende}`
- `\SetKwRepeat{Wiederh}{wiederhole}{bis}`
- `\SetKwIF{Wenn}{SonstWenn}{Sonst}{wenn}{dann}{sonst wenn}{sonst}{Ende}`
- `\SetKwSwitch{Unterscheide}{Fall}{Anderes}{unterscheide}{tue}{Fall}{sonst}{Ende.}`
- `\SetKwFor{Fuer}{für}{tue}{Ende}`
- `\SetKwFor{FuerPar}{für}{tue gleichzeitig}{Ende}`
- `\SetKwFor{FuerJedes}{für jedes}{tue}{Ende}`
- `\SetKwFor{FuerAlle}{für alle}{tue}{Ende}`
- `\SetKwFor{Solange}{solange}{tue}{Ende}`

9.3 Portuguese keywords

- \Entrada{Entrada}
  \Saida{Saída}
  \Dados{Dados}
  \Resultado{Resultado}
- \Ate
  \KwRetorna{[val]}
  \Retorna{[val]}
- \Iniciob{inside block}
- \eSe{condition}{then block}{else block}
  \Se{condition}{then block}
  \uSe{condition}{then block without end}
  \lSe{condition}{then's line text}
  \Senao{else block}
  \uSenao{else block without else}
  \lSenao{else's line text}
  \SenaoSe{condition}{elseif block}
  \uSenaoSe{condition}{elseif block without end}
  \lSenaoSe{condition}{elseif's line text}
- \Selec{condition}{Switch block}
  \Caso{a case}{case block}
  \uCaso{a case}{case block without end}
  \lCaso{a case}{case's line}
  \Outro{otherwise block}
  \lOutro{otherwise's line}
- \ParaPar{condition}{text loop}
  \lParaPar{condition}{line text loop}
- \ParaCada{condition}{text loop}
  \lParaCada{condition}{line text loop}
- \ParaTodo{condition}{text loop}
  \lParaTodo{condition}{line text loop}
- \Enqto{stop condition}{text loop}
  \lEnqto{stop condition}{text loop}
- \Repita{stop condition}{text loop}
  \lRepita{stop condition}{line of the loop}

Here we describe how they are obtained:

1. `\SetKwInput{Entrada}{Entrada}`
   `\SetKwInput{Saida}{Saída}`
   `\SetKwInput{Dados}{Dados}`
   `\SetKwInput{Resultado}{Resultado}`
2. `\SetKw{Ate}{até}` `\SetKw{KwRetorna}{retorna}`
   `\SetKw{Retorna}{retorna}`
3. `\SetKwBlock{Inicio}{início}{fim}`
4. `\SetKwIF{Se}{SenaoSe}{Senao}{se}{então}{senão se}{senão}{fim se}`
5. `\SetKwSwitch{Selec}{Caso}{Outro}{selecione}{faça}{caso}{senão}{fim selec}`
6. `\SetKwFor{Para}{para}{faça}{fim para}`
7. `\SetKwFor{ParaPar}{para}{faça em paralelo}{fim para}`
8. `\SetKwFor{ParaCada}{para cada}{faça}{fim para cada}`
9. `\SetKwFor{ParaTodo}{para todo}{faça}{fim para todo}`
10. `\SetKwFor{Enqto}{enquanto}{faça}{fim enqto}`
11. `\SetKwRepeat{Repita}{repita}{até}`

9.4 Some Czech keywords

Here are some czech keywords, please feel free to send me the others.

- \Vst
- \Vyst
- \Vysl

How they are obtained:

1. `\SetKwInput{Vst}`Vstup
2. `\SetKwInput{Vyst}`Výstup
3. `\SetKwInput{Vysl}`Výsledek

10 Known bugs

- no more known bugs actually; if you find one, please sent it to me.

List of Algorithms

Index