algorithm2e.sty — package for algorithms

release 4.01

Report bugs and comments to christophe.fiorio@univ-montp2.fr
algorithm2esty-annonce@lirmm.fr mailing list for announcements
algorithm2esty-discussion@lirmm.fr mailing list for discussion*†‡§¶‖*∗‡‡

december 14 2009

Contents

*The author is very grateful to David Carlisle, one of the authors of the LaTeX Companion book, for his advices
†Martin Blais for his suggestions
‡David A. Bader for his new option `noend`
§Gilles Geeraerts for his new command `SetKwIfElseIf`
¶Ricardo Fukasawa for the portuguese keywords
‖Christian Icking for the german translation of keywords
**Arnaud Giersch for his suggestions and corrections on SetKwComments
††Nathan Tallent for his \SetKwHangingKw macro
‡‡and the many users as Jean-Baptiste Rouquier 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. See section 3 for two long examples of algorithms written with this package.

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.

Changes from one release to the next are indicated in release notes at the beginning of the packages. For release 4.0, changes are indicated at the end of this document.

2 How to use it: abstract

You must set \usepackage[*options*]{algorithm2e} before \begin{document} command. The available options are described in section 6.

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]
  \SetAlgoLined
  \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}
```

[1] For longer and more complexe examples see section 3

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

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, except that it should be located at the end of the algorithm. It 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.

3 Two more detailed examples

The algorithm 2 and algorithm 3 are written with this package.

3.1 Algorithm disjoint decomposition

Here we suppose that we have done:

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

The algorithm 2 was written in LaTeX2e code as presented next page. You can label lines, and for example line 4 denotes the second `For` (see `\label` command in the example). Notice also some ways of doing comments at lines 8, 12, 14 and 15. Star comment commands are for comment on lines of code, else comment is a line by itself as at line 14. The different option in star comments defines if it is left (`l` and `h`) or right justified (`r` and `f`). The first ones (`l` and `r`) add ; at the end of line code, the second ones (`f` and `h`) doesn't. These last are useful when used in side comment (introduced by `()`) of alternatives of loops keyword commands.

input : A bitmap Im of size $w \times l$
output: A partition of the bitmap

```
1  special treatment of the first line;
2  for i ← 2 to l do
3  |  special treatment of the first element of line i;
4  |  for j ← 2 to w do
5  |  |  left ← FindCompress(Im[i, j − 1]);
6  |  |  up ← FindCompress(Im[i − 1, ]);
7  |  |  this ← FindCompress(Im[i, j]);
8  |  |  if left compatible with this then // O(left,this)==1
9  |  |  |  if left < this then Union(left,this);
10 |  |  |  else Union(this,left);
11 |  |  end
12 |  |  if up compatible with this then                    // O(up,this)==1
13 |  |  |  if up < this then Union(up,this);
14 |  |  |  // this is put under up to keep tree as flat as possible
15 |  |  |  else Union(this,up);                          // this linked to up
16 |  |  end
17 |  end
18 |  foreach element e of the line i do FindCompress(p)
19 end
```

Algorithm 2: disjoint decomposition

```
\IncMargin{1em}
\begin{algorithm}
  \SetKwData{Left}{left}\SetKwData{This}{this}\SetKwData{Up}{up}
  \SetKwFunction{Union}{Union}\SetKwFunction{FindCompress}{FindCompress}
  \SetKwInOut{Input}{input}\SetKwInOut{Output}{output}

  \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 $w$}{\label{forins}
      \Left$\leftarrow$ \FindCompress{$Im[i,j-1]$}\;
      \Up$\leftarrow$ \FindCompress{$Im[i-1,]$}\;
      \This$\leftarrow$ \FindCompress{$Im[i,j]$}\;
      \If(\tcp*[h]{O(\Left,\This)==1}){\Left compatible with \This}{\label{lt}
        \lIf{\Left $<$ \This}{\Union{\Left,\This}}\;
        \lElse{\Union{\This,\Left}\;}
      }
      \If(\tcp*[f]{O(\Up,\This)==1}){\Up compatible with \This}{\label{ut}
        \lIf{\Up $<$ \This}{\Union{\Up,\This}}\;
        \tcp{\This is put under \Up to keep tree as flat as possible}\label{cmt}
        \lElse{\Union{\This,\Up}}\tcp*[r]{\This linked to \Up}\label{lelse}
      }
    }
    \lForEach{element $e$ of the line $i$}{\FindCompress{p}}
  }
  \caption{disjoint decomposition}\label{algo_disjdecomp}
\end{algorithm}\DecMargin{1em}
```

3.2 Algorithm: IntervalRestriction

Here we suppose we that have done:

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

The LaTeX2e code on next page gives algorithm 3. Here lines are not autonumbered but you can number them individually with `\nl` command as for line 1 or line 2. You even can set your own reference with `\nlset` command and get back this reference by simply using classical `\ref`. For example `\ref{InResR}` gives 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 $T[k-1]$}
        $NbPredInMin(y) \longleftarrow 0$
        $NbPredNotInMin(y) \longleftarrow 0$
        $S \longleftarrow S - \{y\}$
        $AppendToMin(y)$
    $RemoveFromMin(x)$

```
\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} $NbPredNotInMin(x) = 0$}{
      $AppendToMin(x)$}
    }
    \nl\While{$S \neq \emptyset$}{\label{InRes1}
    \nlset{REM} remove $x$ from the list of $T$ of maximal index\;\label{InResR}
    \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}
```

4 Compatibility issues

Compatibily with other packages improven by changing name of internal macros. Algorithm2e can now be used with almost all package, as `elsart`, `hermes`, `arabtex` for example, if this last is loaded after algorithm2e package. So, at this time, release 4.01has few known compatibility problem with other packages. The packages or classes that are known to be not compatible with `algorithm2e` package is:

- `ascelike`
- `pstcol`

Nevertheless, when use with some packages, some of their options cannot be used, or you need to specify some particular options (as algo2e to change name of environment if `algorithm` is already defined by the class), either from `algorithm2e` package or from the other packages.

hyperref if you want to compile in PdfLaTeX, you must not use `naturalnames` option. **Beware** this has changed from release 3 where you should use it!

article-hermes is not compatible with relsize used by `algorithm2e` package, so you have to use norelsize option to get algorithm works with `article-hermes` class.

Note also that, if you use packages changing the way references are printed, you must define labels of algorithm **after** the caption to ensure a correct printing. You cannot use `\label` inside a caption without errors.

From release 4.0, some commands have been renamed to have consistent naming (CamlCase syntax) and old commands are no more available. If you doesn't want to change your mind or use old latex files, you have to use oldcommands option to enable old commands back. Here are these commands:

- `\SetNoLine` becomes `\SetAlgoNoLine`
- `\SetVline` becomes `\SetAlgoVlined`
- `\Setvlineskip` becomes `\SetVlineSkip`
- `\SetLine` becomes `\SetAlgoLined`
- `\dontprintsemicolon` becomes `\DontPrintSemicolon`
- `\printsemicolon` becomes `\PrintSemicolon`
- `\incmargin` becomes `\IncMargin`
- `\decmargin` becomes `\DecMargin`
- `\setnlskip` becomes `\SetNlSkip`
- `\Setnlskip` becomes `\SetNlSkip`
- `\setalcapskip` becomes `\SetAlCapSkip`
- `\setalcaphskip` becomes `\SetAlCapHSkip`
- `\nlSty` becomes `\NlSty`
- `\Setnlsty` becomes `\SetNlSty`
- `\linesnumbered` becomes `\LinesNumbered`
- `\linesnotnumbered` becomes `\LinesNotNumbered`

- \linesnumberedhidden becomes \LinesNumberedHidden
- \showln becomes \ShowLn
- \showlnlabel becomes \ShowLnLabel
- \nocaptionofalgo becomes \NoCaptionOfAlgo
- \restorecaptionofalgo becomes \RestoreCaptionOfAlgo
- \restylealgo becomes \RestyleAlgo
- \gIf macros and so on do no more exist

5 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).
- name of the procedure or function set in caption is automatically defined as a KwFunction and so can be used as a macro. For example, if inside a procedure environment you set \caption{myproc()}, you can use \myproc macro in you main text. Beware that the macro is only defined after the \caption!
- nokwfunc unable the feature described above in function and procedure environment. Useful if you use name of procedure or function that cannot be a command name as a math display for example.

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.

If you don't like algorithm or look for something else, you can change the name of algorithm by using command below:

\SetAlgorithmName{algorithmname}{algorithmautorefname}{list of algorithms name} which redefines name of the algorithms and the sentence list of algorithms. Example: \SetAlgorithmName{Protocol}{
of protocols} if you prefer protocol than algorithm. Second argument is the name that \autoref, from hyperref package, will use.

The same exists for procedure and function environment, the difference is that list of algorithms is not change and keep its original name:

\SetAlgoProcName{aname}{anautorefname} sets the name of Procedure printed by procedure environment (the environment prints Procedure by default). Second argument is the name that \autoref, from hyperref package, will use.

\SetAlgoFuncName{aname}{anautorefname} sets the name of Function printed by procedure environment (the environment prints Function by default). Second argument is the name that \autoref, from hyperref package, will use.

6 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. To be used with slide class in order to have correct margins.

norelsize: starting from this release v4.00, algorithm2e package uses relsize package in order to get relative size for lines numbers; but it seems that some rare classes (such as inform1.cls) are not compatible with relsize; to have algorithm2e working, this option makes algorithm2e not to load relsize package and go back to previous definition by using font for lines numbers.

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:*.

onelanguage: allows, if using standard keywords listed below, to switch from one language to another without changing keywords by using appropriate language option:

- KwIn, KwOut, KwData, KwResult
- KwTo KwFrom
- KwRet, Return
- Begin
- Repeat
- If, ElseIf, Else
- Switch, Case, Other
- For, ForPar, ForEach, ForAll, While

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

endfloat: endfloat packages doesn't allow float environment inside other environment. So using it with figure option of algorithm2e makes error. This option enables a new environment algoendfloat to be used instead of algorithm environment that put algorithm at the end. algoendfloat environment make algorithm acting as endfloat figures. This option load endfloat package, so it is required to have it.

algopart: algorithms are numbered within part numbers.

algochapter: algorithms are numbered within chapter numbers.

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

procnumbered: makes the procedure and function to be numbered as algorithm.

boxed: to have algorithms enclosed in a box.

boxruled: surround algorithm by a box, puts caption above and add a line after caption.

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.

tworuled: tworuled acts like ruled but doesn't put a line after the title.

plain: the default, with no feature.

lined: `\SetAlgoLined` becomes the default, see section 8.5 for explanations about the `\SetAlgoLined` macros.

vlined: `\SetAlgoVlined` becomes the default, see section 8.5 for explanations about the `\SetAlgoVlined` macros.

noline: `\SetNoline` becomes the default, see section 8.5 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.

rightnl: put lines numbers to the right of the algorithm instead of left.

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 right)

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

nokwfunc unable the setting in `\KwFunction` of procedure's or function's name (see section 5) of function and procedure environment. Useful if you use name of procedure or function that cannot be a command name as a math display for example.

7 Typesetting

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 10 (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.

8 Commands provided with the package

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.

8.1 basic typesetting commands

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

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

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

\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.

8.2 algorithm environment, caption, list of algorithms, ...

8.2.1 caption, title and changind reference of algorithms

Algorithm environment are float environment. So you can use classical \caption, \listofalgorithms{,} \label. If you want a title but not a caption (for example to not add an enter in the list of algorithm) you have \TitleOfAlgo{.} And if you want to name your algorithm and not number it, you can change the reference of it by using \SetAlgoRefName{ref}:

\caption{thetitle} works as classical caption of figures. It inserts an entry in the list of algorithms. Should be the standard way to put title of algorithms.

\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*.

\SetAlgoRefName{ref} which changes the default ref (number of the algorithm) by the name given in parameter. For example \SetAlgoRefName{QXY} sets reference of the algorithm to QXY. If you label your algorithm and reference it, you will get QXY. On the same manner, entry in the list of algorithm will name it QXY.

\SetAlgoRefRelativeSize{relative integer} which sets the output size of reference in list of algorithms for references set by \SetAlgoRefName. The default is \SetAlgoRefRelativeSize{-2}.

8.2.2 setting style and layout of algorithm, caption and title

The following commands help you to define the style and the layout of the caption:

\SetAlgoCaptionSeparator{sep} which sets the separator between title of algorithms (**Algorithm 1**) and the name of the algorithm. By default it's ':' and caption looks like "**Algorithm 2: name**" but now you can change it by using for example which will give "**Algorithm 3. name**".

\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} sets the lenght of \AlCapSkip) dimension between algorithm body and caption.

\SetAlCapHSkip{length} sets the horizontal skip before Algorithm: in caption when used in ruled algorithm.

\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.

\SetAlgoCaptionLayout{style} sets global style of the caption; style must be the name of a macro taking one argument (the text of the caption). Examples below show how to use it:

- \SetAlgoCaptionLayout{centerline} to have centered caption;
- \SetAlgoCaptionLayout{textbf} to have bold caption.

If you want to apply two styles in the same time, such as centered bold, you have to define you own macro and then use \SetAlgoCaptionLayout with its name. \AlCapFnt and \AlCapNameFnt can change the font used in caption, beware of interactions between this three commands.

Note that two length control the layout of ruled, algoruled, boxruled algorithms caption. \interspacetitleruled and \interspaceboxruled are described section 8.5.

8.3 line numbering

8.3.1 labelling and numbering lines

AlgoLine is the counter used to number the lines. It's a standard counter, so LaTeXcommands works with it.

linesnumbered, linesnumberedhidden and commentsnumbered (see above section 6) are the options controlling auto-numbering of lines. You can also control this feature manually and precisely with the following commands:

\LinesNumbered makes lines of the following algorithms be auto-numbered. This command corresponds to linesnumbered option.

\LinesNumberedHidden makes lines of the following algorithms be auto-numbered, but numbers stay hidden. You have to use \ShowLn and \ShowLnLabel to see them. This command corresponds to linesnumberedhidden option.

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

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

\nl numbers the line: must BEGIN the line. You can use \label to label the line and reference it further.

\lnl{label} numbers and labels the line : must BEGIN the line. Do a **Beware** this has changed from release 3\nl\label{label} in one time. Prefer to use a classical \label as it is more readable.

\nlset{text} works as \nl except that the additional argument is the text to put at the beginning of the line. This text becomes the reference if you label it and \ref will print it instead of classical number.

\lnlset{text}{label} works for \nlset as \lnl for \nl. Prefer to use a classical \label as it is more readable.

\ShowLn shows number of the line when linesnumberedhidden is activated.

\ShowLn{label} same as precedent but with a label. Prefer to use \ShowLn with a classical \label.

8.3.2 setting style of lines

The following command allows you to change the way line numbers are printed:

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

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

\SetAlgoNLRelativeSize{number} sets the relative size of line numbers. By default, line numbers are two size smaller than algorithm text. Use this macro to change this behavior. For example, \SetAlgoNlRelativeSize{0} sets it to the same size, \SetAlgoNlRelativeSize{−1} to one size smaller and \SetAlgoNlRelativeSize{1} to one size bigger.

8.4 standard styles

8.4.1 standard font shapes and styles

Almost every text in algorithm has his own style that can be customized. The following commands correspond to the different styles used by the package. They can be customized by using corresponding "\Set commands" (see section 8.4.3)

\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 to have algorithms typeset in small sf font. Default is nothing so algorithm is typeset as the text of the document.

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

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

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

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

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

\NlSty{<text>} sets <text> in number line typography.

\ProcNameSty{<text>} sets <text> in caption typography of procedure and function environment (by default the same as \AlCapSty{}).

\ProcArgSty{<text>} sets <text> in argument typography of procedure and function environment (by default the same as \AlCapNameSty{}).

8.4.2 caption and title font style

\AlCapSty, \AlCapNameSty, \AlCapFnt, \AlCapNameFnt and corresponding "\Set commands" (see section 8.4.4) \SetAlCapSty, \SetAlCapNameSty, \SetAlCapFnt, \SetAlCapNameFnt control the way caption is printed. \AlCapSty and \AlCapFnt are used to define style and font shape of "Algorithm #:" in caption. \AlCapNameSty and \AlCapNameFnt are used to define style and font shape of the caption text. In fact a caption \my algorithm{i}s printed as follow :
\AlCapSty{\AlCapFnt{Algorithm #:}}\AlCapNameSty{\AlCapNameFnt{my algorithm}}.
By default, \AlCapSty is textbf and \AlCapFnt is nothing. \AlCapNameSty keeps text as it is, and \AlCapNameFnt do nothing.

\AlCapSty{<text>} sets <text> in caption title typography, that is the same used, together with \AlCapFnt, to print Algorithm #:, more precisely it is printed as follow:
\AlCapSty{\AlCapFnt{Algorithm #:}}
which gives actually "**Algorithm #:**". By default \AlCapSty is textbf.

`\AlCapNameSty{<text>}` sets <text> in caption name typography, that is the same used, together with `\AlCapNameFnt` to print the name of caption you set by calling `\caption{name}`. More precisely it is printed as follow:
`\AlCapNameSty{\AlCapNameFnt{name}}`
which gives "name". By default `\AlCapNameSty` is `textnormal` which means print in standard text.

`\AlCapFnt{<text>}` sets <text> in font shape of caption title, that is the same used, together with `\AlCapSty`, to print `Algorithm #:`, more precisely it is printed as follow:
`\AlCapSty{\AlCapFnt{Algorithm #:}}`
which gives actually "**Algorithm #:**". By default `\AlCapFnt` is `\relax` which means keep text as it is.

`\AlCapNameFnt{<text>}` sets <text> in caption name typography, that is the same used, together with `\AlCapNameSty` to print the name of caption you set by calling `\caption{name}`. More precisely it is printed as follow:
`\AlCapNameSty{\AlCapNameFnt{name}}`
which gives "name". By default `\AlCapNameFnt` is `\relax` which means keep text as it is.

`\AlTitleSty{<text>}` is used to typeset "`Algorithm #:`" in title, together with `\AlTitleFnt`. You can use it to have text as your titles. Precisely, titles are typeset as follow:
`\AlTitleSty{\AlTitleFnt{Algorithm #:}}`.

`\AlTitleFnt{<text>}` is used to typeset "`Algorithm #:`" in title, together with `\AlTitleSty`. You can use it to have text as your titles. Precisely, titles are typeset as follow:
`\AlTitleSty{\AlTitleFnt{Algorithm #:}}`.

8.4.3 setting font standard font shapes and styles

With the following commands you can customize the style and have the look you want for your algorithms:

`\SetAlFnt{<font>}` define the fonts used for typesetting algorithms.

You have to give commands to set the font in argument. 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.

The next ones require to give in parameter name of a macro (whithout \) which takes one argument. For example, `\SetAlCapFnt{textbf}` (see section 8.2.2) defines the default behaviour of `\AlCapFnt`. If you want to do more complicated thing, you should define your own macro and give it to `\SetAlCapFnt` or `\SetAlCapNameFnt`. Here are two examples:

- `\newcommand{\mycapfn}[1]{\tiny #1}\SetAlCapNameFnt{mycapfnt}`
- `\newcommand{\mycapfn}[1]{\textsl{\small #1}}\SetAlCapNameFnt{mycapfnt}`

Here is the complete list of these macros:

`\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**).

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

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

`\SetNlSty{<font>}` sets the number line typography (by default: **\relsize{-2}**)

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

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

8.4.4 setting caption and title font style

The following commands allow to redefine Fnt macros. This ones requires to give directly commands that define the font shape you want. They works as \SetAlFnt{d}escribed above. For example you can do \SetAlCapFnt{\large\color{red}} to have Algorithm #: in caption printed in large red font.

\SetAlCapFnt{<font>} sets the font used for {algorithm: } in caption of algorithm (default is set to \relax).

\SetAlCapNameFnt{<font>} sets the font used by caption text. Default is \relax and text is kept as it is.

\SetAlTitleFnt{<font>} sets the font used in \TitleOfAlgo command (default is set to \relax, so text is kept as it is).

The next commands allow to redefine Sty macros for caption or title. As "\Set commands" of basic font style (see section 8.4.3), they require a name of a command in argument, this command have to take one argument, the text to be typeset. Examples of use:

- \newcommand{\mycapfn}[1]{\tiny #1}\SetAlCapNameFnt{mycapfnt}
- \newcommand{\mycapfn}[1]{\textsl{\small #1}}\SetAlCapNameFnt{mycapfnt}

Now the commands:

\SetAlCapSty{<commandname>}: sets the command that will be used by \AlCapSty to define style of Algorithm #: in caption. The argument is a name of a command (without \). This command have to take one argument, the text to be formatted. Default is set to: \SetAlCapSty{textbf}.

\SetAlCapNameSty{<commandname>}: sets the command that will be used by \AlCapNameSty to define style of caption text. The argument is a name of a command (without \). This command have to take one argument, the text to be formatted. Default is set to: \SetAlCapSty{textnormal}.

\SetAlTitleSty{<commandname>} sets the command that will be used by \AlTitleSty to define style of algorithm title given by \TitleOfAlgo (default is set to \SetAlTitleSty{textbf}).

Note that by combining Fnt and Sty macros you can define almost all styles easily. For example, the last example above can be define in a simplier way that previously presented by doing:

- \SetAlCapNameSty{textsl}\SetAlCapNameFnt{\small}

8.5 controlling the layout of algorithms

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

\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).

`\SetAlgoVlined` 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*.

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

`\SetAlgoLongEnd` acts like longend option.

`\SetAlgoShortEnd` acts like shortend option.

`\SetAlgoNoEnd` acts like noend option.

`\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.

`\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}`.

`\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.

`\SetAlgoNlRelativeSize{number}` sets the relative size of line number (see section 8.3) for more details on this command.

`\SetAlgoCaptionLayout{style}` sets the global style of caption (see section 8.2 for more details).

Some length are used to set the layout of ruled, algoruled and boxruled algorithms caption. These length have no particular macro to set them but can be changed by classical `\setlength` commmand:

interspacetitleruled (2pt by defaut) which controls the vertical space between rules and title in ruled and algoruled algorithms.

interspaceboxruled (2\lineskip by default) which controls the vertical space between rules and title in boxruled algorithms.

8.6 comments

There are two ways to do comments in algorithm :

1. by using a comment macro defined by \SetKwComment{command}{right mark}{left mark} (see below) like \tcc;
2. by using side comment, it means comment put in between ( ) after control command like if-then-else, for, ... macros.

At section 9.3, you can see how \tcc is defined and at section 9.4 you can look at some examples how to use it with `if then else` like commands and finally you can look at section 10.4 how to define comments and explanations on the different macros and ways of printing comments. Note also that comments are not numbered by default when using linesnumbered option. You have to set commentsnumbered to number them also.

The following macro control how comment are typeseted.

\SetSideCommentLeft right justified side comments (side comments are flushed to the right), equivalent to scleft option.

\SetSideCommentRight left justified side comments (side comments are put right after the code line) , equivalent to scright option.

\SetFillComment end mark of comment is flushed to the right so comments fill all the width of text, equivalent to fillcomment option.

\SetNoFillComment end mark of comment is put right after the comment, equivalent to nofillcomment option.

9 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 11 for a list of other language keywords. All these keywords are defined using macros provided by the package and described in section 10.

9.1 Input, output macros...

- \KwIn{input}
- \KwOut{output}
- \KwData{input}
- \KwResult{output}

9.2 basic keywords and blocks

1. One simple common keyword:
   - \KwTo
2. One keyword requiring an argument:
   - \KwRet{[value]}
   - \Return{[value]}
3. A block:
   - \Begin{block inside}
   - \Begin(*begin comment*){block inside}

9.3 comments

- \tcc{line(s) of comment}: comment " la" C
- \tcc*{right justified side comment}: comment " la" C
- \tcc*[r]{right justified side comment, ends the line (default)}: comment " la" C
- \tcc*[l]{left justified side comment, ends the line}: comment " la" C
- \tcc*[h]{left justified comment, without end line; useful with "if-then-else" macros for example}: comment " la" C
- \tcc*[f]{right justified comment, without end line; useful with "if-then-else" macros for example}: 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, ends the line (default)}: comment " la" C++
- \tcp*[l]{left justified side comment, ends the line}: comment " la" C++
- \tcp*[h]{left justified comment, without end line; useful with "if-then-else" macros for example}: comment " la" C++
- \tcp*[f]{right justified comment, without end line; useful with "if-then-else" macros for example}: comment " la" C++

You can see some examples of this macros with `if then else` at the end of section 9.4.

9.4 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}

9.5 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}

9.6 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}

9.7 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}

9.8 how default keywords 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}

10 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.

10.1 to define Input, output 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.

10.2 to define basic keywords or blocks

\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.

\SetKwHangingKw{name}{text} (hanging indent with keyword): This creates a hanging indent much like \SetKwInput, except that it removes the trailing ':' and does not reset numbering. It can be used for example to create **let** declarations.

```
text --------------   <= [text] is placed at left
     --------------   <= hanging determined by [text]
```

\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*.

10.3 to define keywords as function

If you want describe the function by an algorithm, use instead *function* or *procedure* environment.

`\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*`)`.

10.4 to define comments

`\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.

10.5 to define if-then-else macros

`\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:

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

⟹

if *cond1* **then**
  a line;
  a line;
else
  another line;
  another line;

```
\SetAlgoNoLine
\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;

```
\SetAlgoVlined
\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;

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

$\Longrightarrow$

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}
```

$\Longrightarrow$

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;
if *test* **then** `/* then comment */`
  then with a comment;
else `/* comment in else */`
  here we are in else;
if *test* **then** `/* then comment */`
  again a comment in then;
else
  but not in else;
if *if-then-else test* **then**
  this time, no comment in then;
else `/* else comment */`
  but one comment in else;
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 */`

10.6 to define multiple condition selection:

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

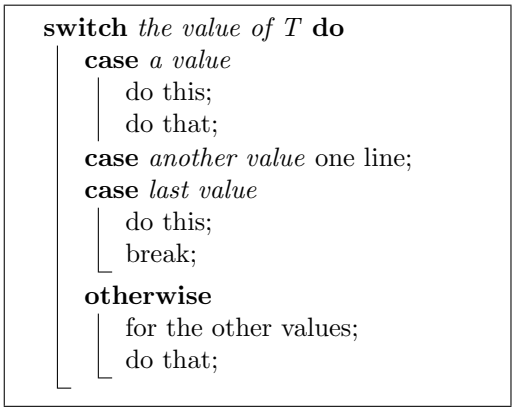
- `\Switch{iden}{switch's block}`

- `\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\;
  }
}
```

$\Longrightarrow$



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

10.7 to define loops with "end condition" test at the beginning

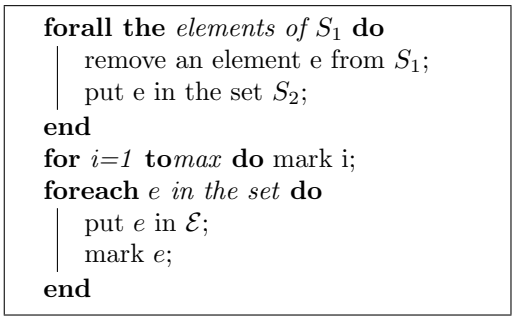
`\SetKwFor{For}{for}{do}{endfor}` defines a loop environment with stop-test done at the beginning of the loop.

- `\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:

```
\SetAlgoLined
\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 $e$\;
}
```

$\Longrightarrow$



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

10.8 to define loops with "end condition" test at the end

`\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}
```

$\Longrightarrow$

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.

11 Other language predefined keywords

11.1 french keywords

Hey, I am a frenchy , so I have defined the same as in section 9 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}`

11.2 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}

11.3 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}
- \Para{condition}{text loop}
  \lPara{condition}{line text loop}
- \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}

11.4 Italian keywords

- \KwIng{Ingresso}
  \KwUsc{Uscita}
  \KwDati{Dati}
  \KwRisult{Risultato}
- \KwA
  \KwRitorna{ritorna}
  \Ritorna{ritorna}
- \Inizio{inside block}
- \Ripeti{stop condition}{text loop}
  \lRipeti{stop condition}{line of the loop}

- \eSea{condition}{then block}{else block}
  \{condition}{then block}
  \uSea{condition}{then block without end}
  \lSea{condition}{then's line text}
  \AltSe{else block}
  \uAltSe{else block without else}
  \lAltSe{else's line text}
  \Altrimenti{condition}{elseif block}
  \uAltrimenti{condition}{elseif block without end}
  \lAltrimenti{condition}{elseif's line text}
- \Switch{condition}{Switch block}
  \Case{a case}{case block}
  \uCase{a case}{case block without end}
  \lCase{a case}{case's line}
  \Other{otherwise block}
  \lOther{otherwise's line}
- \Per{condition}{text loop}
  \lPer{condition}{line text loop}
- \PerPar{condition}{text loop}
  \lPerPar{condition}{line text loop}
- \PerCiascun{condition}{text loop}
  \lPerCiascun{condition}{line text loop}
- \PerTutti{condition}{text loop}
  \lPerTutti{condition}{line text loop}
- \Finche{stop condition}{text loop}
  \lFinche{stop condition}{text loop}

Here we describe how they are obtained:

1. `\SetKwInput{KwIng}{Ingresso}`
2. `\SetKwInput{KwUsc}{Uscita}`
3. `\SetKwInput{KwDati}{Dati}`
4. `\SetKwInput{KwRisult}{Risultato}`
5. `\SetKw{KwA}{a}`
6. `\SetKw{KwRitorna}{ritorna}`
7. `\SetKw{Ritorna}{ritorna}`
8. `\SetKwBlock{Inizio}{inizio}{fine}`
9. `\SetKwRepeat{Ripeti}{ripeti}{finch}`

10. \SetKwIF{Sea}{AltSe}{Altrimenti}{se}{allora}{altrimenti se}{allora}{fine se}
11. \SetKwSwitch{Switch}{Case}{Other}{switch}{do}{case}{otherwise}{endsw}
12. \SetKwFor{Per}{per}{fai}{fine per}
13. \SetKwFor{PerPar}{per}{fai in parallelo}{fine per}
14. \SetKwFor{PerCiascun}{per ciascun}{fai}{fine per ciascun}
15. \SetKwFor{PerTutti}{per tutti i}{fai}{fine per tutti}
16. \SetKwFor{Finche}{finch}{fai}{fine finch}

11.5 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

12 Known bugs

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

Release notes

```
% - december 14 2009 - revision 4.01
% * ADD : new command \SetKwHangingKw{Name}{text} (hanging indent with keyword): This creates a
%         hanging indent much like \texttt{SetKwInput}, except that it removes the trailing ':'
%         and does not reset numbering.
% - november 17 2009  - revision 4.00 -
% * CHANGE : IMPORTANT : some commands have been renamed to have consistent naming (CamlCase
%            syntax) and old commands are no more available. If you doesn't want to change
%            your mind or use old latex files, you can use oldcommands option to enable old
%            commands back.
%            text. Here are these commands:
%            - \SetNoLine becomes \SetAlgoNoLine
%            - \SetVline  becomes \SetAlgoVlined
%            - \Setvlineskip  becomes \SetVlineSkip
%            - \SetLine   becomes \SetAlgoLined
%            - \dontprintsemicolon becomes \DontPrintSemicolon
%            - \printsemicolon becomes \PrintSemicolon
%            - \incmargin becomes \IncMargin
%            - \decmargin becomes \DecMargin
%            - \setnlskip becomes \SetNlSkip
%            - \Setnlskip becomes \SetNlSkip
%            - \setalcapskip becomes \SetAlCapSkip
%            - \setalcaphskip becomes \SetAlCapHSkip
%            - \nlSty becomes \NlSty
%            - \Setnlsty becomes \SetNlSty
%            - \linesnumbered becomes \LinesNumbered
%            - \linesnotnumbered becomes \LinesNotNumbered
%            - \linesnumberedhidden becomes \LinesNumberedHidden
%            - \showln becomes \ShowLn
%            - \showlnlabel becomes \ShowLnLabel
%            - \nocaptionofalgo becomes \NoCaptionOfAlgo
%            - \restorecaptionofalgo becomes \RestoreCaptionOfAlgo
%            - \restylealgo becomes \RestyleAlgo
%            - gIf macros and so on do no more exist
% * NEW: - Compatibily with other packages improven by changing name of internal
%          macros. Algorithm2e can now be used with arabtex for example, if this last is
%          loaded after algorithm2e package.
% * ADD: - OPTION endfloat: endfloat packages doesn't allow float environment inside other
%          environment. So using it with figure option of algorithm2e makes error. This
%          option enables a new environment algoendfloat to be used instead of algorithm
%          environment that put algorithm at the end. algoendfloat environment make
%          algorithm acting as endfloat figures. This option requires endfloat packages.
% * ADD: - OPTION norelsize: starting from this release (v4.00), algorithm2e package uses
%          relsize package in order to get relative size for lines numbers; but it seems
%          that some rare classes (such as inform1.cls) are not compatible with relsize; to
%          have algorithm2e working, this option makes algorithm2e not to load relsize
%          package and go back to previous definition by using \scriptsize font for lines
%          numbers.
% * ADD: - OPTION onelanguage: allow, if using standard keywords listed below, to switch
%          from one language to another without changing keywords by using appropriate
%          language option :
%          . KwIn, KwOut, KwData, KwResult
%          . KwTo KwFrom
```

```
%           . KwRet, Return
%           . Begin
%           . Repeat
%           . If, ElseIf, Else
%           . Switch, Case, Other
%           . For, ForPar, ForEach, ForAll, While
%           .
% * ADD: - OPTION rightnl: put lines numbers to the right of the algorithm instead of left.
% * ADD:   new commands \setRightLinesNumbers and \setLeftLinesNumbers which sets the lines
%          numbers to the right or to the left of the algorithm.
% * ADD/FIX: rules of ruled, algoruled, tworuled styles used rules of different sizes! This
%            is now fixed. Moreover size of the rules is now controlled by a length and so
%            can be customized by the user.
%            \algoheightrule is the height of the rules and can be changed via \setlength
%            \algoheightruledefault is the default height of he rules (0.8pt)
%            \algotitleheightrule is the height of the rule that comes just after the
%            caption in ruled and algoruled style; it can be changed via \setlength
%            \algotitleheightruledefault is the default height of this rules (0.8pt)
%            Thanks to Philippe Dumas who reports the bug and make the suggestion.
% * ADD: - \SetAlgoCaptionSeparator which sets the separator between Algorithm 1 and the
%          title. By default it's ':' and caption looks like "Algorithm 2: title" but now
%          you can change it by using for example \SetAlgoCaptionSeparator{.} which will
%          give "Algorithm 3. title"
% * ADD: - \SetAlgoLongEnd and \SetAlgoShortEnd and \SetAlgoNoEnd commands which act as
%          corresponding package options
% * ADD: - OPTIONS italiano and slovak as new language (thanks to Roberto Posenato and
%          Miroslav Binas)
% * ADD: - \AlCapSty, \AlCapNameSty, \AlCapFnt, \AlCapNameFnt and corresponding "set macro"
%          \SetAlCapSty, \SetAlCapNameSty, \SetAlCapFnt, \SetAlCapNameFnt which control the
%          way caption is printed. Sty macro use command taking one parameter as argument,
%          Fnt macros use directly command. In Fact caption is printed as follow :
%          \AlCapSty{\AlCapFnt Algorithm 1:}\AlCapNameSty{\AlCapNameFnt my algorithm}
%          By default, \AlCapSty is textbf and \AlCapFnt is nothing. \AlCapNameSty keep text
%          as it is, and \AlCapNameFnt do nothing also.
%          You can redefine \AlCapFnt and \AlCapNameFnt by giving macro to \Set commands. For
%          example, you can do \SetAlCapFnt{\large} to see Algorithm printed in \large font.
%          You can redefine \AlCapSty, \AlCapFnt, \AlCapNameSty and \AlCapNameFnt with the
%          corresponding \Set command. For the Sty commands, you have to give in parameter
%          name of a macro (whithout \)  which takes one argument. For example,
%          \SetAlCapFnt{textbf} defines the default behaviour. If you want to do more
%          complicated thing, you should define your own macro and give it to \SetAlCapFnt or
%          \SetAlCapNameFnt. Here are two examples:
%          - \newcommand{\mycapsty}[1]{\tiny #1}\SetAlCapNameSty{mycapsty}
%          - \newcommand{\mycapsty}[1]{\textsl{\small #1}}\SetAlCapNameSty{mycapsty}
%          Or you can combine the two, for the last example you can also do:
%          \SetAlCapNameSty{textsl}\SetAlCapNameFnt{\small}
%          Thanks to Jan Stilhammer who gives me the idea of \AlCapNameFnt.
% * CHANGE \AlTitleFnt to match definition of all other Fnt macros and add a \AlTitleSty
%          macro (see below) . Now you set \AlTitleFnt by calling \SetAlTitleFnt with
%          directly a macro without parameter in argument:
%          Example: \SetAlTitleFnt{\small} to set title in small font.
% * ADD: - \AlTitleSty and \SetAlTitleSty commands to set a style for title. These commands
%          are defined from a macro taking the text in argument, as \textbf for example.
%          To set the TitleSty you have to give name of the macro (without the '\')
```

```
%          to \SetAlTitleSty. For example \SetAlTitleSty{textbf} to set \textbf style.
% * ADD: - new command \SetAlgorithmName{algorithmname}{list of algorithms name} which
%          redefines name of the algorithms and the sentence list of algorithms. Second
%          argument is the name that \autoref, from hyperref package, will use. Example:
%          \SetAlgorithmName{Protocol}{List of protocols} if you prefer protocol than
%          algorithm.
% * ADD: - new \SetAlgoRefName{QXY} which change the default ref (number of the algorithm) by
%          the name given in parameter (QXY in the example).
% * ADD: - new command \SetAlgoRefRelativeSize{-2} which sets the output size of refs, defined
%          by \SetAlgoRefName, used in list of algorithms.
% * ADD: - two dimensions to control the layout of caption in ruled, algoruled and boxruled
%          algorithms:
%          - interspacetitleruled (2pt by defaut) which controls the vertical space between
%            rules and title in ruled and algoruled algorithms.
%          - interspaceboxruled (2\lineskip by default) which controls the vertical space
%            between rules and title in boxruled algorithms.
%          These two dimensions can be changed by using \setlength command.
% * ADD: - With the fix (see below) of procedure and function environments, a new feature has
%          been added: the name of the procedure or function set in caption is automatically
%          defined as a KwFunction and so can be used as a macro. For example, if inside a
%          procedure environment you set \caption{myproc()}, you can use \myproc macro in you
%          main text. Beware that the macro is only defined after the \caption!
% * ADD: - OPTION nokwfunc to unable the new feature described above in function and
%          procedure environment. Useful if you use name of procedure or function that cannot
%          be a command name as a math display for example.
% * ADD: - \SetAlgoNlRelativeSize{number} command which sets the relative size of line
%          numbers. By default, line numbers are two size smaller than algorithm text. Use
%          this macro to change this behavior. For example, \SetAlgoNlRelativeSize{0} sets it
%          to the same size, \SetAlgoNlRelativeSize{-1} to one size smaller and
%          \SetAlgoNlRelativeSize{1} to one size bigger
% * ADD: - \SetAlgoProcName{aname} command which sets the name of Procedure printed by
%          procedure environment (the environment prints Procedure by default). Second
%          argument is the name that \autoref, from hyperref package, will use.
% * ADD: - \SetAlgoFuncName{aname} command which sets the name of Function printed by
%          procedure environment (the environment prints Function by default). Second
%          argument is the name that \autoref, from hyperref package, will use.
% * ADD: - \SetAlgoCaptionLayout{style} command which sets style of the caption; style must
%          be the name of a macro taking one argument (the text of the caption). Examples
%          below show how to use it:
%          . \SetAlgoCaptionLayout{centerline} to have centered caption
%          . \SetAlgoCaptionLayout{textbf} to have bold caption
%          If you want to apply two styles in the same time, such as centered bold, you have
%          to define you own macro and then use \SetAlgoCaptionLayout with its name.
% * ADD: - OPTION procnumbered: which makes the procedure and function to be numbered as
%          algorithm
% * ADD: - OPTIONS tworuled and boxruled
%          these are two new layouts: tworuled acts like ruled but doesn't put a line after
%          caption ; boxruled surround algorithm by a box, puts caption above and add a line
%          after caption.
% * REMOVE: - SetKwInParam has been deleted since not useful itself because of different
%           macros which can do the same in a better and a more consistent way as
%           SetKwFunction or SetKw.
% * FIX: - line number is now correctly vertically aligned with math display.
% * FIX: - references with hyperref. No more same identifier or missing name error. BUT now
```

```
%          you must NOT use naturalnames option of hyperref packages if you do PdfLaTeX
% * FIX: - autoref with hyperref package (thanks to Jrg Sommer who notices the problem).
% * FIX: - titlenumbered was not working! fixed.
% * FIX: - Else(){} acted like uElse. Corrected.
% * FIX: - noend management: when a block was inside another and end of block was following
%          each other, a blank line was added: it's now corrected.
% * FIX: - Function and Procedure environment was no more working as defined originally: the
%          label was no more name of the procedure, it acts always as if procumbered option
%          has been used.
% * FIX: - line numbers had a fixed size which can be bigger than algorithm text accordingly
%          to \AlFnt set (see also new command \SetAlgoNlRelativeSize above)
% * FIX: - semicolon in comments when dontprintsemicolon is used.
% * FIX: - listofalgorithms adds a vertical space before first algo of a chapter as for
%          listoffigures or listoftables
% * FIX: - listofalgorithms with twocolumns mode and some classes which don't allow onecolumn
%          and so don't define \if@restonecol as prescribed in LaTeX (sig-alternate for
%          example)
% * FIX: - algorithm2e now works with elsart cls and some more classes.
% * FIX: - blocks defined by SetKwBlock act now as other blocks (if for instance) and don't
%          write end in vlined mode, instead they print a small horizontal line as required
%          by the option.
% * FIX: - underfull hbox warning at each end of algorithm environment removed.
%
% * INTERNAL CHANGE: - short end keyword are deduce from long end keyword by keeping the
%                      first one. Allows to avoid double definition.
% * INTERNAL CHANGE: - procedure, function and algorithm are now resolved by the same
%          environment to avoid code duplication.
```

List of Algorithms