algorithm2e.sty — package for algorithms

release 5.0

(c) 1995-1997 Christophe Fiorio, Tu-Berlin, Germany
(c) 1998-2013 Christophe Fiorio, LIRMM, Montpellier 2 University, France
Report bugs and comments to cfiorio@um2.fr
algorithm2esty-annonce@lirmm.fr mailing list for announcements
algorithm2esty-discussion@lirmm.fr mailing list for discussion[*][†][‡][§][¶][‖][**][††]

january 06 2013

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

[††] 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 this release (5.0), 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 7.

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, 14, 17 and 19. Star comment commands are for comment on lines of code, else comment is a line by itself as at line 17. 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
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 // O(left,this)==1
            if left < this then Union(left,this);
            ;
            else Union(this,left);
            ;
        end
        if up compatible with this then                    // O(up,this)==1
            if up < this then Union(up,this);
            ;
            // this is put under up to keep tree as flat as possible
            else Union(this,up);
            ;                                              // this linked to up
        end
    end
    foreach element $e$ of the line $i$ do FindCompress($p$);
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 Genericity and example of languages

In this section, we will try to show you main macros and how you can use this package to suit your need. Based on one example using most popular algorithms expressions, we will show you how it can be configured to be display in pseudo-code, in python or in C.

The following code shows how is typeset the generic example we'll use in this section:

```
\Fn(\tcc*[h]{algorithm as a recursive function}){\FRecurs{some args}}{
  \KwData{Some input data\\these inputs can be displayed on several lines and one
    input can be wider than line's width.}
  \KwResult{Same for output data}
  \tcc{this is a comment to tell you that we will now really start code}
  \If(\tcc*[h]{a simple if but with a comment on the same line}){this is true}{
    we do that, else nothing\;
    \tcc{we will include other if so you can see this is possible}
    \eIf{we agree that}{
      we do that\;
    }{
      else we will do a more complicated if using else if\;
      \uIf{this first condition is true}{
        we do that\;
      }
      \uElseIf{this other condition is true}{
        this is done\tcc*[r]{else if}
      }
      \Else{
        in other case, we do this\tcc*[r]{else}
      }
    }
  }
  \tcc{now loops}
  \For{\forcond}{
    a for loop\;
  }
  \While{$i<n$}{
    a while loop including a repeat--until loop\;
    \Repeat{this end condition}{
      do this things\;
    }
  }
  They are many other possibilities and customization possible that you have to
  discover by reading the documentation.
}
```

To handle `if` condition, use a macro to be abble to change it according to language syntax, in particular we will change it for python-style and c-style. We also define a function to write algorithm as a recursive function. These macros are defined as:

```
\newcommand{\forcond}{$i=0$ \KwTo $n$}
\SetKwFunction{FRecurs}{FnRecursive}%
```

The algorithm 4 shows how algorithm is displayed in pseudo-code with default behaviour and options `boxed`, `commentsnumbered` and `longend` set. Note that by default, lines are used to show block of code. Note also that `longend` option makes package use special *end* keyword for each command[2].

[2] Default behaviour uses *short end* keywords, it means typeseting only *end*.

```
Function FnRecursive(some args) /* algorithm as a recursive function */
    Data: Some input data
    these inputs can be displayed on several lines and one input can be wider than line's width.
    Result: Same for output data
    /* this is a comment to tell you that we will now really start code */
    if this is true then /* a simple if but with a comment on the same line */
        we do that, else nothing;
        /* we will include other if so you can see this is possible */
        if we agree that then
            we do that;
        else
            else we will do a more complicated if using else if;
            if this first condition is true then
                we do that;
            else if this other condition is true then
                this is done;                                /* else if */
            else
                in other case, we do this;                   /* else */
            end if
        end if
    end if
    /* now loops */
    for $i = 0$ to $n$ do
        a for loop;
    end for
    while $i < n$ do
        a while loop including a repeat–until loop;
        repeat
            do this things;
        until this end condition;
    end while
    They are many other possibilities and customization possible that you have to discover by reading the documentation.
end
```

Algorithm 4: Generic example with most classical expressions derived in pseudo-code

The algorithm 5 shows how algorithm is displayed using automatic block display (new feature since relase 5.0). To achieve this display, we only add following macros at start of the algorithm:

```
\AlgoDisplayBlockMarkers\SetAlgoBlockMarkers{begin}{end}%
\SetAlgoNoEnd
```

First one tells package to display blocks with keyword markers. Note that the definition of block markers are the one by default. Last macro remove end keywords of commands to avoid a double end (the one of block marker and the one of command).

The algorithm 6 shows how algorithm looks like with a python-style syntax. To achieve this display, we need to make following changes before the algorithm:

```
\SetStartEndCondition{ }{}{}%
\SetKwProg{Fn}{def}{\string:}{}
\SetKwFunction{Range}{range}%%
\SetKw{KwTo}{in}\SetKwFor{For}{for}{\string:}{}%
\SetKwIF{If}{ElseIf}{Else}{if}{:}{elif}{else:}{}%
\SetKwFor{While}{while}{:}{fintq}%
\renewcommand{\forcond}{$i$ \KwTo\Range{$n$}}
\AlgoDontDisplayBlockMarkers\SetAlgoNoEnd\SetAlgoNoLine%
```

`SetStartEndCondition` is used to display alternatives and loops conditions according to python syntax: it means a space before condition, and no space after since ':' marks end of condition. Functions are defined with *def* in python, so we redefine \Fn macro. `Range` is a new macro for *range* python function. Next are redefined *For*, *If* and *While* commands accordingly to python syntax. Note that we do nothing for *repeat-until* command since it doesn't exist in python. For condition is redefined to match python behaviour. At last we tell package to not display block, to not display end keyword and to not print line according to python syntax.

The algorithm 7 shows how algorithm looks like with a c-style syntax. To achieve this display, we need to make following changes before the algorithm:

```
\SetStartEndCondition{ (}{)}{)}\SetAlgoBlockMarkers{\{}{\}}%
\SetKwProg{Fn}{}{}{}\SetKwFunction{FRecurs}{void FnRecursive}%
\SetKwFor{For}{for}{}{}%
\SetKwIF{If}{ElseIf}{Else}{if}{}{elif}{else}{}%
\SetKwFor{While}{while}{}{}%
\SetKwRepeat{Repeat}{repeat}{until}%
\AlgoDisplayBlockMarkers\SetAlgoNoLine%
```

`SetStartEndCondition` set braces around conditions like in C. We want that each block is marked with { at start and } at end, so we set it thanks to `\SetAlgoBlockMarkers` macro. \Fn is redefined with no keyword since in C, name of function defines it. Then we redefin `FnRecursive` with its type. Next, *For*, *If*, *While* and *Repeat* are redefined accordingly to C syntax. At last, we tell the package to display block markers.

The algorithm 8 shows how algorithm looks like with a c-style syntax and a more compact way to mark blocks. To achieve this display, we need to make following changes before the algorithm:

```
\SetStartEndCondition{ (}{)}{)}\SetAlgoBlockMarkers{}{\}}%
\SetKwProg{Fn}{}{\{}{}\SetKwFunction{FRecurs}{void FnRecursive}%
\SetKwFor{For}{for}{\{}{}%
\SetKwIF{If}{ElseIf}{Else}{if}{\{}{elif}{else\{}{}%
\SetKwFor{While}{while}{\{}{}%
\SetKwRepeat{Repeat}{repeat\{}{until}%
\AlgoDisplayBlockMarkers\SetAlgoNoLine%
```

If you look at algorithm 4, you can see that some command doesn't put a end. For example, it is a case for a *if* followed by an *else*, same thing for a *else if*. In C, there is always an end marker. So, to achieve our goal we need to use end marker of blocks. But we don't want displaying begin marker as in algorithm 5 or algorithm 7. If begin block marker is set to empty, then nothing is written (especially not a blank line). So we tell package to use block markers with an empty marker for begin and a } for end. Now we have to tell package to write a { on the same line as commands. This is achieve by redefining *If*, *For*, *While* and *Repeat* command.

```
Function FnRecursive(some args) /* algorithm as a recursive function */
begin
    Data: Some input data
    these inputs can be displayed on several lines and one input can be wider than line's
    width.
    Result: Same for output data
    /* this is a comment to tell you that we will now really start code */
    if this is true then /* a simple if but with a comment on the same line */
    begin
        we do that, else nothing;
        /* we will include other if so you can see this is possible */
        if we agree that then
        begin
            we do that;
        end
        else
        begin
            else we will do a more complicated if using else if;
            if this first condition is true then
            begin
                we do that;
            end
            else if this other condition is true then
            begin
                this is done;                                   /* else if */
            end
            else
            begin
                in other case, we do this;                      /* else */
            end
        end
    end
    /* now loops */
    for $i = 0$ to $n$ do
    begin
        a for loop;
    end
    while $i < n$ do
    begin
        a while loop including a repeat–until loop;
        repeat
        begin
            do this things;
        end
        until this end condition;
    end
    They are many other possibilities and customization possible that you have to discover
    by reading the documentation.
end
```

Algorithm 5: Generic example in pseudo-code with begin-end block set

```
def FnRecursive(some args): /* algorithm as a recursive function */
    Data: Some input data
    these inputs can be displayed on several lines and one input can be wider than line's width.
    Result: Same for output data
    /* this is a comment to tell you that we will now really start code */
    if this is true: /* a simple if but with a comment on the same line */
        we do that, else nothing;
        /* we will include other if so you can see this is possible */
        if we agree that:
            we do that;
        else:
            else we will do a more complicated if using else if;
            if this first condition is true:
                we do that;
            elif this other condition is true:
                this is done;                                   /* else if */
            else:
                in other case, we do this;                         /* else */
    /* now loops */
    for $i$ in range($n$):
        a for loop;
    while $i < n$:
        a while loop including a repeat–until loop;
        repeat
            do this things;
        until this end condition;
    They are many other possibilities and customization possible that you have to discover by reading the documentation.
```

Algorithm 6: Generic example in python-style like syntax

```
void FnRecursive(some args) /* algorithm as a recursive function */
{
    Data: Some input data
    these inputs can be displayed on several lines and one input can be wider than line's width.
    Result: Same for output data
    /* this is a comment to tell you that we will now really start code */
    if (this is true) /* a simple if but with a comment on the same line */
    {
        we do that, else nothing;
        /* we will include other if so you can see this is possible */
        if (we agree that)
        {
            we do that;
        }
        else
        {
            else we will do a more complicated if using else if;
            if (this first condition is true)
            {
                we do that;
            }
            elif (this other condition is true)
            {
                this is done;                                /* else if */
            }
            else
            {
                in other case, we do this;                   /* else */
            }
        }
    }
    /* now loops */
    for ($i$ to range($n$))
    {
        a for loop;
    }
    while ($i < n$)
    {
        a while loop including a repeat–until loop;
        repeat
        {
            do this things;
        }
        until (this end condition);
    }
    They are many other possibilities and customization possible that you have to discover by reading the documentation.
}
```

Algorithm 7: Generic example in c-style like syntax

```
void FnRecursive(some args){ /* algorithm as a recursive function */
    Data: Some input data
    these inputs can be displayed on several lines and one input can be wider than line's width.
    Result: Same for output data
    /* this is a comment to tell you that we will now really start code */
    if (this is true){ /* a simple if but with a comment on the same line */
        we do that, else nothing;
        /* we will include other if so you can see this is possible */
        if (we agree that){
            we do that;
        }
        else{
            else we will do a more complicated if using else if;
            if (this first condition is true){
                we do that;
            }
            elif (this other condition is true){
                this is done;                                  /* else if */
            }
            else{
                in other case, we do this;                     /* else */
            }
        }
    }
    /* now loops */
    for ($i$ to range($n$)){
        a for loop;
    }
    while ($i < n$){
        a while loop including a repeat–until loop;
        repeat{
            do this things;
        }
        until (this end condition);
    }
    They are many other possibilities and customization possible that you have to discover by reading the documentation.
}
```

Algorithm 8: Generic example in c-style like syntax with compact block

5 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 5.0has 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

6 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}{List 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.

7 The options of the package

7.1 language option

croatian: to have for example *Algoritam:* instead of *algorithm:*.

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

english: the default.

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

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

ngerman: to have german option as babel

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

spanish: to have for example *Algoritmo:* 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`

7.2 compatibility issue

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`

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.

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.

slide: require package color. To be used with slide class in order to have correct margins.

nokwfunc disable the setting in `\KwFunction` of procedure's or function's name (see section 6) 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.

7.3 environment display and use

7.3.1 boxed, ruled, plain environment

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.

7.3.2 algorithm numbering

algochapter: algorithms are numbered within chapter numbers.

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

algopart: algorithms are numbered within part numbers.

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

7.3.3 figure and toc

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

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.

7.4 code typesetting

7.4.1 blocks display

lines

lined: `\SetAlgoLined` becomes the default, see section 9.6 for explanations about the `\SetAlgoLined` macros.

vlined: `\SetAlgoVlined` becomes the default, see section 9.6 for explanations about the `\SetAlgoVlined` macros.

noline: `\SetNoline` becomes the default, see section 9.6 for explanations about the `\SetNoline` macros.

block markers

displayblockmarkers `\AlgoDisplayBlockMarkers` becomes the default, see section 9.6 for explanations about the `\AlgoDisplayBlockMarkers` macro.

7.4.2 end keywords

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.

7.4.3 comments

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

7.4.4 lines numbers

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.

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.

7.4.5 title of algorithms

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.

8 Typesetting

There are eight 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 11 (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. (See also function environment at section 6, which defines not only function keyword but algorithm of the function.

   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` and `\ProcNameFnt`): The type style of the caption of *procedure* and *function* environment.

5. The arguments of procedure and function environments style (`\ProcArgSty` and `\ProcArgFnt`): 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. Block markers: style of keywords that are print at begin and end of block when displayblockmarkers option is set or `\AlgoDisplayBlockMarkers` macro used. By default, `\BlockMarkersSty` is set to `\KwSty`.

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

9 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.

9.1 global code 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.

`\SetStartEndCondition{typo1}{typo2}{typo3}` which sets typo around condition in `For`, `If`, `Switch`, `Case` and `Repeat` macros. First two are used around `For`, `If`, `Switch` conditions, First and third are used for `Case` and `Repeat` condition. Default definition is: `\SetStartEndCondition{ }{ }{}`.
A common alternative is `\SetStartEndCondition{ (}{) }{)}`.
It can also be used to remove space around condition, for example if you want python style commands: `\SetStartEndCondition{ }{}{}` and `\SetKwFor{For}{for}{:}{}`

`\AlgoDisplayBlockMarkers` that prints begin and end markers at the start and end of all block. These begin and end keywords could be specified by using `\SetAlgoBlockMarkersbegin keywordsend keywords{c}{o}`mmand. By default, these keywords are not printed but *begin* and *end* are default keywords used if `\AlgoDisplayBlockMarkers` is called.

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

9.2.1 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}`.

9.2.2 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 `\interspacetitleboxruled` are described section 9.6.

9.3 line numbering

9.3.1 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 7) 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.

9.3.2 setting style of lines numbers

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.

Example below shows use of these macros:

```
\SetNlSty{texttt}{[}{]}
\SetAlgoNlRelativeSize{0}
\SetNlSkip{0em}
\nl\KwIn{input data}
\nl\KwOut{output data}
\nl\tcc{a comment line in C-style}
\nl\Repeat{\nl$e<\tau$}{
  \nl$f_n\leftarrow Y_1$\;
  \nl$f_{n+1}\leftarrow f_n\times f_{n-1}$\;
  \nl$e\leftarrow \frac{f_n}{2}$\;
}
\nl\KwRet{$e$}
```

⟹

```
[2]Input: input data
[4]Output: output data
[6]/* a comment line in C-style  */
[8]repeat
[10]    $f_n \leftarrow Y_1$;
[12]    $f_{n+1} \leftarrow f_n \times f_{n-1}$;
[14]    $e \leftarrow \frac{f_n}{2}$;
[16]until $e < \tau$;
[18]return $e$
```

9.4 math display

If you need to use math display to handle complex mathematics as matrix, using standard \[\] or $$ will not allow correct numbering and end line management. If you don't need line numbers, there is no problem. If you want line numbers, please use algomathdisplay environment instead of \[\] or $$. It will work as standard math display but line spacing, line numbers, end line will be managed correcly

9.5 standard styles

9.5.1 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 9.5.3)

\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 procedure style of procedure and function environment (by default the same as \AlCapSty{}). (see section 9.5.2 for more explanations and details)

\ProcFnt{<text>} sets <text> in procedure typography of procedure and function environment (by default the same as \AlCapFnt{}). (see section 9.5.2 for more explanations and details)

\ProcArgSty{<text>} sets <text> in argument style of procedure and function environment (by default the same as \AlCapNameSty{}). (see section 9.5.2 for more explanations and details)

\ProcArgFnt{<text>} sets <text> in argument typography of procedure and function environment (by default the same as \AlCapNameFnt{}). (see section 9.5.2 for more explanations and details)

\BlockMarkersSty{<text>} sets <text> in block markers typography (by default the same as \KwSty{}) (see section 9.6 for more explanations and details on block markers).

9.5.2 caption and title font style

\AlCapSty, \AlCapNameSty, \AlCapFnt, \AlCapNameFnt, \ProcSty, \ProcFnt, \ProcNameSty, \ProcNameFnt, \ProgArgSty, \ProgArgFnt and corresponding "\Set commands" (see section 9.5.4) \SetAlCapSty, \SetAlCapNameSty, \SetAlCapFnt, \SetAlCapNameFnt, \SetProcSty, \SetProcFnt, \SetProcNameSty, \SetProcNameFnt, \SetProgArgSty, \SetProgArgFnt control the way caption of algorithm and procedure/function environment are 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 \caption{my algorithm} is 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.

\ProcSty and \ProcFnt are used to define style and font shape of "Procedure" in caption of procedure and function environment. \ProcNameSty and \ProcNameFnt are used to define style and font shape of the procedure or function name. \ProcArgSty and \ProgArgFnt are used to define style and font shape of arguments in procedure/function environment. In fact a caption \caption{Proc(int i)} of procedure/function environment is printed as follow :

```
\ProcSty{\ProcFnt Procedure}\ProcNameSty{\ProcNameFnt Proc(}%
\ProgArgSty{\ProgArgFnt int i}\ProcNameSty{\ProcNameFnt )}.
```

By default, \ProcSty is \AlCapSty and \ProcFnt is \AlCapFnt. \ProcNameSty keeps text as it is, and \ProcNameFnt 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.

`\ProcSty{<text>}` sets <text> in procedure/function caption title typography, that is the same used, together with `\ProcFnt`, to print `Procedure`, more precisely it is printed as follow:
`\ProcSty{\ProcFnt Procedure}`
which gives actually "**Procedure**". By default `\ProcSty` is `\AlCapSty`.

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

`\ProcArgSty{<text>}` sets <text> in argument of procedure/function typography, that is the same used, together with `\ProcArgFnt`, to print `int i` if `\caption{Proc(int i)}` was called. More precisely it is printed as follow:
`\ProcArgSty{\ProcArgFnt int i}`
which gives actually "int i". By default `\ProcArgSty` is `\AlCapNameSty`;

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

`\ProcNameFnt{<text>}` sets <text> in procedure/function name typography, that is the same used, together with `\ProcNameSty` to print the name of caption you set by calling `\caption{Proc(int i)}`. More precisely it is printed as follow:
`\ProcNameSty{\ProcNameFnt Proc(}`
which gives "Proc(". By default `\ProcNameFnt` is `\relax` which means keep text as it is.

`\ProcArgFnt{<text>}` sets <text> in font shape of argument of procedure/environment caption, that is the same used, together with `\ProcArgSty`, to print `int i` if `\caption{int i}` was called. More precisely it is printed as follow:
`\ProcArgSty{\ProcFnt int i}`
which gives actually "int i". By default `\ProcArgFnt` do nothing.

\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 #:}}.

9.5.3 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 9.2.2) 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{}).

\SetBlockMarkersSty{<font>} sets block markers typography (by default the same as \KwSty{}).

9.5.4 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 9.5.3), they require a name of a command in argument, this command have to take one argument, the text to be typeset. They should be combined with previous commands to redefine display of caption or title. Examples of use:

- `\newcommand{\mycapsty}[1]{\textbf{\emph{#1}}}\SetAlCapNameSty{mycapsty}` caption will be print emphased and in bold face.
- `\SetAlCapNameFnt{\tiny}` set font to tiny size.
- if you combine \SetAlCapNameSty{mycapsty} and \SetAlCapNameFnt{\tiny} will give tiny bold empased caption.

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}`

9.6 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 9.3) for more details on this command.

`\SetAlgoCaptionLayout{style}` sets the global style of caption (see section 9.2 for more details).

`\DisplayBlockMarkers` acts like displayblockmarkers option: each block will be started by a *begin* keyword and be ended by an *end* keywords. This is tricky to use but allows to customize syntax to match almost every language (see section 4 for examples showing how to use it). `\SetAlgoBlockMarkers{begin keyword}{end keyword}` defines *begin* and *end* keywords that will be used by `\DisplayBlockMarkers`. Default keywords are **begin** and **end**. but for example you can set `\DisplayBlockMarkers{\{}{\}}` to match c-style syntax.

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.

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

9.7 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 10.3, you can see how `\tcc` is defined and at section 10.4 you can look at some examples how to use it with `if then else` like commands and finally you can look at section 11.4 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.

10 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 12 for a list of other language keywords. All these keywords are defined using macros provided by the package and described in section 11.

10.1 Input, output macros...

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

10.2 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}

10.3 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 11.5

10.4 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}
- \leIf{condition}{then block}{else block}
- \leIf(*comment*){condition}{then block}{else block}

10.5 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}

10.6 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}

10.7 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}

10.8 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}{endcase}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}`

11 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.

11.1 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.

11.2 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: \Kw{arg}. If so, *arg* is printed in argument typography. For example `\Kw{thetext}` could give: **Kw** *thetext*

`\SetKwHangingKw{Kw}{thetext}` defines a hanging keyword that should act like a combination of `\SetKwInput` and `\SetKw`. In comparison with \SetKwInput, it doesn't print ':' at end of keyword and line is numbering if linesnumbered is set.
For example `\SetKwHangingKw{HData}{Data$\rightarrow$}` could gives:

```
Data→ a list of data and a long description of this data to be sure that text requires
      several lines to be printed;
```

`\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.

`\SetKwArray{Kw}{array}` which defines an array keywords `Kw` called *array* and printed in DataSty style when call with `\Kw`. It can be used with one argument which denotes the element index: `\Kw{n}` prints array[*n*] with *array* in `\DataSty` and *n* in `\ArgSty`.

`\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*.

You can also use alternativ `\uBegin{txt}` which acts as `\Begin{txt}` but without *end*. Useful for example as a part separator that doesn't necessary need an *end* keyword.

`\SetKwProg{Prog}{Title}{is}{end}` Env is a block with 'Title' (in `CapSty` style) at the beginning followed by args followed by '`is`' then '`text`' inside a block ended by '`end`'. If no '`end`' is specified, nothing is written (no blank line is inserted). Useful to typeset function or prog. For example:

```
\SetAlgoLined
\SetKwProg{Fn}{Function}{ is}{end}
\Fn{afunc(i: int) : int}{return 0\;}

\SetKwProg{Def}{def}{:}{}
\Def{afunc(i: int)}{return 0\;}
```

⟹

```
Function afunc(i: int) : int is
  | return 0;
end
def afunc(i: int):
  | return 0;
```

11.3 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*`)`.

11.4 to define comments

`\SetKwComment{Comment}{start}{end}` defines a macro `\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.

11.5 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.
- `\If(comment){cond}{Then's text}`
  as previous but put *comment* after `then` keyword. Usually use with comment macro like `\tcc`*[f]{comment} or `\tcp`*[f]{comment}
- `\ElseIf{ElseIf's text}`
  ElseIf's text is writen in a block and terminating by the `endif`.
- `\ElseIf(comment){ElseIf's text}`
  the same with comment.
- `\Else{Else's text}`
  Else's text is writen in a block and terminating by the `endif`.
- `\Else{Else's text}`
  the same with comment.
- `\lIf{cond}{Then's text}`
  Then's text is written on the same line as `then`. No `endif` is printed. Do not put \; after *Then's text* neither after `\lIf`.

- `\lIf(comment){cond}{Then's text}`
  the same with comment.
- `\lElseIf{ElseIf's text}`
  ElseIf's text is written on the same line as `else if`. No `endif` is printed.
- `\lElseIf(comment){ElseIf's text}`
  the same with comment.
- `\lElse{Else's text}`
  Else's text is written on the same line as `else`. No `endif` is printed.
- `\lElse(comment){Else's text}`
  the same with comment.
- `\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).
- `\uIf(comment){cond}{Then's text}`
  the same with comment.
- `\uElseIf{ElseIf's text}` (for uncomplete elseif)
  Same explanation as for `\uIf` but with `else if`.
- `\uElseIf(comment){ElseIf's text}`
  the same with comment.
- `\uElse{Else's text}` (for uncomplete else)
  Same explanation as for `\uElseIf` but with `else`.
- `\uElse{Else's text}`
  the same with comment.
- `\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}
\leIf{cond4}{ok}{wrong}
```

⟹

```
if cond4 then ok;
else 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\;
}
```

⟹

```
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 */
```

11.6 to define multiple condition selection:

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

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

- `\lOther(comment){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
endsw
```

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

11.7 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}`
- `\For(comment){loop's condition}{For's text}`
- `\lFor{loop's condition}{For's text}`
- `\lFor(comment){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 $e$\;
}
```

⟹

```
forall the elements of $S_1$ do
    remove an element e from $S_1$;
    put e in the set $S_2$;
end
for i=1 tomax 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.

11.8 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}`

- `\Repeat(comment after repeat){end loop condition}{the loop}`
- `\Repeat{end loop condition}{the loop}`(comment after until)
- `\Repeat(comment after repeat){end loop condition}{the loop}`(comment after until)
- `\lRepeat{end loop condition}{only one line}`
- `\lRepeat(comment){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.

12 Other language predefined keywords

12.1 french keywords

Hey, I am a frenchy , so I have defined the same as in section 10 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 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}`

12.2 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 Fall}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}`

12.3 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 caso}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}`

12.4 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}{endcase}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}`

12.5 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

13 Known bugs

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

Release notes

```
% - January 6 2013 - revision 5.0
% * CHANGE: SetKwSwith takes now 9 args: 9th arg is the same as
%            previous 8th arg ('end of switch' keyword). New 8th arg is
%            'end of case' keyword. This is due to change of release
%            3.2 which introduce end after case block... as I never
%            test with longend option, I never see that the 'end
%            switch' used for case was not good.
% * CHANGE: when no end keyword is defined in a block macro, then
%            algorithm2e does no more try to print it. So even with lined or noline
%            option, no empty line is printed (before: a blank end was
%            printed, so a blank line appeared)
% * Internal Change: add some internal function to improve readibility
%                    (thanks to Philip K. F. H\lzenspies)
% * ADD: Block markers.
%        You can now ask package to put begin and end keywords automatically at begin
%        and end of blocks, it means each group of commands shifted and enclosed in
%        braces.
%        This is tricky to use but, combined with \SetStartEndCondition and
%        redefinition of keywords, you should be abble to simulate any syntax. See
%        examples in documentation where a generic example is derived in pseudo-code,
%        python and C by keeping code and changing only style using block markers
%        macros, \SetStartEndCondition and some redefinition of keywords.
%        These new block markers macros are:
%        - \AlgoDisplayBlockMarkers and \AlgoDontDisplayBlockMarkers
%        - \SetAlgoBlockMarkers{begin marker}{end marker}
%        - \BlockMarkersSty{text} and \SetBlockMarkersSty
%        Note that a new option has also been added: displayblockmarkers
% * ADD: \leIf macro automatically defined by \SetKw: allow to define
%        an if-then-else on a single line.
% * ADD: new macro \SetStartEndCondition{typo1}{typo2}{typo3} which
%        sets typo around condition in For, If, Switch, Case and
%        Repeat macros. First two are used around For, If, Swith
%        conditions, First and third are used for Case and Repeat
%        condition. Default definition is \SetStartEndCondition{ }{ }{}.
%        A common alternative is \SetStartEndCondition{ (}{) }{)}
%        Can also be used to remove space around condition, for
%        example if you want python style commands:
%        \SetStartEndCondition{ }{}{} and \SetKwFor{For}{for}{:}{}
% * ADD: new environment algomathdisplay which allow display math (like inside \[ \] or $$ $$)
%        handling end line and line number
% * ADD: new command \SetKwProg{Env}{Title}{is}{end} which defines a macro
%        \Env{args}{text}. Env is a block with 'Title' (in \CapSty) at the beginning
%        followed by args followed by 'is' then 'text' is put below inside a block ended
%        by 'end'. If no 'end' is specified, nothing is written (no
%        blank line is inserted). Useful to typeset function or prog for example:
%        \SetKwProg{Fn}{Function}{is}{end} makes \Fn{afunc(i: int) : int}{return 0\;}
%        writes:
%        Function afunc(i: int) : int is
%        | return 0;
%        end
%        or \SetKwProg{Def}{def}{:}{} makes \Def{afunc(i: int)}{return 0\;} writes:
%        def afunc(i: int):
```

```
%        | return 0
%        Tip: combine it with \SetKwFunction to write recursive function algorithm. With
%        example above, you could define \SetKwFunction{\Afunc}{afunc} and then write:
%        Def{\Afunc{i:int}{\eIf{i>0}{\KwRet \Afunc{i-1}}{\KwRet 0\;}} that writes:
%        def afunc(i: int):
%        | if(i>0):
%        |     return afunc(i-1)
%        | else:
%        |     return 0
%        with appropriate typo.
% * ADD: option croatian: croation keywords (thanks to Ivan Gavran)
% * ADD: option ngerman: same as german option but so can be used with global option ngerman
%        of babel
% * ADD: option spanish: Spanish support (thanks to Mario Abarca)
% * ADD: unterminated block: useful to add part separator that doesn't necessary need an end
%        keyword.
%        Designed on the pattern of unterminated if (see \uIf macro) allowing to
%        add a block that is not terminated by a keyword. Such block are defined in the same
%        time as a block is defined by adding a macro beginning with u. So, for example,
%        predefined \SetKwBlock{Begin}{begin}{end} defines now two commands:
%        - \Begin{} as previously which print a begin - end block
%        - \uBegin{} that defines a begin only block
% * FIX: dotocloa option which was broken
% * FIX: uIf and uCase didn't have same behavior when used with
%        noline, vlined or lined option. This is fixed. Side effect: no empty
%        line after an uIf or uCase when used with options lined or vlined
% * FIX: a bug with Repeat Until command when use with side comment on Until
% * FIX: a bug with side text -- text put into () -- of command macro (SetKwIf and so on)
%        which was always setting a ';' even after a \DontPrintSemicolon
% * FIX: a bug with hyperref and chapter definition (thanks to Hubert Meier)
% * FIX: bugs with l macro and side comment
% * FIX: revision number
% * FIX: fix non ascii character (utf8 not yet recognized by all latex engine)
% * FIX: fnum@algocf had an useless parameter which sometimes broke expansion and output an error
% * FIX: works now with multicol package
```

List of Algorithms