The algxpar package

Jander Moreira moreira.jander@gmail.com

June 26, 2023

Abstract

The algxpar package is an extension of the algorithmicx¹/algpseudocode package to handle multi-line text with proper indentation and provide a number of other improvements.

Contents

1 Introduction 2

2 Package usage and options 2

3 Writing pseudocode 4
  3.1 A preamble on comments ................................. 5
  3.2 A preamble on options .................................. 7
  3.3 Statements .............................................. 7
  3.4 Flow Control Blocks ................................... 7
    3.4.1 The if block ..................................... 7
    3.4.2 The switch block ................................. 8
    3.4.3 The for block ................................... 9
    3.4.4 The while block .................................. 10
    3.4.5 The repeat-until block ............................ 11
    3.4.6 The loop block ................................... 12
  3.5 Constants and Identifiers ............................... 12
  3.6 Assignments and I/O .................................. 13
  3.7 Procedures and Functions .............................. 13
  3.8 Comments ............................................. 15
  3.9 Documentation ........................................ 16

4 Customization and Fine Tuning 17
  4.1 Options .............................................. 18
    4.1.1 Fonts, shapes and sizes .......................... 20
    4.1.2 Colors .......................................... 21
    4.1.3 Paragraphs ...................................... 21
  4.2 Languages and translations ............................ 22
  4.3 Other features ....................................... 23

5 To do 25

¹https://ctan.org/pkg/algorithmicx.

*This document corresponds to algxpar v0.99, dated 2023/06/26. This text was last revised June 27, 2023.
1 Introduction

I teach algorithms and programming and have adopted the \texttt{algorithmicx} package (\texttt{algpseudocode}) for writing my algorithms as it provides clear and easy to read pseudocodes with minimal effort to get a visually pleasing code.

The process of teaching algorithms requires a slightly different use of pseudocode than that normally presented in scientific articles, in which the solutions are presented in a more formal and synthetic way. Students work on more abstract algorithms often preceding the actual knowledge of a programming language, and thus the logic of the solution is more relevant than the variables themselves. Likewise, the use of the development strategy by successive refinements also requires a less programmatic and more verbose code. Thus, when discussing the reasoning for solving a problem, it is common to use sentences such as "accumulate current expenses in the total sum of costs", because \texttt{s \leftarrow s + c} is, in this case, too synthetic and necessarily involves knowing how variables work in programs.

The consequence of more verbose pseudocode leads, however, to longer sentences that often span two or more lines. As pseudocodes, by nature, value visual organization, with regard to control structures and indentations, it became necessary to develop a package that supports the use of commands and comments that could be easily displayed when more than one line was needed.

The \texttt{algorithmx} and \texttt{algpseudocode} packages do not natively support multi-line statements. This package therefore extends several macros to handle multiple lines correctly. Some new commands and a number of features have also been added.

2 Package usage and options

This package depends on the following packages:

\begin{itemize}
\item \texttt{algorithmicx} (https://ctan.org/pkg/algorithmicx)
\item \texttt{algpseudocode} (https://ctan.org/pkg/algorithmicx)
\item \texttt{amssymb} (https://ctan.org/pkg/amsfonts)
\item \texttt{fancyvrb} (https://ctan.org/pkg/fancyvrb)
\item \texttt{pgfmath} (https://ctan.org/pkg/pgf)
\item \texttt{pgfplots} (https://ctan.org/pkg/pgf)
\item \texttt{ragged2e} (https://ctan.org/pkg/ragged2e)
\item \texttt{tcolorbox} (https://www.ctan.org/pkg/tcolorbox)
\item \texttt{varwidth} (https://www.ctan.org/pkg/varwidth)
\item \texttt{xcolor} (https://www.ctan.org/pkg/xcolor)
\end{itemize}

To use the package, simply request its use in the preamble of the document.

\texttt{\usepackage[\textit{package options list}]\{algxpar\}}

Currently, the list of package options includes the following.
Description: LZW Compression using a table with all known sequences of bytes.
Input: A flow of bytes
Output: A flow of bits with the compressed representation of the input bytes

1: Initialize a table with all bytes  \( \rightarrow \) each position of the table has a single byte
2: Initialize sequence with the first byte in the input flow
3: while there are bytes in the input do  \( \rightarrow \) wait until all bytes are processed
4: Get a single byte from input and store it in byte
5: if the concatenation of sequence and byte is in the table then
6: Set sequence to sequence + byte  \( \rightarrow \) concatenate without producing any output
7: else
8: Output the code for sequence  \( \rightarrow \) i.e., the binary representation of its position in the table
9: Add the concatenation of sequence and byte to the table  \( \rightarrow \) the table learns a longer sequence
10: Set sequence to byte  \( \rightarrow \) starts a new sequence with the remaining byte
11: end if
12: end while
13: Output the code for sequence  \( \rightarrow \) the remaining sequence of bits
By default, algorithm keywords are developed in English. The English language keyword set is always loaded. When available, other sets of keywords in other languages can be used simply by specifying the language names. The last language in the list is automatically set as the document’s default language.

Currently supported languages:

- **english** (default language, always loaded)
- **brazilian** Brazilian Portuguese

```latex
\usepackage[brazilian]{algxpar}
```

**language = (language name)**

This option chooses the set of keywords corresponding to `(language name)` as the default for the document. This option is available as a general option (see `language`).

This option is useful when other languages are loaded.

```latex
\usepackage[brazilian, language = english]{algxpar}
```

**noend**

The `noend` suppresses the line that indicates the end of a block, keeping the indentation.

See more information in `end` and `noend` options.

```latex
\usepackage[noend]{algxpar}
```

### 3 Writting pseudocode

Algorithms, following the functionality of the `algorithmicx` package, are written within the `algorithmic` environment. The possibility of using a number to determine how the lines will be numbered is maintained as in the original version.

An algorithm is composed of instructions and control structures such as conditionals and loops. And also, some documentation and comments.

\begin{algorithmic}
\Description Calculation of the factorial of a natural number
\Input $n \in \mathbb{N}$
\Output $n!$
\State \Read $n$
\State \Def{factorial} $\gets 1$ \Comment 0! = 1! = 1
\For{$k \gets 2$ \To $n$} \Comment from 2 up
\State \Def{factorial} $\gets \Def{factorial} \times k$ \Comment $(k-1)! \times k$
\EndFor
\State \Write \Def{factorial}
\end{algorithmic}

### 3.1 A preamble on comments

This is the Euclid’s algorithm as provided in the \texttt{algorithmicx} package documentation\(^2\).

\begin{algorithmic}[1]
\Procedure{Euclid}{$a,b$} \Comment{The g.c.d. of a and b}
\State $r \gets a \bmod b$
\While{$r \neq 0$} \Comment{We have the answer if r is 0}
\State $a \gets b$
\State $b \gets r$
\State $r \gets a \bmod b$
\EndWhile
\State \textbf{return} $b$ \Comment{The gcd is b}
\EndProcedure
\end{algorithmic}

Comments are added \textit{in loco} with the \texttt{\Comment} macro, which makes them appear along the right margin. The \texttt{algxpar} package embeds comments as part of the commands themselves in order to add multi-line support.

Until \texttt{algxpar} v0.95, they could be added as an optional parameter before the text, in the style of most \LaTeX macros.

\footnotesize\(^2\)A label was suppressed here.
Using the comment before the text always bothered me somewhat, as it seemed more natural to put it after. Thus, as of v0.99, the comment can be placed after the text (as the second parameter of the macro), certainly making writing algorithms more user-friendly. To maintain backward compatibility, the use of comments before text is still supported, although it is discouraged.

In addition to this change, the use of comments in the new format has been extended to most pseudocode macros, such as \EndWhile for example.

Using \Comment{} still produces the expected result, although it may break automatic tracking of longer lines.

Throughout this documentation, former style comments are denoted as ⟨\texttt{comment*}⟩, while the new format uses ⟨\texttt{comment}⟩.

See more about comments in section 3.8.
3.2 A preamble on options

As of version 0.99, a list of options can be added to each command, changing some algorithm presentation settings. These settings are optional and must be entered using angle brackets at the end of the command.

\begin{algorithmic}<keyword font = \scshape\bfseries, comment width = nice>
\If{$a > b$}[check conditions]
\While{$a > 0$}<keyword color = blue!70>
\Statep\{Call\{Process\}{$a$}\}[process current data]
\EndWhile
\EndIf
\end{algorithmic}

There is a lot of additional information about options and how they can be used. See discussion and full list in section 4.

3.3 Statements

The macros \State and \Statex defined in algorithmicx can still be used for single statements and have the same general behaviour.

For automatic handling of comments and multi-line text, the \Statep macro is available, which should be used instead of \State.

\Statep\{\langle comment*\rangle\}{\langle text\rangle}{\langle comment\rangle}<\langle options\rangle>

The \Statep macro corresponds to an statement that can extrapolate a single line. The continuation of each line is indented from the baseline and this indentation is based on the value indicated in the statement indent option. Any \langle options\rangle specified uniquely affect this macro.

As an example, observe lines 8 and 9 of the LZW compression algorithm on page 26.

3.4 Flow Control Blocks

Flow control is essentially based on conditionals and loop.

3.4.1 The if block

This block is the standard if block.

\begin{algorithmic}
\State \Read $v$
\If{$v < 0$}[is it negative?]
\Statep{$v \gets -v$}[make it positive]
\EndIf
\end{algorithmic}

As an example, observe lines 8 and 9 of the LZW compression algorithm on page 26.
\If\langle\text\rangle\langle\options\rangle
\If shows \text (the condition) and must be closed with an \EndIf, creating a block of nested commands.
Any of the \options specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

\EndIf\langle\options\rangle
\EndIf closes its respective \If.
Any \options specified uniquely affect this macro.

\Else\langle\options\rangle
This macro defines the else part of the \If statement.
Any of the \options specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

\Elsif\langle\text\rangle\langle\options\rangle
\ElsIf defines the \If chaining. The argument \text is the new condition.
Any of the \options specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

3.4.2 The switch block

\begin{algorithmic}
\Statex Get \Id{option}
\Switch{\Id{option}}
\Case{1}[inserts new record]
\State\Call{Insert}{\Id{record}}
\EndCase
\Case{2}[deletes a record]
\State\Call{Delete}{\Id{key}}
\EndCase
\Otherwise
\State \Call{Print \textbar invalid option\textbar}"
\EndOtherwise
\EndSwitch
\end{algorithmic}

Get option
switch option
    case 1 do
        \Call{Insert}{record}
    end case
    case 2 do
        \Call{Delete}{key}
    end case
    otherwise
        Print “invalid option”
    end otherwise
end switch
The \Switch is closed by a matching \EndSwitch. Any of the \langle options \rangle specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

This macro closes a \Switch block. Any \langle options \rangle specified uniquely affect this macro.

When the result of the switch expression matches one of the constants in \langle constants-list \rangle, then the case is executed. Usually the \langle constant-list \rangle is a single constant, a comma-separated list of constants or some kind of range specification. Any of the \langle options \rangle specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

This macro closes a corresponding \Case statement. Any \langle options \rangle specified uniquely affect this macro.

A switch structure can optionally use an otherwise clause, which is executed when no previous cases had a hit. Any of the \langle options \rangle specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

This macro closes a corresponding \Otherwise statement. Any \langle options \rangle specified uniquely affect this macro.

3.4.3 The for block

The for loop uses \For and is also flavored with two variants: for each (\ForEach) and for all (\ForAll).
\begin{algorithmic}
\For{$i \gets 0$ \To $n$}
\State Do something with $i$
\EndFor
\ForAll{$\text{item} \in C$}
\State Do something with $\text{item}$
\EndFor
\ForEach{$\text{item}$ in queue $Q$}
\State Do something with $\text{item}$
\EndFor
\end{algorithmic}

\begin{Verbatim}
for $i \leftarrow 0$ to $n$ do
  Do something with $i$
end for
for all item $\in C$ do
  Do something with item
end for
for each item in queue $Q$ do
  Do something with item
end for
\end{Verbatim}

\For[\langle comment*\rangle\{\langle text\rangle\}][\langle comment\rangle\langle options\rangle]
  \langle text\rangle is used to establish the loop scope.
  Any of the \langle options\rangle specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.
\EndFor[\langle comment\rangle\langle option\rangle]

This macro closes a corresponding \For, \ForEach or \ForAll. Any \langle options\rangle specified uniquely affect this macro.

\ForEach[\langle comment*\rangle\{\langle text\rangle\}][\langle comment\rangle\langle options\rangle]
  Same as \For.
\EndFor[\langle comment\rangle\langle option\rangle]

\ForAll[\langle comment*\rangle\{\langle text\rangle\}][\langle comment\rangle\langle options\rangle]
  Same as \For.
\EndFor[\langle comment\rangle\langle option\rangle]

3.4.4 The while block

\While is the loop with testing condition at the top.
\begin{algorithmic}
  \While{$n > 0$}
    \State Do something
    \State $n \gets n - 1$
  \EndWhile
\end{algorithmic}

\begin{algorithmic}
  \Repeat
    \State Do something
    \State $n \gets n - 1$
  \Until{$n \leq 0$}
\end{algorithmic}

3.4.5 The repeat-until block
The loop with testing condition at the bottom is the \texttt{\Repeat/\Until} block.

\begin{algorithmic}
  \Repeat
    \State Do something
    \State $n \gets n - 1$
  \Until{$n \leq 0$}
\end{algorithmic}

\textbf{\Repeat} \textit{[\texttt{(comment)}]} \textit{[\texttt{(text)}]} \textit{[\texttt{(comment)}]} \textit{[\texttt{(options)}]}
This macro starts the repeat loop, which is closed with \texttt{\Until}. Any of the \texttt{(options)} specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

\textbf{\Until} \textit{[\texttt{(comment)}]} \textit{[\texttt{(text)}]} \textit{[\texttt{(comment)}]} \textit{[\texttt{(options)}]}
In \texttt{(text)} is the boolean expression that, when \texttt{\True}, will end the loop. Any \texttt{(options)} specified uniquely affect this macro.
3.4.6 The loop block

A generic loop is built with \Loop.

\begin{algorithmic}
  \Loop
    \State Do something
    \State $n \gets n + 1$
    \If{$n$ is multiple of 5}
      \State \Continue \hfill \textit{restarts loop}
    \EndIf
    \State Do something else
    \If{$n \leq 0$}
      \State \Break \hfill \textit{ends loop}
    \EndIf
    \State Keep working
  \EndLoop
\end{algorithmic}

\Loop[⟨comment⟩][⟨options⟩]

The generic loop starts with \Loop and ends with \EndLoop. Usually the infinite loop is interrupted by an internal \Break or restarted with \Continue. Any of the ⟨options⟩ specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

\EndLoop[⟨comment⟩][⟨options⟩]

\EndLoop closes a matching \Loop block. Any ⟨options⟩ specified uniquely affect this macro.

3.5 Constants and Identifiers

A few macros for well known constants were defined: \True (TRUE), \False (FALSE), and \Nil (NIL).

The macro \Id was created to handle “program-like” named identifiers, such as \textit{sum}, \textit{word_counter} and so on.

\Id{⟨identifier⟩}

Identifiers are in italics: \Id{value} is value. Its designed to work in both text and math modes: $\Id{offer}_k$ is offer$_k$.  

12
3.6 Assignments and I/O

To support teaching-like, basic pseudocode writing, the macros \texttt{\Read} and \texttt{\Write} are provided.

\begin{algorithmic}
\State \texttt{\Read $v_1, v_2$}
\State $\texttt{mean} \gets \frac{v_1 + v_2}{2}$ [calculate]
\State \texttt{\Write $\texttt{mean}$}
\end{algorithmic}

\begin{verbatim}
read $v_1, v_2$
mean \leftarrow \frac{v_1 + v_2}{2} \triangleright calculate
write mean
\end{verbatim}

The macro \texttt{\Set} can be used for assignments.

\texttt{\Set{$\langle lvalue \rangle$}{$\langle expression \rangle$}} (deprecated)

This macro expands to \texttt{\Id{$\langle #1 \rangle$} \texttt{\gets #2}}.

As the handling of text and math modes should be done and its usage brings no evident advantage, this macro will no longer be supported. It will be kept as is for backward compatibility however.

3.7 Procedures and Functions

Modularization uses \texttt{\Procedure} or \texttt{\Function}.

\begin{algorithmic}
\Procedure{SaveNode}{\Id{node}}
[\texttt{\begin{verbatim}
[\texttt{saves a B$^+$-tree node to disk}]
\If{\Id{node}.\Id{is\_modified}}
\If{$\Id{node}.\Id{address} \texttt{\gets} -1$} \triangleright \texttt{creates a new node on disk}
\Else \State \texttt{\Id{node}.\Id{address} \texttt{\gets} \Id{node}.\Id{address}} \triangleright \texttt{updates the node}
\EndIf
\State \texttt{Write \Id{node} to disk}
\State $\Id{node}.\Id{is\_modified} \texttt{\gets} \texttt{False}$
\EndIf
\EndProcedure
\end{verbatim}}\end{algorithmic}
\begin{algorithmic}
\Function{Factorial}{$n$}[\hspace{1em}n \geq 0$]
\If{$n \in \{0, 1\}$}
\Statep{\Return 1}[base case]
\Else
\Statep{\Return $n \times \Call{Factorial}{n-1}$}[recursive case]
\EndIf
\EndFunction
\end{algorithmic}

\begin{center}
\begin{tabular}{l}
function \texttt{Factorial}(n) \hspace{1em} \triangleright \ n \geq 0 \\
\hspace{1em} if \ n \in \{0, 1\} then \\
\hspace{2em} return 1 \hspace{1em} \triangleright \ \text{base case} \\
\hspace{1em} else \\
\hspace{2em} return \ n \times \texttt{Factorial}(n-1) \hspace{1em} \triangleright \ \text{recursive case} \\
\hspace{1em} end if \\
end function \\
\end{tabular}
\end{center}

\Procedure{(name)}{(argument list)}{(comment)}\langle\hspace{0.5em}options\rangle\end\Procedure\langle\hspace{0.5em}comment\rangle\langle\hspace{0.5em}options\rangle
This macro creates a procedure block that must be ended with \EndProcedure. Any of the \langle options\rangle specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

\EndProcedure\langle\hspace{0.5em}comment\rangle\langle\hspace{0.5em}options\rangle
This macro closes the \Procedure block. Any \langle options\rangle specified uniquely affect this macro.

\Function{(name)}{(argument list)}{(comment)}\langle\hspace{0.5em}options\rangle
This macro creates a function block that must be ended with \EndFunction. A \Return is defined. Any of the \langle options\rangle specified in this macro will affect this command and all items in the inner block, propagating up to and including the closing macro.

\EndFunction\langle\hspace{0.5em}comment\rangle\langle\hspace{0.5em}options\rangle
This macro closes the \Function block. Any \langle options\rangle specified uniquely affect this macro.

For calling a procedure or function, \Call should be used.

\Call{(name)}{(arguments)}\langle\hspace{0.5em}options\rangle
\Call is used to state a function or procedure call. The module's \langle name\rangle and \langle arguments\rangle are mandatory. Any \langle options\rangle specified uniquely affect this macro.
3.8 Comments

The \Comment macro defined by \texttt{algorithmicx} has the same original behavior and has been redefined to handle styling options.

\Comment\{\langle\text\rangle\}<\langle\text\rangle>

The redesigned version of \Comment can be used with \State, \Statex and \Statep. When used with \Statep, it must be enclosed inside the text braces, but multi-line statements should work differently than expected. Any \langle\text\rangle specified uniquely affect this macro.

\begin{minipage}{7.5cm}
\begin{algorithmic}<\textcolor{blue}{\textbf{for viewing purposes only}}\Comment\{first assignment\}\\
\State Store the value zero in variable $x$\Comment{first assignment}\\
\Statep Store the value zero in variable $x$\Comment{first assignment}\\% best choice\\
\Statep Store the value zero in variable $x$[first assignment]\\% best choice\\
\end{algorithmic}
\end{minipage}

\CommentIn\{\langle\text\rangle\}<\langle\text\rangle>

While \Comment pushes text to the end of the line, the macro \CommentIn is “local”. In other words, it just puts a comment in place. Local comments follows regular text and no line changes are checked. Any \langle\text\rangle specified uniquely affect this macro.

\begin{algorithmic}
\If{$a > 0$\Comment{special case}\\
or\\
$a < b$\Comment{general case}\}
\Statep Process data\Comment{may take a while}\}
\EndIf
\end{algorithmic}

\CommentIn\{\langle\text\rangle\}<\langle\text\rangle>

\CommentIn is an alternative to line comments which usually extends to the end of the line. This macro defines a comment with a begin and an end. A comment starts with \texttt{▷} and ends with \texttt{◁}. Any \langle\text\rangle specified uniquely affect this macro.
\begin{algorithmic}
  \If{$a > 0$ \CommentIn{special case} or $a < b$ \CommentIn{general case}}
  \Statep{Process data \CommentIn{may take a while}}
  \EndIf
\end{algorithmic}

if $a > 0$ ▷ special case ◁ or $a < b$ ▷ general case ◁ then
  Process data ▷ may take a while
end if

3.9 Documentation

A series of macros are defined to provide the header documentation for a pseudocode.

```latex
\begin{algorithmic}
  \Description Calculation of the factorial of a natural number through successive multiplications
  \Require $n \in \mathbb{N}$
  \Ensure $f = n!$
\end{algorithmic}
```

**Description**: Calculation of the factorial of a natural number through successive multiplications

**Require**: $n \in \mathbb{N}$

**Ensure**: $f = n!$

\Description \textit{(description text)}

The \texttt{Description} is intended to hold the general description of the pseudocode.

\Require \textit{(pre-conditions)}

The required initial state that the code relies on. These are \textit{pre-conditions}.

\Ensure \textit{(post-conditions)}

The final state produced by the code. These are \textit{post-conditions}.

```latex
\begin{algorithmic}
  \Description Calculation of the factorial of a natural number through successive multiplications
  \Input $n$ (integer)
  \Output $n!$ (integer)
\end{algorithmic}
```

**Description**: Calculation of the factorial of a natural number through successive multiplications

**Input**: $n$ (integer)

**Output**: $n!$ (integer)

\Input \textit{(inputs)}

This works as an alternative to \texttt{Require}, presenting \texttt{Input}.  

16
\textbf{Output \{outputs\}}

This works as an alternative to \textbf{Ensure}, presenting \textbf{Output}.

\section{Customization and Fine Tuning}

As of version 0.99 of \texttt{algxpar}, a series of options have been introduced to customize the presentation of algorithms. Colors and fonts that only apply to keywords, for example, can be specified, providing an easier and more convenient way to customize each algorithm.

The \texttt{AlgSet} macro serves this purpose.

\texttt{\AlgSet\{\{options list\}\}}

This macro sets algorithmic settings as specified in the \{\textit{options list}\}, which is key/value comma-separated list. All settings will be applied to the entire document, starting from the point of the macro call. The scope of a definition made with \texttt{AlgSet} can be restricted to a part of the document simply by including it in a TeX group.

\begin{verbatim}
\AlgSet{algorithmic indent = 1.5cm}
\begin{algorithmic}
  \Statep{Read $k$}
  \If{$k < 0$}
    \Statep{$k \gets -k$}
  \EndIf
  \Statep{Write $k$}
\end{algorithmic}
\end{verbatim}

If the settings are only applied to a single algorithm and not a group of algorithms in a text section, the easiest way is to include the options in the \texttt{algorithmicx} environment.

\begin{verbatim}
\begin{algorithmic}<keyword font = \sffamily\bfseries\itshape>
  \Statep{Read $k$}
  \If{$k < 0$}
    \Statep{$k \gets -k$}
  \EndIf
  \Statep{Write $k$}
\end{algorithmic}
\end{verbatim}

Named styles can also be defined using the \texttt{pgfkeys} syntax.
Sometimes some settings need to be applied exclusively to one command, for example to highlight a segment of the algorithm.

\AlgSet{
  highlight/.style = {
    text color = red!50!black,
    keyword color = red!50!black,
  }
}
\begin{algorithmic}
\Statep{\Commentl{Process $k$}}
\Statep{\Read $k$}
\If{$k < 0$}<highlight>
  \Statep{$k \gets -k$}[back to positive]
\EndIf
\Statep{\Write $k$}
\end{algorithmic}
▷ Process $k$
read $k$
if $k < 0$ then
  $k \gets -k$ \hspace{1cm} // back to positive
end if
write $k$

4.1 Options

This section presents the options that can be specified for the algorithms, either using \AlgSet or the \langle options \rangle parameter of the various macros.

\textbf{language = } \langle language \rangle \hspace{1cm} \textbf{Default: english}

This key is used to choose the keyword language set for the current scope. The language keyword set should already have been loaded through the package options (see section 2).
Structured algorithms use blocks for its structures, marking their begin and end. In pseudocode it is common to use a line to finish a block. Using the option `end`, this line is suppressed. The result is similar to a program written in Python.

This option reverses the behaviour of `end`, and the closing line of a block presented.

```algorithmic
<noend>
\For{$i \gets 0$ \To $N - 1$}
  \For{$j \gets$ \To $N - 1$}
    \If{$m_{ij} < 0$}
      \Statep{$m_{ij} \gets 0$}
    \EndIf
  \EndFor
\EndFor
</noend>
```

This option allows to change a keyword (or define a new one). See section 4.2 for more information on keywords and translations.

```algorithmic
\begin{algorithmic}
  \keywords = {\list of keywords assignments}
  \end{algorithmic}
```
algorithmic indent = \langle width \rangle 

Default: 1em

The algorithmic indent is the amount of horizontal space used for indentation
inner commands.
This option actually sets the algorithmicx’s \algorithmicindent.

comment symbol = \langle symbol \rangle 

Default: \triangleright

The default symbol that preceeds the text in comments is \triangleright (▷),
as used by algorithmicx, and can be changed with this key.
The current comment symbol is available with \CommentSymbol. Do not change
this symbol by redefining \CommentSymbol, as font, shape and color settings will
no longer be respected. Always use \CommentSymbol.

comment symbol right = \langle symbol \rangle 

Default: \triangleleft

This is the symbol that closes a \CommentIn. This symbol is set to < and can be
retrieved with the \CommentSymbolRight macro. Do not attempt to change the
symbol by redefining \CommentSymbolRight, as font, shape and color settings will
no longer be respected. Always use \CommentSymbolRight.

4.1.1 Fonts, shapes and sizes

The options ins this section allows setting font family, shape, weight and size for several
parts of an algorithm.
Notice that color are handled separately (see section 4.1.2) and using \color with
font options will tend to break the document.

text font = \langle font, shape and size \rangle 

Default: –empty–

This setting corresponds to the font family, its shape and size and applies to the
\langle text \rangle field in each of the commands.

comment font = \langle font, shape and size \rangle 

Default: \slshape

This setting corresponds to the font family, its shape and size and applies to all
comments.

keyword font = \langle font, shape and size \rangle 

Default: \bfseries

This setting sets the font family, shape, and size, and applies to all keywords, such
as function or end.

constant font = \langle font, shape and size \rangle 

Default: \scshape

This setting sets the font family, shape, and size, and applies to all constants,
such as NIL, TRUE and FALSE.
This setting also applies when \Constant is used.

module font = \langle font, shape and size \rangle 

Default: \scshape

This setting sets the font family, shape, and size, and applies to both procedure
and function identifiers, as well as their callings with \Call.
4.1.2 Colors

Colors are defined using the \texttt{xcolors} package.

\texttt{text color = \{color\}} \hspace{1cm} \text{Default: . (dot)}

This setting corresponds to the color that applies to the \texttt{text} field in each of the commands.

\texttt{comment color = \{color\}} \hspace{1cm} \text{Default: .!70}

This setting corresponds to the color that applies to all comments.

\texttt{keyword color = \{color\}} \hspace{1cm} \text{Default: . (dot)}

This key is used to set the color for all keywords.

\texttt{constant color = \{color\}} \hspace{1cm} \text{Default: . (dot)}

This setting corresponds to the color that applies to the defined constant (see section 3.5) and also when macro \texttt{\textbackslash Constant} is used.

\texttt{module color = \{color\}} \hspace{1cm} \text{Default: . (dot)}

This color is applied to the identifier used in both \texttt{Procedure} and \texttt{Function} definitions, as well as module calls with \texttt{\textbackslash Call}. Notice that the arguments use \texttt{text color}.

4.1.3 Paragraphs

Multi-line support are internally handled by \texttt{parboxes}.

\texttt{procedure Euclid(a, b)} \hspace{1cm} \texttt{\Comment \{ The g.c.d. of a and b \}}
\texttt{ \hspace{1cm} r \leftarrow \text{a} \mod \text{b} \hspace{1cm} \Comment \{ We have the answer if r is 0 \}}
\texttt{ \hspace{1cm} while r \neq 0 do \hspace{1cm} \Comment \{ The g.c.d. is b \}}
\texttt{ \hspace{1cm} \hspace{1cm} a \leftarrow b \hspace{1cm} \Comment \{ The g.c.d. is b \}}
\texttt{ \hspace{1cm} \hspace{1cm} b \leftarrow r \hspace{1cm} \Comment \{ The g.c.d. is b \}}
\texttt{ \hspace{1cm} \hspace{1cm} r \leftarrow a \mod b \hspace{1cm} \Comment \{ The g.c.d. is b \}}
\texttt{ \hspace{1cm} \hspace{1cm} end while \hspace{1cm} \Comment \{ The g.c.d. is b \}}
\texttt{ \hspace{1cm} \hspace{1cm} return b \hspace{1cm} \Comment \{ The g.c.d. is b \}}
\texttt{ \hspace{1cm} \hspace{1cm} end procedure \hspace{1cm} \Comment \{ The g.c.d. is b \}}

The options in this section should be used to set how these paragraphs will be presented.

\texttt{text style = \{style\}} \hspace{1cm} \text{Default: \texttt{\RaggedRight}}

This \texttt{\{style\}} is applied to the paragraph box that holds the \texttt{\{text\}} field in all commands.

\texttt{comment style = \{style\}} \hspace{1cm} \text{Default: \texttt{\RaggedRight}}

This \texttt{\{style\}} is applied to the paragraph box that holds the \texttt{\{comment\}} field in all algorithmic commands. This setting will not be used with \texttt{\Comment, \Commentl} or \texttt{\CommentIn}.
**comment separator width = \textit{(width)}**  \textit{Default: 1em}

The minimum space between the text box and the \texttt{\CommentSymbol}. This affects the available space in a line for keywords, text and comment.

**statement indent = \textit{(width)}**  \textit{Default: 1em}

This is the \texttt{\hangindent} set inside \texttt{\Statep} statements.

**comment width = \textit{auto|nice|\textit{(width)}}**  \textit{Default: auto}

There are two ways to balance the lengths of \textit{(text)} and \textit{(comments)} on a line, each providing different visual experiences.

In automatic mode (\textit{auto}), the balance is chosen considering the widths that the actual text and comment have, trying to reduce the total number of lines, given there is not enough space in a single line for the keywords, text, comment and comment symbol. The consequence is that each line with a comment will have its own balance.

The second mode, \textit{nice}, sets a fixed width for the entire algorithm, maintaining consistency across all comments. In that case, longer comments will tend to span a larger number of lines. The “nice value” is hardcoded and sets the comment width to \textit{0.4\linewidth}.

Also, a fixed comment width can be specified.

### 4.2 Languages and translations

A simple mechanism is employed to allow keywords to be translated into other languages.

```latex
\begin{algorithmic}<language = brazilian>
  \Procedure{Euclid}{$a,b$}
    \State $r \gets a \bmod b$
    \While{$r \neq 0$}
      \State $a \gets b$
      \State $b \gets r$
      \State $r \gets a \bmod b$
    \EndWhile
    \Statep{\Keyword{return} $b$}
  \EndProcedure
\end{algorithmic}
```

Creating a new keyword set uses the \texttt{\AlgLanguageSet} macro.
This macro sets new values for known keywords as well as new ones. Once created, keywords cannot be deleted.
In case a default keyword is not reset, the English version will be used.
To create a new set, copy the file `algxpar-english.kw.tex` and edit it accordingly.
Note that there is a set of keywords for the lines that close each block. These keys are provided to allow for more versatility in changing how these lines are presented. It is highly recommended that references to other keywords use the `Keyword` macro so that font, color and language changes can be made without any problems.
In translations, these compound keywords do not necessarily need to appear (see file `brazilian.kw.tex`, which follows the settings in `algxpar-english.kw.tex`). However, if defined, there will be different versions for each language.

The mechanism behind \AlgLanguageSet uses the \SetKeyword macro, which is called to adjust the value of a single keyword. To retrieve the value of a given keyword, the \Keyword macro must be used. It returns the formatted value according to the options currently in use for keywords.

\SetKeyword[\langle language\rangle]{\langle keyword\rangle}{\langle value\rangle}

The macro \SetKeyword changes a given \langle keyword\rangle to \langle value\rangle if it exists; otherwise a new keyword is created.
If \langle language\rangle is omitted, the language currently in use is changed.
See also the `keywords` option.

\Keyword[\langle language\rangle]{\langle keyword\rangle}

This macro expands to the value of a keyword in a \langle language\rangle using the font, shape, size, and color determined for the keyword set.
If \langle language\rangle is not specified, the current language is used. \langle keyword\rangle is any keyword defined for a language, including custom ones.

\SetKeyword[german]{if}{wenn} % new
Depending on the language, a keyword can take different forms: \Keyword{if}
\rightarrow \langle English\rangle, \Keyword[german]{if} \langle german\rangle or \Keyword[brazilian]{if} \langle Brazilian Portuguese\rangle.

\Constant[\langle name\rangle]

This macro presents \langle name\rangle using font, shape, size and color defined for constants.

---

3Macros like `algorithmicwhile` from the algorithmicx package are no longer used.
% English keywords
% Moreira, J. (moreira.jander@gmail.com)
\AlgLanguageSet{english}{
  description = Description,
  input = Input,
  output = Output,
  require = Require,
  ensure = Ensure,
  end = end,
  if = if,
  then = then,
  else = else,
  switch = switch,
  of = of,
  case = case,
  otherwise = otherwise,
  do = do,
  while = while,
  repeat = repeat,
  until = until,
  loop = loop,
  foreach = {for~each},
  forall = {for~all},
  for = for,
  to = to,
  downto = {down~to},
  step = step,
  continue = continue,
  break = break,
  function = function,
  procedure = procedure,
  return = return,
  true = True,
  false = False,
  nil = Nil,
  read = read,
  write = write,
  set = set,
}

% Compound keywords
\AlgLanguageSet{english}{
  endwhile = \Keyword{end}~\Keyword{while},
  endfor = \Keyword{end}~\Keyword{for},
  endloop = \Keyword{end}~\Keyword{loop},
  endif = \Keyword{end}~\Keyword{if},
  endswitch = \Keyword{end}~\Keyword{switch},
  endcase = \Keyword{end}~\Keyword{case},
  endotherwise = \Keyword{end}~\Keyword{otherwise},
  endprocedure = \Keyword{end}~\Keyword{procedure},
  endfunction = \Keyword{end}~\Keyword{function},
}
\Module{(name)}

This macro presents \( (name) \) using font, shape, size and color defined for procedures and functions.

5 To do

This is a todo list:

- Add font, shape, size and color settings to a whole algorithm;
- Add font, shape, size and color settings to line numbers;
- Add font, shape, size and color settings to identifiers.

6 Examples

6.1 LZW revisited

\AlgSet{
  comment color = purple,
  comment width = nice,
  comment style = \raggedleft,
}

\Description{LZW Compression using a table with all known sequences of bytes.}
\Input{A flow of bytes}
\Output{A flow of bits with the compressed representation of the input bytes}

1: Initialize a table with all bytes  \( \text{each position of the table has a single byte} \)
2: Inilize sequence with the first byte in the input flow
3: while there are bytes in the input do  \( \text{wait until all bytes are processed} \)
4: Get a single byte from input and store it in byte
5: if the concatenation of sequence and byte is in the table then
6: Set sequence to sequence + byte  \( \text{concatenate without producing any output} \)
7: else
8: Output the code for sequence  \( \text{i.e., the binary representation of its position in the table} \)
9: Add the concatenation of sequence and byte to the table  \( \text{the table learns a longer sequence} \)
10: Set sequence to byte  \( \text{starts a new sequence with the remaining byte} \)
11: end if
12: end while
13: Output the code for sequence  \( \text{the remaining sequence of bits} \)

6.2 LZW revisited again

\AlgSet{
  keyword font = \ttfamily,
  keyword color = green!40!black,
  text font = \itshape,
}
Description: LZW Compression using a table with all known sequences of bytes.
Input: A flow of bytes
Output: A flow of bits with the compressed representation of the input bytes

1: Initialize a table with all bytes ▷ each position of the table has a single byte
2: Initialize sequence with the first byte in the input flow
3: while there are bytes in the input do ▷ wait until all bytes are processed
4: Get a single byte from input and store it in byte
5: if the concatenation of sequence and byte is in the table then
6: Set sequence to sequence + byte ▷ concatenate without producing any output
7: else
8: Output the code for sequence ▷ i.e., the binary representation of its position in the table
9: Add the concatenation of sequence and byte to the table ▷ the table learns a longer sequence
10: Set sequence to byte ▷ starts a new sequence with the remaining byte
11: Output the code for sequence ▷ the remaining sequence of bits
Index

\AlgLanguageSet, 22
algorithmic indent, 19
\AlgSet, 16
brazilian, 3
\Break, 11
\Call, 14
\Case, 8
\Comment, 14
comment color, 21
comment font, 20
comment separator width, 21
comment style, 21
comment symbol, 19
comment symbol right, 20
comment width, 22
\CommentIn, 15
\Commentl, 14
\CommentSymbol, 20
\CommentSymbolRight, 20
\Constant, 23
constant color, 21
constant font, 20
\Continue, 11
\Description, 15
\Else, 7
\Elif, 7
de, 18
\EndCase, 8
\EndFor, 9
\EndFunction, 14
\EndIf, 7
\EndLoop, 11
\EndOtherwise, 9
\EndProcedure, 13
\EndSwitch, 8
\EndWhile, 10
english, 3
\Ensure, 16
\False, 11
\For, 9
\ForAll, 9
\ForEach, 9
\Function, 14
\Id, 11
\If, 7
\Input, 16
\Keyword, 23
keyword color, 21
keyword font, 20
keywords, 19
language, 18
\Loop, 11
\Module, 23
module color, 21
module font, 20
\Nil, 11
noend, 3, 18
\Otherwise, 8
\Output, 16
\Procedure, 13
\Read, 12
\Repeat, 10
\Require, 15
\Return, 14
\Set, 12
\SetKeyword, 23
statement indent, 22
\Statep, 6
\Switch, 8
text color, 20
text font, 20
text style, 21
\True, 11
\Until, 10
\While, 10
\Write, 12