

# The algxpar package\*

Jander Moreira  
moreira.jander@gmail.com

March 4, 2021

## Abstract

The `algxpar` packages is an extension of the `algorithmicx` package to handle multiline text with the proper indentation.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Instalation</b>	<b>2</b>
<b>3</b>	<b>Usage</b>	<b>2</b>
<b>4</b>	<b>Writing pseudocode</b>	<b>3</b>
4.1	Header . . . . .	3
4.2	Constants and identifiers . . . . .	4
4.3	Assignment, reading and writing . . . . .	4
4.4	Comments . . . . .	5
4.5	Statements . . . . .	5
4.6	Conditionals . . . . .	6
4.7	Loops . . . . .	8
4.8	Procedures and functions . . . . .	10
<b>5</b>	<b>Extras</b>	<b>11</b>
<b>6</b>	<b>Implementation</b>	<b>13</b>
<b>7</b>	<b>Customization</b>	<b>18</b>
<b>8</b>	<b>To do...</b>	<b>20</b>
<b>A</b>	<b>An example</b>	<b>20</b>

---

\*This document corresponds to `algxpar` v0.91, dated 2020/05/30.

# Change History

v0.9		
General: Initial version	.....	1
v0.91		
\Id: Macro now can be used as super-/subscripts in math formulas, while still preventing hyphenation in text mode.	.....	13
	\Set: New macro for assignments, using $\leftarrow$	..... 14
	\Set1: New macro for assignments (verbose)	..... 14
	General: Small fix in the position of the triangle in line numbers.	17

## 1 Introduction

I teach algorithms and programming and adopted the `algorithmicx` package (`algpseudocode`) to typeset my code, as it provides a clean, easy to read pseudolanguage algorithms with a minimum effort to write.

As part of the teaching process, I use very verbose commands in my algorithms before the students start to use more sintetic text. For example, I use “*Iniciate a counter  $c$  with the value 0*”, what will become “ $c \leftarrow 0$ ” later. This leads to sentences that often span the text for multiple lines, specially in two-column documents with nested structures.

Unfortunately, `algorithmicx` has no support for multiline statements natively, but it can adapted to use `\parboxes` to achive this goal.

This package, therefore, extends macros to handle multiple lines in a seamlessly way. Some new commands and features are also added.

## 2 Instalation

The package `algxpar` is provided by the files `algxpar.ins` and `algxpar.dtx`.

If the `.sty` file is not available, it can be generated by running the following at a command line prompt.

```
latex algxpar.ins
```

Then the generated `algxpar.sty` must be copied to a directory searched by L<sup>A</sup>T<sub>E</sub>X. Package dependencies can be checked in section 6.

## 3 Usage

The package must be loaded using

```
\usepackage[options]{algxpar}
```

The only option to the package is `brazilian`, which sets the pseudocode “reserved words” to Brazilian Portuguese, so `\While` is rendered **enquanto** instead

of **while**, for example. No other language is supported so far, but a translation can be easily achieved (see section 7).

## 4 Writing pseudocode

The algorithms must be written using the `algorithmic` environment and use basically the same set of macros defined by `algpseudocode`.

```
\begin{algorithmic}
  \langle contents \rangle
\end{algorithmic}
```

### Example

Consider the following code.

```
\begin{algorithmic}
\Function{Max}{$a, b$}
  \If{$a > b$}
    \State{\Return $a$}
  \Else
    \State{\Return $b$}
  \EndIf
\EndFunction
\end{algorithmic}
```

The corresponding typeset is shown below.

```
function MAX( $a, b$ )
  if  $a > b$  then
    return  $a$ 
  else
    return  $b$ 
  end if
end function
```

### 4.1 Header

A header for the algorithm is proposed so the algorithm can provide a description, its inputs and outputs, as well as the preconditions and post-conditions. Therefore, new macros are defined.

<pre>\Description \Input \Output \Require \Ensure</pre>	<p>A description can be provided for the sake of code documentation. The macro <code>\Description</code> is used to provide such a text. The input requirements for the algorithm uses the clause <code>\Input</code> and the produced by the code should be expressed with <code>\Output</code>. Also, the possibility to use <code>\Require</code> and <code>\Ensure</code> remains.</p>
---	--

## Examples

```
\Description Evaluates and prints the factorial of $n$
\Input A non-negative integer number $n$
\Output The value of the factorial $n$
```

---

**Description:** Evaluates and prints the factorial of  $n$   
**Input:** A non-negative integer number  $n$   
**Output:** The value of the factorial  $n$

```
\Require $n \in \{1, 2, \ldots, 10\}$
\Ensure $k = \max(1, 2, \ldots, 10)$
```

---

**Pre:**  $n \in \{1, 2, \dots, 10\}$   
**Post:**  $k = \max(1, 2, \dots, 10)$

## 4.2 Constants and identifiers

\True	Some additional macros were added: \True, \False, and \Nil, producing TRUE,
\False	FALSE, and NIL, respectively.
\Nil	The macro \Id{\langle id \rangle} was included to support long variable names, such as
\Id	<i>maxval</i> or <i>count</i> , for example. This macro handles better ligatures and accented characters than the regular math mode. <code>\$offered\$</code> results in <i>offered</i> and <code>\Id{offered}</code> produces <i>offered</i> . With accented characters, <code>\$magnético\$</code> and <code>\Id{magnético}</code> result in <i>magnético</i> and <i>magnético</i> , respectively.
\TextString	For literal constants, usually represented quoted in programs and algorithms, the macro <code>\TextString{\langle text \rangle}</code> is provided, so <code>\TextString{Error}</code> produces “Error”.
\VisibleSpace	An additional macro called <code>\VisibleSpace</code> is also provided to produce <code>␣</code> . Sometimes the number of spaces is relevant in text strings, so one can write <code>\TextString{a\VisibleSpace\VisibleSpace\VisibleSpace b}</code> to get “a␣␣␣b”. The macros <code>\Id</code> and <code>\TextString</code> work in text and math modes.

## 4.3 Assignment, reading and writing

\gets	The default symbol for assigning values to variables is $\leftarrow$ , provided by <code>\gets</code> . This is a clearer option, once the equal sign is left just for comparisons.
\Read	Although not common in algorithms published in scientific journals, explicit
\Write	reading and writing is necessary for basic algorithms. Therefore <code>\Read</code> and <code>\Write</code> fulfills this need.

```
\Statep{\Read\ $a, b$}
\Statep{$s \gets a + b$}
\Statep{\Write\ $$}
```

▷

---

```

read  $a, b$ 
 $s \leftarrow a + b$ 
write  $s$ 

```

`\Set` Besides `\gets`, the macros `\Set` and `\Setl` can be used for assignments.  
`\Setl` `\Set{⟨id⟩}{⟨value⟩}` is a shortcut to `\Id{⟨id⟩} \gets value`. The “long” version for the assignment is `\Setl{⟨id⟩}{⟨value⟩}`, to get the verbose “Set *id* to *value*”.

## 4.4 Comments

Comments use the symbol  $\triangleright$  preceding the commented text and stay close to the left margin. Comment macros are intended to be used with `\State` or `\Statex`, when no multiline handling is done. Comments with multiline control are considered starting at section 4.5.

`\Comment` The macro `\Comment{⟨text⟩}` puts *⟨text⟩* at the end of the line.  
`\Commentl` A variant, `\Commentl{⟨text⟩}`, places the commented text without moving it to the left margin. It is a “local” comment.  
`\CommentIn` A third option is `\CommentIn{⟨text⟩}`, that places the comment locally, but finishes it with  $\triangleleft$ . Yes, that is really ugly.

```

\State\Commentl{Simple counter}
\State $c \gets 1\Comment{initialize conter}
\State $n \gets \Call{FirstInstance}{}$
\While{$n < 0$}
  \State $c \gets c + 1\Comment{counts one more}
  \State $n \gets \mbox{\CommentIn{all new} } \Call{NewInstance}{}$
\EndWhile

```

---

```

 $\triangleright$  Simple counter
 $c \leftarrow 1$   $\triangleright$  initialize conter
 $n \leftarrow \text{FIRSTINSTANCE}()$ 
while  $n < 0$  do
   $c \leftarrow c + 1$   $\triangleright$  counts one more
   $n \leftarrow \triangleright$  all new  $\triangleleft$  NEWINSTANCE()
end while

```

## 4.5 Statements

`\Statep` The statements should use `\Statep{⟨text⟩}`, which defines a hang indent for continued lines. The `algorithmicx`’s `\State` and `\Statex` can be used as well.  
`\State`  
`\Statex` In opposition to `\State` and `\Statex`, which uses justified text, `\Statep` aligns only to the left, what is aesthetically better than justification in my opinion.

Since `\Statep` uses a `\parbox` to span the text over multiple lines, no room is left for a comment. When needed a comment can be added through the optional argument: `\Statep[<comment>]{<text>}`.

### Example

```
\Statep{Calculate the value of  $x$  using  $k$  and  $m$ ,
considering the stochastic distribution}
\Statep[ $k \neq 0$ ,  $m > k$ ]{Calculate the value of  $x$ 
using  $k$  and  $m$ , considering the stochastic distribution}
```

---

Calculate the value of  $x$  using  $k$  and  $m$ , considering the stochastic distribution  
Calculate the value of  $x$  using  $k$  and  $m$ , considering  $\triangleright k \neq 0, m > k$  the stochastic distribution

## 4.6 Conditionals

The traditional **if-then-else** structure is supported, handling nested commands as well. An **else if** construction avoids nesting **ifs** and getting too much indentation. The macros are: `\If`, `\Else`, and `\ElsIf`.

<code>\If</code>	<code>\If[<i>&lt;comment&gt;</i>]{<i>&lt;condition&gt;</i>}</code> is used for conditional execution and is ended
<code>\Else</code>	with a <code>\EndIf</code> . The optional <i>&lt;comment&gt;</i> is typeset to the left and the <i>&lt;condition&gt;</i> is put in a <code>\parbox</code> . Regular <code>\Comment</code> and <code>\Commentl</code> can be used after <code>\Else</code> .
<code>\ElsIf</code>	The <b>else if</b> clause is specified by <code>\ElsIf[<i>&lt;comment&gt;</i>]{<i>&lt;condition&gt;</i>}</code> .
<code>\Switch</code>	Flow control using a selection structure are provided by the macro
<code>\EndSwitch</code>	<code>\Switch[<i>&lt;comment&gt;</i>]{<i>&lt;selector&gt;</i>}</code> , ended with <code>\EndSwitch</code> . Each matching clause
<code>\Case</code>	uses <code>\Case[<i>&lt;comment&gt;</i>]{<i>&lt;value&gt;</i>}</code> and <code>\EndCase</code> . The default uses <code>\Otherwise</code>
<code>\EndCase</code>	and <code>\EndOtherwise</code> .
<code>\Otherwise</code>	To specify ranges, the macro <code>\Range[<i>&lt;step&gt;</i>]{<i>&lt;start&gt;</i>}{<i>&lt;end&gt;</i>}</code> can be used. For
<code>\EndOtherwise</code>	example, <code>\Range{1}{10}</code> outputs 1..10 and <code>\Range[2]{0}{10}</code> prints 0..10:2.

### Examples

```
\If{ $a < 0$ }
\Statep{ $a$  gets 0}
\EndIf
```

---

**if**  $a < 0$  **then**  
 $a \leftarrow 0$   
**end if**

```

\If[closing doors]{the building is empty and the
  security system is active}
  \Statep{$\Id{status} \gets \TextString{ok}$}
\Else
  \Statep{$\Id{status} \gets \TextString{not ok}$}
\EndIf

```

---

```

if the building is empty and the security system is    ▷ closing doors
  active then
    status ← “ok”
  else
    status ← “not ok”
  end if

```

```

\If[desired status]{ $n \geq 0.8$ }
  \Statep{$\Id{status} \gets \TextString{excellent}$}
\ElseIf{ $n \geq 0.7$ }
  \Statep{$\Id{status} \gets \TextString{great}$}
\ElseIf{ $n \geq 0.5$ }
  \Statep{$\Id{status} \gets \TextString{good}$}
\ElseIf{ $n \geq 0.2$ }
  \Statep{$\Id{status} \gets \TextString{not so good}$}
\Else\Comment{minimum not achieved}
  \Statep{$\Id{status} \gets \TextString{call for help}$}
\EndIf

```

---

```

if  $n \geq 0.8$  then                                ▷ desired status
  status ← “excellent”
else if  $n \geq 0.7$  then
  status ← “great”
else if  $n \geq 0.5$  then
  status ← “good”
else if  $n \geq 0.2$  then
  status ← “not so good”
else                                                ▷ minimum not achieved
  status ← “call for help”
end if

```

```

\Switch[$1 \leq \Id{month} \leq 12$]{\Id{month}}
  \Case{2}
    \If{\Call{IsLeapYear}{\Id{year}}}
      \Statep{$n_{\text{days}} \gets 29$}
    \Else
      \Statep{$n_{\text{days}} \gets 28$}
    \EndIf

```

▷

```

\EndCase
\Case{4, 6, 9, 11}
  \Statep{$n_{\text{days}}$ \gets 30$}
\EndCase
\Otherwise\Comment{1, 3, 5, 7, 8, 10, 12}
  \Statep{$n_{\text{days}}$ \gets 31$}
\EndOtherwise
\EndSwitch

```

---

```

with month of                                ▷  $1 \leq \textit{month} \leq 12$ 
  case 2 do
    if ISLEAPYEAR(year) then
       $n_{\text{days}} \leftarrow 29$ 
    else
       $n_{\text{days}} \leftarrow 28$ 
    end if
  end case
  case 4, 6, 9, 11 do
     $n_{\text{days}} \leftarrow 30$ 
  end case
  otherwise do                                ▷ 1, 3, 5, 7, 8, 10, 12
     $n_{\text{days}} \leftarrow 31$ 
  end otherwise
end with

```

## 4.7 Loops

Loops uses **while**, **repeat until**, and **for** flow control.

**\While**      Loops with condition on top uses **\While**[\comment]{\condition} and are  
**\EndWhile**    ended with **\EndWhile**.

**\Repeat**      When loops have their termination condition tested at the bottom, the macros  
**\Until**      **\Repeat** and **\Until**[\comment]{\condition} are used.

**\For**          The **for** loop starts with **\For**[\comment]{\condition} and ends with  
**\ForAll**      **\EndFor**. To make things more versatile, **\For** can be replaced by **\ForAll** or  
**\ForEach**    **\ForEach**.

**\To**           Some macros for supporting loops are also provided: **\To**, **\DownTo**, and **\Step**,  
**\DownTo**      which defaults to **to**, **downto**, and **step**, respectively.

**\Step**

### Examples

```

\While{there is data in the input stream and no
  termination signal was received}
  \Statep{Get element $$ from the input stream}

```

▷



```

\Statep{\Call{Process}{$e$}}
\EndWhile

```

---

**while** there is data in the input stream and no termination signal was received **do**  
     Get element  $e$  from the input stream  
     PROCESS( $e$ )  
**end while**

```

\Statep[$n_1, n_2 > 0$]{Let $n_1$ and $n_2$
be the two integers in order to find the greatest
number that divides both}
\Repeat
  \Statep[$n_1 \bmod n_2$]{Set \Id{rest} as the
    rest of the integer
    division of $n_1$ by $n_2$}
  \Statep{Redefine $n_1$ with the value of $n_2$}
  \Statep{Redefine $n_2$ with the value of \Id{rest}}
\Until[terminates]{\Id{rest} = 0$}
\Statep[greatest common divisor]{Set $m$ to the value of $n_1$}

```

---

Let  $n_1$  and  $n_2$  be the two integers in order to find the greatest number that divides both  $\triangleright n_1, n_2 > 0$   
**repeat**  
     Set  $rest$  as the rest of the integer division of  $n_1$  by  $n_2$   $\triangleright n_1 \bmod n_2$   
     Redefine  $n_1$  with the value of  $n_2$   
     Redefine  $n_2$  with the value of  $rest$   
**until**  $rest = 0$   $\triangleright terminates$   
 Set  $m$  to the value of  $n_1$   $\triangleright greatest\ common\ divi-$   
 $sor$

```

\For{$i \ gets\ n-1$ \DownTo\ $0$}
  \Statep{$s \ gets\ s + i$}
\EndFor

```

---

**for**  $i \leftarrow n - 1$  **downto** 0 **do**  
      $s \leftarrow s + i$   
**end for**

```

\ForEach[main transactions]{transaction $t$ in the flow
of transactions for month $m$}
  \Statep{\Call{ProcessTransaction}{$t$}}
\EndFor

```

$\triangleright$

---

```

for each transaction  $t$  in the flow of transac-    ▷ main transactions
    tions for month  $m$  do
    PROCESSTRANSACTION( $t$ )
end for

```

```

\ForAll{ $e$  in set  $M$ }
  \State{\Call{ProcessElement}{ $e$ }}
\EndFor

```

---

```

for all  $e$  in set  $M$  do
  PROCESSELEMENT( $e$ )
end for

```

## 4.8 Procedures and functions

\Procedure    Procedure and functions are supported with \Procedure{ $\langle name \rangle$ }{ $\langle arguments \rangle$ }  
 \EndProcedure    and \EndProcedure and \Function{ $\langle name \rangle$ }{ $\langle arguments \rangle$ } and \EndFunction.  
               \Function    The return value for functions uses \Return.  
 \EndFunction  
               \Return

### Examples

```

\Procedure{PrintError}{ $code$ }
  \Switch{ $code$ }
    \Case{1}
      \State{\Write\ \TextString{Not found}}
    \EndCase
    \Case{2}
      \State{\Write\ \TextString{Access denied}}
    \EndCase
    \Case{3}
      \State{\Write\ \TextString{Blocked}}
    \EndCase
    \Otherwise
      \State{\Write\ \TextString{Unknown}}
    \EndOtherwise
  \EndSwitch
\EndProcedure

```

---

```

procedure PRINTERROR( $code$ )
  with  $code$  of
    case 1 do
      write "Not found"
    end case

```

▷

```

    case 2 do
        write "Access denied"
    end case
    case 3 do
        write "Blocked"
    end case
    otherwise do
        write "Unknown"
    end otherwise
end switch
end procedure

```

```

\Function{CelsiusToFahrenheit}{t$}
  \Statep{\Return $\dfrac{9}{5}t + 32$}
\EndFunction

```

---

```

function CELSIUSTOFAHRENHEIT( $t$ )
  return  $\frac{9}{5}t + 32$ 
end function

```

```

\Function[many parameters]{MyFunction}
  {$a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$, $i$, $j$, $k$, $l$}
  \Statep{\Return $\dfrac{a+b+c+d}{f+g+hi^{\sim{j}}}$kl$}
\EndFunction

```

---

```

function MYFUNCTION( $a, b, c, d, e, f, g, h, i, j, k, l$ )
   $\triangleright$  many parameters
  return  $\frac{a+b+c+d}{f+g+hi^j}kl$ 
end function

```

## 5 Extras

**\NewLine** Sometimes just letting the **\parbox** handle the line breaks is not enough. The macro **\NewLine** can be used to manually break lines.

**DefineCode** It is possible to define pieces of code for later use. Using the environment **\UseCode** **DefineCode** with a  $\langle name \rangle$ , a part of the pseudocode can be specified and used with **\UseCode** $\{\langle name \rangle\}$ . The  $\langle name \rangle$  provided should be unique; when repeated the code is overwritten. The macro **\ShowCode** $[\langle options \rangle]\{\langle name \rangle\}$  displays the saved code *verbatim*. Any option for **\VerbatimInput** from **fancyvrb** can be specified in  $\langle options \rangle$ . All chunks of code are written to temporary files.

## Examples

```
\If{$h > 0$ and\NewLine
      ($n_1 \neq 0$ or $n_2 < n_1$) and \NewLine
      $p \neq \text{Nil}$}
  \Statep{\Call{DoSomething}{}}
\Else
  \Statep{\Call{DoSomethingElse}{}}
\EndIf
```

---

```
if  $h > 0$  and
  ( $n_1 \neq 0$  or  $n_2 < n_1$ ) and
   $p \neq \text{NIL}$  then
  DO_SOMETHING()
else
  DO_SOMETHING_ELSE()
end if
```

```
\begin{DefineCode}{half_in_out}
  \Input A number  $n$ 
  \Output Half of  $n$  (i.e.,  $n/2$ )
\end{DefineCode}
\begin{DefineCode}{half_code}
  \Statep[in]{Get  $n$ }
  \Statep[out]{Print  $n/2$ }
\end{DefineCode}
```

Inside algorithmic one can use the following definitions.

```
\UseCode{half_in_out}
\Statep{\Commentl{Code}}
\UseCode{half_code}
```

---

**Input:** A number  $n$

**Output:** Half of  $n$  (i.e.,  $n/2$ )

▷ *Code*

Get  $n$

Print  $n/2$

▷ *in*

▷ *out*

The source is shown by `\ShowCode{half_code}`.

```
\Statep[in]{Get  $n$ }
\Statep[out]{Print  $n/2$ }
```

## 6 Implementation

This package is algxpar v0.91 – L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>.

```

1 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
2 \ProvidesPackage{algxpar}
3 [2020/05/30 v0.91 Algorithms with multiline/paragraph support]

4 \newif\ifaxp@brazilian\axp@brazilianfalse
5 \DeclareOption{brazilian}{\axp@braziliantrue}
6 \DeclareOption*{\PackageWarning{algxpar}{Unknown ‘\CurrentOption’}}
7 \ProcessOptions\relax

   ragged2e: for \RaggedRight
   listings: to get accented characters in verbatim mode (pt_BR)
   amsmath, amssymb: for \triangleright and \triangleleft
   xcolor: gray color for \VisibleSpace
   tcolorbox: verbatim save to file
   fancyvrb: verbatim read from file with tabs

8 \RequirePackage{algorithmicx}
9 \RequirePackage{algpseudocode}
10 \RequirePackage{ragged2e}
11 \RequirePackage{listings}
12 \RequirePackage{amsmath, amssymb}
13 \RequirePackage{xcolor}
14 \RequirePackage{tcolorbox} % to save verbatim
15 \RequirePackage{fancyvrb} % to load verbatim preserving tabs

\True
\False 16 \algnewcommand\algorithmictrue{True}
\Nil 17 \algnewcommand\algorithmicfalse{False}
\Id 18 \algnewcommand\algorithmicnil{Nil}
\TextString 19 \algnewcommand\True{\ensuremath{\textsc{\rmfamily \algorithmictrue}}}
\VisibleSpace 20 \algnewcommand\False{\ensuremath{\textsc{\rmfamily \algorithmicfalse}}}
21 \algnewcommand\Nil{\ensuremath{\textsc{\rmfamily \algorithmicnil}}}
22 \newcommand{\Id}[1]{\ensuremath{\textit{\rmfamily#1}}}
23 \newcommand{\TextString}[1]{\textrm{\normalfont‘‘{\ttfamily\mbox{#1}}’’}}
24 \algnewcommand{\VisibleSpace}{\textrm{\color{black!70}\textvisiblespace}}

\Description
\Input 25 \algnewcommand\algorithmicdescription{\textbf{Description}}
\Output 26 \algnewcommand\algorithmicinput{\textbf{Input}}
\Ensure 27 \algnewcommand\algorithmicoutput{\textbf{Output}}
\Require 28 \algnewcommand\algorithmicensure{\textbf{Ensure}}
29 \algnewcommand\algorithmicrequire{\textbf{Require}}
30 \algnewcommand\Description{\item[\algorithmicdescription:]}
31 \algnewcommand\Input{\item[\algorithmicinput:]}
32 \algnewcommand\Output{\item[\algorithmicoutput:]}
33 \algnewcommand\Ensure{\item[\algorithmicensure:]}
34 \algnewcommand\Require{\item[\algorithmicrequire:]}

```

```

\Read
\Write 35 \alnewcommand{\algorithmicread}{\textbf{read}}
\Set 36 \alnewcommand{\algorithmicwrite}{\textbf{write}}
\Setl 37 \alnewcommand{\algorithmicset{Set}
38 \alnewcommand{\algorithmicsetto{to}
39 \alnewcommand{\Set}[2]{\Id{#1} $\gets$ #2}
40 \alnewcommand{\Setl}[2]{\algorithmicset\ #1 \algorithmicsetto\ #2}
41 \alnewcommand{\Read}{\algorithmicread}
42 \alnewcommand{\Write}{\algorithmicwrite}

\Comment
\Commentl 43 \newcommand{\axp@commentleftsymbol}{$\triangleright$}
\CommentIn 44 \newcommand{\axp@commentrightsymbol}{$\triangleleft$}
45 \alnewcommand{\CommentIn}[1]{\axp@commentleftsymbol~%
46 \textsl{#1}~\axp@commentrightsymbol}
47 \alnewcommand{\Commentl}[1]{\axp@commentleftsymbol~\textsl{#1}}
48 \algrenewcommand{\algorithmiccomment}[1]{%
49 \def\tmp{#1}%
50 \ifx\tmp\empty\else%
51 \hfill\Commentl{#1}%
52 \fi
53 }

\Statep
54 \newlength{\axp@stateindent}
55 \setlength{\axp@stateindent}{\dimexpr\algorithmicindent/2\relax}
56 \alnewcommand{\Statep}[2][\State\algpabox{#1}{#2}{\axp@stateindent}}

\While
\EndWhile 57 \newlength{\axp@whilewidth}
58 \algblockdefx{While}{EndWhile}%
59 [2][\%
60 \settowidth{\axp@whilewidth}{\algorithmicwhile\ }%
61 \algpabox{#1}{\algorithmicwhile\ #2~\algorithmicdo}{\axp@whilewidth}%
62 ]%
63 {\algorithmicend\ \algorithmicwhile}

\Repeat
\Until 64 \newlength{\axp@untilwidth}
65 \algblockdefx{Repeat}{Until}%
66 {\algorithmicrepeat}%
67 [2][\%
68 \settowidth{\axp@untilwidth}{\algorithmicuntil\ }%
69 \axp@algpabox{#1}{\algorithmicuntil\ #2}{\axp@untilwidth}{0}%
70 }

\If
\Else 71 \newlength{\axp@ifwidth}
\ElseIf 72 \newlength{\axp@elseifwidth}
\EndIf

```

```

73 \algblockdefx[If]{If}{EndIf}%
74 [2] []{%
75 \settowidth{\axp@ifwidth}{\algorithmicif\ }%
76 \algpabox[#1]{\algorithmicif\ #2~\algorithmicthen}{\axp@ifwidth}%
77 }
78 {\algorithmicend\ \algorithmicif}
79 \algcblockx[If]{If}{ElsIf}{EndIf}
80 [2] []{%
81 \settowidth{\axp@elseifwidth}{\algorithmicelse\ \algorithmicif\ }%
82 \algpabox[#1]{\algorithmicelse~\algorithmicif\ #2~\algorithmicthen}{\axp@elseifwidth}%
83 }
84 {\algorithmicend\ \algorithmicif}
85 \algcblockx[If]{Else}{EndIf}
86 {\textbf{\algorithmicelse}}
87 {\textbf{\algorithmicend~\algorithmicif}}

\Switch
\EndSwitch 88 \algnewcommand{\algorithmicswitch}{\textbf{switch}}
\Case 89 \algnewcommand{\algorithmicof}{\textbf{of}}
\EndCase 90 \algnewcommand{\algorithmiccase}{\textbf{case}}
\Otherwise 91 \algnewcommand{\algorithmicotherwise}{\textbf{otherwise}}
\EndOtherwise 92 \newlength{\axp@switchwidth}
\Range 93 \algblockdefx{Switch}{EndSwitch}%
94 [2] []{%
95 \settowidth{\axp@switchwidth}{\algorithmicswitch\ }%
96 \algpabox[#1]{\algorithmicswitch\ #2~\algorithmicof}{\axp@switchwidth}%
97 }
98 {\algorithmicend~\algorithmicswitch}
99 \newlength{\axp@casewidth}
100 \algblockdefx{Case}{EndCase}%
101 [2] []{%
102 \settowidth{\axp@casewidth}{\algorithmiccase\ }%
103 \algpabox[#1]{\algorithmiccase\ #2~\algorithmicdo}{\axp@casewidth}%
104 }
105 {\algorithmicend~\algorithmiccase}
106 \algblockdefx{Otherwise}{EndOtherwise}%
107 {\algorithmicotherwise~\algorithmicdo}%
108 {\textbf{\algorithmicend\ \algorithmicotherwise}}
109 \newcommand{\Range}[3] []{%
110 \ensuremath{%
111 #2%
112 \def\temp{#1}%
113 \mathcal{\ldotp\ldotp}\#3
114 \ifx\temp\empty\relax\else\ensuremath{\mathcal{:}\#1}}\fi%
115 }%
116 }

\For
\ForEch 117 \algnewcommand{\To}{\textbf{to}}
\ForAll 118 \algnewcommand{\DownTo}{\textbf{downto}}
\EndFor
\To
\DownTo
\Step

```

```

119 \alnewcommand{\Step}{\textbf{step}}
120 \newlength{\axp@forwidth}
121 \algblockdefx{For}{EndFor}%
122 [2] []{%
123 \settowidth{\axp@forwidth}{\algorithmicfor\ }%
124 \algpabox[#1]{\algorithmicfor\ #2~\algorithmicdo}{\axp@forwidth}%
125 }
126 {\algorithmicend\ \algorithmicfor}
127 \alnewcommand{\algorithmicforeach}{\textbf{for~each}}
128 \newlength{\axp@foreachwidth}
129 \algblockdefx{ForEach}{EndFor}%
130 [2] []{%
131 \settowidth{\axp@foreachwidth}{\algorithmicforeach\ }%
132 \algpabox[#1]{\algorithmicforeach\ #2~\algorithmicdo}{\axp@foreachwidth}%
133 }
134 {\algorithmicend\ ~\algorithmicfor}
135 \newlength{\axp@forallwidth}
136 \algblockdefx{ForAll}{EndFor}%
137 [2] []{%
138 \settowidth{\axp@forallwidth}{\algorithmicforall\ }%
139 \algpabox[#1]{\algorithmicforall\ #2~\algorithmicdo}{\axp@forallwidth}%
140 }%
141 {\algorithmicend\ \algorithmicfor}

\Procedure
\EndProcedure 142 \newlength{\axp@procedurewidth}
\Function 143 \newlength{\axp@namewidth}
\EndFunction 144 \algblockdefx{Procedure}{EndProcedure}%
\Call 145 [3] []{%
146 \settowidth{\axp@procedurewidth}{\algorithmicprocedure~}%
147 \settowidth{\axp@namewidth}{\textsc{#2}()}%
148 \addtolength{\axp@procedurewidth}{0.6\axp@namewidth}%
149 \algpabox[#1]{\algorithmicprocedure\ \textsc{#2}(\#3)}{\axp@procedurewidth}
150 }%
151 {\algorithmicend\ \algorithmicprocedure}
152 \newlength{\axp@functionwidth}
153 \algblockdefx{Function}{EndFunction}%
154 [3] []{%
155 \settowidth{\axp@functionwidth}{\algorithmicfunction~}%
156 \settowidth{\axp@namewidth}{\textsc{#2}()}%
157 \addtolength{\axp@functionwidth}{0.6\axp@namewidth}%
158 \algpabox[#1]{\algorithmicfunction\ \textsc{#2}(\#3)}{\axp@functionwidth}
159 }%
160 {\algorithmicend\ \algorithmicfunction}
161 \algrenewcommand\Call[2]{%
162 \def\argstmp{\#2}%
163 \textsc{#1}\ifx\argstmp\empty\mbox{(\hskip0.5ex)}\else(\#2)\fi%
164 }

\NewLine

```



```

165 \newcommand{\NewLine}{\\}

DefineCode
\UseCode 166 \newenvironment{DefineCode}[1]
\ShowCode 167 {\begingroup\tcbverbatimwrite{\jobname_code_#1.tmp}}
168 {\endtcverbatimwrite\endgroup}
169 \newcommand{\UseCode}[1]{\input{\jobname_code_#1.tmp}}
170 \newcommand{\ShowCode}[2] []{{\small\VerbatimInput[tabsize=4, #1]%
171 {\jobname_code_#2.tmp}}}

\alglinenumber
172 \algrenewcommand{\alglinenumber}[1]%
173 {\hspace{-1.5em}\color{black!35}{\scriptsize#1}\raisebox{0.2ex}{\tiny$\blacktriangleright$}}

\exp@alparbox
174 \newlength{\exp@commentwidth}
175 \setlength{\exp@commentwidth}{0pt}
176 \newcommand{\alparbox}[3] []{\exp@alparbox{#1}{#2}{#3}{1}}
177
178 \newlength{\exp@largestcommentwidth}
179 \setlength{\exp@largestcommentwidth}{0.3\linewidth}
180 \newcommand{\exp@alparbox}[4]{%
181 \def\temp{#1}%
182 \ifx\temp\empty%
183 \setlength{\exp@commentwidth}{-2em}%
184 \else%
185 \settowidth{\exp@commentwidth}{\exp@commentleftsymbol\ #1}%
186 \ifdim\exp@commentwidth>\exp@largestcommentwidth\relax%
187 \setlength{\exp@commentwidth}{\exp@largestcommentwidth}%
188 \fi%
189 \fi%
190 \renewcommand{\NewLine}{\\hspace{#3}}%
191 \parbox[t]{\dimexpr\linewidth-\exp@commentwidth-
192 (\algorithmicindent)*(\theALG@nested - #4)-2em}%
193 {\RaggedRight\setlength{\hangindent}{#3#2\strut}}%
194 \ifx\temp\empty\else%
195 \hfill\exp@commentleftsymbol\hspace{0.5em}%
196 \parbox[t]{\exp@commentwidth}{\slshape\RaggedRight#1}%
197 \fi%
198 \renewcommand{\NewLine}{\\}%
199 }

200 \lstset{
201 literate=
202 {á}{\ 'a}}1 {ê}{\ 'e}}1 {í}{\ 'i}}1 {ô}{\ 'o}}1 {ú}{\ 'u}}1
203 {Á}{\ 'A}}1 {Ê}{\ 'E}}1 {Î}{\ 'I}}1 {Ô}{\ 'O}}1 {Û}{\ 'U}}1
204 {à}{\ 'a}}1 {è}{\ 'e}}1 {ì}{\ 'i}}1 {ö}{\ 'o}}1 {ü}{\ 'u}}1
205 {À}{\ 'A}}1 {È}{\ 'E}}1 {Ì}{\ 'I}}1 {Ö}{\ 'O}}1 {Ü}{\ 'U}}1
206 {ä}{\ "a}}1 {ë}{\ "e}}1 {ï}{\ "i}}1 {ö}{\ "o}}1 {ü}{\ "u}}1

```

```

207 {\~a}{\~a}}1 {\~o}{\~o}}1
208 {\~A}{\~A}}1 {\~O}{\~O}}1
209 {\~A}{\~A}}1 {\~E}{\~E}}1 {\~I}{\~I}}1 {\~O}{\~O}}1 {\~U}{\~U}}1
210 {\~a}{\~a}}1 {\~e}{\~e}}1 {\~i}{\~i}}1 {\~o}{\~o}}1 {\~u}{\~u}}1
211 {\~A}{\~A}}1 {\~E}{\~E}}1 {\~I}{\~I}}1 {\~O}{\~O}}1 {\~U}{\~U}}1
212 {\c}{\c}}1 {\C}{\C}}1
213 {\o}{\o}}1 {\a}{\a}}1 {\A}{\A}}1
214 {\oe}{\oe}}1 {\OE}{\OE}}1 {\ae}{\ae}}1 {\AE}{\AE}}1
215 {\ss}{\ss}}1
216 {\H}{\H}}1 {\U}{\U}}1 {\o}{\o}}1 {\O}{\O}}1
217 {\pounds}{\pounds}}1
218 {\guillemotleft}{\guillemotleft}}1
219 {\guillemotright}{\guillemotright}}1
220 {\n}{\n}}1 {\N}{\N}}1 {\j}{\j}}1
221 }

```

## 7 Customization

By default, the longest width for a comment at the right margin is  $0.3\text{\linewidth}$ . This can be changed using something like the code below.

```

\makeatletter
\setlength{\axp@largestcommentwidth}{new length}
\makeatother

```

The assignment sign can be changed from  $\leftarrow$  to anything else, as well as the symbols used in comments.

```

\renewcommand{\gets}{\mathop{:=}}
\renewcommand{\axp@commentleftsymbol}{\texttt{//}}
\renewcommand{\axp@commentrightsymbol}{\texttt{*/}}

```

To handle languages, the macro `\algxparset` should be used.

```

222 \pgfkeys{
223 algxpar/.cd,
224 brazilian/.code = {\axp@languagebrazilian},
225 english/.code = {\axp@languageenglish},
226 default/.code = {\axp@languageenglish},
227 }
228 \newcommand{\algxparset}[1]{
229 \pgfkeys{
230 algxpar/.cd,
231 #1
232 }
233 }
234 \newcommand{\axp@languagebrazilian}{
235 \algnewcommand\algorithmicdescription{\textbf{Descrição}}

```

```

236 \algrenewcommand\algorithmicinput{\textbf{Entrada}}
237 \algrenewcommand\algorithmicoutput{\textbf{Saída}}
238 \algrenewcommand\algorithmicrequire{\textbf{Pré}}
239 \algrenewcommand\algorithmicensure{\textbf{Pós}}
240 \algrenewcommand\algorithmicend{\textbf{fim}}
241 \algrenewcommand\algorithmicif{\textbf{se}}
242 \algrenewcommand\algorithmicthen{\textbf{então}}
243 \algrenewcommand\algorithmicelse{\textbf{senão}}
244 \algrenewcommand\algorithmicswitch{\textbf{escolha}}
245 \algrenewcommand\algorithmicof{\textbf{de}}
246 \algrenewcommand\algorithmiccase{\textbf{caso}}
247 \algrenewcommand\algorithmicotherwise{\textbf{caso~contrário}}
248 \algrenewcommand\algorithmicfor{\textbf{para}}
249 \algrenewcommand\algorithmicdo{\textbf{faça}}
250 \algrenewcommand\algorithmicwhile{\textbf{enquanto}}
251 \algrenewcommand\algorithmicrepeat{\textbf{repita}}
252 \algrenewcommand\algorithmicuntil{\textbf{até que}}
253 \algrenewcommand\algorithmicloop{\textbf{repita}}
254 \algrenewcommand\algorithmicforeach{\textbf{para~cada}}
255 \algrenewcommand\algorithmicforall{\textbf{para~todo}}
256 \algrenewcommand\algorithmicfunction{\textbf{função}}
257 \algrenewcommand\algorithmicprocedure{\textbf{procedimento}}
258 \algrenewcommand\algorithmicreturn{\textbf{retorne}}
259 \algrenewcommand\algorithmictrue{Verdadeiro}
260 \algrenewcommand\algorithmicfalse{Falso}
261 \algrenewcommand\algorithmicnil{Nulo}
262 \algrenewcommand\algorithmicread{\textbf{leia}}
263 \algrenewcommand\algorithmicwrite{\textbf{escreva}}
264 \algrenewcommand\algorithmicset{Defina}
265 \algrenewcommand\algorithmicsetto{como}
266 \algrenewcommand\To{\textbf{até}}
267 \algrenewcommand\DownTo{\textbf{decrecente~até}}
268 \algrenewcommand\Step{\textbf{passo}}
269 }
270 \newcommand\axp@languageenglish{
271 \algrenewcommand\algorithmicdescription{\textbf{Description}}
272 \algrenewcommand\algorithmicinput{\textbf{Input}}
273 \algrenewcommand\algorithmicoutput{\textbf{Output}}
274 \algrenewcommand\algorithmicrequire{\textbf{Pre}}
275 \algrenewcommand\algorithmicensure{\textbf{Post}}
276 \algrenewcommand\algorithmicend{\textbf{end}}
277 \algrenewcommand\algorithmicif{\textbf{if}}
278 \algrenewcommand\algorithmicthen{\textbf{then}}
279 \algrenewcommand\algorithmicelse{\textbf{else}}
280 \algrenewcommand\algorithmicswitch{\textbf{swith}}
281 \algrenewcommand\algorithmicof{\textbf{of}}
282 \algrenewcommand\algorithmiccase{\textbf{case}}
283 \algrenewcommand\algorithmicotherwise{\textbf{otherwise}}
284 \algrenewcommand\algorithmicfor{\textbf{for}}
285 \algrenewcommand\algorithmicdo{\textbf{do}}

```

```

286 \algrennewcommand{\algorithmicwhile}{\textbf{while}}
287 \algrennewcommand{\algorithmicrepeat}{\textbf{repeat}}
288 \algrennewcommand{\algorithmicuntil}{\textbf{until}}
289 \algrennewcommand{\algorithmicloop}{\textbf{loop}}
290 \algrennewcommand{\algorithmicforeach}{\textbf{for~each}}
291 \algrennewcommand{\algorithmicforall}{\textbf{for~all}}
292 \algrennewcommand{\algorithmicfunction}{\textbf{function}}
293 \algrennewcommand{\algorithmicprocedure}{\textbf{procedure}}
294 \algrennewcommand{\algorithmicreturn}{\textbf{return}}
295 \algrennewcommand{\algorithmictrue}{True}
296 \algrennewcommand{\algorithmicfalse}{False}
297 \algrennewcommand{\algorithmicnil}{Nil}
298 \algrennewcommand{\algorithmicread}{\textbf{read}}
299 \algrennewcommand{\algorithmicwrite}{\textbf{write}}
300 \algrennewcommand{\algorithmicset}{Set}
301 \algrennewcommand{\algorithmicsetto}{to}
302 \algrennewcommand{\To}{\textbf{to}}
303 \algrennewcommand{\DownTo}{\textbf{downto}}
304 \algrennewcommand{\Step}{\textbf{step}}
305 }
306 \axp@languageenglish % default language
307 \ifexp@brazilian\algxparset{brazilian}\fi

```

## 8 To do...

There are lots of improvements to make in the code. I recognize it!

## Appendix

### A An example

```

\Description Inserts a new item in the B-tree structure,
             handling only the root node
\Input The \Id{item} to be inserted
\Output Returns \True\ in case of success, \False\ in
         case of failure (i.e., duplicated keys)
\Function{Insert}{\Id{item}}
  \If{\Id{tree.root address} is \Nil}
    \State{\Comment{Create first node}}
    \State[\Nil\ = new node]{\Id{new root node}
      \gets \Call{GetNode}{\Nil}}
    \State[only item]{Insert \Id{item} in \Id{new
      root node} and set both its left and right
      childs to \Nil; also set \Id{new root
      node.count} to 1}

```

▷

```

\Statep[first node is always a leaf]{Set \Id{new
    root node.type} to \Leaf}
\Statep[flag that node must be updated in file]
    {Set \Id{new root node.modified} to \True}
\Statep{\Call{WriteNode}{\Id{new root node}}}
\Statep{\Id{tree.root address} \gets
    \Id{new root node.address}}
\Statep[update root address in file]
    {\Call{WriteRootAddress}{}}
\Statep{\Return \True}
\Else
\Statep{\Commentl{Insert in existing tree}}
\Statep[]{\Id{success}$,
    $\Id{promoted item}$, $\Id{new node address} \gets
        \Call{SearchInsert}{\Id{tree.root address},
            \Id{item}}}$}
\If[root has splitted]{\Id{success} and
    ${\Id{new node address}}\neq\Nil}$}
\Statep[new root]{${\Id{new root node}} \gets
    \Call{GetNode}{\Nil}}
\Statep{Insert \Id{promoted item} in \Id{new
    root node} and set \Id{new root node.count}
    to 1}
\Statep[tree height grows]{Set \Id{item}'s
    left child to \Id{tree.root
        address} and right child to \Id{new
        node address}}
\Statep[not a leaf]{Set \Id{new root
    node.type} to \Internal}
\Statep{Set \Id{new root node.modified}
    to \True}
\Statep{\Call{WriteNode}{\Id{new root
    node}}}
\Statep{\Id{tree.root address} \gets
    \Id{new root node.address}}
\Statep[update root address in
    file]{\Call{WriteRootAddress}{}}
\EndIf
\Statep[insertion status]{\Return \Id{success}}
\EndIf
\EndFunction

```

---

**Description:** Inserts a new item in the B-tree structure, handling only the root node

**Input:** The *item* to be inserted

**Output:** Returns TRUE in case of success, FALSE in case of failure (i.e., ▷

duplicated keys)

**function** INSERT(*item*)

**if** *tree.root address* is NIL **then**

▷ Create first node

*new root node* ← GETNODE(NIL) ▷ NIL = new node

Insert *item* in *new root node* and set both its left and right child to NIL; also set
 ▷ only item

*new root node.count* to 1

Set *new root node.type* to LEAF ▷ first node is always a leaf

Set *new root node.modified* to TRUE ▷ flag that node must be updated in file

WRITENODE(*new root node*)

*tree.root address* ← *new root node.address*

WRITEROOTADDRESS( ) ▷ update root address in file

**return** TRUE

**else**

▷ Insert in existing tree

*success, promoted item, new node address* ←

SEARCHINSERT(*tree.root address, item*)

**if** *success* and ▷ root has splitted

*new node address* ≠ NIL **then**

*new root node* ← GETNODE(NIL) ▷ new root

Insert *promoted item* in *new root node* and set

*new root node.count* to 1

Set *item's* left child to ▷ tree height grows

*tree.root address* and right child to

*new node address*

Set *new root node.type* to INTERNAL ▷ not a leaf

Set *new root node.modified* to TRUE

WRITENODE(*new root node*)

*tree.root address* ← *new root node.address*

WRITEROOTADDRESS( ) ▷ update root address in file

**end if**

**return** *success* ▷ insertion status

**end if**

**end function**

# Index

<b>C</b>		
\Case	6	
\Comment	5	
\CommentIn	5	
\Comment1	5	
<b>D</b>		
DefineCode (environment)	11	
\Description	3	
\DownTo	8	
<b>E</b>		
\Else	6	
\ElsIf	6	
\EndCase	6	
\EndFunction	10	
\EndOtherwise	6	
\EndProcedure	10	
\EndSwitch	6	
\EndWhile	8	
\Ensure	3	
environments:		
DefineCode	11	
<b>F</b>		
\False	4	
\For	8	
\ForAll	8	
\ForEach	8	
\Function	10	
<b>G</b>		
\gets	4	
<b>I</b>		
\Id	4	
\If	6	
\Input	3	
<b>N</b>		
\NewLine	11	
\Nil	4	
<b>O</b>		
\Otherwise	6	
\Output	3	
<b>P</b>		
\Procedure	10	
<b>R</b>		
\Read	4	
\Repeat	8	
\Require	3	
\Return	10	
<b>S</b>		
\Set	5	
\Set1	5	
\ShowCode	11	
\State	5	
\Statep	5	
\Statex	5	
\Step	8	
\Switch	6	
<b>T</b>		
\TextString	4	
\To	8	
\True	4	
<b>U</b>		
\Until	8	
\UseCode	11	
<b>V</b>		
\VisibleSpace	4	
<b>W</b>		
\While	8	
\Write	4	