The algxpar package*

Jander Moreira
moreira.jander@gmail.com
July 29, 2020

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

Contents
1 Introduction 2
2 Instalation 2
3 Usage 2
4 Writing pseudocode 3
  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
5 Extras 11
6 Implementation 13
7 Customization 18
8 To do... 20
A An example 20

*This document corresponds to algxpar v0.91, dated 2020/05/30.
1 Introduction

I teach algorithms and programming and adopted the \texttt{algorithmicx} package (\texttt{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 "\texttt{Initiate a counter c with the value 0}" what will become "c ← 0" later. This leads to sentences that often span the text for multiple lines, specially in two-column documents with nested structures.

Unfortunately, \texttt{algorithmicx} has no support for multiline statements natively, but it can adapted to use \texttt{parboxes} to acheive this goal.

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

2 Installation

The package \texttt{algxpar} is provided by the files \texttt{algxpar.ins} and \texttt{algxpar.dtx}.

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

\begin{verbatim}
latex algxpar.ins
\end{verbatim}

Then the generated \texttt{algxpar.sty} must be copied to a directory searched by \LaTeX. Package dependencies can be checked in section 6.

3 Usage

The package must be loaded using

\begin{verbatim}
\usepackage[⟨options⟩]{algxpar}
\end{verbatim}

The only option to the package is \texttt{brazilian}, which sets the pseudocode "reserved words" to Brazilian Portuguese, so \texttt{While} is rendered \texttt{enquanto} instead
of \texttt{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 \texttt{algorithmic} environment and use basically the same set of macros defined by \texttt{algpseudocode}.

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

Example

Consider the following code.

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

The corresponding typeset is shown below.

\begin{verbatim}
function Max(a, b)
    if a > b then
        return a
    else
        return b
end if
end function
\end{verbatim}

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.

\begin{verbatim}
\Description A description can be provided for the sake of code documentation. The macro \Description is used to provide such a text. The input requirements for the algorithm uses the clause \Input and the produced by the code should be expressed with \Output. Also, the possibility to use \Require and \Ensure remains.
\Input
\Output
\Require
\Ensure
\end{verbatim}
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: $n \in \{1, 2, \ldots, 10\}$
\Post: $k = \max(1, 2, \ldots, 10)$

4.2 Constants and identifiers

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

The macros \Id and \TextString work in text and math modes.

4.3 Assignment, reading and writing

\gets
The default symbol for assigning values to variables is ←, provided by \gets. This is a clearer option, once the equal sign is left just for comparisons.
\Read
\Write
Although not common in algorithms published in scientific journals, explicit reading and writing is necessary for basic algorithms. Therefore \Read and \Write fulfills this need.
\Statep{\Read \ $a$, \ $b$}
\Statep{\Sum \ $a$ + \ $b$}
\Statep{\Write \ $s$}
\read a, b
\write s

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

4.4 Comments

Comments use the symbol >> 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 \Commentl \CommentIn

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

\Comment
\Commentl
\CommentIn

▷ Simple counter
$c \gets 1$ \Comment{initialize counter}
$n \gets \Call{FirstInstance}{}$
\While{$n < 0$}
$c \gets c + 1$ \Comment{x}
$n \gets \CommentIn{all new} \Call{NewInstance}{}$
\EndWhile

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}

\text{Calculate the value of $x$ using $k$ and $m$, considering the stochastic distribution}

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

4.6 Conditionals

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

\begin{Verbatim}
\If\[\langle\text{comment}\rangle\]{\langle\text{condition}\rangle}
\Else
\ElsIf\[\langle\text{comment}\rangle\]{\langle\text{condition}\rangle}
\Switch\[\langle\text{selector}\rangle\] ended with \texttt{\EndSwitch}. Each matching clause uses \texttt{\Case\[\langle\text{value}\rangle\]} and \texttt{\EndCase}. The default uses \texttt{\Otherwise} and \texttt{\EndOtherwise}.
\EndOtherwise
\end{Verbatim}

To specify ranges, the macro \texttt{\Range\[\langle\text{step}\rangle\]{\langle\text{start}\rangle}{\langle\text{end}\rangle}} can be used. For example, \texttt{\Range{1}{10}} outputs 1..10 and \texttt{\Range{2}{0}{10}} prints 0..10:2.

Examples

\begin{Verbatim}
\If{$a < 0$}
\Statep{$a \gets 0$}
\EndIf
\end{Verbatim}

\texttt{if $a < 0$ then}
\texttt{\hspace{1em} $a \gets 0$}
\texttt{end if}
\If{closing doors}{the building is empty and the security system is active}
   \State{\Id{status} \gets \Text{ok}}
\Else
   \State{\Id{status} \gets \Text{not ok}}
\EndIf

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

\If{desired status}{$n \geq 0.8$}
   \State{\Id{status} \gets \Text{excellent}}
\ElsIf{$n \geq 0.7$}
   \State{\Id{status} \gets \Text{great}}
\ElsIf{$n \geq 0.5$}
   \State{\Id{status} \gets \Text{good}}
\ElsIf{$n \geq 0.2$}
   \State{\Id{status} \gets \Text{not so good}}
\Else\Comment{minimum not achieved}
   \State{\Id{status} \gets \Text{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
   status ← “call for help”  ⊲ minimum not achieved
end if

\Switch{$1 \leq \Id{month} \leq 12$}{\Id{month}}
   \Case{2}
      \If{\Call{IsLeapYear}{\Id{year}}}
         \State{$n_{days} \gets 29$}
      \Else
         \State{$n_{days} \gets 28$}
      \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
```

```
swith month of
  case 2 do
    if \text{ISLEAPYEAR}(\text{year}) then
      $n_{\text{days}} \gets 29$
    else
      $n_{\text{days}} \gets 28$
    end if
  end case
  case 4, 6, 9, 11 do
    $n_{\text{days}} \gets 30$
  end case
  otherwise do $n_{\text{days}} \gets 31$
end otherwise
end with
```

4.7 Loops

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

- Loops with condition on top uses `\While[⟨comment⟩]{⟨condition⟩}` and are ended with `\EndWhile`.
- When loops have their termination condition tested at the bottom, the macros `\Repeat` and `\Until[⟨comment⟩]{⟨condition⟩}` are used.
- The `for` loop starts with `\For[⟨comment⟩]{⟨condition⟩}` and ends with `\EndFor`. To make things more versatile, `\For` can be replaced by `\ForAll` or `\ForEach`.
- Some macros for supporting loops are also provided: `\To`, `\DownTo`, and `\Step`, which defaults to `to`, `downto`, and `step`, respectively.

**Examples**

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

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

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

\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{$n_1 \mod n_2$}{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{\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
\Repeat
\Statep{$n_1 \mod n_2$}{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{\Id{rest} = 0}
\Statep{greatest common divisor}{Set $m$ to the value of $n_1$}

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

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

\ForEach{main transactions}{transaction $t$ in the flow of transactions for month $m$}
\Statep{\Call{ProcessTransaction}{$t$}}
\EndFor
for each transaction $t$ in the flow of transactions for month $m$ do
  \textit{main transactions}
  \texttt{PROCESS TRANSACTION}($t$)
end for

\textit{For all $e$ in set $M$}
\texttt{\textit{Statep}}\{\texttt{Call(ProcessElement}($e$))\}
\textit{EndFor}

for all $e$ in set $M$ do
  \texttt{PROCESS ELEMENT}($e$)
end for

4.8 Procedures and functions

Procedure and functions are supported with \texttt{\textit{Procedure}}\{\texttt{name}\}\{\texttt{arguments}\} and \texttt{\textit{EndProcedure}} and \texttt{\textit{Function}}\{\texttt{name}\}\{\texttt{arguments}\} and \texttt{\textit{EndFunction}}. The return value for functions use \texttt{\textit{Return}}.

Examples

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

\texttt{\textit{procedure}} \texttt{PRINT ERROR($code$)}
  \texttt{\textit{switch}} $code$ \texttt{of}
    \texttt{case} 1 \texttt{do}
      \texttt{write} “Not found”
    \texttt{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

procedure

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

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

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

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

5 Extras

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

It is possible to define pieces of code for later use. Using the environment 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 \textit{verbatim}. Any option for \VerbatimInput from fancyvrb can be specified in \langle options \rangle. All chunks of code are written to temporary files.
Examples

\begin{verbatim}
if $h > 0$ and ($n_1 \neq 0$ or $n_2 < n_1$) and $p \neq \text{Nil}$
\Statep \Call{DoSomething}{}
\Else
\Statep \Call{DoSomethingElse}{}
\EndIf
\end{verbatim}

if $h > 0$ and
($n_1 \neq 0$ or $n_2 < n_1$) and
$p \neq \text{Nil}$
\State \Call{DoSomething}()
\Else
\State \Call{DoSomethingElse}()
\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}
\State \Commentl{in} Get $n$
\State \Commentl{out} Print $n/2$
\end{DefineCode}

Inside algorithmic one can use the following definitions.

\UseCode{half_in_out}
\State \Commentl{Code} \UseCode{half_code}
\UseCode{half_code}

\textbf{Input}: A number $n$
\textbf{Output}: Half of $n$ (i.e., $n/2$)
\Commentl{Code}
\State \Commentl{in} Get $n$
\State \Commentl{out} Print $n/2$

The source is shown by \ShowCode{half_code}.

\State \Commentl{in} Get $n$
\State \Commentl{out} Print $n/2$
6 Implementation

This package is algxpar v0.91 – \LaTeX2ε.

\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{algxpar}[2020/05/30 v0.91 Algorithms with multiline/paragraph support]
\newif\ifaxp@brazilian\axp@brazilianfalse
\DeclareOption{brazilian}{\axp@braziliantrue}
\DeclareOption*{\PackageWarning{algxpar}{Unknown \texttt{\CurrentOption}}}\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
\RequirePackage{algorithmicx}
\RequirePackage{algpseudocode}
\RequirePackage{ragged2e}
\RequirePackage{listings}
\RequirePackage{amsmath, amssymb}
\RequirePackage{xcolor}
\RequirePackage{tcolorbox} % to save verbatim
\RequirePackage{fancyvrb} % to load verbatim preserving tabs
\algnewcommand\algorithmictrue{True}
\algnewcommand\algorithmicfalse{False}
\algnewcommand\algorithmicnil{Nil}
\algnewcommand\True{\ensuremath{\textsc{\rmfamily \algorithmictrue}}}
\algnewcommand\False{\ensuremath{\textsc{\rmfamily \algorithmicfalse}}}
\algnewcommand\Nil{\ensuremath{\textsc{\rmfamily \algorithmicnil}}}
\newcommand{\Id}{\ensuremath{\textit{\rmfamily#1}}}
\newcommand{\TextString}{\textrm{''\ttfamily\mbox{#1}''}}
\algnewcommand{\VisibleSpace}{\textrm{\color{black!70}\textvisiblespace}}
\Description
\Input
\Output
\Ensure
\Require
\algnewcommand{\algorithmicdescription}{\textbf{Description}}
\algnewcommand{\algorithmicinput}{\textbf{Input}}
\algnewcommand{\algorithmicoutput}{\textbf{Output}}
\algnewcommand{\algorithmicensure}{\textbf{Ensure}}
\algnewcommand{\algorithmimicroinput}{\textbf{Input}}
\algnewcommand{\algorithmimicrooutput}{\textbf{Output}}
\algnewcommand{\algorithmimicroensure}{\textbf{Ensure}}
\algnewcommand{\algorithmimicrodescription}{\textbf{Description}}
\algnewcommand{\Step}{\textbf{step}}
\newlength{\axp@forwidth}
\algblockdefx{For}{EndFor}{\algorithmicfor\ #2\algorithmicdo}{\axp@forwidth}
\algnewcommand{\algorithmicforeach}{\textbf{for~each}}
\newlength{\axp@foreachwidth}
\algblockdefx{ForEach}{EndFor}{\algorithmicforeach\ #2\algorithmicdo}{\axp@foreachwidth}
\newlength{\axp@forallwidth}
\algblockdefx{ForAll}{EndFor}{\algorithmicforall\ #2\algorithmicdo}{\axp@forallwidth}
\newlength{\axp@procedurewidth}
\newlength{\axp@namewidth}
\algblockdefx{Procedure}{EndProcedure}{\algorithmicprocedure~\textsc{#2}(#3 )}{\axp@procedurewidth}
\newlength{\axp@functionwidth}
\algblockdefx{Function}{EndFunction}{\algorithmicfunction~\textsc{#2}(#3) }{\axp@functionwidth}
\algrenewcommand{\Call}[2]{\textsc{#1}\ifx\argstmp\empty\mbox{(hskip0.5ex)}\else(#2)\fi}
\NewLine
\Procedure
\EndProcedure
\Function
\EndFunction
\Call
\NewLine
\newcommand{\NewLine}{\}

\DefineCode
\UseCode
\ShowCode
\newenvironment{DefineCode}[1]{\begingroup\tcbverbatimwrite{\jobname_code_#1.tmp}}{\endtcbverbatimwrite\endgroup}
\newcommand{\UseCode}[1]{\input{\jobname_code_#1.tmp}}
\newcommand{\ShowCode}[2][{{\small VerbatimInput[tabsize=4, #1]}]{\jobname_code_#2.tmp}}

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

\algparbox
\newlength{\axp@commentwidth}
\setlength{\axp@commentwidth}{0pt}
\newcommand{\axp@algparbox}[3][{1}]{\axp@algparbox{#1}{#2}{#3}{1}}
\newlength{\axp@largestcommentwidth}
\setlength{\axp@largestcommentwidth}{0.3\linewidth}
\newcommand{\axp@algparbox}[4][{1}]{\def\temp{#1}\ifx\temp\empty\setlength{\axp@commentwidth}{-2em}\else\settowidth{\axp@commentwidth}{\axp@commentleftsymbol \ #1}\ifdim\axp@commentwidth>\axp@largestcommentwidth\relax\setlength{\axp@commentwidth}{\axp@largestcommentwidth}\fi\fi\renewcommand{\NewLine}{\hspace{#3}}\parbox[t]{\dimexpr\linewidth-\axp@commentwidth-(\algorithmicindent)*(#4)-2em}{\RaggedRight\setlength{\hangindent}{#3}#2\strut}\ifx\temp\empty\else\hfill\axp@commentleftsymbol\hspace{0.5em}\parbox[t]{\axp@commentwidth}{\slshape\RaggedRight#1}\fi\renewcommand{\NewLine}{\}}}

\lstset{\literate=\literate={á}{{á}}1 {é}{{é}}1 {í}{{í}}1 {ó}{{ú}}1 {ú}{{ú}}1 {À}{{à}}1 {È}{{è}}1 {Ì}{{ì}}1 {Ò}{{ò}}1 {Ù}{{ù}}1 {ä}{{ä}}1 {ë}{{ë}}1 {ï}{{ï}}1 {ö}{{ö}}1 {ü}{{ü}}1}
7 Customization

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

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

The assignment sign can be changed from ← 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.

\pgfkeys{
  algxpar/.cd,
  brazilian/.code = {\axp@languagebrazilian},
  english/.code = {\axp@languageenglish},
  default/.code = {\axp@languageenglish},
}
\newcommand{\algxparset}[1]{
  \pgfkeys{algxpar/.cd,#1}
}
\newcommand{\axp@languagebrazilian}{
  \algrenewcommand{\algorithmicdescription}{\textbf{Descrição}}
}
\newcommand{\axp@languageenglish}{
  \algrenewcommand{\algorithmicdescription}{\texttt{Description}}
}
\algxparset{english}
There are lots of improvements to make in the code. I recognize it!

Appendix

A An example

\begin{description}
\item[\Description] Inserts a new item in the B-tree structure, handling only the root node
\item[\Input] The \Id{item} to be inserted
\item[\Output] Returns \True\ in case of success, \False\ in case of failure (i.e., duplicated keys)
\item[\Function{Insert}]{\Id{item}}
\item[\If{\Id{tree.root address} is \Nil}]
\item[\Statep\Comment{Create first node}]
\item[\Statep\Nil = new node{\Id{new root node}}]
\item[\Call{GetNode}{\Nil}]
\item[\Statep{only item}{\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]}
\end{description}
\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]{$\Id{item}$’s left child to \Id{tree.root address} and right child to \Id{new node address}}
\Statep[not a leaf]{$\Id{new root node.type}$ 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

\textbf{Description}: Inserts a new item in the B-tree structure, handling only the root node  
\textbf{Input}: The \textit{item} to be inserted  
\textbf{Output}: Returns \texttt{TRUE} in case of success, \texttt{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 childs to NIL; also set
  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
    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.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

C
\Case .......................... 6
\Comment ....................... 5
\CommentIn ..................... 5
\CommentI ...................... 5

D
\DefineCode (environment) ... 11
\Description ................... 3
\DownTo ........................ 8

E
\Else ............................ 6
\ElsIf ........................... 6
\EndCase ........................ 6
\EndFunction .................... 10
\EndOtherwise ................... 6
\EndProcedure ................... 10
\EndSwitch ....................... 6
\Ensure ......................... 3
environments:
  \DefineCode .................... 11

F
\False .......................... 4
\For ............................. 8
\ForAll .......................... 8
\ForEach ........................ 8
\Function ....................... 10

G
\gets ............................ 4

I
\Id .............................. 4
\If ............................... 4
\Input ........................... 3

N
\NewLine ......................... 11

O
\Nil ............................. 4
\Otherwise ....................... 6
\Output ........................... 3

P
\Procedure ....................... 10

R
\Read ............................ 4
\Repeat ........................... 8
\Require .......................... 3
\Return .......................... 10

S
\Set ............................. 5
\SetI ............................. 5
\ShowCode ......................... 11
\State ............................ 5
\StateP ........................... 5
\StateX ........................... 5
\Step ............................. 8
\Switch ........................... 6

T
\TextString ....................... 4
\To ............................... 8
\True ............................. 4

U
\Until ............................ 8
\UseCode ......................... 11

V
\VisibleSpace ..................... 4

W
\While ............................ 8
\Write ............................ 4