The algxpar package*

Jander Moreira
moreira.jander@gmail.com
November 7, 2019

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

Contents
1 Introduction 2
2 Installation 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... 19
A An example 19

*This document corresponds to algxpar v0.9, dated 2019/10/24.
Change History

v0.9
General: Initial version . . . . . . . 1

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{Iniciate a counter} \texttt{c with the value} 0", what will become "\texttt{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{\parbox} 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 \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.

\texttt{latex algxpar.ins}

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[\langle options\rangle]{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 algorithmic environment and use basically the same set of macros defined by 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.

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

\Description\Input\Output\Require\Ensure

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.
Examples

\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 $k = \max(1, 2, \ldots, 10)$

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

4.2 Constants and identifiers

\True Some additional macros were added: \True, \False, and \Nil, producing \text{TRUE}, \text{FALSE}, and \text{NIL}, respectively.
\False The macro \Id{⟨id⟩} was included to support long variable names, such as \text{maxval} or \text{count}, for example. This macro handles better ligatures and accented characters than the regular \text{math} mode. \text{offered} results in \text{offered} and \Id{offered} produces \text{offered}. With accented characters, \text{magnético} and \Id{magnético} result in \text{magnético} and \text{magnético}, respectively.
\Nil An additional macro called \VisibleSpace is also provided to produce \_\_.
\Id maxval or count, for example. Thus, \Id{maxval} or \Id{count}, for example. This macro handles better ligatures and accented characters than the regular \text{math} mode. \text{offered} results in \text{offered} and \Id{offered} produces \text{offered}. With accented characters, \text{magnético} and \Id{magnético} result in \text{magnético} and \text{magnético}, respectively.
\VisibleSpace An additional macro called \VisibleSpace is also provided to produce \_\_.
\Id maxval or count, for example. Thus, \Id{maxval} or \Id{count}, for example. This macro handles better ligatures and accented characters than the regular \text{math} mode. \text{offered} results in \text{offered} and \Id{offered} produces \text{offered}. With accented characters, \text{magnético} and \Id{magnético} result in \text{magnético} and \text{magnético}, respectively.
\TextString For literal constants, usually represented quoted in programs and algorithms, the macro \TextString{⟨text⟩} is provided, so \TextString{Error} produces \text{“Error”}.
\Read An additional macro called \VisibleSpace is also provided to produce \_\_.
\Write 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 \text{math} modes.

4.3 Assignment, reading and writing

\gets The default symbol for assigning values to variables is $\rightarrow$, provided by \gets. This is a clearer option, once the \text{equal} sign is left just for comparisons.
\Read 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.
\Write \Write \Write \Write \Write
\begin{verbatim}
read a, b
s ← a + b
write s
\end{verbatim}

### 4.4 Comments

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

\begin{verbatim}
\Comment The macro \texttt{\textbackslash Comment\{text\}} puts \texttt{text} at the end of the line.
\Commentl A variant, \texttt{\textbackslash Commentl\{text\}}, places the commented text without moving it to the left margin. It is a "local" comment.
\CommentIn A third option is \texttt{\textbackslash CommentIn\{text\}}, that places the comment locally, but finishes it with \texttt{\textbackslash Comment}. Yes, that is really ugly.
\State\Commentl Simple counter
\State \Commentl c ← 1 \Comment initialize counter
\State \Commentl n ← FirstInstance()
\While {n < 0}
\Commentl c ← c + 1 \Comment counts one more
\Commentl n ← \texttt{all new} \CommentIn NewInstance()
\EndWhile
\end{verbatim}

\begin{verbatim}
▷ Simple counter
  c ← 1 \Comment initialize counter
  n ← FIRSTINSTANCE()
  while n < 0 do
    c ← c + 1 \Comment counts one more
    n ← \texttt{all new} \CommentIn NEWINSTANCE()
  end while
\end{verbatim}

### 4.5 Statements

The statements should use \texttt{\textbackslash Statep\{(text)\}}, which defines a hang indent for continued lines. The \texttt{\textbackslash algorithmicx}'s \texttt{\textbackslash State} and \texttt{\textbackslash Statex} can be used as well.

In opposition to \texttt{\textbackslash State} and \texttt{\textbackslash Statex}, which uses justified text, \texttt{\textbackslash Statep} aligns only to the left, what is aesthetically better than justification in my opinion.

Since \texttt{\textbackslash Statep} uses a \texttt{\textbackslash 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: \texttt{\textbackslash 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

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 \ElseIf. \If{}[(comment)]{}{(condition)} is used for conditional execution and is ended with a \EndIf. The optional \comment{} is typeset to the left and the \condition{} is put in a \parbox. Regular \Comment and \Commentl can be used after \Else. The else if clause is specified by \ElseIf{}[(comment)]{}{(condition)}. Flow control using a selection structure are provided by the macro \Switch and ended with \EndSwitch. Each matching clause uses \Case{}[(comment)]{}{(value)} and \EndCase. The default uses \Otherwise and \EndOtherwise. To specify ranges, the macro \Range{[step]}{[start]}{[end]} can be used. For example, \Range{1}{10} outputs 1..10 and \Range{2}{0}{10} 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 active then
status ← “ok”
else
status ← “not ok”
end if

\If[desired status]{$n \geq 0.8$}
\Statep{$\Id{status} \gets \TextString{excelent}$}
\ElsIf{$n \geq 0.7$}
\Statep{$\Id{status} \gets \TextString{great}$}
\ElsIf{$n \geq 0.5$}
\Statep{$\Id{status} \gets \TextString{good}$}
\ElsIf{$n \geq 0.2$}
\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
status ← “excelent”
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”
end if

\Switch[$1 \leq \Id{month} \leq 12$]{\Id{month}}
\Case{2}
\If\Call{IsLeapYear}{\Id{year}}
\Statep{$\Id{n}_{\text{days}} \gets 29$}
\Else
\Statep{$\Id{n}_{\text{days}} \gets 28$}
\EndIf
\EndCase
\Case{4, 6, 9, 11}
\Statep{$\Id{n}_{\text{days}} \gets 30$}
\EndCase
Otherwise \Comment{1, 3, 5, 7, 8, 10, 12}
\State{$n_{\text{days}} \gets 31$}
\EndOtherwise
\EndSwitch

\begin{verbatim}
switch month of
  case 2 do
    if ISLEAPYEAR(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 switch
\end{verbatim}

4.7 Loops

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

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

Examples

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


\[\textbf{while} \text{ there is data in the input stream and no termination signal was received \textbf{do}}\]

\[\text{Get element } e \text{ from the input stream}\]

\[\text{PROCESS}(e)\]

\[\textbf{end while}\]

\[\text{\State} \text{[n}_1, \text{n}_2 > 0\text{]}\{\text{Let } n_1 \text{ and } n_2 \text{ be the two integers in order to find the greatest number that divides both}\}\]

\[\text{\Repeat}\]

\[\text{\State} \text{[n}_1 \bmod n_2\text{]}\{\text{Set } \text{Id}\{\text{rest}\} \text{ as the rest of the integer division of } n_1 \text{ by } n_2\}\]

\[\text{\State} \text{Redefine } n_1 \text{ with the value of } n_2\]

\[\text{\State} \text{Redefine } n_2 \text{ with the value of } \text{Id}\{\text{rest}\}\]\n
\[\text{\Until}[\text{terminates}][\{\text{Id}\{\text{rest}\} = 0\}\}

\[\text{\State} \text{[greatest common divisor]}\{\text{Set } m \text{ to the value of } n_1\}\]

\[\text{Let } n_1 \text{ and } n_2 \text{ be the two integers in order to find the greatest number that divides both}\]

\[\textbf{repeat}\]

\[\text{Set } \text{rest} \text{ as the rest of the integer division of } n_1 \text{ by } n_2\]

\[\text{Redefine } n_1 \text{ with the value of } n_2\]

\[\text{Redefine } n_2 \text{ with the value of } \text{rest}\]

\[\textbf{until } \text{rest} = 0\]

\[\text{Set } m \text{ to the value of } n_1\]

\[\textbf{\For}{i \gets n-1 \text{ \DownTo} \ 0}\]

\[\text{\State} s \gets s + i\]

\[\textbf{\EndFor}\]

\[\text{\For}{i \gets n-1 \text{ downto } 0}\]

\[\text{\State} s \gets s + i\]

\[\textbf{\EndFor}\]

\[\textbf{\ForEach}{(main transactions)}\{\text{transaction } t \text{ in the flow of transactions for month } m}\]

\[\text{\State} \text{Call \{ProcessTransaction\}(t)}\]

\[\textbf{\EndFor}\]
for each transaction \( t \) in the flow of transactions for month \( m \) do

\( \text{PROCESS TRANSACTION}(t) \)
end for

\texttt{ForAll}\{$e$ in set $M$}\{
\texttt{Statep}\{$\text{Call\{ProcessElement\}}(e)$}\}
\texttt{EndFor}

for all \( e \) in set \( M \) do

\( \text{PROCESS ELEMENT}(e) \)
end for

4.8 Procedures and functions

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

Examples

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

\texttt{procedure PrintError\{(code)\}}
\texttt{switch \ text{code} of}
\texttt{case 1 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
direct procedure
end procedure

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

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

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

\Function{MyFunction}{$a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$, $i$, $j$, $k$, $l$}
  \Return \frac{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

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

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

Inside algorithmic one can use the following definitions.
\UseCode{half_in_out}\NewLine
\Statep{\Commentl{Code}}\NewLine
\UseCode{half_code}\NewLine

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

The source is shown by \ShowCode{half_code}.\NewLine
\Statep[in]{Get $n$}\NewLine
\Statep[out]{Print $n/2$}
6 Implementation

This package is algxpar v0.9 – \LaTeX\ 2e.

1 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
2 \ProvidesPackage{algxpar}
3 [2019/10/24 v0.9 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
8 \ragged2e: for \RaggedRight
9 listings: to get accented characters in verbatim mode (pt_BR)
10 amsmath, amssymb: for \triangleright \text{ and } \triangleleft
11 xcolor: gray color for \VisibleSpace
12 tcolorbox: verbatim save to file
13 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
16 \algnewcommand\algorithmictrue{\textsc{True}}
17 \algnewcommand\algorithmicfalse{\textsc{False}}
18 \algnewcommand\algorithmicnil{\textsc{Nil}}
19 \algnewcommand\True{\mbox{\algorithmictrue}}
20 \algnewcommand\False{\mbox{\algorithmicfalse}}
21 \algnewcommand\Nil{\mbox{\algorithmicnil}}
22 \newcommand[1]{\Id}[1]{\mbox{\textit{\rmfamily #1}}}
23 \newcommand[1]{\TextString}[1]{\textrm{\normalfont''{\ttfamily\mbox{#1}}''}}
24 \algnewcommand\VisibleSpace{\textrm{\color{black!70}\textvisiblespace}}
25 \algnewcommand\algorithmicdescription{\textbf{Description}}
26 \algnewcommand\algorithmicinput{\textbf{Input}}
27 \algnewcommand\algorithmicoutput{\textbf{Output}}
28 \algnewcommand\algorithmicensure{\textbf{Ensure}}
29 \algnewcommand\algorithmicrequire{\textbf{Require}}
30 \algnewcommand{\Description}[1]{\item[\algorithmicdescription:]\item[\algorithmicinput:]}
31 \algnewcommand{\Input}[1]{\item[\algorithmicinput:]}
32 \algnewcommand{\Output}[1]{\item[\algorithmicoutput:]}
33 \algnewcommand{\Ensure}[1]{\item[\algorithmicensure:]}
34 \algnewcommand{\Require}[1]{\item[\algorithmicrequire:]}
7 Customization

By default, the longest width for a comment at the right margin is 0.3\textwidth. 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{*/}}

The translation to Brazilian Portugues is straightforward.

\ifaxp@brazilian
\RequirePackage{icomma} % comma as decimal separator
\algrenewcommand{\algorithmicdescription}{\textbf{Descrição}}
\algrenewcommand{\algorithmicinput}{\textbf{Entrada}}
\algrenewcommand{\algorithmicoutput}{\textbf{Saída}}
\algrenewcommand{\algorithmicrequire}{\textbf{Pré}}
\algrenewcommand{\algorithmicensure}{\textbf{Pós}}
\algrenewcommand{\algorithmicend}{\textbf{fim}}
\algrenewcommand{\algorithmicif}{\textbf{se}}
\algrenewcommand{\algorithmicthen}{\textbf{então}}
\algrenewcommand{\algorithmicelse}{\textbf{senão}}
\algrenewcommand{\algorithmicswitch}{\textbf{escolha}}
\algrenewcommand{\algorithmicof}{\textbf{de}}
\algrenewcommand{\algorithmiccase}{\textbf{caso}}
\algrenewcommand{\algorithmicotherwise}{\textbf{caso~contrário}}
\algrenewcommand{\algorithmicfor}{\textbf{para}}
\algrenewcommand{\algorithmicdo}{\textbf{faça}}
\algrenewcommand{\algorithmicwhile}{\textbf{enquanto}}
\fi
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}
    \Statep{\Commentl{Create first node}}
    \Statep{\Nil = new node}{\Id{new root node}}
    \Statep{\Call{GetNode}{\Nil}}
    \Statep{\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{\Call{WriteRootAddress}{}}
\Statep{\Return \True}
\Else
\Statep{\Commentl{Insert in existing tree}}
\Statep{\Id{success} = \Id{promoted item} \gets \Call{SearchInsert}{\Id{tree.root address}, \Id{item}}}
\If[root has splitted]{\Id{success} and \Id{new node address} \neq \Nil}
\Statep{\Id{new root node} \gets \Call{GetNode}{\Nil}}
\Statep{\Call{WriteNode}{\Id{new root node}}}
\Statep{\Id{tree.root address} \gets \Id{new root node.address}}
\Else
\Statep{\Commentl{tree height grows}}
\EndIf
\Statep{\Call{WriteRootAddress}{}}
\EndIf
\Statep{\Return \Id{success}}
\EndFunction

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

Input: The \textit{item} to be inserted.

Output: Returns True in case of success, False in case of failure (i.e., duplicated keys).

\textbf{function} \textit{INSERT(item)}
\hspace*{1em} \textbf{if} \ \textit{tree.root address} is NIL \ \textbf{then}
\hspace*{2em} \triangleright \ \textit{Create first node}
new root node ← GetNode(Nil)  \quad \triangleright \quad \text{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  \quad \triangleright \quad \text{first node is always a leaf}

Set new root node.modified to True  \quad \triangleright \quad \text{flag that node must be updated in file}

\text{WriteNode}(\text{new root node})

\text{tree.root address ← new root node.address}

\text{WriteRootAddress()}  \quad \triangleright \quad \text{update root address in file}

\text{return True}

\text{else}
\quad \triangleright \quad \text{Insert in existing tree}
\quad \text{success, promoted item, new node address} ← \text{SearchInsert}(\text{tree.root address, item})

\text{if success and new node address} \neq \text{Nil then}
\quad \text{new root node ← GetNode(Nil)}  \quad \triangleright \quad \text{new root}
\quad \text{Insert promoted item in new root node and set new root node.count to 1}
\quad \text{Set item’s left child to tree.root address and right child to new node address}
\quad \text{Set new root node.type to INTERNAL}  \quad \triangleright \quad \text{not a leaf}
\quad \text{Set new root node.modified to True}
\quad \text{WriteNode(new root node)}
\quad \text{tree.root address ← new root node.address}
\quad \text{WriteRootAddress()}  \quad \triangleright \quad \text{update root address in file}
\quad \text{end if}
\text{return success}  \quad \triangleright \quad \text{insertion status}
\text{end if}
\text{end function}
Index

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

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

E
\Else ............................ 6
\ElsIf ............................ 6
\EndCase ........................ 6
\EndFunction .................... 10
\EndOtherwise .................. 6
\EndProcedure ................... 10
\EndSwitch ....................... 6
\EndWhile ........................ 8
\Ensure ........................... 3
environments:
  DefineCode .................... 11

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

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

I
\Id ................................. 4
\If ................................. 6
\Input ............................. 3

N
\NewLine .......................... 11
\Nil ................................. 4

O
\Otherwise ........................ 6
\Output ............................. 3

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

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

S
\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