Algpseudocodex Package Documentation

Christian Matt
https://github.com/chrmatt/algpseudocodex

February 17, 2023
v1.1.0

Abstract

This package allows typesetting pseudocode in LaTeX. It is based on `algpseudocode` from the `algorithmicx` package and uses the same syntax, but adds several new features and improvements. Notable features include customizable indent guide lines and the ability to draw boxes around parts of the code for highlighting differences. This package also has better support for long code lines spanning several lines and improved comments.

Contents

1 Basic Usage **3**
- 1.1 Algorithmic Block . . . 3
- 1.2 Simple Statements and Commands . . . 3
- 1.3 Blocks . . . 3
  - 1.3.1 While Loop . . . 3
  - 1.3.2 For Loop . . . 3
  - 1.3.3 For-All Loop . . . 4
  - 1.3.4 Loop . . . 4
  - 1.3.5 Repeat-Until Loop . . . 4
  - 1.3.6 If Statement . . . 4
  - 1.3.7 Procedure . . . 4
  - 1.3.8 Function . . . 4
- 1.4 Require and Ensure . . . 4
- 1.5 Comments . . . 5

2 Boxes **5**
- 2.1 Boxes Around Multiple Lines of Code . . . 5
- 2.2 Boxes Inside Single Line . . . 5

3 Package Options **6**
- 3.1 noEnd . . . 6
- 3.2 indLines . . . 6
- 3.3 spaceRequire . . . 6
- 3.4 italicComments . . . 7
- 3.5 rightComments . . . 7
- 3.6 commentColor . . . 7
- 3.7 beginComment and endComment . . . 8
- 3.8 beginLComment and endLComment . . . 8

4 Customization **8**
- 4.1 Style of Indent Guide Lines . . . 8
- 4.2 Default Style of Boxes . . . 8
- 4.3 Changing Keywords . . . 9

5 Revision History **9**

1 Basic Usage

To use the package, load it in your preamble:

```
\usepackage{algpseudocodex}
```

Basic usage is identical to `algpseudocode` from the `algorithmicx` package. Pseudocode written for that package should also be compatible with `algpseudocodex`.

1.1 Algorithmic Block

Pseudocode can be typeset inside a algorithmic blocks:

```
\begin{algorithmic}[line numbering]
    ...
\end{algorithmic}
```

The optional argument specifies how lines are numbered. 0 means no numbering, $n > 0$ means every nth line gets a number. The default is 0, i.e., no line numbers will be typeset if no optional argument is provided.

1.2 Simple Statements and Commands

Statements start with `\State`. The command `\Statex` can be used to start a new line that does not get a new line number.

The commands `\Return` and `\Output` can be used for return values of functions and outputs. They do not start a new line on their own, so they need to be used together with `\State`.

The `\Call` command is used for procedure calls. It takes two arguments: The first one is the name of the procedure and the second one are the arguments.

Example

```
\begin{algorithmic}[1]
    \State first line
    \Statex continuing first line
    \State \Call{Proc}{a1, a2}
    \State \Output Hello World!
\end{algorithmic}
```

1: first line
   continuing first line
2: PROC(a1, a2)
3: **output** Hello World!

1.3 Blocks

Blocks are used for loops, conditional statements, and functions. Blocks can also be nested within other blocks.

1.3.1 While Loop

```
\While{condition}
    \State body
\EndWhile
```

while condition **do**
  body

1.3.2 For Loop

```
\For{$n = 1, \dots, 10$}
    \State body
\EndFor
```

for $n = 1, \dots, 10$ **do**
  body

1.3.3 For-All Loop

```
\ForAll{$n \in \{1, \dots, 10\}$}
    \State body
\EndFor
```

for all $n \in \{1, \dots, 10\}$ **do**
    body

1.3.4 Loop

```
\Loop
    \State body
\EndLoop
```

loop
    body

1.3.5 Repeat-Until Loop

```
\Repeat
    \State body
\Until{$n > 10$}
```

repeat
    body
until $n > 10$

1.3.6 If Statement

```
\If{condition}
    \State body
\ElsIf{condition}
    \State body
\Else
    \State body
\EndIf
```

if condition **then**
    body
else if condition **then**
    body
else
    body

The `\ElsIf` and `\Else` parts are optional.

1.3.7 Procedure

```
\Procedure{name}{parameters}
    \State body
\EndProcedure
```

procedure NAME(parameters)
    body

1.3.8 Function

```
\Function{name}{parameters}
    \State body
\EndFunction
```

function NAME(parameters)
    body

1.4 Require and Ensure

To specify conditions on the inputs and outputs of an algorithm, `\Require` and `\Ensure` can be used.

Example

```
\begin{algorithmic}[1]
    \Require $x \in \{0,1\}$
    \Ensure $y \in \{1,2\}$
    \State $y \gets x+1$
    \State \Return $y$
\end{algorithmic}
```

Require: $x \in \{0, 1\}$
Ensure: $y \in \{1, 2\}$
1: $y \leftarrow x + 1$
2: **return** y

1.5 Comments

There are two ways to typeset code comments: The command `\Comment` can be used to add shorts comments to the end of the current line. The command `\LComment` can be used to typeset long comments that can span multiple lines. Comments with `\LComment` start on a new line.

Example

```
\begin{algorithmic}[1]
    \State $x \gets y^2$
    \LComment{The next two lines
        increment both $x$ and $y$.}
    \State $x \gets x + 1$
    \Comment{Increment $x$.}
    \State $y \gets y + 1$
    \Comment{Increment $y$.}
\end{algorithmic}
```

1: $x \leftarrow y^2$
2: ▷ *The next two lines increment both x and y.* ◁
3: $x \leftarrow x + 1$ ▷ *Increment x.*
4: $y \leftarrow y + 1$ ▷ *Increment y.*

2 Boxes

A unique feature of the `algpseudocodex` package is the ability to draw boxes around pieces of code. There are two different methods to do so: One for drawing boxes around multiple lines of code, and another one for drawing a box around a string on a single line of code.

2.1 Boxes Around Multiple Lines of Code

The command `\BeginBox[style]` is used to set the beginning of the box. The optional argument determines the style of the drawn box. The boxes are drawn using TikZ, so any TikZ style can be used. The default style can be changed as described in Section 4.2. The command `\EndBox` is used to set the end of the last started box. Boxes can be nested arbitrarily, but every `\BeginBox` needs a matching `\EndBox`.

Example

```
\begin{algorithmic}
    \BeginBox
        \State first line
        \BeginBox[fill=yellow]
            \State second line
            \State another line
        \EndBox
    \EndBox
    \BeginBox[draw=blue,dashed]
        \State last line
    \EndBox
\end{algorithmic}
```

first line
second line
another line
last line

2.2 Boxes Inside Single Line

The command `\BoxedString[style]{text}` is used to typeset text with a box around it. The optional argument determines the style of the box, as in `\BeginBox`. The default style is the same as for `\BeginBox`.

Example

```
\begin{algorithmic}
    \State first line
    \State second line with
    \BoxedString[fill=yellow]{box}
    \State last line
\end{algorithmic}
```

first line
second line with box
last line

3 Package Options

When loading `algpseudocodex` the options describe in this section can be set. They syntax for setting `option1` to `value1` and `option2` to `value2` is:

```
\usepackage[option1=value1,option2=value2]{algpseudocodex}
```

3.1 noEnd

possible values: `true`, `false`

default: `true`

If `false`, the end of blocks are marked with the expression "end" followed by the name of the block.

Example

`noEnd=false`:

if $x > 0$ **then**
  $x \leftarrow x - 1$
end if

`noEnd=true`:

if $x > 0$ **then**
  $x \leftarrow x - 1$

3.2 indLines

possible values: `true`, `false`

default: `true`

If `true`, indent guide lines are drawn. The style of the lines can be customized as described in Section 4.1.

Example

`indLines=false`:

if $x > 0$ **then**
  $x \leftarrow x - 1$

`indLines=true`:

if $x > 0$ **then**
  $x \leftarrow x - 1$

3.3 spaceRequire

possible values: `true`, `false`

default: `true`

If `true`, vertical space is added before every `\Require` except the one on the first line. This is useful for specifying different behaviors depending on the provided input.

Example

`spaceRequire=false`:

Require: $x \in \{0, 1\}$
  return x
Require: $x \in \{1, 2\}$
  return $x - 1$

`spaceRequire=true`:

Require: $x \in \{0, 1\}$
  return x

Require: $x \in \{1, 2\}$
  return $x - 1$

3.4 italicComments

possible values: `true`, `false`

default: `true`

If `true`, all comments are typeset in italic font. If `false`, comments are typeset in roman font.

Example

`italicComments=false`:

▷ Long comment. ◁
$x \leftarrow 0$ ▷ Short comment.

`italicComments=true`:

▷ *Long comment.* ◁
$x \leftarrow 0$ ▷ *Short comment.*

3.5 rightComments

possible values: `true`, `false`

default: `true`

If `true`, comments typeset with `\Comment` are right justified on the current line. If a comment does not fit on the current line, no justification is applied. If `false`, all comments are typeset right after the end of the current line.

Does not affect long comments typeset with `\LComment`.

Example

`rightComments=false`:

▷ *No effect on long comments.* ◁
$x \leftarrow 0$ ▷ *Short comment.*
$x \leftarrow x^2$ ▷ *Does not fit on the current line and is thus not justified.*

`rightComments=true`:

▷ *No effect on long comments.* ◁
$x \leftarrow 0$ ▷ *Short comment.*
$x \leftarrow x^2$ ▷ *Does not fit on the current line and is thus not justified.*

3.6 commentColor

possible values: Any color that can be used in `\textcolor`.

default: `gray`

Defines the color in which comments are typeset.

Example

`commentColor=black`:

▷ *Long comment.* ◁
$x \leftarrow 0$ ▷ *Short comment.*

`commentColor=blue`:

▷ *Long comment.* ◁
$x \leftarrow 0$ ▷ *Short comment.*

3.7 beginComment and endComment

possible values: Any string that can be typeset in text mode.

default: `$\triangleright$~` and (empty)

Used to indicate the beginning and end of comments typeset with `\Comment`, respectively.

Example

`beginComment=//~`:

▷ *Long comment.* ◁
$x \leftarrow 0$ *// Short comment.*

`beginComment=/*~, endComment=~*/`:

▷ *Long comment.* ◁
$x \leftarrow 0$ */* Short comment. **/*

3.8 beginLComment and endLComment

possible values: Any string that can be typeset in text mode.

default: `$\triangleright$~` and `~$\triangleleft$`

Used to indicate the beginning and end of long comments typeset with `\LComment`, respectively.

Example

`beginLComment=/*~, endLComment=~*/`:

/ Long comment. **/*
$x \leftarrow 0$ ▷ *Short comment.*

4 Customization

4.1 Style of Indent Guide Lines

Indent guide lines are drawn using TikZ and consequently any TikZ style can be used. To set the style, use:

```
\tikzset{algpxIndentLine/.style={style}}
```

The default style is `draw=gray,very thin`.

Example

`algpxIndentLine/.style={draw=blue,dashed}`:

if $x > 0$ **then**
  $x \leftarrow x - 1$

4.2 Default Style of Boxes

Boxes are drawn using TikZ and consequently any TikZ style can be used. To set the default style, use:

```
\tikzset{algpxDefaultBox/.style={style}}
```

The default style is `draw`.

4.3 Changing Keywords

As in the `algorithmicx` package, keywords can be renamed using the syntax:

`\algrenewcommand\keyword{new name}`

The following keywords can be customized:

- `\algorithmicend` Default: `\textbf{end}`
- `\algorithmicdo` Default: `\textbf{do}`
- `\algorithmicwhile` Default: `\textbf{while}`
- `\algorithmicfor` Default: `\textbf{for}`
- `\algorithmicforall` Default: `\textbf{for all}`
- `\algorithmicloop` Default: `\textbf{loop}`
- `\algorithmicrepeat` Default: `\textbf{repeat}`
- `\algorithmicuntil` Default: `\textbf{until}`
- `\algorithmicprocedure` Default: `\textbf{procedure}`
- `\algorithmicfunction` Default: `\textbf{function}`
- `\algorithmicif` Default: `\textbf{if}`
- `\algorithmicthen` Default: `\textbf{then}`
- `\algorithmicelse` Default: `\textbf{else}`
- `\algorithmicrequire` Default: `\textbf{Require:}`
- `\algorithmicensure` Default: `\textbf{Ensure:}`
- `\algorithmicreturn` Default: `\textbf{return}`
- `\algorithmicoutput` Default: `\textbf{output}`

5 Revision History

v1.1.0 (2023-02-17)

- Added support for indent guide lines spanning multiple pages.

v1.0.2 (2022-10-07)

- Fixed bug with incorrectly ended indent block for nested statements.

v1.0.1 (2021-12-05)

- Fixed bug regarding alignment of comments after end if, end for etc.

v1.0 (2020-08-16)

- Initial release.