Pseudocode in LyX

Paul A. Rubin (rubin@msu.edu)

January 3, 2019

This document pertains to (beta) version 0.2.

1 Introduction

The `algorithmicx` LaTeX package provides commands for typesetting algorithms written in pseudocode. This document describes a LyX module that (hopefully) facilitates entering such listings in LyX. It uses the `algpseudocode.sty` style file, which is part of `algorithmicx`. For details on how the package works, please see section 3.1 of the documentation for `algorithmicx`.

1.1 Changes in this version

- The order and grouping of styles in the environment drop-down list has been modified (hopefully moving more common styles higher in the list).
- Three styles have been added[1]. "Input block" and "Output block" are intended to be inserted in front of a function or procedure, letting you explain the inputs (arguments) and outputs (return values). One or more instances of "Declare argument" is inserted within one of those blocks (one per input or output value). It provides two boxes side by side, similar to the standard "Description" environment (but without the use of bold face font), where you can specify an argument and a description of the argument.

2 Installation

First, of course, you have to install the `algorithmicx` package. After that, download the `pseudocode.module` file and place it in your local layouts directory. (If you are not sure where that directory is, select `Help > About LyX`, which will tell you the path for your "user directory". The local layouts directory is a child of the user directory.) Once the module is in the local layouts directory, execute `Tools > Reconfigure` and restart LyX.

3 Usage

In your document, go to `Document > Settings... > Modules`, select "Pseudocode" and click the Add button, then OK. The module adds a number of paragraph environments to the drop-down list in the tool bar.

- The "Pseudocode" group contains a paragraph environment for algorithms and paragraph styles for functions and procedures. Functions, procedures and pretty much everything else should be nested

[1] These additions are not part of the algpseudocode package, but are motivated by an answer by Andrew Swann to a question on Stack Exchange.

inside algorithms. In addition, it provides one paragraph style ("Return") that the `algpseudocode.sty` style does not provide. This simply creates a statement that begins with the word "return" in bold, after which you can enter a return value (or not).

- The "Statements" group provides the "State" command, which is the style to use for plain statements in a pseudocode listing. It also provides "Call" (for calling a procedure or function).
- The "Conditional" group provides "if", "else" and "else if" paragraph styles.
- The "Looping" group contains paragraph styles for the various types of loops ("for", "repeat until", "while" and an indefinite "loop") supported by the LaTeX package.
- The "Other" group provides "Input block" and "Output block", which can be inserted at the start of an algorithm to specify inputs and outputs. The "Declare argument" style is used inside either of these blocks to specify individual input or output values. This group also contains "Ensure" and "Require" (to state conditions for an algorithm), and an environment that inserts a blank line that will not be numbered.

The module also supports the `\Comment` command, which inserts comments in listings. Because comments are embedded in paragraphs of other types (most typically, though not exclusively, statements), this is implemented not as a paragraph style but as a custom inset. You add a comment using `Insert > Custom Insets > Code comment`.

There are some details regarding usage that are worth noting.

- The algorithm *paragraph style* is separate from algorithm *floats*. If you want an algorithm inside a float, use `Insert > Float > Algorithm` to create the float. Inside the float (and outside its label) create a paragraph and choose `Algorithm` from the environment list.
- Loops require you to enter two paragraphs, one to start the loop and the other to end it. Functions and procedures behave similarly. The relevant styles are paired in the environment selection list. For instance, to create a for loop, set one paragraph to the "For" style and another to the "EndFor" style, then insert any code inside the loop in paragraphs between those two.
- Most statements will automatically nest inside algorithms. If you need to force something to nest that does not do so automatically, use the "Increase depth" button on the tool bar or the corresponding command on the `Edit` menu.
- Some commands (and the algorithm environment) have required or optional arguments. An inset for the first such argument appears automatically. Due to a limitation in LyX, the inset for a second argument (if one exists) does not appear automatically. For instance, when inserting a procedure, the inset for the procedure's name appears automatically, but the inset for the parameters of the procedure does not. To access the second inset, do one of the following:
  - right-click next to the inset and select the entry that describes the argument you need to insert;
  - select the `Insert` menu and find the correct item to insert; or
  - use the accelerator keys to insert it (shown in the Insert menu, probably `Alt+A 2`).

  Note that math mode can be used within insets.
- Labels can be inserted, in the usual way, in any line of a pseudocode listing.
- Lines in the LyX source code do not indent (other than nesting inside the algorithm delimiters), but the output will indent properly.
- You will frequently want to enter several consecutive statements. The LyX function "paragraph-break inverse" (bound to Alt+Return in the default cua bind file) will start a new statement. This also works for other styles introduced by the module, but "State" is the only style you are likely to want to apply to consecutive paragraphs.

- The "Declare argument" style takes three arguments ("item", "description" and "box width"). The first names the thing you are declaring, and the second describes it. The third ("box width") is an optional argument that adjusts the amount of space allocated for the name portion. If you look at the example in section 5.8 below, you will see that the output argument has its box width set to 3em (wider than the default), while the two input arguments use the default width. The extra width for the output argument is necessary to prevent the item name "gcd(a,b)" from breaking the left margin.

4 License

In keeping with the rest of LyX, this is released under the GNU General Public License version 2 (GPLv2).

5 Examples

To demonstrate use of the module, we reproduce a number of examples from section 3.1 of the `algorithmicx` manual[2], showing first the original code (in ERT) and then the equivalent LyX code using the module. By compiling this file to PDF, you can test the correctness of the module code.

5.1 Euclid's algorithm

Raw LaTeX version:

1: **procedure** EUCLID(a, b) ▷ The g.c.d. of a and b
2: $r \leftarrow a \bmod b$
3: **while** $r \neq 0$ **do** ▷ We have the answer if r is 0
4: $a \leftarrow b$
5: $b \leftarrow r$
6: $r \leftarrow a \bmod b$
7: **end while**
8: **return** b ▷ The gcd is b
9: **end procedure**

Module version:

1: **procedure** EUCLID(a, b) ▷ The g.c.d. of a and b
2: $r \leftarrow a \bmod b$
3: **while** $r \neq 0$ **do** ▷ We have the answer if r is 0
4: $a \leftarrow b$
5: $b \leftarrow r$
6: $r \leftarrow a \bmod b$
7: **end while**
8: **return** b ▷ The gcd is b
9: **end procedure**

The labeled lines are 7 and 7 respectively.

5.2 For block

Raw LaTeX version:

1: $sum \leftarrow 0$
2: **for** $i \leftarrow 1, n$ **do**
3: $sum \leftarrow sum + i$

[2]Copyright 2005 by Szász János (szaszjanos@users.sourceforge.net)

4: **end for**

Module version:

1: $sum \leftarrow 0$
2: **for** $i \leftarrow 1, n$ **do**
3: $\quad sum \leftarrow sum + i$
4: **end for**

5.3 While block

Raw LaTeX version:

1: $sum \leftarrow 0$
2: $i \leftarrow 1$
3: **while** $i \leq n$ **do**
4: $\quad sum \leftarrow sum + i$
5: $\quad i \leftarrow i + 1$
6: **end while**

Module version:

1: $sum \leftarrow 0$
2: $i \leftarrow 1$
3: **while** $i \leq n$ **do**
4: $\quad sum \leftarrow sum + i$
5: $\quad i \leftarrow i + 1$
6: **end while**

5.4 Repeat block

Raw LaTeX version:

1: $sum \leftarrow 0$
2: $i \leftarrow 1$
3: **repeat**
4: $\quad sum \leftarrow sum + i$
5: $\quad i \leftarrow i + 1$
6: **until** $i > n$

Module version:

1: $sum \leftarrow 0$
2: $i \leftarrow 1$
3: **repeat**
4: $\quad sum \leftarrow sum + i$
5: $\quad i \leftarrow i + 1$
6: **until** $i > n$

5.5 If block

Raw LaTeX version:

1: **if** $quality \geq 9$ **then**
2: $\quad a \leftarrow perfect$
3: **else if** $quality \geq 7$ **then**

```
    a ← good
else if quality ≥ 5 then
    a ← medium
else if quality ≥ 3 then
    a ← bad
else
    a ← unusable
end if
```

Module version:

```
if quality ≥ 9 then
    a ← perfect
else if quality ≥ 7 then
    a ← good
else if quality ≥ 5 then
    a ← medium
else if quality ≥ 3 then
    a ← bad
else
    a ← unusable
end if
```

5.6 Require/ensure

Raw LaTeX version:

Require: $x \geq 5$
Ensure: $x \leq -5$

```
while x > −5 do
    x ← x − 1
end while
```

Module version:

Require: $x \geq 5$
Ensure: $x \leq -5$

```
while x > −5 do
    x ← x − 1
end while
```

5.7 Call

Raw LaTeX version:

```
CREATE(10)
```

Module version:

```
CREATE(10)
```

5.8 Input/output example

This repeats Euclid's algorithm, but with input and output descriptions.

```
1: Input
2:         a      the first integer
3:         b      the second integer
4: Output
5:   gcd(a,b)      the greatest common divisor of a and b
6: procedure EUCLID(a, b)                          ▷ The g.c.d. of a and b
7:     r ← a mod b
8:     while r ≠ 0 do                              ▷ We have the answer if r is 0
9:         a ← b
10:        b ← r
11:        r ← a mod b
12:    end while
13:    return b                                    ▷ The gcd is b
14: end procedure
```