# The `algorithmicx` package*

Szász János
szaszjanos@users.sourceforge.net

December 17, 2003

**Abstract**

The `algorithmicx` package provides many possibilities to customize the layout of algorithms. You can use one of the predefined layouts (for **pseudocode**, **pascal** and **c** look), with or without modifications, or you can define a completely new layout for your specific needs.

## Contents



*This is the documentation for the version 1.0 of the package.

# 1 Introduction

All this has begun in my last year at the university. The only thing that I knew of LaTeX was that it exists, and that it is "good". I started using it, but I needed to display some algorithms. So I begun searching for a good algorithmic style, and I have found the `algorithmic` package. It was a great joy for me, and I started to use it... Well... Everything went nice, until I needed some block that wasn't defined in there. What to do? I was no LaTeX guru, in fact I only knew the few basic macros. But there was no other way, so I opened the style file, and I copied one existing block, renamed a few things, and voilà! This (and some other small changes) where enough for me...

One year later — for one good soul — I had to make some really big changes on the style. And there on a sunny day came the idea. What if I would write some macros to let others create blocks automatically? And so I did! Since then the style was completely rewritten several times...

I had fun writing it, may you have fun using it! I am still no LaTeX guru, so if you are, and you find something really ugly in the style, please mail me! All ideas for improvements are welcome!

Thanks go to Benedek Zsuzsa, Ionescu Clara, Szőcs Zoltán, Cseke Botond, Kanoc and many-many others.

# 2 The predefined layouts

The `algorithmicx` package has the following predefined layouts:

**algpseudocode** has (almost) the same look as the one defined in the `algorithmic` package. The main difference is that while the `algorithmic` package doesn't allow you to modify predefined structures, or to create new ones, the `algorithmicx` package gives you full control over the definitions (ok, there are some limitations — you can not send mail with a, say, `\For` command).

**algpascal** aims to create a formatted pascal program, it performs automatic indentation (!), so you can transform a pascal program into an **algpascal** algorithm description with some basic substitution rules.

**algc** – yeah, just like the **algpascal**... but for c...

Each algorithm begins with the `\begin{algorithmic}[lines]` command, the optional `lines` controls the line numbering: 0 means no line numbering, 1 means number every line, and $n$ means number lines $n$, $2n$, $3n$... until the `\end{algorithmic}` command, witch ends the algorithm.

## 2.1 The algpseudocode layout

If you are familiar with the `algorithmic` package, then you'll find it easy to switch, and you have good chances to use previously written algorithms without changing them (some comments needs to be placed to their right place, but you will be happy to have them there). You only need to read the section 2.1.9 and 2.1.10.

The first algorithm one should write is the first algorithm ever (ok, an improved version), *Euclid's algorithm*:

**algorithm 1** Euclid's algorithm

```
1: procedure EUCLID(a, b)                          ▷ The g.c.d. of a and b
2:     r ← a mod b
3:     while r ≠ 0 do                              ▷ We have the answer if r is 0
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     end while
8:     return b                                    ▷ The gcd is b
9: end procedure
```

Created with the following source:

```
\begin{algorithm}
\caption{Euclid's algorithm}
\begin{algorithmic}[1]
\Procedure{Euclid}{$a,b$}\Comment{The g.c.d. of a and b}
   \State $r\leftarrow a\bmod b$
   \While{$r\not=0$}\Comment{We have the answer if r is 0}
      \State $a\leftarrow b$
      \State $b\leftarrow r$
      \State $r\leftarrow a\bmod b$
   \EndWhile
   \State \textbf{return} $b$\Comment{The gcd is b}
\EndProcedure
\end{algorithmic}
\end{algorithm}
```

The `\State` stands at the beginning of each simple statement; the respective statement is put in a new line, with the needed indentation. The `\Procedure` ... `\EndProcedure` and `\While` ... `\EndWhile` blocks (like any block defined in the **algpseudocode** layout) automatically indent their content. The indentation of the source doesn't matter, so

```
\begin{algorithmic}[1]
\Repeat
\Comment{forever}\State this\Until{you die.}
\end{algorithmic}
```

results

1: **repeat** ▷ forever
2: this
3: **until** you die.

But, generally, it is a good idea to keep the source indented, since you will find errors much easier. And your tex file looks better!

### 2.1.1 The for block

The **for** block may have one of the forms:

```
\For{<text>}
   <body>
\EndFor
```

or

```
\ForAll{<text>}
   <body>
\EndFor
```

Example:

```
\begin{algorithmic}[1]
\State $sum\leftarrow 0$
\For{$i\leftarrow 1, n$}
   \State $sum\leftarrow sum+i$
\EndFor
\end{algorithmic}
```

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

### 2.1.2 The while block

The **while** block has the form:

```
\While{<text>}
   <body>
\EndWhile
```

Example:

```
\begin{algorithmic}[1]
\State $sum\leftarrow 0$
\State $i\leftarrow 1$
\While{$i\le n$}
   \State $sum\leftarrow sum+i$
   \State $i\leftarrow i+1$
\EndWhile
\end{algorithmic}
```

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**

### 2.1.3 The repeat block

The **repeat** block has the form:

```
\Repeat
   <body>
\Until{<text>}
```

Example:

```
\begin{algorithmic}[1]
\State $sum\leftarrow 0$
\State $i\leftarrow 1$
\Repeat
   \State $sum\leftarrow sum+i$
   \State $i\leftarrow i+1$
\Until{$i>n$}
\end{algorithmic}
```

1: $sum \leftarrow 0$
2: $i \leftarrow 1$
3: **repeat**

4: $sum \leftarrow sum + i$
5: $i \leftarrow i + 1$
6: **until** $i > n$

### 2.1.4 The if block

The **if** block has the form:

```
\If{<text>}
   <body>
[
\ElsIf{<text>}<body>
...
\ElsIf{<text>}<body>
]
[
\Else
   <body>
]
\EndIf
```

Example:

```
\begin{algorithmic}[1]
\If{$quality\ge 9$}
   \State $answer\leftarrow perfect$
\ElsIf{$quality\ge 7$}
   \State $answer\leftarrow good$
\ElsIf{$quality\ge 5$}
   \State $answer\leftarrow medium$
\ElsIf{$quality\ge 3$}
   \State $answer\leftarrow bad$
\Else
   \State $answer\leftarrow unusable$
\EndIf
\end{algorithmic}
```

1: **if** $quality \geq 9$ **then**
2: $answer \leftarrow perfect$
3: **else if** $quality \geq 7$ **then**
4: $answer \leftarrow good$
5: **else if** $quality \geq 5$ **then**
6: $answer \leftarrow medium$
7: **else if** $quality \geq 3$ **then**
8: $answer \leftarrow bad$
9: **else**
10: $answer \leftarrow unusable$
11: **end if**

### 2.1.5 The procedure block

The **procedure** block has the form:

```
\Procedure{<text>}{<text>}
   <body>
\EndProcedure
```

Example: See Euclid's algorithm somewhere on the first pages...

### 2.1.6 The function block

The **function** block has the same syntax as the **procedure** block:

```
\Function{<text>}{<text>}
   <body>
\EndFunction
```

### 2.1.7 The loop block

The **loop** block has the form:

```
\Loop
   <body>
\EndLoop
```

### 2.1.8 Require and Ensure

The starting conditions for the algorithm can be described with the **require** instruction, and its result with the **ensure** instruction.:

```
\Require something
\Ensure something
```

Example:

```
\begin{algorithmic}[1]
\Require $x\ge5$
\Ensure $x\le-5$
\Statex
\While{$x>-5$}
   \State $x\leftarrow x-1$
\EndWhile
\end{algorithmic}
```

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

1: **while** $x > -5$ **do**
2: $\quad x \leftarrow x - 1$
3: **end while**

### 2.1.9 Placing comments in sources

Comments may be placed everywhere in the source (there are no limitations like those in the `algorithmic` package), feel the freedom!

If you would like to change the form in witch comments are displayed, just change the `\algorithmiccomment` macro:

```
\renewcommand{\algorithmiccomment}[1]{\hskip 3em$\rightarrow$ #1}
```

will result:

1: $x \leftarrow x + 1$ → Here is the new comment

### 2.1.10 Options

The `algpseudocode` package supports the following options:

**noend/end**
: With **noend** specified all **end ...** lines are omitted. You get a somewhat smaller algorithm, and the ugly feeling, that something is missing... The **end** value is the default, it means, that all **end ...** lines are in their right place.

**compatible/noncompatible**
: If you would like to use old algorithms, written with the `algorithmic` package without (too much) modification, then use the **compatible** option. This option defines the uppercase version of the commands. Note that you still need to remove the `[...]` comments (these comments appeared due to some limitations in the `algorithmic` package, these limitations and comments are gone now). The default **noncompatible** does not define the all uppercase commands.

### 2.1.11 Changing command names

One common thing for a pseudocode is to change the command names. Many people use many different kind of pseudocode command names. This package uses something like the following definitions for command names (you can change any of them):

```
\newcommand{\algorithmicend}{\textbf{end}}
\newcommand{\algorithmicwhile}{\textbf{while}}
\newcommand{\algorithmicfor}{\textbf{for}}
<and many others>
```

The following command changes the name of the **While** command:

```
\renewcommand{\algorithmicwhile}{\textbf{am\'ig}}
```

After this all **while**'s in the pseudocode will be replaced with **amíg**.

If you need to change the order of the words (or the number of parameters) then redefine one of the following:

```
\renewcommand{\algorithmicWhile}[1]%
   {\algorithmicwhile\ #1 \algorithmicdo}
\renewcommand{\algorithmicFor}[1]%
   {\algorithmicfor\ #1 \algorithmicdo}
\renewcommand{\algorithmicIf}[1]%
   {\algorithmicif\ #1 \algorithmicthen}
\renewcommand{\algorithmicElse}{\algorithmicelse}
\renewcommand{\algorithmicElsIf}[1]%
   {\algorithmicelse\ \algorithmicif\ #1 \algorithmicthen}
\renewcommand{\algorithmicEndWhile}%
   {\algorithmicend\ \algorithmicwhile}
\renewcommand{\algorithmicEndFor}%
   {\algorithmicend\ \algorithmicfor}
<and many others>
```

The following makes the `\EndWhile` have 2 params, and the order of the words changed:

```
\renewcommand{\algorithmicEndWhile}[2]%
   {#1\algorithmicwhile\ #2\algorithmicend}
```

After this all `\EndWhile` commands must be followed by 2 params.

## 2.2 The algpascal layout

One of the most important features of the **algpascal** layout is that *it performs automatically the end-of-block indentation*. Here is an example to demonstrate this feature:

1: **begin**
2: $sum := 0$;
3: **for** $i = 1$ **to** $n$ **do**
4: $sum := sum + i$;
5: writeln($sum$);
6: **end**.

is obtained from:

```
\begin{algorithmic}[1]
\BEGIN
\State $sum:=0$;
\For{i=1}{n}
   \State $sum:=sum+i$;
\State writeln($sum$);
\END.
\end{algorithmic}
```

Note, that the `\For` is not closed explicitly, its end is detected automatically. Again, the indentation in the source doesn't affect the output. In this layout every parameter passed to an instruction is put in mathematical mode.

### 2.2.1 The Begin... End block

```
\Begin
   <body>
\End
```

The **Begin**... **End** block (and the **Repeat**... **Until** block) are the only blocks in the **algpascal** style (instead of `\Begin` you may write `\Asm`). This means, that every other loop is ended automatically after the following command (another loop, or a block).

### 2.2.2 The For loop

```
\For{<assignment>}{<expression>}
   <command>
```

The **For** loop (as all other loops) ends after the following command (a block counts also as a command). So if you write:

```
\begin{algorithmic}[1]
\Begin
\State $sum := 0$;
\State $prod := 1$;
   \For{i = 1}{10}
      \Begin
         \State $sum := sum + i$;
         \State $prod := prod * i$;
      \End
\End.
\end{algorithmic}
```

you'll get:

1: **begin**
2: $sum := 0$;
3: $prod := 1$;
4: **for** $i = 1$ **to** 10 **do**
5: **begin**
6: $sum := sum + i$;
7: $prod := prod * i$;
8: **end**
9: **end**.

### 2.2.3 The While loop

```
\While{<expression>}
   <command>
```

I think the syntax is enough...

### 2.2.4 The Repeat... Until block

```
\Repeat
   <body>
\Until{<expression>}
```

### 2.2.5 The If command

```
\If{<expression>}
   <command>
[
\Else
   <command>
]
```

Every **Else** matches the nearest **If**.

### 2.2.6 The Procedure command

```
\Procedure<some text>
```

**Procedure** just writes the "procedure" word on a new line... You will probably put a **Begin**... **End** block after it.

### 2.2.7 The Function command

```
\Function<some text>
```

Just like **Procedure**.

## 2.3 The algc layout

Sorry, the **algc** layout is unfinished. The commands defined are:

- `\{`... `\}` block
- `\FOR` with 3 params
- `\IF` with 1 param
- `\ELSE` with no params
- `\WHILE` with 1 param

- `\DO` with no params
- `\FUNCTION` with 3 params
- `\RETURN` with no params

# 3 Custom algorithmic blocks

## 3.1 Blocks and loops

Most of environments defined in the standard layouts (and most probably the ones you will define) are divided in two categories:

**Blocks** are the environments witch contain an arbitrary number of commands, and nested blocks. Each block has a name, begins with a starting command and ends with an ending command. The commands in a block are indented by `\algorithmicindent` (or another amount).

If your algorithm ends without closing all blocks, the `algorithmicx` package gives you a nice error. So be good, and close them all!

Blocks are all the environments defined in the `algpseudocode` package, the `\Begin` ... `\End` block in the `algpascal` package, and some other ones.

**Loops** (Let us call them loops...) The loops are environments that include only one command, loop or block; a loop is closed automatically after this command. So loops have no ending commands. If your algorithm (or a block) ends before the single command of a loop, then this is considered an empty command, and the loop is closed. Feel free to leave open loops at the end of blocks!

Loops are most of the environments in the `algpascal` and `algc` packages.

For some rare constructions you can create mixtures of the two environments (see the `\algsetblock` macro). Each block and loop may be continued with another one (like the `If` with `Else`).

## 3.2 algblock

With `\algblock` you can create a block:

```
\algblock[Name]{Start}{Stop}
\begin{algorithmic}[1]
\Start
   \Start
   \Stop
   \Start
   \Stop
\Stop
\end{algorithmic}
```

1: **Start**
2: **Start**
3: **Stop**
4: **Start**
5: **Stop**
6: **Stop**

The block will have the name `Name`, start with the `\Start` command, and end with the `\Stop` command. If the `Name` is missing then `Start` is used instead.

## 3.3 algcblock

With `\algcblock` you can create a block that continues an open block or loop.

```
\algblock[Name]{Start}{Stop}
\algcblock[CName]{Name}{CStart}{CStop}
\begin{algorithmic}[1]
\Start
   \Start
   \CStart
   \CStop
\CStart
   \Start
   \Stop
\CStop
\end{algorithmic}
```

1: **Start**
2: **Start**
3: **CStart**
4: **CStop**
5: **CStart**
6: **Start**
7: **Stop**
8: **CStop**

This creates a `\Start`...`\Stop` block, and a block named `\CName` that continues the `Name` block, begins with `CStart`, and ends with `CStop`.

The most relevant use of `\algcblock` is in the definition of `\If`...`\ElsIf`...`\Else`...`\EndIf`:

```
\algblock{If}{EndIf} % the If block
\algcblock[If]{If}{ElsIf}{EndIf} % \ElsIf switches from If to If
\algcblock{If}{Else}{EndIf} % and \Else may follow only once
```

## 3.4 algloop

```
\algloop[Name]{Start}
```

This creates a loop named `Name`, starting with `\Start`. The loop has no ending command, since it ends after the first state, block or loop that follows this loop.

```
\algloop{For}
\algblock{Begin}{End}
\begin{algorithmic}[1]
\For
   \Begin
   \For
      \For
         \Begin
         \End
   \Begin
   \End
   \End
\end{algorithmic}
```

```
1: For
2:     Begin
3:         For
4:             For
5:                 Begin
6:                 End
7:         Begin
8:         End
9:     End
```

## 3.5 algcloop

With `\algcloop` you can create a loop that continues an open block or loop.

```
\algloop{If}
\algcloop{If}{Else}
\algblock{Begin}{End}
\begin{algorithmic}[1]
\If
   \Begin
   \End
\Else
   \If
      \Begin
      \End
\end{algorithmic}
```

**If**
    **Begin**
    **End**
**Else**
    **If**
        **Begin**
        **End**

Note that you can continue a loop with a block, and a block with a loop — I don't know what use this has, but there is no reason (and good implementation) to restrict this. If you have found a good example for the mixed use, then be proud for it!

## 3.6 algsetblock

The `\algsetblock` macro does not really create a block — at least not always. It gives you the means to create a custom block or loop or whatever it comes out.

```
\algsetblock[Name]{Start}{Stop}{3}{1cm}
\begin{algorithmic}[1]
\Start
   \State 1
   \State 2
   \State 3
\State 4
\Start
   \State 1
\Stop
\State 2
\end{algorithmic}
```

**Start**
        1
        2
        3
4
**Start**
        1
**Stop**
2

This creates an environment named `Name`, starting with `\Start` and possibly ending with `\Stop`, with a lifetime of 3 statement (or blocks or loops), and with 1 cm indentation.

The created environment behaves as follows:

- You can start it with `\Start`. The nested environments are indented by 1 cm.
- If it is followed by at least 3 environments, then it closes automatically after the third one. If the lifetime parameter is left empty, then it supports an infinite number of environments, and it must be closed manually (block).
- If you put a `\Stop` before the automatic closure, then this `\Stop` closes the environment. If the lifetime is infinite (left empty) then it MUST be closed with `\Stop`, or another closing command.
- If you define later a block that continues this one, then on the starting command of the second block the first one is closed.
- If you leave the Stop parameter empty, then the lifetime must be finite, or you must give a block or loop that continues this one. In this case the continuation is not optional.

## 3.7 algsetcblock

This macro creates a custom block, that continues another one.

```
\algsetblock[Name]{Start}{Stop}{3}{1cm}
\algsetcblock[CName]{Name}{CStart}{CStop}{2}{2cm}
\begin{algorithmic}[1]
\Start
   \State 1
\CStart
   \State 1
   \State 2
\State 3
\Start
   \State 1
\CStart
   \State 1
\CStop
\end{algorithmic}
```

1: **Start**
2: 1
3: **CStart**
4: 1
5: 2
6: 3
7: **Start**
8: 1
9: **CStart**

10: 1
11: **CStop**

# 4 Bugs

At this time there are no known bugs. If you find some, please contact me on **szaszjanos@users.sourceforge.net**.