The DuckBoat — News from TEX.SE: The Morse code of Ti*k*Z

Herr Professor Paulinho van Duck

Abstract

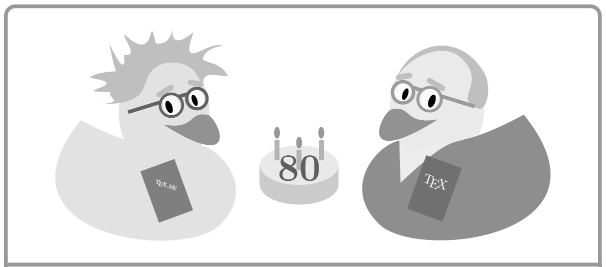
For this installment, Prof. van Duck would like to tell you how the *duck mania* began and infected many TEX.SE users. In the second Quack Guide, you will find a beginner's approach to Ti*k*Z, a powerful package to draw your graphics directly in LATEX.

1 Here I am, again!

Hi, (LA)TEX friends!

If you missed the last issue, I am Prof. van Duck, and I enjoy helping beginners like me!

First of all, let me celebrate a very significant date, and thank our Jedi Master Prof. Knuth for creating the best typesetting system in the world!



Happy Birthday, Prof. DEK!

Secondly, I would like to thank all the TEX.SE friends who warmly appreciated the first DuckBoat. Some of them also used a link to it in their comments, to explain to new users how to ask.

Peter Wilson himself wrote to me suggesting a new topic (something like: *How to add code and images to TEX.SE posts?*), which will be treated in one of the next Quack Guides.

Prof. Enrico Gregorio was so kind to take me with him on stage during his talk at last October's conference of the Italian TEX User Group, the annual GuIT*meeting* (I thank him also for his editing and suggestions about this article).

By the way:

```
\begin{advertising}
```

Are you an Italian (LA)TEX user?

Join the GuIT!

`http://www.guitex.org`

```
\end{advertising}
```

I am very proud of all that, quack!

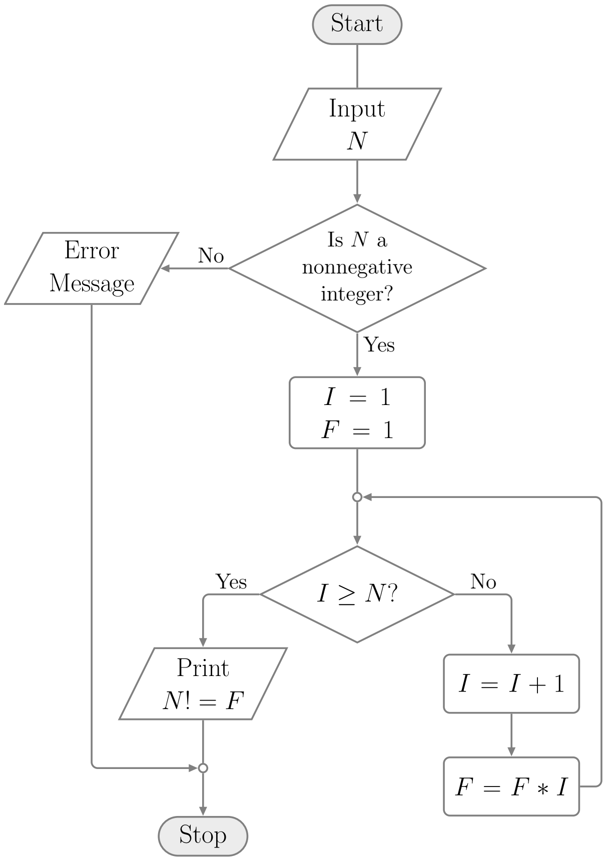


Figure 1: Case study: a flowchart to compute the factorial of a number.

Last time I talked about the *just-do-it-for-me* questions. Since a lot of them refer to Ti*k*Z, learning some tips and tricks could be useful.

I will show you some of them drawing the flowchart in Figure 1. I chose it because I often see questions about similar diagrams on TEX.SE.

Of course, as always in LATEX, there are many ways to skin a duck, er, to make such a diagram. Some of them are even more efficient and fun than the one I will show here, for example with a matrix or a chain (how many topics I do have for the next issues, quack!). But I would like to proceed one step at a time. I also will not be very rigorous, I hope the experts will forgive me.

Anyway, I thank all the users whose answers helped me to build the example. They are too many to list them all here, quack!

2 Origin and evolution of the *duck mania*

I would like to tell you a story which is both moving and funny; it concerns the origin of the duck mania. The protagonist is, of course, Paulo Cereda. I will just report (more or less exactly) what he said in chat about it.

It happened that, one day, he was in a conference and saw a lone girl in the corner of the auditorium. He decided at once to talk to her (he never loses an opportunity to make new friends). But when he said "Hi!", she did not reply; she seemed not even to notice his presence. When he came near her, she looked at him, saying no words, and wrote in a piece of paper that she was hearing impaired. Of course, this was not a problem for Paulo, who promptly used his notebook to write sentences which she could read.

To impress her, he mentioned that he studied ASL (American Sign Language), even if he remembered almost nothing. When she asked him to try a sentence for her, he got stuck because the only words he could remember were: *I love you* and, of course, *duck*.

Since saying "I love you" to a girl you have known for just a few minutes is not very appropriate, he chose — in his own words — the second best sentence ever known to mankind: "I love ducks".

Eventually, after much gesticulation, the girl, obviously, started laughing a lot!

This good memory is the reason why Paulo began spreading the duck mania all over the world.

One way to infect other people was to offer a hand puppet duck as a prize in a TEX.SE (Meta) contest. I moved to Milan, to my friend Carla's, on that occasion (she won the contest).

Images or words related to ducks have been used in TEX.SE posts for years; some users have a duck as their avatar. The peak of the infection was reached with the creation of `tikzducks`,[1] and since the package is growing bigger and bigger, the duck joke will last for many years to come.

3 Quack Guide No. 2 The Morse code of Ti*k*Z

At first sight, Ti*k*Z may scare newbies due to its huge package documentation [1], but its usage is not so difficult as it may seem.

Its logic is simple: like the Morse code uses *dots* and *dashes* to translate any text, Ti*k*Z uses *nodes* and *paths* to draw any picture!

If you look at Figure 1, you will see some geometric shapes, connected by lines (in this case arrows): the former are nodes, the latter are paths. Are you looking forward to learning how to draw them? Just load the `tikz` package, add a `tikzpicture` environment to your document, and start!

[1] `https://ctan.org/pkg/tikzducks`.

3.1 Nodes

The syntax of the node command is more or less:
`\node[⟨options⟩] (⟨name⟩) at (⟨coord⟩) {⟨text⟩};`

Only `{⟨text⟩}`, i.e., the text within the node, is mandatory, although it can be empty: `{}`.

⟨*name*⟩ is the identifier by which the node will be referenced in your `tikzpicture`; it can also be set with the option `name=⟨name⟩`.

The coordinates where it will be located are (⟨*coord*⟩), they can be Cartesian, polar or spherical; the default is `(0,0)`.

As for ⟨*options*⟩, you can play around setting dimensions, aspect, positioning, labels, you name it. Of course, I cannot list all of them in these few pages, I will only highlight the ones who surprised me when I first met them, quack!

Figure 2 shows the options for setting the node dimensions. All of these are followed by the actual desired value, e.g., `inner sep=⟨dimension⟩` and they are not mandatory; if not explicitly set, they assume a default value.

For instance, the default value for `text width` is the natural width of your node text; let us call the latter w, for convenience. If your node does not have an explicit `text width`, it is set to w, as in the first node of the following example. If a `text width` less than w is indicated, your text will be broken onto more than one line, as in the second node. If it is greater than w, the remaining space will be filled with spaces, as in the third node.

```
\begin{tikzpicture}[every node/.style={draw}]
  \node {We love ducks};
  \node[text width=4em] at (3,0)
    {We love ducks};
  \node[text width=10em] at (1.5,-1)
    {We love ducks};
\end{tikzpicture}
```

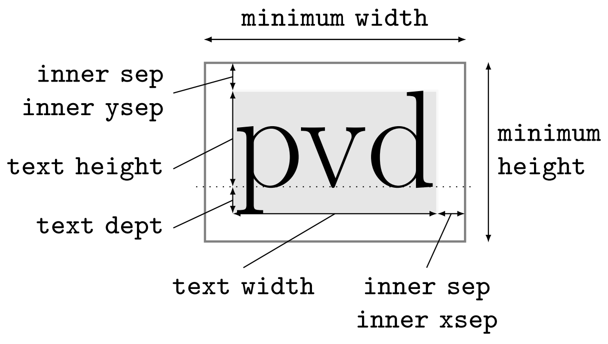
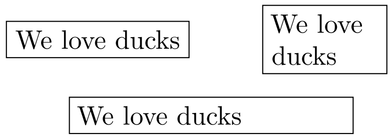


Figure 2: Node dimensions. The dotted line is the baseline; the text has a gray background to better highlight its dimensions. For the border style see Figure 5. *(The first one who guesses what* pvd *means wins a rubber duck.)*



The first strangeness you might notice in Figure 2 is that there are `minimum width/height` but *not* the corresponding maximum. Indeed, if you consider a node border with no thickness, the node width is the sum of the `text width` and the double of the `inner xsep`, whereas the node height is the sum of `text height`, `text depth` and the double of the `inner ysep`. Hence, for instance, you will usually act on the `text width` to set a maximum width.

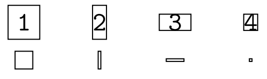
The `text height` is the piece of text above the baseline; the `text depth` is the one below.

The `inner sep` is the gap between the text and the node border. You can set the horizontal/vertical value separately with `inner xsep/inner ysep`.

If the border has a thickness, half of its thickness will be inside the shape and half outside, so to compute precisely the total node width/height you should also add the line width of the border.

Have you got a headache yet? Don't worry, quack! Let me show you an example:

```
\begin{tikzpicture}[
  every node/.style={draw, font=\ttfamily}
  ]
  \node {1};
  \node[inner xsep=0em] at (1,0) {2};
  \node[inner ysep=0em] at (2,0) {3};
  \node[inner sep=0em] at (3,0) {4};
  \node at (0,-.5) {};
  \node[inner xsep=0pt] at (1,-.5) {};
  \node[inner ysep=0pt] at (2,-.5) {};
  \node[inner sep=0pt] at (3,-.5) {};
\end{tikzpicture}
```



The nodes in the first column have the standard inner sep dimension (which is `.3333em`), with and without text. As you can see, even if there is no text, the node has a width and a height, due to the inner sep. In the second and third columns, there are nodes respectively with no horizontal and no vertical gaps between the text and the border, whereas, in the last column, there are no gaps at all. To have a point with no dimensions, you have to nullify also the `inner sep`, as in the last node (if you only need an actual geometric point, you can use `\coordinate`, but I will not talk about it this time).

Did you notice the options of the `tikzpicture` environment? `draw` means that you want the node borders visible, and you can set the font used for the node text with `font=`⟨*font commands*⟩.

Imagine that you have a picture with a lot of nodes — writing these options for every node could be boring! But LaTeX is fun; it is made to avoid code repetition, quack! The Ti*k*Z way to do this is to create a `style`. You can make the style valid for all the nodes, as in the previous example, or only for some of them, giving your own name to the style. In our case study, for instance, we will create a style for the terminal blocks, one for the instructions, another for the tests, and so on.

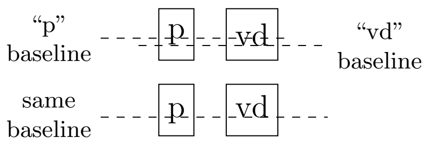
If you specify them, like in the previous example, as options of your `tikzpicture` environment, you can use them only locally. To make them valid for all the pictures of our document, you can use `\tikzset`, and write them in our preamble or anywhere before using them:

`\tikzset{`⟨*style name*⟩`/.style={`⟨*options*⟩`},...}`

Another example is a handy application of `text height` and `text depth`: alignment of texts in different nodes.

Look at this code snippet and its output (for now, do not worry about the `\draw` commands, I will explain them in Section 3.2):

```
\begin{tikzpicture}[
  mylabel/.style={font=\small, align=center},
  mynode/.style={draw, font=\large,
                 minimum height=4.5ex},
  mynodeok/.style={draw, font=\large,
                   text height=1.75ex,
                   text depth=.5ex,
                   minimum height=4.5ex}]
  \node[mynode] (p) {p};
  \node[mynode] (vd) at (1,0) {vd};
  \draw[dashed] (p.base) +(-1,0)
    node[mylabel, left] {‘‘p’’ \\ baseline}
    -- +(1.5,0);
  \draw[dashed] (vd.base) +(-1.5,0) -- +(1,0)
   node[mylabel, right] {‘‘vd’’ \\ baseline};
  \node[mynodeok] (pok) at (0,-1) {p};
  \node[mynodeok] at (1,-1) {vd};
  \draw[dashed] (pok.base) +(-1,0)
   node[mylabel, left] {same \\ baseline}
   -- +(2,0);
\end{tikzpicture}
```



It is evident that the nodes of the first row have different baselines, but adding the appropriate text height and depth, you get two nodes with a perfectly aligned text (second row).

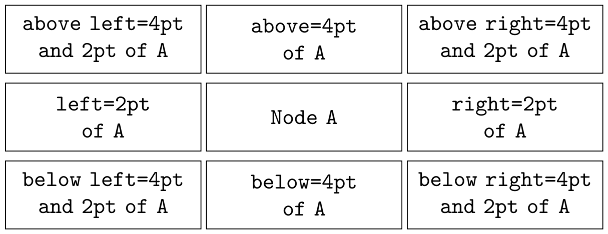

above left=4pt and 2pt of A
above=4pt of A
above right=4pt and 2pt of A
left=2pt of A
Node A
right=2pt of A
below left=4pt and 2pt of A
below=4pt of A
below right=4pt and 2pt of A

Figure 3: Node locating with Ti*k*Z library `positioning`. See Section 17.5 of [1].

The option `align=`⟨*alignment option*⟩, which I used for the side descriptions, sets up the alignment for multi-line text inside a node.

So far I have explicitly set the coordinates to locate the nodes. With a complex picture, it could be not only dull, but you may also be obliged to recalculate the coordinates of many nodes for a small change to your image, even if their relative positions, with respect to other nodes, remain the same.

In such cases, the Ti*k*Z library `positioning` could be your friend!

What is a Ti*k*Z library? You know that Ti*k*Z is a huge package, but usually you do not need all its possible features; a Ti*k*Z library allows you to load some specific additional ones. Just add

`\usetikzlibrary{`⟨*list of libraries*⟩`}`

after loading Ti*k*Z to use them.

Figure 3 shows some options you can use with `positioning`. The locating options (`above`/`below`, `right`/`left` and their combinations) are followed by a ⟨*shifting part*⟩ and an ⟨*of-part*⟩, and they are both optional. Please note that the equal sign must be located before the ⟨*of-part*⟩, even if the ⟨*shifting part*⟩ is not present.

In the ⟨*shifting part*⟩, you can indicate, for instance, a ⟨*dimension*⟩, which represents the distance between the borders of the nodes you would like to set. For the options like `above left`, you can also differentiate between the vertical and horizontal distances, writing ⟨*vdimension*⟩ `and` ⟨*hdimension*⟩. If they are the same for all your nodes, you could add a single option to your environment:

`node distance=`⟨*shifting part*⟩

In the ⟨*of-part*⟩, you can tell Ti*k*Z with respect to which node or coordinate your node should be placed.

To tell the truth, you could use `above` & Co. also without any library, and you can also use anchors (see below) to locate nodes. However, I still advise using `positioning` because it is simpler and has more features.

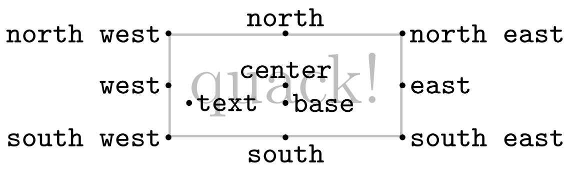
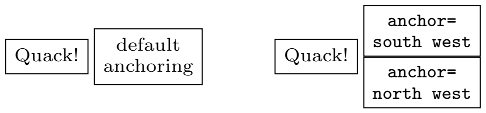


Figure 4: The main anchors of a rectangular node. For further details and other shapes see Section 67 of [1].

Figure 4 shows the main anchors of a rectangular shape. Other shapes may have other, possibly non-intuitive, anchors. Anchors could be used for node positioning (see the following example). In our case study, I will use one also as a starting point of a path. Indeed, anchors are genuine coordinates; you can refer to them with ⟨*node name*⟩`.`⟨*anchor*⟩.

```
\usetikzlibrary{positioning}
...
\begin{tikzpicture}[node distance=2pt,
  every node/.style={draw,
    align=center, font=\scriptsize}]
  \node (a) {Quack!};
  \node[right=of a] {default\\ anchoring};
  \node[right=7em of a] (b) {Quack!};
  \node[right=of b, font=\scriptsize\ttfamily,
    anchor=north west] {anchor=\\ north west};
  \node[right=of b, font=\scriptsize\ttfamily,
    anchor=south west] {anchor=\\ south west};
\end{tikzpicture}
```



With `positioning`, the default anchor for a node positioned to the right of another one is `west` (see the left side of the above picture), but you can change this behavior, setting an `anchor` option explicitly (as on the right side).

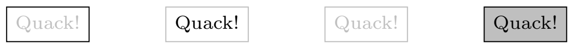
Ti*k*Z offers countless node shapes; `rectangular` is the default one. To draw our flowchart you also need a `circle`, which does not require any additional library, a `rounded rectangle`, for which you need `shapes.misc`, and, lastly, a `diamond` and a `trapezium` of `shapes.geometric`.

Some shapes may have additional options. For example, in our case study, we will modify the standard `trapezium` side angles; for the `diamond`, we will change the ratio between its width and height with `aspect=`⟨*number*⟩ (if the option is not present, it is set to 1).

Let us see some options to color our nodes: `text=`⟨*color*⟩ colors the text, `draw=`⟨*color*⟩ colors the borders, simply ⟨*color*⟩ colors both, whereas for the background there is `fill=`⟨*color*⟩.

In the following example I am using `lightgray` for editorial reasons, but, of course, you can use any color you prefer.

```
\begin{tikzpicture}[every node/.style={draw,
    font=\scriptsize}]
 \node[text=lightgray] (a) {Quack!};
 \node[right=of a, draw=lightgray](b){Quack!};
 \node[right=of b, lightgray] (c) {Quack!};
 \node[right=of c, fill=lightgray] {Quack!};
\end{tikzpicture}
```



3.2 Paths

Having created and located our nodes, let us learn the command to link them:

`\path[`⟨*option*⟩`]` ⟨*path specification*⟩`;`

Since almost all paths are drawn, there is also the abbreviation `\draw` for `\path[draw]`.

For ⟨*option*⟩, again, you can put whatever you like for changing the aspect of the line. There are also a lot of path specifications. Here I will show only the ones used in our case study.

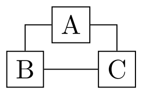
The simplest one is:

`\draw (`⟨*starting point*⟩`) -- (`⟨*target*⟩`);`

which produces a straight line; the ⟨*starting point*⟩ and the ⟨*target*⟩ could be nodes or coordinates, and it is also possible to add other points to the path.

If you use `|-` or `-|`, instead of a straight line you will have a line with a 90° angle, respectively starting vertically and going on horizontally, or vice-versa; see the following example:

```
\begin{tikzpicture}[every node/.style=draw,
  node distance=4pt]
  \node (a) {A};
  \node[below left=of a] (b) {B};
  \node[below right=of a] (c) {C};
  \draw (a) -| (b) -- (c) |- (a);
\end{tikzpicture}
```



The line thickness and its pattern can be customized extensively; you will find some examples in Figure 5. If your desired thickness is not among the predefined ones, you can set it to any value you like with `line width=`⟨*dimension*⟩.

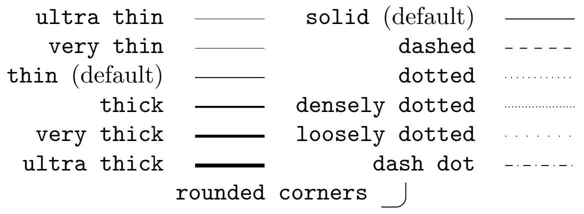


Figure 5: Examples of path thicknesses and patterns. The same options are valid for node borders. See Section 15.3 of [1] for more details.

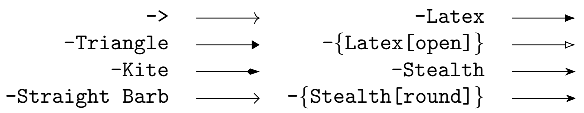
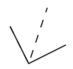


Figure 6: Arrow tips, see Section 16.5 of [1] for a complete list and options.

A useful feature is the possibility of indicating a coordinate of the path relative to another point, with `+(`⟨*shift*⟩`)` and `++(`⟨*shift*⟩`)`. The difference between the two notations is that `++` updates the current point while `+` does not. The current point is the one from which the ⟨*shift*⟩ will be applied; of course, it could be either positive or negative, and any kind of dimension or coordinates can be used. Let us look at an example:

```
\begin{tikzpicture}
  \draw (0,0) -- ++(1,-2) -- +(2,1);
  \draw[dashed] (0,0) -- +(1,-2) -- +(2,1);
\end{tikzpicture}
```



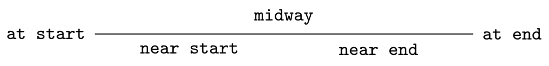
The first segments coincide because the second vertex is `(0,0)+(1,-2)=(1,-2)` for both the solid and the dashed paths; whereas the second segments differ: indeed, the last vertex is `(1,-2)+(2,1)=(3,-1)` for the solid path whereas `(0,0)+(2,1)=(2,1)` for the dashed one.

In our case study, however, there are not simple lines but arrows. The Ti*k*Z library `arrow.meta` provides many different arrow tips, which are then further customizable.

In our flowchart, I used a triangular tip with a smaller width: `-{Triangle[width=5pt]}`. You will find some other examples in Figure 6. To create an arrow path it is enough to put the kind of arrow tip you like in one of these ways: `-`⟨*arrow tip*⟩, ⟨*arrow tip*⟩`-`, or ⟨*arrow tip*⟩`-`⟨*arrow tip*⟩, depending on if you need the tip at the beginning of your path, at the end of it or both.

Eventually, it may be useful to put some nodes along the path; see, for instance, the “Yes” and “No” exits of the tests of our flowchart. It can be done easily by putting a `node` (without the backslash because it is not a macro but a path option) in an appropriate position. Note also where the semicolon is positioned in the following example:

```
\begin{tikzpicture}[
  every node/.style={font=\scriptsize\ttfamily}
  ]
  \draw (0,0) -- +(5.5,0)
    node[at start, left] {at start}
    node[near start, below] {near start}
    node[midway, above] {midway}
    node[near end, below] {near end}
    node[at end, right] {at end};
\end{tikzpicture}
```



3.3 Let us put them together

Now you have all the tools needed to understand the complete code of our case study.

```
\documentclass[tikz]{standalone}
\usetikzlibrary{positioning,
  shapes.geometric, shapes.misc, arrows.meta}
\begin{document}
\begin{tikzpicture}[
  every path/.style={gray, very thick,
rounded corners,-{Triangle[width=5pt]}},
  basenode/.style={draw, sharp corners,
text=black},
  terminator/.style={basenode,font=\LARGE,
rounded rectangle,minimum height=6ex,
text width=5em,text height=2.25ex,
    text depth=.25ex,
    fill=lightgray!30,align=center,},
  inout/.style={basenode,font=\LARGE,
    text width=5.8em,minimum height=11ex,
    align=center,trapezium,trapezium stretches,
trapezium left angle=60,
trapezium right angle=120,},
  block/.style={basenode,font=\LARGE,
    text width=9em,minimum height=9ex,
rounded corners,inner sep=0pt,
    align=center,},
  decision/.style={basenode,diamond,
    align=flush center,aspect=2,font=\LARGE,
    minimum height=15ex,minimum width=30ex,
inner sep=0pt,},
  joining/.style={basenode,circle,
    inner sep=2pt,},
  yesno/.style={font=\Large,near start,black},
  ]
% nodes
  \node[terminator](start) {Start};
  \node[inout, below=of start](input)
    {Input\\ $N$};
  \node[decision, below=of input,
    font=\Large](dqtest)
    {Is $N$ a\\ nonnegative \\ integer?};
  \node[inout, left=4.5em of dqtest](error)
    {Error Message};
  \node[block, below=of dqtest,
    minimum height=11ex](setvar)
    {$I = 1$\\$F = 1$};
  \node[joining, below=of setvar](join1){};
  \node[decision, below=of join1](looptest)
    {$I \ge N$?};
  \node[inout,
    below left=10ex and 7em of looptest,
    anchor=center](output){Print\\ $N! = F$};
  \node[block,
    below right= 10ex and 7em of looptest,
    anchor=center](increase){$I = I + 1$};
  \node[joining, below=of output](join2){};
  \node[block, below=of increase](multiply)
    {$F = F \ast I$};
  \node[terminator, below=of join2](stop)
    {Stop};
% paths
  \draw (start) -- (input) -- (dqtest);
  \draw (dqtest) -- node[yesno, right] {Yes}
    (setvar);
  \draw (dqtest) -- node[yesno, above] {No}
    (error);
  \draw (setvar) -- (join1) -- (looptest);
  \draw (looptest) -| node[yesno, above] {Yes}
    (output);
  \draw (looptest) -| node[yesno, above] {No}
    (increase);
  \draw (output) -- (join2) -- (stop);
  \draw (error) |- (join2);
  \draw (increase) -- (multiply);
  \draw (multiply.east) -- +(.5,0) |- (join1);
\end{tikzpicture}
\end{document}
```

4 Conclusions

I hope you liked my explanation, and if you have trouble in using Ti*k*Z, remember:

A duck makes you laugh!

References

[1] Till Tantau. The Ti*k*Z and PGF packages. http://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf. Package page: https://ctan.org/pkg/pgf.

◇ Herr Professor Paulinho van Duck
Quack University Campus
Sempione Park Pond, Milano
Italy
paulinho dot vanduck (at) gmail dot com