# Building a tabular environment, via LaTeX macros…

I wanted to write a LaTeX macro to output something like the following:

```
\left(\begin{tabular}{cccc}
  1 & 2 & 3 & 4 \\%
  #1 & #2 & #3 & #4
\end{tabular}\right)
```

It would turn out o be quite the project. But on a first approach, it was rather simple.

```
\newcommand\perm[4]{\left(\begin{tabular}{cccc}
  1 & 2 & 3 & 4 \\%
  #1 & #2 & #3 & #4
\end{tabular}\right)}
```

This command can then be invoked as, for example, `\perm{1}{2}{3}{4}`, for the identity permutation. But what I really wanted was a command I could invoke like this: `\perm{3, 1, 4, 2, 5, 6}`, and that would produce:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 1 & 4 & 2 & 5 & 6 \end{pmatrix} \tag{1}$$

That is, I wanted the number of columns to be detected automatically, from the macro's input. The first task, is to detect the number of arguments. This is done as follows:

```
\usepackage{etoolbox} % Required for \forcsvlist.

\newcounter{lmax}

\setcounter{lmax}{0}
\forcsvlist{\incCounterMax}{#1}

\newcommand{\incCounterMax}[1]{\stepcounter{lmax}}
```

Recall that `#1` contains `3, 1, 4, 2, 5, 6`. The `\forcsvlist` calls the macro `\incCounterMax` for each element in the comma separated list, which increments by 1 the `lmax`, once for each element in the list. Thus `lmax` now has the value 6.

**A note about counters.** To be able to call the macro multiple times, any `\newcounter{countername}` command must be outside of the macro definition. Otherwise, when running it for the second time, LaTeX will complain that the counter is already defined… Furthermore, one must, insofar as possible, try to use unique counter names, because they

all inhabit the same space. That is why they are prefixed with 'l' (standing for "local").

Moving on, the next task, is to produce the first line, which in our example, consists of the numbers 1 to 6. This is done as follows (already inside the `tabular` environment):

```
\usepackage{forloop} % Required for \forloop.

\newcounter{lctr}

\newcounter{laux}
\setcounter{laux}{0}
\addtocounter{laux}{\value{lmax}+1}

\left(\begin{tabular}{*{\thelmax}{c}}
  1%
  \forloop[1]{lctr}{2}{\value{lctr} < \value{laux}}{%
    &\arabic{lctr}%
  }\\%
\end{tabular}\right)
```

First, in the declaration of the `tabular` environment, the `*{\thelmax}{c}` is a trick specify the number of columns dinamically—in this case, it is stored in the `lmax` counter, the value of which, **for non-arithmetic purposes**, is access as `\thelmax`.

Next, the `\forloop` works much like a C for loop; it is equivalent to the following `for(lctr = 2; lctr < laux; lctr++)` (the optional argument for `\forloop` specifies the increment value, in this case 1). The values start at 2 to be able to stack an ampersand (&) between them, but not after the last one (or before the first). But there is another catch: we want the values from 1 to 6, so the condition such be `lctr <= lmax` (lmax = 6); however, <= is invalid in LaTeX, and so I use another counter, `laux`, to store the value $6 + 1 = 7$, obtaining the condition `lctr < 7`, which also works. The `\arabic` command simply tells LaTeX to print the number in the counter, as an Arabic numeral. But that **for arithmetic purposes**, we need to access the counter value as `\value{couternam e}`.

Next came the really really tricky part: to iterate over the numbers given in the input, and to produce the second row, **adding an ampersand (&) between them!** To cut a very long story short, this cannot be done inside the tabular environment, because in LaTeX, the ampersand actually stands for `}{`, which wreaks havoc among `tabular`'s internals (and many other things). This means that one has to build the

second line, in a separate macro...(recall that `#1` contains `3, 1, 4, 2, 5, 6`)[1]

```
\usepackage{pgffor} % For \foreach command.

\def\zz{}
\foreach \i [count=\ni] in {#1} {%
  \ifnum\ni=1 \xdef\zz{\i} \else \xdef\zz{\zz & \i}\fi}
```

It is the `\zz` that then gets called inside the `tabular` environment. Here is the full code (note that inside a `\newcommand`, there can be no empty lines!):

```
\usepackage{etoolbox}
\usepackage{forloop}
\usepackage{pgffor}

% the 'l' prefix stands for local, to try to avoid counter name clashes...
\newcounter{lmax}
\newcounter{laux}
\newcounter{lctr}

\newcommand{\incCounterMax}[1]{\stepcounter{lmax}}

\newcommand\perm[1]{%
  \def\zz{}
    \foreach \i [count=\ni] in {#1} {%
      \ifnum\ni=1 \xdef\zz{\i} \else \xdef\zz{\zz & \i}\fi}
  %
  \setcounter{lmax}{0}
  \forcsvlist{\incCounterMax}{#1}
  %
  \setcounter{laux}{0}
  \addtocounter{laux}{\value{lmax}+1}
  %
  \left(\begin{tabular}{*{\thelmax}{c}}
    1%
    \forloop[1]{lctr}{2}{\value{lctr} < \value{laux}}{%
      &\arabic{lctr}%
    }\\%
    \zz
\end{tabular}\right)}
```

[1]The code below was partially taken from https://tex.stackexchange.com/questions/432274/use-array-meaningful-ampersand-to-change-column-in-foreach-loop.

When used like this:

```
\begin{equation}
  \perm{3, 1, 4, 2, 5, 6}
\end{equation}
```

The result is:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 1 & 4 & 2 & 5 & 6 \end{pmatrix} \tag{2}$$