low level TeX

grouping

Contents

1 Introduction

This is a rather short explanation. I decided to write it after presenting the other topics at the 2019 ConTEXt meeting where there was a question about grouping.

2 Pascal

In a language like Pascal, the language that TEX has been written in, or Modula, its successor, there is no concept of grouping like in TEX. But we can find keywords that suggests this:

```
for i := 1 to 10 do begin ... end
```

This language probably inspired some of the syntax of TEX and MetaPost. For instance an assignment in MetaPost uses `:=` too. However, the `begin` and `end` don't really group but define a block of statements. You can have local variables in a procedure or function but the block is just a way to pack a sequence of statements.

3 TEX

In TEX macros (or source code) the following can occur:

```
\begingroup
    ...
\endgroup
```

as well as:

```
\bgroup
    ...
\egroup
```

Here we really group in the sense that assignments to variables inside a group are forgotten afterwards. All assignments are local to the group unless they are explicitly done global:

```
\scratchcounter=1
\def\foo{foo}
\begingroup
    \scratchcounter=2
    \global\globalscratchcounter=2
    \gdef\foo{FOO}
\endgroup
```

Here `\scratchcounter` is still one after the group is left but its global counterpart is now two. The `\foo` macro is also changed globally.

Although you can use both sets of commands to group, you cannot mix them, so this will trigger an error:

```
\bgroup
\endgroup
```

The bottomline is: if you want a value to persist after the group, you need to explicitly change its value globally. This makes a lot of sense in the perspective of TEX.

4 MetaPost

The MetaPost language also has a concept of grouping but in this case it's more like a programming language.

```
begingroup ;
    n := 123 ;
engroup ;
```

In this case the value of `n` is 123 after the group is left, unless you do this (for numerics there is actually no need to declare them):

```
begingroup ;
    save n ; numeric n ; n := 123 ;
engroup ;
```

Given the use of MetaPost (read: MetaFont) this makes a lot of sense: often you use macros to simplify code and you do want variables to change. Grouping in this language serves other purposes, like hiding what is between these commands and let the last expression become the result. In a `vardef` grouping is implicit.

So, in MetaPost all assignments are global, unless a variable is explicitly saved inside a group.

5 Lua

In Lua all assignments are global unless a variable is defines local:

```
local x = 1
local y = 1
for i = 1, 10 do
    local x = 2
    y = 2
end
```

Here the value of `x` after the loop is still one but `y` is now two. As in LuaTEX we mix TEX, MetaPost and Lua you can mix up these concepts. Another mixup is using `:=`, `endfor`, `fi` in Lua after done some MetaPost coding or using `end` instead of `endfor` in MetaPost which can make the library wait for more without triggering an error. Proper syntax highlighting in an editor clearly helps.

6 C

The Lua language is a mix between Pascal (which is one reason why I like it) and C.

```
int x = 1 ;
int y = 1 ;
for (i=1; i<=10;i++) {
    int x = 2 ;
    y = 2 ;
}
```

The semicolon is also used in Pascal but there it is a separator and not a statement end, while in MetaPost it does end a statement (expression).

6 Colofon

Author	Hans Hagen
ConTEXt	2021.09.06 11:47
LuaMetaTEX	2.0923
Support	www.pragma-ade.com contextgarden.net