Nicola Vitacolonna

Objective METAPOST

In METAPOST, you declare variables like that:

```
    pair p[], q; picture a[]b[]; string s[][];
```

Have you ever wished you could declare your own “data types” the same way? Now you can! No more generisize! Just include the following pair of macros:

```
    def _do_declare(text var) =
      gobble begingroup
        save [ ; save ] ; let [ = \ ; let ] = \ ;
        if str var = "0": _eat? := false; def _eatvar = enddef; fi;
      endgroup
      endgroup
      if _eat?: yield(var); fi;
      gobble begingroup save , ; let , = ) ; _eatvar
    enddef;
    def declare(text type) text vars =
      begingroup
        save _eat ; boolean _eat? ; _eat? = true ;
        save _eatvar; def _eatvar text t = _do_declare ( t enddef;
        save yield; type;
        gobble begingroup save , ; let , = ) ; _do_declare( vars ) 0 ); endgroup
      endgroup
    enddef;
```

Here is a contrived example of usage:

```
    def cpaths =
      def yield(text __) =
        path __;
        vardef __.c = center #@ enddef;
      enddef;
    enddef;
    declare(cpaths) a[]b[], foo, bar[]; a1b2 = fullcircle; draw a1b2.c--(4,5);
```

Even templates are possible. For example, let us define a parametrised stack:

```
    def _stack_impl(text type) =
      def yield(text __) =
        type __[]; numeric __;
        vardef __.new = #@ := 0; enddef;
        vardef __.push(expr v) = #@[incr(#@)] := v; enddef;
        vardef __.pop = gobble(decr(#@)) enddef;
        vardef __.top = #@[#@] enddef;
      enddef;
    enddef;
    def stacks(text type) text vars = declare(_stack_impl(type)) vars; enddef;
    stacks(string) P, Q[]R[], S[]; S0.new; S0.push("x"); show S0.top; S0.pop;
```

To define a new “data type”, just define a macro that defines yield. In yield’s body declare the “attributes” (variables) and “methods” (vardefs) that make up your “objects”, using yield’s argument as a variable prefix. Such definitions lead naturally to making heavy use of two real METAPOST gems: the implicit (and underestimated) suffix parameters `#@` and `@`. The possibilities are endless. The above is the cornerstone of the author’s METAppeal library, in its early stage of development.