Metapost control structures

(conditions and loops)

Taco Hoekwater

September 16, 2022

- We will talk about conditions and loops
- This will be a fairly short because it is quite a simple topic

- Conditions are always wrapped inside `if ..: .. fi`:
- Conditions can be inserted (almost) everywhere
- Does not have to adhere to syntactical structure
- Try to think of it as an in-line preprocessor

The ⟨boolean variable⟩s *true* and *false* are primitives:

```
if true:
  message "hi";
fi
```

Boolean variables can be declared, of course:

```
boolean mystate;
mystate = true;
```

(they start off in the *false* state)

And you can these them in *if* expressions:

```
if mystate:
  message "hi";
fi
```

You can use parentheses to create a nested expression:

```
if (mystate):
  message "hi";
fi
```

Or grouping:

```
if begingroup
     mystate := false ;
     mystate
   endgroup:
   message "hi";
fi
```

You can test if a value is (un)known:

```
if known mystate:
  message "known";
fi
if unknown mystate:
  message "unknown";
fi
```

Boolean variables are unknown unless initialized, but indeed known when they are `false`.

You can test for the variable type:

```
if boolean mystate:
  message "boolean";
fi
```

this works for all other variable types as well (`if path mystate:` et cetera).

You can ask if something is a cyclic path:

```
if cycle fullcircle:
  message "cyclic path";
fi
```

the test works for anything, but is only true for cyclic paths.

You can ask if a ⟨numeric primary⟩ is odd:

```
if odd 5.5:
 message "odd";
fi
```

This test rounds up first, so -5.5 is odd and 5.5 is even.

An ⟨boolean primary⟩ can be inverted:

```
if not known mystate:
  message "known";
fi
```

There are special *if* tests for objects inside pictures

```
if filled p:
   message "filled";
fi
```

`filled`	true for filled paths inside pictures
`stroked`	true for stroked paths inside pictures
`clipped`	true for clip objects inside pictures
`bounded`	true for setbounds objects inside pictures
`textual`	true for typeset text inside pictures

actually tests the first item in the (sub)picture

⟨boolean primary⟩ can be composed of ⟨boolean secondary⟩ using *and*:

```
boolean mycondition;
if mystate and unknown mycondition:
  message "state true but condition unknown"
fi
```

⟨boolean secondary⟩ can be composed of ⟨boolean tertiary⟩ using *or*:

```
if mystate or unknown mycondition:
  message "state true or condition unknown"
fi
```

And tertiaries can by built up from expressions:

```
if 5 < 6:
  message "universe still sane";
fi
```

relation tests are: **<**, **<=**, **>**, **>=**, **=**, **<>**.

There is also a possible *else* clause:

```
if 5 < 6:
  message "universe still sane";
else:
  message "the sky is falling";
fi
```

And lastly, there is a chained *if* possible:

```
if 5 < 6:
  message "universe still sane";
elseif mystate:
  message "in limbo";
else:
  message "the sky is falling";
fi
```

*elseif*s can be repeated.

- Loops start with a ⟨loop header⟩ and end with *endfor*
- Loops can also be inserted (almost) everywhere
- Does not have to adhere to syntactical structure
- Try to think of it as an in-line preprocessor

Loops can be made with an explicit expression list:

```
for a = "1","2","3":
  message (a);
endfor
```

Or using a numeric progression:

```
for a = 1 step 1 until 3:
  message (decimal a);
endfor
```

Or using a suffix list:

```
vardef mymessage @# =
  message (decimal @#)
enddef;

forsuffixes a = 1, 2:
  mymessage.a;
endfor
```

Or even forever:

```
forever:
  message ("eternal");
endfor
```

Especially with that last option, it is also useful to be able to abort a loop:

```
a = 0;
forever:
  message ("eternal");
  exitif a>10;
  a := a + 1;
endfor
```

Finally, there is a way to loop over a picture's content:

```
for a within currentpicture:
  if stroked a: message "stroked"; fi
endfor
```

That's all!