Mathematics 308—January 9, 1996

Subroutines in PostScript

We can draw a square with the sequence

```
newpath
0 0 moveto
a 0 lineto
a a lineto
0 a lineto
0 0 lineto
closepath
stroke
```

but if we want to draw several squares this involves a lot of typing. It is more efficient and flexible if we have a subroutine to do the drawing, so that in the middle of a long program we can just type

```
square
```

and have a square drawn in the program.

A **procedure** in PostScript is any sequence of commands inside curly brackets `{` and `}`. Procedures can be treated just like any other objects in PostScript so that the effect of the sequence

```
/square {
newpath
0 0 moveto
a 0 lineto
a a lineto
0 a lineto
0 0 lineto
closepath
stroke
} def
```

is to let the single word `square` be assigned to the procedure inside the curly brackets. The effect of using the word `square` in a program is thereafter to replace it by the long sequence inside the brackets. *This is how procedures in PostScript always work: they just amount to defining a single word to stand for a complicated sequence of commands.*

As in other programming languages, you can pass arguments to procedures, but through the very simple method of putting them on the stack before you call the procedure. Thus we can rewrite the procedure above to make it more flexible.

```
/square {
/a exch def
newpath
0 0 moveto
a 0 lineto
a a lineto
0 a lineto
0 0 lineto
closepath
stroke
} def
```

has the effect of drawing a square whose side is the number placed on the stack just before the word `square`. Thus with this definition the sequence

```
3 square
```

will draw a square of side 3 units. The line `/a exch def` will probably seem a bit peculiar, but is very simple to understand if you keep in mind that a procedure just substitutes the commands inside `{`, `}` for the procedure name. Thus the command `3 square` will transform to

```
3 /a exch def ...
```

What is the effect? What we want is

```
/a 3 def
```

but we can't do that directly because this assignment takes place inside the procedure. However, the total effect of `3 /a exch` is to put the sequence `/a 3` on the stack, since `exch` just exchanges the top two elements on the stack. Then the `def` makes the assignment we want.

The routine above is still somewhat awkward because there may already be a variable `/a` in your program, and you probably wouldn't want to change its value. What you want is what is called a **local variable** for your procedure. This is a bit awkward in PostScript, but perfectly possible. I won't explain all the details of why it works, but what you do is add a few lines to the procedure establishing a **local dictionary** of variable names, like this:

```
/square {
1 dict begin
/a exch def
newpath
0 0 moveto
a 0 lineto
a a lineto
0 a lineto
0 0 lineto
closepath
stroke
end
} def
```

The extra lines you add are

```
1 dict begin
...
end
```

at the beginning nad end of your procedure. The variable `a` referred to here will be one used only in this procedure. The number `1` refers to the fact that you are only going to use one local variable. If you want to use more, say so. It does little harm to be over-generous, so that `5 dict` would be OK. This would be useful if you think you might want to add more variables later.

Here is a procedure with two arguments:

```
/rectangle {
2 dict begin
/h exch def
/w exch def
newpath
0 0 moveto
```

```
w 0 lineto
w h lineto
0 h lineto
0 0 lineto
closepath
stroke
end
} def
```

It assumes that you call it with two items on the stack, the width and the height in that order. When called, it removes the height and assigns its value to a local variable, then does the same for width. Then it draws the rectangle. Thus `2 3 rectangle` draws a rectangle of width 2 units and height 3 units.

A few fine points. The procedures above are perfectly correct, but they is probably not the most useful one you might want to write. For example, if you want to fill a square instead of stroking it you would have to write another procedure to do that, too. Instead, in PostScript it is usually a good idea to use procedures to add paths to ones you have already constructed, instead of drawing and stroking or filling the path all in one procedure. Thus it would probably be more useful in the long run if you were to define a procedure

```
/square {
1 dict begin
/a exch def
0 0 moveto
a 0 lineto
a a lineto
0 a lineto
0 0 lineto
end
} def
```

so that

```
newpath
3 square
stroke
```

would draw a square and

```
newpath
3 square
fill
```

would fill it.

This illustrates that *for good programs in general flexibility is an important quality.*