Mathematics 308—January 23, 1996

Happier PostScript programming hints

Many people are spending a lot of unnecessary time programming PostScript and getting extremely frustrated. I would therefore like to make the following remarks.

1. So far repeats and procedures have caused a lot of trouble. They are a bit tricky. So the best thing to do first is get a picture—any picture even distantly related to the assignment—up on the screen. Forget about repeats and procedures in the first attempt. Then you have something to gauge your progress against.

2. Even at this stage, however, it will help you to have variable names for certain constants like the pagewidth or size of margin. Give them readable names like `pagewidth` or `margin`. The purpose of using variables rather than raw numbers like `11.15` etc. is to reduce your mental effort, and readable names help a lot. define these **global variables** right at the beginning of your program, and don't change them again.

3. Do not at first put separate problems together. This slows everything up, and confuse things badly. Put each picture in a separate file at first. Then when each one is working on its own, you can put them all together. The best way to do this is with the UNX command `cat`. Thus

```
cat prob1.ps prob2.ps > hw.ps
```

typed within an `xterm` window will copy `prob1.ps` and `prob2.ps` in that order into the file `hw.ps`. If that file already exists, use `>!` instead of `>`, so it will be overwritten. Then you will have to do a little 'light editing' in `hw.ps` (removing initial setup, etc; adding showpages perhaps) to make it ready to hand in. Above all, try out `gs` on the final file before sending it along to me to be sure it all fits together. Remember that two lines

```
72 72 scale
72 72 scale
```

are cumulative and will scale to 72×72! `gsave` and `grestore` are ways around this.

4. Using `xterm` and `gs` is messy unless you remember my advice to begin by launching two copies of `xterm` by typing

```
xterm -sb -sl 500 &
```

in your login window when you first start up. Use one to run `gs` in and the other to do Other Useful Stuff like `cat` or `ls` or `mail`. Perhaps I haven't mentioned the `-sl 500` before—it remembers the last `500` lines so you can scroll back to them. The `&` is very important!

5. If you feel confident, just jump in and start drawing. But when things go wrong, keep in mind that the process is hypnotic. Take time out for a cup of coffee or whatever and sketch your PostScript in a notebook before you go back to programming.

6. Use variables only when it seems really useful. And as I mentioned above, give them good names.

7. To find out what `x` is equal to at some point in your program insert code

```
(x =) == x ==
```

This looks cryptic, but what it says is to display the string "x =" then the current value of `x`. For serious stuff insert `pstack`.

8. Do not confuse the **name** of a variable `/b` with its **value** `b`. As far as I can see, the only time so far in this course that you should use the name `/b` is when you assign a value to it.

9. Make lots of comments. In PostScript you should do this to keep track of what's on the stack.

10. Keep your stack clean. Be sure that at the end of procedures the stack has on it just what you want it to and no more extra junk.

11. Keep in mind that drawing in PostScript is like drawing by hand. `newpath` amounts to picking up a pencil. `moveto` amounts to moving it. `lineto` amounts to drawing a straight line from the current location of the pencil to the next one. But the thing you have drawn is not visible until the end, when you apply

```
stroke
fill
clip
```

Keep in mind that the last two are different from the first in that the path you draw must be *continuous* and *closed*, or stuff will 'leak out'.

12. How the path is actually drawn depends on the **graphics state** only at the moment you finish it. The graphics state takes into account (•) the current transformation; (•) the current colour; (•) current dash pattern; (•) current linewidth; and a few things you won't have to know about. *If you want to draw paths which are different in any of these aspects then you must finish one path before taking up another.* Like putting down your current pencil and picking up another one. In particular it does not make much sense to change any of these things in the middle of drawing a path, between the first `newpath` and the last `stroke`. It only confuses things. Make all changes of this kind *before* you put down a newpath.

13. Keep the lines which actually draw your paths as uncluttered as you can. It is hard enough to draw stuff, trying to keep track of where your pencil point is, so to speak, without worrying about how variables are changing; and unnecessary if those variables do not have to chnage while you are drawing. Put as much stuff outside path-drawing as possible.

14. You have seen most of PostScript. There are only a few more things you need to learn that you haven't acquired already. The main one is how to do fancier curves than just lines and arcs of circles, and that is coming up.

15. Programming is always potential chaos, since computers are so fast and yet so stupid. Anything you can do to make your programs simpler to understand and easier to read is always a good idea.

Floppy disks

16. You can move stuff from your home machines to the system in the lab by using the ancient IBM clone in the corner. To transfer `myfile` from your disk to your home directory, put the disk in the piece of junk and then:

```
cd \tmp
copy b:myfile .
ftp gamba
```

log into `gamba` with your id and password, and then

```
put myfile
```

To retrieve the stuff from the lab, do things in the opposite order, and use `get` instead of `put`.

```
cd tmp
ftp gamba
(login)
get myfile
copy myfile b:
```

Online help

17. David Maxwell (bless his heart) has just last week put up a large new help system for PostScript on the undergraduate lab network. A link to it can be found on the undergraduate lab home page `http://gamba.math.ubc.ca/localdoc`. This is in addition to online copies of my notes availble on the course home page `http://www.math.ubc.ca/~cass/courses/m308.html`.