Mathematics 308—Fall 1996

More 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. Remarks repeated from the first handout of this kind

1. 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.

2. 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`.

2. New advice

3. I repeat again more strongly one thing I have said a number of times. In this course and *when solving any kind of difficult problem, it is always a good idea to break the problem up into small chunks and solve them individually.* In this course this means essentially specific thing: *Use separate, isolated procedures as often as possible, procedures which should do something relatively simple, and which can be debugged simply.* You should do this just about whenever you have to do a job which is simple and clearly defined. In the current assignment, for example, it is a good idea to have procedures to calculate $\arccos(x)$ for a number between -1 and 1, to calculate the angle between two vectors, and replace an array of spherical coordinates (ρ, φ, θ) by an array of equivalent rectangular coordinates (x, y, z).

4. In the first set of remarks I said this:

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.*

But now things have changed, because in calling the routine `mkpath` you pass the name of the function you are plotting. For example in the line

```
[R] /circle 0 2 Pi mul 8 mkpath
```

you are passing the name of the routine `/circle` to `mkpath`. You can't type `circle` instead, because then you would be attempting to do some calculation. *The role of the function* `/circle` *is that, given the variable t, it calculates the two-dimensional point on the path you are drawing corresponding to the parametrizing variable t.* The point of the 'bag of parameters' you pass in an array (here `[R]`) is that what you pass in the array of quanities which determine which of several paths you are drawing. For example, a circle around the origin has a parametrization

$$t \mapsto (R\cos t, R\sin t)$$

but here different values of R will give you different circles.

5. In the present assignment, you will need some kind of a function `great-circle` to use in drawing a great-circle path of some kind. There are lots of great circles possible, so you will need to pass some parameters also. And you will have to decide what parameters to pass. You have to pass parameters which determine a particular great circle completely. You can't just pass the axis, for example, because that won't determine where the path starts. You could pass a pair of vectors w_0 and w_1 on the circle, where the path starts at w_0 and goes towards w_1, but then you would have to calculate the normalized α and the vector $w_* = \alpha \times w_0$ in order to calculate the point

on the path at time t. You will wind up doing all this calculation all over each time you add a Bezier segment to your path. This is unnecessary. It is much more efficient to calculate w_* once ahead of time and pass that and w_0 as parameters.

Summary of how to use `mkpath`

The array listed first contains a list of things necessary to specify your path from a whole family of paths. Next comes the name of the fundction which parametrizes your path. This function has a well defined interface: it assumes two things on the stack when it is called, the array **`[...]`** *passed to* **`mkpath`** *and the parameter t which varies along the path.* (Keep in mind: there are two uses of the word ‘parameter’ here: one for the things which determine which path in a family you are drawing, and one for the thing which varies to specify the path.)

6. This assignment is somewhat more complicated than earlier ones. The first question is intended to get you familiar with **`mkpath`**. But to jump right in and try doing the question is probably not a good idea. Instead, try some related uses of **`mkpath`**, for example try drawing various complete great circles on a sphere instead of the great-circle arc between two points. To do this would require perhaps in the parameter array an a ‘north pole’ (determining the axis) and a starting point on the great-circle, which is to say some other point on the unit sphere 90° away from the pole. I repeat what I have said before: *It is always a good idea to some kind of picture drawn right at the beginning, and then think about ways to modify that picture to get exactly what you want.* For problem #1 on the assignment, the simplest useful thing to draw would be the equator. Try using **`mkpath`** to do it. Or try drawing specific great-circle paths in a program rtaher than trying to make a procedure to draw an arbitrary one, and then gradually modify what you have done to get a procedure.

7. What I am saying is good advice for all problem solving: **Isolate difficulties!** Break a large problem into bite-sized pieces instead of trying to swallow it whole. Assemble smaller items to buid a large object . . .

8. The idea of isolation applies in other ways. *Different parts of your programs should do different things.* For example, I saw yesterday where a student with otherwise almost impeccable style had included inside a procedure a **showpage** operation. This is almost certainly a mistake—suppose that this procedure were to be required several times on one page? Separate function as much as possible to your programs **flexible**.