Interlude

At this point you have seen just about all the basic PostScript commands you'll ever use. The rest of this book will be spent showing you how to combine them together to make some very complicated—one would hope even very beautiful—figures.

In principle you should by now have no trouble drawing simple figures, but are starting to worry about how to do more difficult things. I want to offer you some advice, in part just to collect in one place remarks I have made throughout the text. Much of what I have to say is not specific to PostScript, but might well be made about programming in general.

- Most programs you write will be just an assembly of smaller chunks that are themselves quite simple. Your coding should reflect this, making the chunks in your code as independent of each other as possible. Just like your pages should be independent of each other.
- The reason you should try to arrange your program so it looks as much as possible like an assembly of smaller chunks is that you can then concentrate on getting each chunk completely correct. The most important thing to keep in mind in good programming is, as the real estate agents (don't) say, **locality**, **Locality**, **LOCALITY**. The effects of your code should be carefully set up to be local in nature, affecting if possible only data needed at the moment it is running. One example of this I have already remarked on is that procedures above all, which may be called from anywhere in your code, should use local variables and have few or no side effects. Those it does have should be clearly specified.
- Procedures should be as isolated as possible from the rest of your code. Best is to put them in separate files and `run` these. Then you can test your procedures independently of the rest of your program. With the PERL script described in Appendix 2, embedding files that are `run` during development into the final program is simple.
- In particular, make up a file defining your favourite constants like e, π etc. and run that file to obtain access to them. One thing I haven't mentioned is that for efficiency you can get PostScript to embed these numbers directly if you write using `//`. For example, `//pi` will immediately substitute $3.1415\ldots$ if you have previously written `/pi 3.1415926535 def`. (Whereas ordinarily it defers evaluating the expression `pi` because you might very well have redefined it. It doesn't know that you mean it to be a constant.) I don't suppose that for the programs we are concerned with here that this really increases speed much, but it makes you feel good.
- As a programming language, PostScript is special because of its direct link to graphics. Use this feature. When starting to draw a picture, begin by getting *something* up on the screen that looks roughly like what you want and then begin to modify it. Visual debugging compensates somewhat for the otherwise terrible debugging environment of PostSCript.
- Debugging PostScript using Ghostscript is nasty. The only way to avoid it, however, is to write only perfect lines of code that never need rewriting. But for those presumably rare moments when things aren't going quite right, you'll have to descend to the land of mortals. So far I have mentioned the techniques of spilling out data in the terminal window and running `gsnd`. To make this easier I myself use a procedure `display` with one argument n that spills out in an array, without destruction, the top n items on the stack:

```
/display { 1 dict begin
  /n exch def
  n copy [ n 1 add 1 roll ] ==
end } def
```

I put this in a file `display.inc` and run it at the top of nearly all my programs. For line breaks to make output more readable, use `() =`. You can also use `quit` to break your code off at a selected spot.

- Keep your stack clean. A common error is to forget to take everything off the stack in procedures. You can check this by running `gs`, which indicates the stack size at the end.
- Remember that coordinate changes are cumulative.
- The part of your code that actually does the drawing should be as clean and readable as possible. Do all necessary calculations ahead of time. Path drawing is the cockpit of PostScript programming. Leave unnecessary items at the door when you enter.
- At the beginning of a project, use lots of variables and procedures. Readability at that stage is extremely important. Comment freely. Slim down your code when and if necessary.
- Make your code readable, not only by adding comments but by separating different parts by dividers, say like this;

```
% --- this part does blah blah ----------------------------------------
```

so you can scan your file easily to get where you want to go. As for comments, the most important ones are those that describe procedures—tell what arguments they need, what they return, and what side effects they have. Procedures will usually be called many times in many different environments, and you will not likely want to read the whole procedure over again to figure out what it's doing.