§12. Excerpts from class, October 23 [notes by PMR]

One of the chief aspects of WEB is to encourage better programming, not just better exposition of programs. For example, many people say that about 25% of any piece of software should be devoted to error handling and user guidance. But this will typically mean that a subroutine might have 15 lines of 'what to do if the data is faulty' followed by one or two lines of 'what to do in the normal course of events'. The subroutine then looks very much like an error-handling routine. This fails to motivate the writer to do a good job; his heart just isn't in the error handling. WEB provides a solution to this. The procedure can have a single line near the beginning that says ⟨ Check if the data is wrong 28 ⟩ and points to another module. Thus the proper focus is maintained: In the main module we have code devoted to handling the normal cases, and elsewhere we have all the error-case instructions. The programmer never feels that he's writing a whole lot of stuff where he'd really much rather be writing something else; in module 28, it feels right to do the best error detection and recovery. Don showed us an example of this from his undergraduate class in which a routine had two references of the form

if ... **then** ... **else** *char_error*

pointing to a very brief error-reporting module.

We looked at a program written by another student who had the temerity to include some comments critical of WEB. Don struck back with the following:

> It is good practice to use italics for the names of variables when they appear in comments.
>
> Let the variables in the module title correspond to the local parameters in the module itself.
>
> According to this student's comments, his algorithm uses 'tail recursion'. This is an impressive phrase, helpful in the proper context; but unfortunately that is not the kind of recursion his program uses.

However, Don did grant that his exposition was good, and said that it gave a nice intuition about the functions of the modules.

We saw a second program by the same student. It had the usual sprinkling of "wicked whiches"—'which's that should have been 'that's. The purpose of the program was to "enforce" the triangle inequality on a table of data that specified the distances between pairs of large cities in the US. Don commented here that his project (from which these programs came) intends to publish interesting data sets so that researchers in different places can replicate each other's results. He also observed that a program running on a table of "real data," as here (the actual "official" distances between the cities in question) is a lot more interesting than the same program running on "random data." Returning to the nitty-gritty of the program, Don observed that the student had made a good choice of variable names—for instance '*villains*' for those parts of the data that were causing inconsistencies. This fitted in nicely with the later exposition; he could talk about 'cut throats' and so forth. (Don added that we nearly always find villainy pretty unamusing in real life, but the word makes for a witty exposition in artificial life; the English language has lots of vocabulary just waiting for such applications.)

Don wondered aloud why it is that people talk about "the n^{th} and m^{th} positions" (as this student had) thereby reversing the natural (or at any rate alphabetical) order?

He also pointed to an issue that arises with the move from typewriters to computer typesetting—the fact that we now distinguish between opening and closing quotes. We saw an example where the student had written "main program". To add to the confusion, different languages have different conventions for quotes; in German they appear like this: „The Name of the Rose". How to represent this in a standard ASCII file remains a mystery.

Back to the triangle inequality. Don pointed out that one obvious check for bad data in the distance table follows from the fact that the road distance can not be less than a Great Circle route. ("It could, if you had a tunnel" commented a New Yorker in the audience.) The student had written a nice group of modules based on this fact, and it illustrated the `WEB` facility of being able to put displayed equations into comments.

"So `WEB` effectively just does macro substitution?" asked another member of the class. Exactly, said Don. In fact the macros he uses are not very general—they really allow only one parameter. This means he doesn't need a complex parser, but in fact one can do a great deal within this restriction. For instance, it is not difficult to simulate two-parameter macros if we wish.

Someone in the class commented that it seemed a little strange to put variable declarations in a different module from their use. Don said that this was OK as long as they are close to their use, but large procedures should have their local variables "distributed" as the exposition proceeds.

Don recalled that older versions of Algol allowed you to declare a variable in the middle of a block. This fits in nicely with the `WEB` philosophy, but unfortunately cannot be done in modern Pascal. Indeed, Don became painfully aware of the limitations of Pascal for system programming when he was writing `WEB`—you can't have an array of file names, for example. He got around them, though, with macros.

One example of improving Pascal via macros is to define (in `WEB`)

$$\mathit{string\_type}(\#) \equiv \textbf{packed array}\ [1\,..\,\#]\ \textbf{of}\ \mathit{char}$$

so that you can say things like

$$\mathit{name\_code}\colon\ \mathit{string\_type}(2)$$

when declaring a two-letter string variable.

At this point, prompted by a note from Tracy, Don announced that 23 copies of the *Handbook for Scholars* had arrived in the Bookstore, with more to come. A resounding cheer echoed throughout Terman.

Don commented that the student had given a certain variable the name '*scan*'. Since this variable was essentially a place marker, Don thought that a noun would be much

better than a verb—'*place*', perhaps. Let the function determine the part of speech; think of it as a kind of Truth in Naming. Verbs are for procedures, not data.

The last student had written a program to handle graph structures based on encounters between the characters in novels. He too had made the "quote mistake". The student gave a nice characterization of the input and output of the program, using the typewriter font to illustrate data as it appears in a file.

This student also showed a bit of inconsistency in the use of 'it' and 'we' as the personification of his program. We seem to be finding the same old faults over and over now, Don said, so perhaps that indicates that we have found them all. Discuss.