shows, all recently entered text has been analyzed as a side-effect of the language-based commands (analysis can never cause a command to fail, nor can it place the user in any special modes), and the cursor now rests at a statement — both structurally, revealed by shading the entire statement with a pale blue background, and textually, as shown by the usual inverted box. The user may now (as always) move forward either structurally or textually (probably easiest) to the erroneous operator "+", delete the character textually, type in the correct operator "`*`", and proceed with other tasks.

Pan can be configured to make available many kinds of information *about* documents, depending on the task at hand, using a variety of techniques. For example, Figures 3.5 and 3.6 show some of these techniques applied to language diagnostics. In Figure 3.5 a **panel flag** in the form of a red exclamation point appears near the upper right corner of the window. This particular flag reveals that some number of language errors are known to the analyzer; in some *Pan* configurations this will be the only clue available that the program is not well-formed. In this configuration, however, a **highlighter** for language errors is also available and has been toggled on by the user. This highlighter has been configured to render malformed code with red ink, and three such lines appear in Figure 3.5 (including one under the structural cursor). Highlighting draws attention to error sites, and is often all the notification that an expert programmer needs. Furthermore, the user may navigate to error sites by setting the level appropriately ("`Language Error`" is configured to be a component of the level menu) and invoking structural navigation commands (forward, backward, mouse click, and the like). Figure 3.5 show the result of such a navigation command, with the cursor landing on the "`IMPORT`" statement. Navigation helps locate error sites that may be scrolled off screen, and as a useful side effect it announces in the panel the diagnostic at each site reached. The announcement is context sensitive, however, and navigation to the same site at the "`Statement`" level would produce no such message. Finally, an alternate view may be requested that summarizes a particular kind of information, language errors in this case, as shown in Figure 3.6. Structural navigation is shared among all views, so setting the structure cursor on the line "`malformed expression`" in the language error view also causes the cursor in the primary view to be set and the window scrolled appropriately.

Chapter 4 describes in considerably more detail how *Pan*'s language-based mechanisms are constructed, and Chapter 5 discusses how they can be applied to a wide variety of useful tasks for users.

3.3 Design Principles

This section introduces the general design principles that guided work on *Pan*. All are rooted in the observation that a successful system addressing *Pan*'s goals must reflect good user interface design on two fronts.

1. The system itself must strike users as familiar, simple, and easily learned in productive increments.

2. The entire system must be considered an interface between users and documents, helping to make those documents comprehensible.

is a weak input mode that controls generic language-based commands, but does not interfere with textual commands.

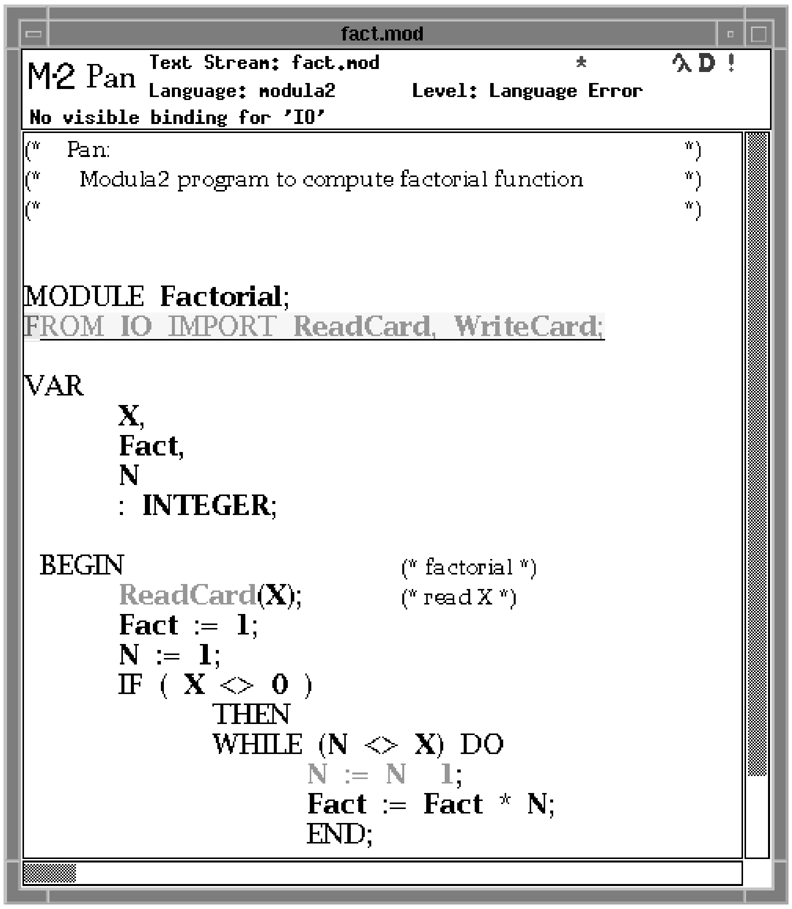

Figure 3.5: Using *Pan*: Highlighting and Finding Errors