PL/CT, Another Approach to Two Problems in Interactive PL/I

Steven Worona

TR 75-236

April 1975

Department of Computer Science
Cornell University
Ithaca, NY 14853

PL/CT, Another Approach to Two Problems in Interactive PL/I

Steven Worona
Department of Computer Science
Cornell University
Ithaca, NY 14853

Introduction

In "Interactive PL/I" [1] Zelkowitz describes how STREAM output and ON-conditions cause difficulties for the PLUM implementation of interactive PL/I. Wolman, however, claims in "Reply to 'Interactive PL/I'" [2] that these problems are "defects in the operating system or the implementation", rather than "deficiencies in the language" as Zelkowitz calls them.

This note reports how these two PL/I features are handled in another recent implementation of interactive PL/I, hopefully providing some insight as to where the "deficiencies" really lie.

PL/CT

PL/C [3] is an in-core compiler for a large subset of the PL/I language. (The major omissions are the list-processing features.) The compiler itself repairs syntactic and semantic errors [4], producing machine code which executes under the supervision of a monitor. Hardware- and software-detected run-time errors are analyzed, diagnosed, and repaired by the monitor, which also prints appropriate error messages.

PL/CT [5], the interactive version of the PL/C system, runs on IBM hardware under both TSO and CMS. Modification of the basic compiler consisted simply of replacing the operating-system interface modules, and adding a relatively small run-time debugging package. The debugging package accepts commands which set and display variables, establish breakpoints, and control the manner in which execution is resumed.

STREAM Output

Our experiences with PL/CT lead to the feeling that Wolman is wrong in ascribing Zelkowitz's STREAM output problems to the operating system used or implementation chosen. The difficulties are inherent in the PL/I language, which was clearly not designed for use in an interactive environment. (The language definition considered is that of IBM's PL/I-F, with which PL/C is compatible. Different language definitions — the proposed ANSI standard, for example — may require different implementations for interactive use. It is felt, however, that the considerations discussed here will apply to most PL/I dialects.)

Consider the following section of code:

```
PUT SKIP LIST('ENTER NEW VALUE OF I');
GET LIST(I);
PUT SKIP LIST('NEW VALUE OF I =',I);
```

Normal (non-interactive) output from this program would include

```
ENTER NEW VALUE OF I
NEW VALUE OF I =            10
```

In interactive systems, the standard input and output files are generally associated with the terminal. This creates the problem of synchronization of terminal output and input. Implementors have several options in dealing with synchronization. (In the following examples, characters typed by the programmer are underlined.)

Option 1: Do nothing

Under either TSO or CMS, this approach would produce this conversation from the sample program:

```
10
ENTER NEW VALUE OF I
NEW VALUE OF I =            10
```

That is, I/O progresses just as if the terminal were a card reader and line printer. While such devices (and the spooling systems frequently used) don't care whether a prompting line is printed before or after the corresponding input card is read, programmers cannot be expected to display such tolerance. In many cases, the user will want to see the contents of "partial" output lines.

It is essential to realize that programmed I/O requests are not the only causes of terminal activity in an interactive system. In particular, when a program is interrupted--because of an error, designated breakpoint, or "attention" signal--control generally returns to the terminal for entry of debugging commands. Proper analysis of program status in these cases very frequently depends on "output" which has been formatted but not printed.

Option 2: Flush buffers after each PUT

This is the PLUM approach. Unfortunately, although the desired synchronization is obtained, the statements

```
PUT LIST(X);
PUT LIST(Y);
PUT LIST(Z);
GET LIST(I);
```

cause three lines to be printed when one would have served.

Aside from wasting paper, this causes terminal output to look markedly different from batch-mode output, which would include only the one line specified by the program. This is a distinct disadvantage in the often-occurring case where formatting is a major part of the program's function.

Option 3: Flush buffers before each GET

This is somewhat better than the PLUM solution, achieving synchronization without extra output. It is, partially, the PL/CT implementation, although a somewhat more complicated strategy was chosen--and another is still under consideration--because of reasons to be discussed later.

Option 4: Character-by-character printing

In Wolman's implementation (run under the Multics system), terminal output appears character-by-character, rather than line-by-line. Thus, according to an example he cites, the statement

```
put list('Hello there');
```

immediately prints the designated string, leaving the terminal ready to type at the very next character position on the line. He notes that "this behavior is simply achieved by having the I/O module that processes PUT statements empty the buffer associated with a file after each PUT statement."

Note that this is exactly the PLUM strategy (Option 2)! In both cases, it is necessary to modify the "standard" PL/I output rule of emptying a buffer only when it is full or a SKIP is processed. Multics does seem to offer an advantage in allowing the separation of buffer flushing and carriage return. The question remains, though, as to what happens to terminal input. If this is simply appended to the most recent program-output line, the conversation records will be totally garbled. If each request for terminal input is preceded by a carriage return, as seems likely, then this reduces to Option 3. In either case, it appears that printing character-by-character offers no advantages to line-by-line mode.

(I should stress that this last statement applies only to the case at hand, namely, implementation of PL/I STREAM output. There are certainly examples of implementations for which a character-mode output capability is of the utmost importance.)

Option 5: Minimizing formatting differences

As already mentioned, it is often important that the terminal output produced while interactively debugging a program accurately represent the format of batch output. A proposal for reconciling this with the goal of synchronization is still under consideration for PL/CT.

A similar problem has already been solved in the standard version of PL/C. As opposed to PL/I-F, run-time error messages produced by PL/C do not terminate execution. Rather, each reports an error and a corrective action taken by the monitor to allow execution to continue. Adherence to "normal" output format was considered an important design goal, but it was obviously essential that "partial" output lines be printed to pinpoint program status. The result was a technique which produces the output lines indicated from the following program:

```
T: PROC OPTIONS(MAIN);
DCL I(5) INIT(1,2,3,4,5);
PUT SKIP EDIT((I(J) DO J=1 TO 7))(F(5,2));
END T;
```

```
 1.00 2.00 3.00 4.00 5.00
*****ERROR IN STMT 3  SUBSCRIPT 1 OF I IS OUT OF BOUNDS (6). 5 IS U
                                         5.00
*****ERROR IN STMT 3  SUBSCRIPT 1 OF I IS OUT OF BOUNDS (7). 5 IS U
                                              5.00
```

Thus, although error messages interrupt the vertical integrity of the output format, they do not affect the horizontal integrity.

The implementation of this feature is completely trivial, involving only the saving and restoring of some pointers by the error-message writer. An analogous strategy can be adopted in interactive systems when buffers must be flushed for synchronization purposes.

Multiple Terminal Output Files

It is not uncommon to have more than one file producing terminal output simultaneously. In PL/CT, for example, programmed PUT statements default to the terminal file SYSPRINT. At the same time, output from the run-time debugging package appears on SYSTERMO. The separation is provided so that programmed output can be easily re-directed to a line printer, disk data set, etc.

In general, when a program is interrupted, the run-time debugging package issues a prompting message indicating the location and cause of interrupt before accepting debug commands. Clearly, then, flushing buffers in any of the options discussed previously applies to both "program" files (e.g. SYSPRINT) and "system" files (e.g. SYSTERMO). Moreover, if several program files are all producing terminal output, all should be flushed, with the system files flushed last. Of course, the horizontal integrity described in Option 5 applies on a file-by-file basis, and only to program files.

ON-Conditions

Zelkowitz reports that the implementors of PLUM found ON-conditions ill-suited to an interactive environment, and therefore omitted them. Interactive debugging mode is entered whenever an error condition occurs. Wolman's Multics system retains ON-conditions, invoking a debugging routine when a condition arises for which no ON-unit is pending. The debugging routine is sometimes (it's not clear just when) a "Multics command processor", which then accepts debugging commands.

In implementing PL/CT, absolutely no questions arose concerning ON-conditions. It appears, however, that this was due to the error-correcting nature of PL/C, rather than the "suitability" of ON-conditions for interactive use.

ON-conditions in PL/CT are treated just as in PL/C. When a condition is raised,

(1) If it is disabled: do nothing; continue execution
(2) If it is enabled, but there is <u>no</u> pending ON-unit: print an error message, perform the required repair, and continue execution
(3) If it is enabled, but there <u>is</u> a pending ON-unit: perform the required repair if necessary, enter the ON-unit, continue execution.

This seems quite well-suited to an interactive environment, with the following additional rule: After any run-time error message is printed, interactive debug mode is entered. This rule is necessary to handle the common PL/C error messages not associated with PL/I conditions. It also applies very nicely to the error messages printed as part of the standard no-ON-unit response to raising an enabled condition.

Summary

Zelkowitz claims that PL/I is "deficient" for use in an interactive environment. Wolman asserts that any deficiencies are in Zelkowitz's operating system or implementation. It should be clear by now, however, that the PL/I definition of STREAM output is not well-suited to interactive use. Although a technique exists for maintaining terminal synchronization without sacrificing much format integrity, things would obviously have been less complicated in a language like FORTRAN. The villain, of course, is the lack of an implied end-of-record following each PUT. Difficulty of interactive use is the price paid for this added flexibility.

The ON-condition argument seems less clear. The PLUM view is that, given the capability for interactive error analysis, there is no need for the relatively inflexible actions of ON-units. Multics and PL/CT use interactive debugging in cases where ON-units are not specified. However, the only obvious uses of ON-units in an interactive environment are in programs being developed for batch execution, and in those cases where exceptional conditions (i.e., overflow) occur in pre-determined contexts. Considered simply as general-purpose debugging aids, ON-units are no match for a live programmer.

References

1. Zelkowitz, M., "Interactive PL/I", SIGPLAN Notices 9, No. 9, September, 1974 (29-31).

2. Wolman, B., "Reply to 'Interactive PL/I'", SIGPLAN Notices 10, No. 4, April, 1975 (46-48).

3. Worona, S. et al, User's Guide to PL/C, June, 1974.

4. Conway, R. and T. R. Wilcox, "Design and Implementation of a Diagnostic Compiler for PL/I", CACM 16, No. 3, March, 1973 (169-179).

5. Moore, C. G., S. L. Worona, and R. Conway, User's Guide to PL/CT, February 1975, Cornell University Technical Reports 75-230 and 75-231.