# Discussion and Correspondence

## A note on program debugging in an on-line environment

*By* D. W. Barron*

It is commonly asserted that one of the main benefits of an on-line system is that it simplifies the task of getting a program to work: indeed, this has become part of the gospel of multi-access. This note examines this assertion, and suggests that it is true, though not for the reasons usually given.

The obvious advantages of a multi-access system are twofold: a filing system that allows programs to be stored and readily altered, and almost instant access to the machine for test runs. Anyone who has used such a system for program development will testify that these are great benefits; however, comparable benefits can be obtained at less cost. Programs on punched cards can be modified fairly easily, and a full-scale filing system can be maintained quite separately from a remote console system. The same goes for access to the machine: given a ten-minute turnround on test runs most programmers would be happy, and it is certain that many of the people in universities who ask for multi-access really want rapid turnround. Though most conventional systems provide a turnround measured in hours or even days, this should not obscure the fact that rapid turnround can be achieved without a full-scale remote console system.[1,2,3,4]

However, supposing that we have a multi-access system available, with a filing system and instant turnround, what do we pay (in programmer convenience, not machine efficiency) for these benefits? The most significant trade-off is that since we are communicating with the machine through a narrow bandwidth channel, we are restricted in what we get from the machine to what can be printed in a reasonable time at ten characters per second. No listings, no core dumps: with a compiler that was written for off-line use, this makes life difficult. The minimal facility necessary is the ability to record a core-dump in a file, for later examination. If on-line debugging is to be effective one needs an elaborate interrogation program which can access the core dump and the symbol table produced at load-time, so that the programmer can ask for information identified by symbolic name, and receive replies in source-language form.

It might seem from the foregoing that a console is not a very effective debugging aid. However, a full-scale multi-access system has one trump card: the ability to interact with a program whilst it is running. Any programmer knows that finding a program bug from a core-dump taken after the event is at best an unsatisfactory procedure. If the bug is at all subtle its effects will be far from obvious, and it may require a substantial intellectual effort to work backwards to what actually went wrong. (Consider, for example, the case of a program that goes wrong then immediately overwrites the offending section with an overlay.) Dynamic monitoring of a program as it runs is a much better way of proceeding, and since single-shotting is not practicable on a large machine this has led to the development of tracing systems. Most of these, however, suffer from the defect of producing too much output, since the user cannot be very selective in what he asks for. With a conversational system it is possible to be highly selective. The debugging system can allow the programmer to intercept his program at a particular point, or when a particular set of conditions is satisfied, control then reverting to the console. The programmer can then type in questions to find the values of variables or the content of store registers, he can change the contents of store registers if he wishes, and he can then resume the program, either where it was interrupted, or at some other place.

It is not the purpose of this note to describe such systems in detail (descriptions can be found in the literature,[5,6]) but to make the point that having remote consoles and a multi-access system is not going to remove debugging problems, unless the consoles are backed up by a lot of sophisticated software.

* *Department of Mathematics, The University, Southampton*

**References**

1. LYNCH, W. C. (1966). Description of a High Capacity, Fast Turnaround University Computing Center, *Comm. ACM*, Vol. 9, p. 117.
2. IRONS, E. T. (1965). A Rapid Turnaround Multi-Programming System, *Comm. ACM*, Vol. 8, p. 152.
3. SHANTZ, P. W. *et al.* (1967). WATFOR—The University of Waterloo FORTRAN IV Compiler, *Comm. ACM*, Vol. 10, p. 41
4. ROSEN, S. *et al.* (1965). PUFFT—The Purdue University Fast FORTRAN Translator, *Comm. ACM*, Vol. 8, p. 661.
5. BOILEN, S. *et al.* (1963). A Time-Sharing Debugging System for a Small Computer, *Proc. SJCC*, p. 51.
6. CRISMA, P. A. (editor) (1965). Sections on FAPBUG and MADBUG in *The Compatible Time-Sharing System: A Programmers' Guide*, 2nd edition, The M.I.T. Press.

# Correspondence

*To the Editor*
*The Computer Journal*

**What is an analyst?**

Sir,

Recently a quantity of perfectly good paper, on which programs might otherwise have been written or circuits drawn, has been expended in the attempt to define hierarchies in computer skills. Some dregs of classical education may perhaps help resolve one point in this difficult and important exercise. According to its Greek roots, 'analyst' should be the opposite of 'catalyst.' Taking the dictionary definition of catalyst and applying a single negation, one accordingly finds the following:

'An analyst is one who while taking no essential part in a process nevertheless impedes its progress.'

For those who through no fault of their own are called systems analysts, I should add that I do not really mean it. Nevertheless, it is true that 'programmer' remains the most honorific term in my vocabulary of this subject.

Yours faithfully,

PETER FELLGETT

Department of Applied Physical Sciences,
Whiteknights, Reading.
28 October 1968