

Vijay Kumar Garg
IIT, Kanpur, F302, Hall One
INDIA

SCREEN-ORIENTED HIGHLEVEL DEBUGGER(SHD) for PASCAL

There are certain golden maxims in programming. One of them is, that no matter how meticulous a programmer is, he spends major portion of his software development time on debugging. In such wise, we must have good debuggers to reduce debugging time. As Lauesen[1] puts it, the superiority of a debugger lies in helping user catch more bugs in given time. Accordingly, power of a debugger is the sole yardstick for goodness of it. Power in this context mean both, functionality and convenience. Other factors, such as, extra information it requires from compiler, its bigger size etc. are irrelevant so long as it is extra powerful. The complexity of the debugger, in terms of efforts required to program it should not be a prohibiting factor either. A debugger is used, as often as, if not more than the compiler itself. Conventional Debuggers[2] which call themselves 'High Level Debugger' are no doubt better than 'Dumpers' but are still too primitive to help user in any substantial manner. Most of them are confined to setting up of breakpoints and examination of unstructured variables. SHD strives to match its level of commands with the thought process of the user for maximum efficiency in detecting bugs.

SHD IS AT HIGHER LEVEL BECAUSE

1) It talks with user in entities like tree, linked-list, graph, stack, table and such other frequently used highlevel structures. This is unlike CD which adhere to bare pointers, integers, arrays and records. The lower level of communication has many disadvantages which are accentuated in dynamic structures. One, in case of graph, trees etc. the user has to figure out the connection between his various nodes by himself. He is given an integer corresponding to each dynamically allotted record. The pointers are shown to have this value. Using these integers he draws his data structure manually. The process becomes very complicated as the number of nodes increases. Two, the user is forced to type things like First^.next^.next^.next^ to see fourth node in his linked-list and sure enough he would have never seen thirteenth node in his list. Three, by far the most serious drawback is that the user has to lower the level of his mental process from that of logical data structures to that provided by the programming language. This drastically cuts down his efficiency by placing him into tormenting world of lower level details. On the other hand, the level used by SHD suits his mental process. Data structures are displayed to him the way he conceptualizes them. For example, SHD when given following commands displays a symbol tree as shown in fig.1.

```
SET SYMROOT^ TREE LLINK RLINK  
DISPLAY SYMROOT^
```

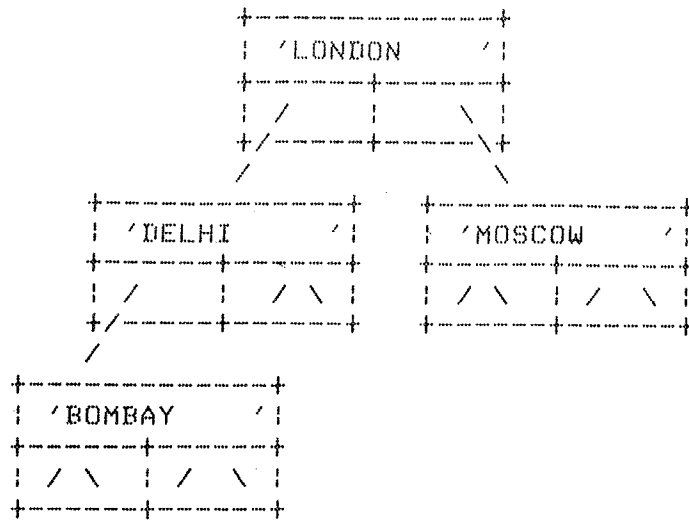


Fig.1

For graphs, methodology as discussed in Graphtrace[3] is used. To see a graph in the format desired, the user sets each pointer as horizontal, vertical, direct or invisible. As a further assistance, he is given a flexibility to set any of fields in record as invisible or to declare the fieldlist he wants to inspect in a record with untaged variants[6]. Consequently, the user can reduce the representation of any data structure to the form he would like to conceive as.

2) It gives user an ability to make assertions about his data structures. Much has been talked upon assertions in programming languages but it remains to be seen how many programmers will spend their time thinking and putting rigorous assertions in their program. Mostly it is only when the program does not work, that the programmer has any urge or tendency to make assertions. Moreover, many times the programmer might like to make assertions which are dependent on input data and therefore can not be made in program anyway. SHD lets user make assertions in two ways:-

IMMEDIATE ASSERTION:: This assertion is made and checked for validity at a breakpoint. For example, suppose at some breakpoint he thinks that his matrix A should be symmetric. In CD, he would be forced to check this using sequence of displays. Using SHD, he can make assertion of the form

(FOR_ALL I=1,N)((FOR_ALL J=I,N)(A[I,J]=A[J,I]))

SHD reports to the user validity of his assertion, and, if A turns out to be unsymmetric, it provides the user a way of finding these offending indices.

LINE ASSERTION:: This assertion though issued at a breakpoint is not meant to be checked then. Line Assertion is equivalent to putting the assertion before all the lines in the portion of the program specified by the user. This has two implications. One, the program is running while the assertion is being checked and is interrupted only when an assertion is found invalid. Contrast it with Immediate Assertion which is checked when the program is already interrupted and SHD is at top level for accepting commands. Two, values of all variables used in assertion are calculated afresh when such an assertion is encountered during program execution.

3) It does not require user to refer his program listing for line numbers. In general, a nontrivial program has not one but many bugs. After correcting one bug, however small it may be, the programmer is compelled to take a new printout of his program. This is because either line no.'s have changed or some new lines have come in. This is undesirable both because of time and paper the user needs to spend for a printout. This is more so, in case of a network, where a lineprinter is being shared and is either distant from terminal or altogether unavailable at that time. In order to avoid taking a printout, when an user fixes a bug, he may be expected to make the small change in his printout itself. But it would be unreasonable to expect him to note new line numbers in his old printout. SHD solves this problem by incorporating a readonly editor in it. The user can ask SHD to print certain lines of his source program or find a string in his source program. With help of these commands he can find out required line numbers to make an assertion or to put a breakpoint.

4) It tries to minimize number of times user has to refer line numbers in his program. Infuriating variables [4,5] serve as a good example here. In CD the user has to look for lines in program where the variable is assigned a new value either by direct assignment or by a function or procedure call. After that he puts breakpoint at those lines and on reaching these breakpoints he asks CD to display it. This is a clumsy and cumbersome way of doing it. In SHD, a high level construct BREAK ASSIGNMENT allows user to do all this in one stroke. It stops whenever an assignment is made to the infuriating variable and informs the user of its new value. At this point an user can ask it to print surrounding few lines to avoid referring program listings. As another example suppose that user suspects certain procedure to be foul. In this case, he can use BREAK CALL which breaks the program on call of the procedure after which the user can step-execute.

5) Other highlevel facilities include

* Trace of a variable.

The program while running prints the value of the variable and the line number without stopping at those lines. Note the way it differs from BREAK ASSIGNMENT.

* Dynamic nesting at any breakpoint.

He can optionally examine parameters of procedures in dynamic nesting.

* Dynamic Trace of Procedures.

This feature displays various calls to procedures while the program is running. It also optionally gives value of parameters passed and values returned. Such feature is generally present in interpreters, but for no good reason, is missing from debuggers using compiled code.

REFERENCES::

- [1] Debugging Techniques. Lauesen S., Softw. Pract. Exper. 9,1 (Jan 1979) 51-64
- [2] A High Level Debugger for PL/I, Fortran and Basic. Elliot B. Softw. Pract. Exper. 12,4 (April 1982) 331-340
- [3] A Very High Level Interactive Graphical Trace for the Pascal heap. Getz et. al. IEEE Trans. Softw. Enss. 9,2 (March 1983) 179-185
- [4] Runtime Print Values. Finkel R., SIGPLAN notices 18,2 (Feb 1983)
- [5] Tracing Offensive Values from Exceptions. Gutfreund S. SIGPLAN notices, 18,7 (July 1983)
- [6] Pascal, User Manual and Report, Jensen & Wirth.