Quaestiones Informaticae

Vol. 2 No. 2 May, 1983

Quaestiones Informaticae

An official publication of the Computer Society of South Africa
'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika

Editor: Prof G. Wiechers
Department of Computer Science and Information Systems
University of South Africa
P.O. Box 392, Pretoria 0001

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R6	$7	£3,0
Institutions	R12	$14	£6,0

Circulation manager
Mr E Anderssen
Department of Computer Science
Rand Afrikaanse Universiteit
P O Box 524
Johannesburg 2000
Tel.: (011) 726-5000

Quaestiones Informaticae is prepared for publication by Thomson Publications South Africa (Pty) Ltd for the Computer Society of South Africa.

NOTE FROM THE EDITOR

Three points must be made by way of introduction to the second issue of Volume 2 of Quaestiones Informaticae.

Firstly, an apology is in order for the mistake in the date (November 1983 instead of 1982) at the foot of my note introducing the preceding issue. Lacking the services of a professional proof reader, printing errors are bound to show up from time to time, but it is hoped that their number will be kept to a minimum!

Secondly, it is a pleasure to announce that this journal will not only serve to publish papers of a scientific or technical nature on computing matters under the auspices of the Computer Society of South Africa. An agreement has been reached to share the facilities of Quaestiones Informaticae between the CSSA and SAICS, the South African Institute of Computer Scientists. Henceforth this journal will also be used to publish the Transactions of this Institute. This implies certain changes to the cover pages which will be implemented in future issues. I shall continue to serve as editor, but on behalf of SAICS Prof R. J. van den Heever will share some of my duties and act as co-editor.

Finally Mr Edwin Anderssen, of Rand Afrikaanse Universiteit, has agreed to serve as circulation manager for Quaestiones Informaticae. I am grateful indeed that he is willing to serve the journal in this capacity, and look forward to a long period of fruitful cooperation.

G WIECHERS

May, 1983

Data Structure Traces

S. R. Schach
University of Cape Town

Abstract
Three levels of traces for data structures (as opposed to simple variables) are defined. A machine-code core dump is essentially a low level trace. A high level trace reflects the high level language in which the data structure being traced has been implemented. A very high level trace displays the data structure in the format in which the programmer conceptualizes it. Three traces written by the author (a graphical FORTRAN array trace, a portable trace for the Pascal heap, and a graphical Pascal data structure trace) are described, and the level of each trace is then analyzed.

INTRODUCTION

Just as languages may be described as high level or low level, so we may classify the level of a trace. For example, suppose we wish to trace the Pascal statement

total := total + 50

If the value of total was 1 000 before the statement was executed and the statement itself is at line 123, a high level trace might print something like

line 123: total = 1000/1050

Such a trace may be described as high level because it reflects the high level language in which the statement being traced was written.

But if the trace prints the machine-code equivalent of the original Pascal statement, or the trace takes the form of a core dump, then such a trace is low level.

When tracing data structures, rather than simple variables, a third category of trace may be defined. If the output of the trace displays a data structure in the format in which the programmer conceptualizes it then such a trace will be termed very high level. For example, if tracing an array causes the elements of that array to be displayed arranged in rows and columns, or if a tree or doubly-linked list appears as such, then since the "shape" of the trace output corresponds to the "shape" of that data structure within the programmer's mind this is a very high level trace.

In this paper the above three categories of data structure trace are analyzed and evaluated. Then three data structure traces written by the author are described, and the category into which each falls considered. The traces are:
(1) A FORTRAN array trace [6].
(2) A trace for the Pascal heap [7].
(3) An interactive graphical trace of the Pascal heap [2].

COMPARISON OF DATA STRUCTURE TRACES

Low level data structure traces

Debugging a program written in a high level language by analyzing a machine-code core dump of a data structure is most undesirable. In the first place, the programmer is required to have detailed knowledge of the internal representation of his program and data, defeating the whole purpose of high level language programming. Furthermore, the ability to understand core dumps is becoming increasingly rare as fewer and fewer programmers are being trained in low level languages. But even if a programmer does possess this skill, it is strictly non-portable from machine to machine, and generally from compiler to compiler, as there is certainly no guarantee that (say) two Pascal compilers will store packed records in the same way. Thus low level data structure traces should be employed only if there is no alternative form of tracing available.

High level data structure traces

When programming in a high level language such as FORTRAN or Pascal, it should be possible for a user at all times to "think high level". That is to say, he should almost never have to consider how the machine is handling either his program or the data on which it operates. The only occasion when the high level language programmer should have to descend to code level is in order to optimize critical loops to speed up program execution; this in itself should be very infrequent for the vast majority of programmers.

The ideal situation is that if a data structure needs to be traced then the output from the trace should resemble the original high level source code as closely as possible. For example, the actual user-defined names of variables should appear (rather than their addresses); furthermore, the names of fields within records should be specified. In this way the programmer is freed from the burden not ony of having to understand the internal machine representation of his data structure, but also of having to think in terms of that internal representation, thereby defeating the whole purpose of programming in a high level language [8].

Very high level data structure traces

It can be very helpful for a user to "see" a data structure which he has conceived in the format in which he has conceived it. For example, if a user has conceptualized a two-dimensional array in terms of rows and columns, but has for some reason transposed the elements of such an array, then a graphical display will quickly show up his mistake. On the other hand, merely listing the array elements, even in a high level format, may not solve the problem; the user may simply not appreciate that (say) the first index, rather than the second, corresponds to "row number".

At a more advanced level, the fact that a language like Pascal allows literally an infinite number of different possible data structures means that very complex structures can arise; presenting a user with graphical output depicting his data in a form as close as possible to the way he "sees" it is again a quick and helpful debugging aid. (The reader will no doubt at this point recall the oft-quoted Confucian proverb relating the comparative worth of a picture and 1K words of natural language).

EXAMPLES OF DATA STRUCTURE TRACES

A Fortran array trace

The author has constructed an interactive graphical trace for FORTRAN arrays [6]. The system permits the contents of up to four arrays (of one or two dimensions) to be displayed simultaneously on a TEKTRONIX [5] graphics screen. At any one time no more than a 10 x 10 portion of each array can appear, but the entire array may be displayed piecewise by choosing the appropriate options. The user may decide how the arrays selected for tracing are to be arranged on the screen by means of the graphics cursor.

Typical output from the array trace is shown in Figure 1. Four arrays are being traced, a double-precision array DUBBLE(50,30), an integer array IARRAY(50,30), a complex array COMP(50,30) and a real vector A(100). The current value of any element appears in the top half of the corresponding

"box", which is labelled by its row and column index; a new value appears in the lower half. If the value again were to change, this would result in the entry in the lower half being overwritten; option 'R'(Refresh) causes the screen to be redrawn with the latest values once more in the top half of each rectangle, thus obviating the possibility of overwriting values.

The system is implemented in the form of a pre-processor which transforms the user's FORTRAN program into a FORTRAN program containing the relevant calls on subroutines which perform the plotting. It is written in FORTRAN IV, and hence is fully portable. For further details the reader is referred to Schach [6]; here we are more concerned with the level of this trace.

The fact that each array is specifically labelled on the screen with its name as given by the user in his or her FORTRAN source code, and that each element bears the appropriate row and column number means that we are dealing with a high level trace; the output is in a format closely resembling the original high level language source program. But further, since the data structures (arrays) are displayed in terms of their rows and columns as the user visualizes them means that this is in fact a very high level trace.

A trace for the Pascal heap

The package HEAPTRACE [7] is a precompiler for Pascal programs which enables the user to trace the heap, selectively dumping dynamically created records in a high level format. Each field of the record is named, and its value given in a form as close as possible to the original source code. Each record to be traced is assigned a sequence number as it is created on the heap, and these numbers not only provide unique identification during program execution, but are used when tracing pointer (e.g. POINTER P POINTS TO NODE — 456).

Figure 2 shows the output produced when HEAPTRACE was applied to the example on pages 44-46 of the Pascal User Manual [4]. When HEAPTRACE intervenes, the user is informed of the line number in his original Pascal program. When a node is dumped, its type identifier (in this case *person*) as named by the user is given. Then each component field is named, and its value given.

Integers, reals and characters are output in the conventional way, while the values of types defined by enumeration (including Boolean) are explicitly printed out. The user is informed if a field is of type set, and the contents of the set (if any) are printed out as above. For arrays, the indices and values of the first and last elements are printed. For example, if the program includes the declaration

```
specimen : array [ -4..9,false..true,34..45] of real,
```

then the output from HEAPTRACE would include

```
specimen : array
  specimen [-4,false,34] : 72.96
  specimen [9,true,45] : -7.52
```

For packed arrays of char, the first and last character strings are given.

A record field within a record is identified as such, and its fields are in turn indented a further four spaces (see *birth* and *ddate* in Figure 2). Indentation is also used for tagfields (*ms* in the figure), and for the fields of variant parts.

But despite the fact that the underlying structure of each record is reflected through indentation, HEAPTRACE is not a very high level trace. The reason is that while the contents of any one individual record of the data structure are provided, the user is not given the overall picture of his data structure in the format in which he conceptualizes it. In terms of the terminology of this paper, HEAPTRACE is thus a strictly high level trace.

An interactive trace for the Pascal Heap

The Pascal pre-processor HEAPTRACE described in the section above provides information as to the contents of selected Pascal dynamic records. GRAPHTRACE, on the other hand, allows the user to "see" the overall shape of his data structures at a graphics screen. Each record is represented as a node, and the nodes are interconnected by directed links representing the pointers of the data structure. However, the contents of the individual fields do not appear on the screen.

Typical GRAPHTRACE output is shown in Figure 3a. Each record to be traced is assigned a number, as before. There are nodes of two distinct types, represented here by circles and diamonds respectively, corresponding to the two record types in the user's program. The three types of pointer defined in the program being traced are also distinguishable. The user is provided with a key to enable him to match the node or pointer to the name he gave it in his Pascal source.

The user specifies which subset of the records he wishes to see displayed in the current plot (or all of them), and may also select the root of the graph to be drawn. He may specify that certain links are to be drawn horizontally or drawn vertically, or are to be ignored. On the other hand, if he is unsure of the exact shape of his data structure he may simply allow the package to draw it as it sees fit, and the user will then refine his instructions stepwise, indicating that a particular record is the root of a tree, and so on. Figure 3b shows the same data structure as before, but with node 10 specified to be the root of a tree.

Figures 3c and 3d again show the identical data structure, but the user has specified various choices of directions for drawing his three types of pointer (or has chosen to suppress one type).

The method used for displaying the graph is used on the UDRL algorithm of Becker and Schach[1], but modified to allow for links which are neither horizontal nor vertical to be superimposed on the basic structure. For further details see Getz et al [2].

With regard to the level of GRAPHTRACE, the fact that the data structures may be displayed precisely as the user conceives them means that this is a very high level trace. But at the same time, GRAPHTRACE is not a high level trace.

The reason is that the only Pascal variable names with which GRAPHTRACE is concerned are the names of the types of the records, and of the pointers. As mentioned above, the contents of the nodes themselves are not displayed on the screen. Thus, strictly speaking, GRAPHTRACE is not a high level trace. However, GRAPHTRACE does allow the user to interact with the HEAPTRACE routines at any time. The user is permitted to dump selectively the contents of the heap at a printer or VDU screen.

By combining GRAPHTRACE with HEAPTRACE we thus have both a high level and a very high level trace, which together provide the user with maximal information for tracing the Pascal heap.

CONCLUSION

High level languages like Pascal or Ada[7] support powerful and flexible variable types, thus permitting highly sophisticated and complex data structures to be constructed. The price that must be paid for this is that if there is an error within a complicated data structure, then it is often not easy to detect and correct it. Use of a low level trace is entirely unsatisfactory, and at the very least a high level trace should be employed, and preferably a very high level one. Such traces do not exist for the new language Ada, but as soon as Ada compilers become available it would be advantageous for high level and very high level traces to be written.

```
 OPTIONS? <D,I,N OR R>     LINE NUMBER          38        38        38        52        52        52
>N                                              52
```

DUBBLE

	25	26	27
41	.ØØØØØ	.ØØØØØ	.ØØØØØ
42	.ØØØØØ	.ØØØØØ	.ØØØØØ
43	.ØØØØØ	.ØØØØØ	.ØØØØØ

IARRAY

	1	2	3
2	21	22	23
3	31	32	33
4	41 41	Ø 42	Ø 43

COMP

	3	4	5	6				
1	13.ØØØ	113.ØØ	14.ØØØ	114.ØØ	15.ØØ	115.ØØ	16.ØØØ	116.ØØ
2	23.ØØØ	123.ØØ	24.ØØØ	124.ØØ	25.ØØ	125.ØØ	26.ØØØ	126.ØØ
3	33.ØØØ	133.ØØ	34.ØØØ	134.ØØ	35.ØØ	135.ØØ	36.ØØØ	136.ØØ
4	.ØØØØØ 43.ØØØ	.ØØØØØ 143.ØØ	.ØØØØØ 44.ØØØ	.ØØØØØ 144.ØØ	.ØØØØØ 45.ØØØ	.ØØØØØ 145.ØØ	.ØØØØØ 46.ØØØ	.ØØØØØ 146.ØØ

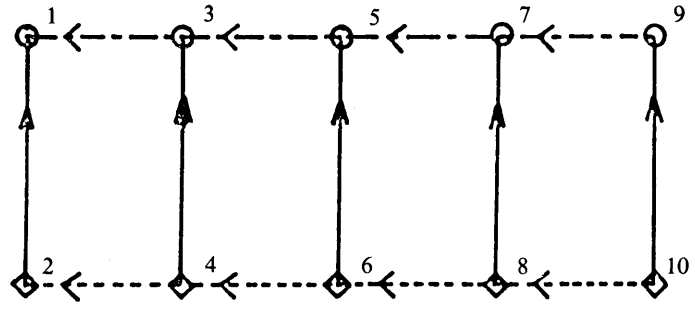
A

	1
3	3.ØØØØ
4	4.ØØØØ
5	5.ØØØØ
6	6.ØØØØ

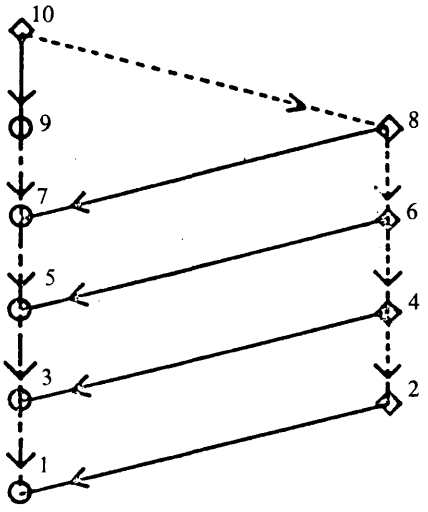
Figure 1: Sample FORTRAN Array Trace Output

```
***** HEAPTRACE CALLED AT LINE   44

NODE #  1     TYPE -  PERSON

 NAME          :     RECORD
     FIRST          :    ARRAY
                   STRING : EDWARD
     LAST           :    ARRAY
                   STRING : WOODYARD
 SS            :  845680539
 SEX           :  MALE
 BIRTH         :     RECORD
     MO             :  AUG
     DAY            :  30
     YEAR           :  1941
 DEPTDS        :  1
   MS             :  SINGLE
     INDEPDT        :  TRUE


NODE #  2     TYPE -  PERSON

 NAME          :     RECORD
     FIRST          :    ARRAY
                   STRING : NICOLAS
     LAST           :    ARRAY
                   STRING : ROBERTSMAN
 SS            :  627259003
 SEX           :  MALE
 BIRTH         :     RECORD
     MO             :  MAR
     DAY            :  15
     YEAR           :  1932
 DEPTDS        :  4
   MS             :  DIVORCED
     DDATE          :     RECORD
         MO             :  FEB
         DAY            :  23
         YEAR           :  1972
     FIRSTD         :  FALSE
```

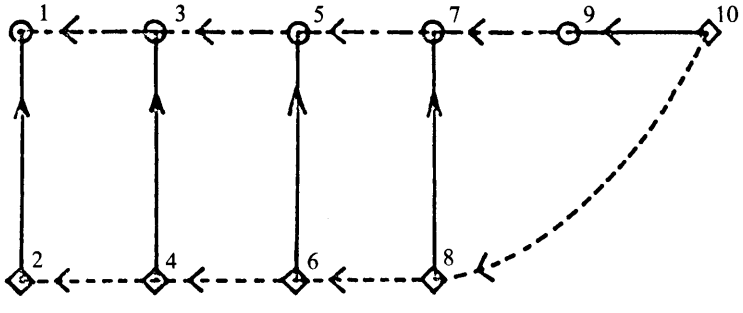
Figure 2: Sample HEAPTRACE Output.



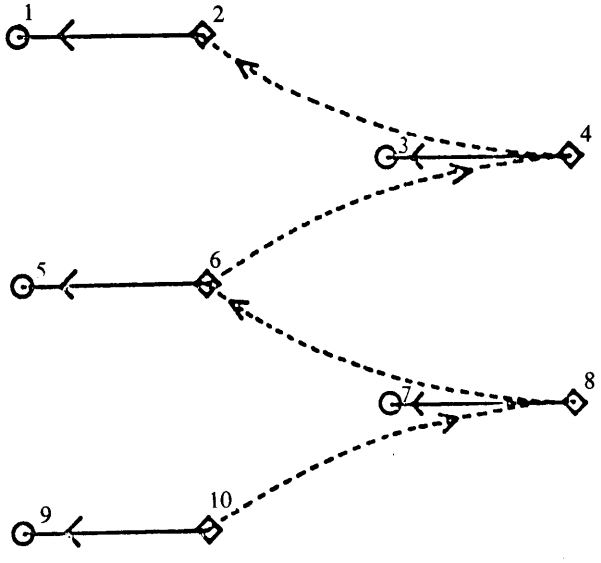
(a)



(b)



(c)



(d)

Figure 3: The same data structure drawn by GRAPHTRACE with various choices of the direction for the 3 types of pointers.

REFERENCES

[1] R. I. Becker & S. R. Schach, Drawing labelled binary graphs on a grid. Submitted to Networks.

[2] S. L. Getz, G. Kalligiannis & S. R. Schach, A very high-level interactive graphical trace for the Pascal heap. To appear in IEE Trans. on Software Engineering.

[3] J. D. Ichbiah et al, Preliminary Ada reference manua, ACM SIGPLAN Notices, Vol. 14, Number 6, 1979.

[4] K. Jensen & N. Wirth *Pascal User Manual and Report,* 2nd Edition, Lecutre notes in Computer Science 18, Springer-Verlag, Berlin, 1974.

[5] PLOT-10 Terminal Control System, User's Manual No. 062-1474-00, Tektronix, Inc., Beaverton, Oregon, 1974.

[6] S. R. Schach, An interactive graphical array trace. Quaest. Inform. Vol. 2, No. 1, pp 23-26.

[7] S. R. Schach, A portable trace of the Pascal heap, Software — Practice and Experience, Vol. 10, 1980, pp. 421-426.

[8] D. A. Watt & W. Findlay, A Pascal diagnostics system. In: *Pascal: The Language and its Implementation,* D. W. Barron, Editor, Wiley, Chichester, 1981.

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to: Prof. G. Wiechers at:

Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM,* **9**, 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of context-free Languages,* McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Quaestiones Informaticae

Contents/Inhoud

*Presented at the second South African Computer Symposium held on 28th and 29th October, 1981.

Printed by Studio Press, Johannesburg