Quaestiones Informaticae

An official publication of the Computer Society of South Africa
'\nAmptelike 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

Editorial Advisory Board

PROFESSOR D. W. BARRON
Department of Mathematics
The University
Southampton S09 5NH
England

PROFESSOR K. GREGGOR
Computer Centre
University of Port Elizabeth
Port Elizabeth 6001
South Africa

PROFESSOR K. MACGREGOR
Department of Computer Science
University of Cape Town
Private Bag
Rondebosch 7700
South Africa

PROFESSOR M. H. WILLIAMS
Department of Computer Science
Herriot-Watt University
Edinburgh
Scotland

MR P. P. ROETS
NRIMS
CSIR
P.O. Box 395
Pretoria 0001
South Africa

PROFESSOR S. H. VON SOLMS
Department of Computer Science
Rand Afrikaans University
Auckland Park
Johannesburg 2001
South Africa

DR H. MESSERSCHMIDT
IBM South Africa
P.O. Box 1419
Johannesburg 2000

MR P. C. PIROW
Graduate School of Business Administration
University of the Witwatersrand
P.O. Box 31170
Braamfontein 2017
South Africa

Subscriptions

Annual subscriptions are as follows:

<table>
<thead>
<tr>
<th></th>
<th>SA</th>
<th>US</th>
<th>UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals</td>
<td>R6</td>
<td>$7</td>
<td>£3.0</td>
</tr>
<tr>
<td>Institutions</td>
<td>R12</td>
<td>$14</td>
<td>£6.0</td>
</tr>
</tbody>
</table>

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

S. R. Schach
Computer Science Department, University of Cape Town

Abstract
The use of a high-level language generally masks the essential difference between the programmer’s conception of a data structure and its computer implementation.

However, in the course of debugging a program written in a high-level language, the user is sometimes forced to try to understand the low-level implementation of his data structure, a task that may well be beyond his ability.

In an attempt to obviate this problem a trace package has been written which allows a user at a TEKTRONIX Graphics Terminal to “see” arrays as two-dimensional data structures with rows and columns, and to examine interactively the contents of up to four arrays of any FORTRAN IV program while it is being executed.

The package consists of a precompiler and a set of graphical routines, both designed to be fully portable. It has been used to debug programs with large arrays (which a standard line-by-line variables trace cannot comfortably handle).

Introduction
One of the advantages of using a high-level language is that the essential difference between a data structure and its computer implementation is transparent to the user.

Consider, for example, a binary tree. We all have a mental image of an archetypal binary tree; the fact that this image differs radically from, say, its Pascal [1] implementation as a linked list, is irrelevant to most users.

Similarly, to the majority of FORTRAN programmers a matrix is a two-dimensional data structure with rows and columns. Again, its computer implementation (as a one-dimensional linear list) is generally of no concern to the user.

However, if a program containing a data structure does not work, the user may be forced to acquire a detailed knowledge of the computer implementation of that data structure.

For instance, suppose a FORTRAN program which uses arrays contains bugs. One approach is to insert judiciously selected WRITE statements at appropriate places in the code. However, this presupposes that the user has already discovered sufficient information as to the nature of his error so as to be able to determine with a reasonably high degree of accuracy where in the matrix (and where in the code) the problem lies.

Some systems, e.g. UNIVAC Re-entrant FORTRAN [2], allow a “trace” facility whereby, for example, a 50 x 30 matrix can be traced, with the machine printing out every one of the 1500 elements of the array as each line of the source code is executed. Clearly this approach is unworkable for large FORTRAN arrays.

The last resort, namely investigation of an assembler listing or a machine-code core dump, requires an intimate knowledge of the precise way the relevant data structures are handled, a level of expertise many programmers simply do not have.

In the course of descending from a high-level language to its low-level implementation, the protection from the implementation details of data structures by the high-level language is destroyed to such an extent that this form of debugging is out of reach of the average computer user. In addition, expertise with one machine may be irrelevant when using the computer of a different manufacturer because of the non-Portability of assembler and machine codes.

To my mind, the “obvious” way to trace a data structure is graphically. One must allow the user to see his data structure on a VDU screen in what is for him the most natural format, namely in the shape of the mental picture with which he personally visualizes that structure. For instance, if he is working with a binary tree, then the trace must cause a binary tree to appear on the screen; if a matrix, he must see in front of him a two-dimensional structure with rows and columns. Furthermore, as such mental pictures clearly vary from individual to individual, the system must be sufficiently powerful and flexible to allow the user to specify exactly what he requires. To give a trivial example, it must permit representations of trees growing downwards (as computer scientists believe they do) as well as upwards (the way botanists claim trees behave).

My eventual aim is the construction of a package to trace arbitrary Pascal data structures by means of a display on a graphics terminal. In this paper I describe the first stage of the project, namely a graphical trace for FORTRAN arrays, consisting of a totally portable package written in strict FORTRAN IV [3], together with the standard TEKTRONIX graphics routines [4].

Input to the Precompiler
The system consists of a precompiler and a set of graphical routines, both designed to be fully portable. Input to the precompiler consists of the user’s FORTRAN program interspersed with commands which control the tracing options. In addition the user must provide (in the form of data cards) the names of the up to four arrays that he wishes to be traced simultaneously.

The trace directives, punched from column 1, must be preceded by an asterisk (see Figure 1). The three commands currently implemented are:

- **ON** — turn on the trace until further notice
- **OFF** — turn off the trace until further notice
- **SNAPSHOT** — allow the examination of the current contents of all arrays being traced.

<table>
<thead>
<tr>
<th>COMPLEX COMP</th>
<th>DOUBLE PRECISION DUBBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ON</em> DO 41 I = 1,1000</td>
<td></td>
</tr>
<tr>
<td>2100 FORMAT (02,15)</td>
<td></td>
</tr>
<tr>
<td>IF(I.GE.3) OR (I.LE.5) WRITE(5,2100)(IARRAY(K),K = 1,144)</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td><em>OFF</em> DO 42 J = 1,50</td>
<td></td>
</tr>
<tr>
<td>IF(I.GEQ.3) AND (I.LE.5) WRITE(5,2100)(IARRAY(K), K = 1,144)</td>
<td></td>
</tr>
<tr>
<td>DO 42 J = 1,30</td>
<td></td>
</tr>
<tr>
<td>JIVAL = 10**J + J</td>
<td></td>
</tr>
<tr>
<td>IARRAY(J) = JIVAL</td>
<td></td>
</tr>
<tr>
<td>DUBBLE(I,J) = DBLE(FLOAT(IVAL)*J)</td>
<td></td>
</tr>
<tr>
<td><em>ON</em> DO 42 CONTINUE</td>
<td></td>
</tr>
<tr>
<td><em>OFF</em> STOP</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 1
Sample Output

Before describing the details of the program, let us examine some typical output. As soon as the first *ON directive is encountered, the user is asked to specify which rows and columns of the array(s) to be traced he would like to view at that point (see Figure 2). Once this has been determined, the user must state how he would like the arrays to be displayed on the screen. He may select either the "standard" arrangement — each array occupying one quarter of the available area, or he may use the graphics cursor to delimit the screen as he desires. When the user is satisfied with the arrangement, the current values of the arrays being traced are inserted in the top half of the rectangles. Each time the contents of an array element currently on the screen are altered, the new value is inserted in the lower half of the rectangle. The current line number of the FORTRAN program is also drawn in (Figure 3).

The user has the option of refreshing the screen when one of three events occurs:

(i) A value is inserted into the top-left or bottom-right hand rectangle of an array as it currently appears on the screen.
(ii) The whole array is altered (either by a READ statement, or by a CALL to a subroutine with the entire array as an argument).
(iii) The screen area for indicating the current line number is filled (there is room for 30 line numbers).

Whenever one of the above takes place, the user can exercise one or more of the following options:

(a) Do nothing.
(b) Refresh the screen i.e. erase it, redraw the frames and insert the current values in the top half of the rectangles.
(c) Select different areas of the arrays for viewing.
(d) Alter the arrangement of the arrays on the screen.

At no time can more than a 10 x 10 portion of any one array be displayed on the screen, but the user can view the entire array piecewise by choosing the appropriate options. This is particularly valuable when used in conjunction with the *SNAPSHOT feature — inserting this command at the point where the program appears to go wrong allows the user to examine the contents of all arrays to find their current values. Thereafter, the program reverts to normal execution.

A problem with using the package is that of distinguishing the extra output which it produces from that produced by the

![Figure 2](image-url)

**ARRAY**: COMP
- **TYPE**: COMPLEX
- **MAXIMUM DISPLAYABLE DIMENSIONS**: 10 x 5
- **CURRENT X-LIMITS**
  - LOWER: 1
  - UPPER: 50
- **TOO LARGE, LIMIT IS 10 ELEMENTS.**
- **GIVE NEW LOWER X-LIMIT**
  - > 1
- **GIVE NEW UPPER X-LIMIT**
  - > 4
- **CURRENT X-LIMITS**
  - LOWER: 1
  - UPPER: 4
- **IS THIS SATISFACTORY?**: <Y OR N>
  - > Y
- **CURRENT Y-LIMITS**
  - LOWER: 1
  - UPPER: 30
- **TOO LARGE, LIMIT IS 5 ELEMENTS.**
- **GIVE NEW LOWER Y-LIMIT**
  - > 3
- **GIVE NEW UPPER Y-LIMIT**
  - > 6
- **CURRENT Y-LIMITS**
  - LOWER: 3
  - UPPER: 6
- **IS THIS SATISFACTORY?**: <Y OR N>
  - > Y

![Figure 3](image-url)

**Figure 3**

**Options**? (D, I, N OR R) **LINE NUMBER**

- D
- I
- N
- R

- **OPTIONS?**: (D, I, N OR R)
- **LINE NUMBER**: 38
- **ARRAY**: COMP
  - **TYPE**: COMPLEX
  - **MAXIMUM DISPLAYABLE DIMENSIONS**: 10 x 5
  - **CURRENT X-LIMITS**
    - LOWER: 1
    - UPPER: 50
  - **TOO LARGE, LIMIT IS 10 ELEMENTS.**
  - **GIVE NEW LOWER X-LIMIT**
    - > 1
  - **GIVE NEW UPPER X-LIMIT**
    - > 4
  - **CURRENT X-LIMITS**
    - LOWER: 1
    - UPPER: 4
  - **IS THIS SATISFACTORY?**: <Y OR N>
    - > Y
  - **CURRENT Y-LIMITS**
    - LOWER: 1
    - UPPER: 30
  - **TOO LARGE, LIMIT IS 5 ELEMENTS.**
  - **GIVE NEW LOWER Y-LIMIT**
    - > 3
  - **GIVE NEW UPPER Y-LIMIT**
    - > 6
  - **CURRENT Y-LIMITS**
    - LOWER: 3
    - UPPER: 6
  - **IS THIS SATISFACTORY?**: <Y OR N>
    - > Y

**Figure 2**

At no time can more than a 10 x 10 portion of any one array be displayed on the screen, but the user can view the entire array piecewise by choosing the appropriate options. This is particularly valuable when used in conjunction with the *SNAPSHOT feature — inserting this command at the point where the program appears to go wrong allows the user to examine the contents of all arrays to find their current values. Thereafter, the program reverts to normal execution.

A problem with using the package is that of distinguishing the extra output which it produces from that produced by the

**Figure 3**

- **DUBBLE**
  - **ARRAY**
    - **1**: 21 22 23
    - **2**: 31 32 33
    - **3**: 41 42 43

- **COMP**
  - **1**: 13.00 113.00 14.00 114.00 15.00 115.00 16.00 116.00
  - **2**: 23.00 123.00 24.00 124.00 25.00 125.00 26.00 126.00
  - **3**: .0000 .0000 .0000 .0000 .0000 .0000 .0000 .0000 .0000
  - **4**: .0000 .0000 .0000 .0000 .0000 .0000 .0000 .0000 .0000

- **A**
  - **1**: 3.0000
  - **2**: 4.0000
  - **3**: 5.0000
  - **4**: 6.0000
LOGICAL GRMXON
COMPLEX COMP
DOUBLE PRECISION DUBBLE
DIMENSION A(1000),IARRAY(50,30),COMP(50,30),DUBBLE(50,30)

C***** TRACE ON *****
GRMXON=.TRUE.
CALL GRMXO1( 7,4,1, 1000, 1,2, A, IARRAY, COMP, DUBBLE, 1HA, 1H ,1H ,1H ,1H ,1H ,1 , IGRMXO) 
CALL GRMXO1( 7,4,1, 50, 30,1, A, IARRAY, COMP, DUBBLE, 1HA, 1H, 1HR, 1HR, 1HA, IHY, 6, IGRMXO) 
CALL GRMXO1( 7,4,3, 50, 30,4, A, IARRAY, COMP, DUBBLE, 1HC, 1HO, 1HM, 1HP, 1H ,1H ,4, IGRMXO) 
CALL GRMXO1( 7,4,4, 50, 30,5, A, IARRAY, COMP, DUBBLE, 1HD, 1HU, 1HB, 1HB, 1HL, 1HE, 6, IGRMXO) 
DO 77777 I= 1, 1000
RENUMBD
FORMAT(12,15)
IF(.NOT. I((I.EQ.3).OR.(I.EQ.7)) )GOTO 77777
WRITE(5,2100)(IARRAY(K), K = 1,144)
77777 CONTINUE
CALL GRMXO2(GRMXON, 24,1,A ,IARRAY,COMP ,DUBBLE,1,1) 
C ***** TRACE OFF *****
GRMXON=.FALSE.
DO 42 I= 1,50
IF(.NOT. I((I.GE.3).OR.(I.LE.5)) )GOTO 77779
CALL GRMXO3(IGRMXO, 32,5 )
WRITE (5,2100)(IARRAY(K),K= 1,144)
77779 CONTINUE
CALL GRMXO2 (GRMXON, 38,2,A ,IARRAY,COMP ,DUBBLE,1,J) 
CALL GRMXO2 (GRMXON, 40,4,A ,IARRAY,COMP ,DUBBLE,1,J) 
C ***** TRACE ON *****
GRMXON=.TRUE.
CALL GRMXO1( 43,4,1, 1000, 1,2, A, IARRAY, COMP, DUBBLE, 1HA, 1H ,1H ,1H ,1H ,1H ,1 , IGRMXO) 
CALL GRMXO1( 43,4,2, 50, 30,1, A, IARRAY, COMP, DUBBLE, 1HA, 1HR, 1HR, 1HA, IHY, 6, IGRMXO) 
CALL GRMXO1( 43,4,3, 50, 30,4, A, IARRAY, COMP, DUBBLE, 1HC, 1HO, 1HM, 1HP, 1H ,1H ,4, IGRMXO) 
CALL GRMXO1( 43,4,4, 50, 30,5, A, IARRAY, COMP, DUBBLE, 1HD, 1HU, 1HB, 1HB, 1HL, 1HE, 6, IGRMXO) 
CALL GRMXO2 (GRMXON, 52,3,A ,IARRAY,COMP ,DUBBLE,1,J) 
42 CONTINUE
GRMXON=.FALSE.
C ***** TRACE OFF *****
STOP
DATA IGRMXO/0/,GRMXON/.FALSE./
END

FIGURE 4
user's WRITE statements, especially on the limited area of a terminal screen. There is no point in letting the precompiler direct all output to a file, because the program output is itself a valuable debugging aid when used in conjunction with the graphical routines. Instead, whenever a WRITE statement is encountered while tracing is in progress, the user is informed of the current line number as well as the number of the relevant output file. He then has the option of doing nothing (as in the case of output to be written to a tape or disk file), or of clearing the screen to receive the output. In the latter case a flag is set so that when another call to a graphing routine is encountered the screen is again cleared, the rectangles redrawn and the system is ready to insert the new value. The fact that the user must manually initiate both the clearing of the screen, as well as the re-instatement of the graphical trace, gives him time to make a copy of the current output using the hard-copy facility of the TEKTRONIX terminal.

The Precompiler — Implementation Details
We now examine the system in more detail. The precompiler processes the original text adding appropriate calls to graphing subroutines (output from which has already been described)
which are then included in the object program by the loader. In broad outline, the precompiler operates as follows:

(1) After encountering a *ON (trace on) command, it inserts the statement

\[ \text{GRMXON} = \text{.TRUE.} \]

followed by calls to the initializing routine GRMX01.

(2) Similarly, after a *OFF command, GRMXON is set to .FALSE.

(3) After every CALL, READ or assignment statement which alters the value of an element (or elements) of an array being traced, a call to the appropriate graphing routine is inserted in the user's FORTRAN program.

(4) A call to the subroutine GRMX03 (which informs the user of impending output) is inserted before every WRITE statement (see Figures 1 and 4).

At execution time, the calling of the graphing routines is controlled by two variables - the previously mentioned GRMXON which turns the trace on or off, and IGRMX0 which prevents both re-initialization, as well as calls to GRMX03 before tracing has started. Figure 4 illustrates this point.

Certain sequences of FORTRAN statements are easier to precompile than others. Suppose an array \( A(1000) \) is to be traced. Then if the precompiler encounters an instruction such as:

\[ A(I) = 1 \]

it merely follows it by the statement

\[ \text{CALL GRAPHTRACE} \]

(where GRAPHTRACE stands for the appropriate subroutine with its argument list).

However, an instruction like

\[ \text{IF}(A(I) < 3.4) \ A(I) = 3.4 \]

must be replaced by the sequence

\[ \text{IF}(.\text{NOT.}(A(I) < 3.4)) \ GOTO \ 77777 \]
\[ A(I) = 3.4 \]
\[ \text{CALL GRAPHTRACE} \]
\[ 77777 \text{ CONTINUE} \]

Similarly, the DO loop

\[ \text{DO} \ 14 \ I = 1,1000 \]
\[ 14 \ A(I) = 1 \]

must be altered to read

\[ \text{DO} \ 77778 \ I = 1,1000 \]
\[ 77778 \text{ CONTINUE} \]

Efficiency

The precompiler in its current form is somewhat slow — on average it requires about one second of CPU time per ten lines of input program. In addition to this, FORTRAN I/O is notoriously unhurried, and each of the two passes requires the complete program to be read from, and written to, disk storage. This slowness is largely due to the stipulation of true portability. For example, every line of code must be read in 80Al format. Furthermore, much time is spent in circumventing possible problems with overflow resulting from arithmetic performed on Hollerith variables. Each test for equality requires a subroutine to be called — use of the standard logical operator .EQ. might easily cause overflow on some machines.

The user's program is read into core one FORTRAN statement at a time. The statement (which could be up to twenty lines long) is parsed, modified if necessary, and then written to mass storage. Allowing the user's entire program to reside in core would certainly obviate the need for a second pass, thereby halving the I/O time, but the use of Al format requires eighty words of core per line of program.

In addition, further space would be required for tables of pointers for insertions of calls into the program.

The package has been implemented on the UNIVAC 1106 computer at the University of Cape Town, where the precompiler and the graphical routines each occupy about 25K words, but with overlaying and segmentation this can be reduced to under 20K. If one wants the user's entire program to reside in core during precompilation, this would require approximately an additional 10K words of core per 100 lines of input program.

Applications

This system has been successfully used to debug various FORTRAN programs incorporating large arrays. In most cases the errors occurred when the last few elements of the arrays were being computed. By restricting the trace to the bottom right-hand corner of each matrix a considerable amount of computer time was saved; a non-graphical line-by-line trace would have generated thousands of lines of output before reaching this area of the arrays.

The package has been used to debug itself. This might represent another step forward towards the dream of computers writing their own programs!

References

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 I, 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:


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

The Design Objectives of Quadlisp* ........................................... 3
  S W Postma
A CSP Description of some Parallel Sorting Algorithms* ............. 7
  M H Linck
The Design and Microprogrammed Implementation of a Structured
Language Machine.............................................................. 13
  G R Finnie
Micro-Code Implementation of Language Interpreters* ................. 19
  P P Roets
An Interactive Graphical Array Trace* ................................... 23
  S R Schach
An Efficient Implementation of an Algorithm for Min-Max
  Tree Partitioning............................................................. 27
  Ronald I Becker, Yehoshua Perl, Stephen R Schach
The Relative Merits of Two Organisational Behaviour Models for Struct-
turing a Management Information System................................ 31
  Peter Pirow

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