QI QUAESTIONES INFORMATICAE

Vol. 3 No. 3

August, 1985

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUAESTIONES INFORMATICAE

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R10	$7	£5
Institutions	R15	$14	£10

Circulation and Production Manager

MR. C. S. M. Mueller
Department of Computer Science
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg 2001

Quaestiones Informaticae is prepared by the Computer Centre and Central Graphics Unit, and printed by the Central Printing Unit, all of the University of the Witwatersrand, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial Note

I am happy to present the current issue of Quaestiones Informaticae. As indicated in the last issue, Vol.3 No.2, it was necessary to look for a new method to produce the journal. Thanks to the efforts of Prof. Judy Bishop we have found a way to continue at an acceptable cost, and the result is in your hands. Wits University's Computer Centre will do the 'typesetting' using a laser printer. Judy will use Computer Science students to do the proof reading, and Wits will also do the printing. On behalf of myself, the Computer Society of SA and the SA Institute of Computer Scientists, I want to express my appreciation for all their efforts.

Conrad Mueller, of Wits' Computer Science department has agreed to become the circulation manager for QI. He will handle the business side of our publication. A hearty welcome to him!

G. WIECHERS
Editor.

STOP PRESS

QI is now on the official supplementary list of journals considered by the National Department of Education for subsidy purposes.

High-level Debugging Systems for Pascal: Interpreter versus Compiler

MICHAEL HIRSCH
*Department of Applied Mathematics,
The Weizemann Institute of Science,
Rehovot,
76100 Israel*

STEPHEN R. SCHACH
*Department of Computer Science, Vanderbilt University,
Nashville, Tennessee 97235, U.S.A.*

WILLEM R. VAN BILJON
*Computer Science Division, NRIMS, CSIR
Pretoria, 7700 South Africa*

ABSTRACT

Two approaches to high-level debugging systems for Pascal are compared, namely the use of a debugging precompiler as against a debugging interpreter. A description is given of a high-level Pascal debugging interpreter which has the power of the precompiler approach, but with none of the disadvantages.

INTRODUCTION

A *high-level* debugger is one whose output reflects the high-level language in which the program being debugged was written. Three systems that support high-level debugging of Pascal programs are the PASCAL-I program development environment[1], HEAPTRACE, a portable trace for the Pascal heap[2], and the Pascal compiler of Watt and Findlay[3] which provides forward and retrospective trace facilities and a post-mortem dump. A *very high-level* debugger displays a user's data structure in a format as close as possible to the way he conceptualizes that data structure, be it a tree, or a doubly linked list; GRAPHTRACE, a very high-level trace for the Pascal heap[4], is an example of a very high-level debugger for Pascal.

The systems of Cichelli, and of Watt and Findlay, have a drawback in that neither supports dynamic variables. PASCAL-I is an interactive program development system for Pascal-S[5], a subset of Pascal that excludes dynamic (heap) variables, while Watt and Findlay's high-level post mortem dump includes only static variables. A further disadvantage of PASCAL-I is that in order to to make use of of the wide variety of powerful features provided, the user has to learn a fairly extensive command language.

While GRAPHTRACE and HEAPTRACE specifically provide high-level debugging of the heap, they suffer from major limitations as a consequence of the fact they are written in standard Pascal (in order to achieve true portability) and as a result of the implementation strategies chosen. In both packages the user inserts debugging commands (in the form of comments) within his source code, which is passed through a precompiler. The Pascal output from the precompiler is then compiled and linked in the usual way. Tracing of the heap is achieved as follows: whenever the user creates (via the **new** command) a record to be traced, the system creates a "hyper-record" which is two-way linked to the corresponding record. The hyper-records themselves are two-way linked. In this way the system can access any user defined record, even if the user has mismanaged his pointers.

There are four major problems with HEAPTRACE and GRAPHTRACE.

1. No information can be gained from either system after a run-time failure as, in order to achieve portability, both systems are dependent on the Pascal run-time environment.
2. No interactive modification of the trace commands is possible. The commands are inserted into the source code at places where the user surmises that diagnostic information would prove helpful; if the user now wishes to obtain information from elsewhere in the program, he must go through the edit-precompile-compile-link cycle.
3. Both systems disable the (non-standard) garbage disposal procedure **release**, because the portability constraint requires the hyper-records to be stored on the heap itself, and hence no interference with the structure of the heap can be permitted.
4. Finally, the problems of uninitialized variables in general (and pointers in particular) require the user to initialize all fields of records to be traced.

In this note we describe a debugging system that solves problems 1) to 4) above, while still providing high-level debugging of all variables, including heap variables. In addition, the user need not learn a complex command language; debugging commands are almost all specified in the form of Pascal statements. Finally, the system is fully portable, being written in standard Pascal.

SYSTEM OVERVIEW

The debugging interpreter supports all the high-level heap debugging features of HEAPTRACE[2], as well as conventional high-level debugging tools. These include a line-by-line execution trace in which each source line is printed before it is executed, and a control flow trace (at each branch in the program the source line is printed, and information as to which branch was taken is given). There is also a variables trace; for each l-value, the appropriate source line, the scope of the variable, its name, old value and new value are printed.

The system consists of the portable P compiler[6], minimally modified, together with a debugging P-code interpreter. The compiler generates P-code from the user's Pascal source code. The source code, P-code, and compiler symbol table are all passed to the interpreter. Hyper-records are maintained in a similar fashion to HEAPTRACE and GRAPHTRACE, but on a separate heap. The user interacts with the system at run-time to enter debugging statements expressed in Pascal.

This method solves all four of the above problems. Firstly since the interpreter flags all run-time errors when they occur, the user may enter any debugging statements after his program has crashed, including the equivalent of requesting a selective post-mortem dump. The debugging statements can be entered either in the source code, or interactively at run-time, or both, thus solving problem two. The separate heap for the hyper-records solves the third problem; if a record is deleted from the program heap via a **release** command, the heap of hyper-records is appropriately modified. Finally, the interpreter is able to distinguish uninitialized scalar variables thus enabling the user to trace or to dump uninitialized variables.

The system has no explicit debugging language. It is assumed that a programmer debugging a Pascal program is already familiar with Pascal, and thus it is the obvious command language. Typical commands are:

```
                    writeln(employeeptr↑.socialsecuritynumber);
```

or (to change the values of two variables)

```
                    begin
                              y2:=1;
                              y3:=2

                    end;
```

However, it was found that Pascal is inadequate for certain debugging functions, especially examining the heap. The decision was therefore taken to augment Pascal by ten additional standard procedures, such as **breakpoint** (to effect a suspension of execution at a desired point), and **heapdump**.

IMPLEMENTATION NOTES

The system effectively consists of three components, the compiler, the interpreter, and the debugger. The compiler generates P-code (for a hypothetical stack computer) from the user's source program when called by the interpreter, and from the debugging commands when called by the debugger. The interpreter simulates the hypothetical stack computer. The debugger is invoked by the interpreter whenever a run-time exception occurs.

The P compiler was extended to accept the ten additional standard procedures, and to generate two additional stack computer instructions, namely LNO (line number), generated at the start of each statement and after each condition, but before the conditional jump, and CDP (call debugging procedure), generated by each of the additional standard procedures. CDP is analogous to CSP (call standard procedure). Finally changes were made so that the system runs in compile-and-go fashion.

The interpreter is based on the hypothetical stack computer model. Extensions were made to this model to allow for a simple tagged architecture, to monitor the heap, and to interpret the added instructions.

The architecture of the stack computer has three main stores, namely the data store, the code store, and the hyper-heap (where the hyper-records are stored), as well as six registers. The registers are Stack Pointer (SP), Program Counter (PC), Markstack Pointer (MP), Extremestack Pointer, New Pointer and Line Number. The SP, PC and MP registers are implemented as value parameters of the interpreter, so that recursive calls on the interpreter implicitly permit the enviroment to be saved.

The data store is implemented as an array of variant records, with the type of that word of store (integer, character, etc.) as tag field. Two Boolean flags are provided, one to denote if the variable is undefined, the other to indicate whether the word in question is being traced. The data store contains the stack growing up from 0, and the heap growing down. The code store is maintained separately from the data store.

The hyper-heap is implemented as an array of hyper-records. The fields of a hyper-record include a pointer to the start of the corresponding structure on the heap (i.e. in the data store), and a pointer to the record in the symbol table containing the type information for that structure.

The task of the debugger is exception handling. It reports the exception to the user, then reconstructs the display from the static link, and calls the compiler to solicit a Pascal statement from the user. Finally, it calls the interpreter recursively to execute this statement. If possible, execution of the user's program is then resumed.

RESULTS AND CONCLUSIONS

The system has demonstrated that it is possible to avoid the major drawbacks of Pascal high-level debugging systems based on precompilers by using an interpreter. The system is portable, and uses Pascal as command language, thereby obviating the need for the user to learn a separate command language.

It must be pointed out that, despite this success, the debugger-interpreter has some disadvantages of its own. As a consequence of the decision to make only minimal changes to the P compiler, certain limitations have been forced on the system. Firstly since the P compiler treats enumeration types as integer constants, no type information is on hand when certain errors occur. Insufficient information is available to supply high-level debugging information for handling a 'value out of range' error (generated by instruction CHK) or a 'no case provided for value' error (generated by UJC); the relevant enumeration value cannot be displayed without the introduction of a third additional P-code.

More serious is the fact that the compiler does not support any tagged features, forcing items to be traced or debugged by address. When tracing an array, the address of the array compiles to the same address as its first element, causing only the first element to be traced. Similarly, when dumping structures, the largest possible structure starting at the given address is printed. A request for a dump of the first element of an array will thus result in the printing of the whole array. Analogous action is taken in the case of records.

However, these problems can be solved by writing a Pascal interpreter (or an interpreter for an abstract syntax tree representation of a Pascal program), rather than modifying an existing compiler. All data type and lexical information can then immediately be made available.

On the grounds of our experiences with precompiler-based Pascal debuggers as well as the debugging interpreter described above, it appears that a Pascal debugging interpreter can provide all the facilities of a precompiler-based debugger, but with none of the disadvantages. It remains to be seen whether this result is equally applicable to other programming languages.

ACKNOWLEDGEMENT

The work of Michael Hirsch and Willem van Biljon was supported (in part) by the South African Council for Scientific and Industrial Research, Pretoria.

REFERENCES

1. R. Cichelli, Pascal-I - Interactive, conversational Pascal-S, *Assoc. Comput. Mach. SIGPLAN Notices*, **15**(1), 34-44 (1980).

2. S. R. Schach, A portable trace for the Pascal heap, *Software - Practice and Experience*, **10**, 421-426(1980).

3. D. A. Watt and W. Findlay, A Pascal diagnostic system, in *Pascal: The Language and its Implementation* (Ed. D. H. Barron), Wiley, Chichester, 1981.

4. S. L. Getz, G. Kalligiannis and S. R. Schach, A very high-level interactive graphical trace for the Pascal heap, *IEEE Trans. Software Engineering*, **SE-9**, 179-185 (1983).

5. N. Wirth, Pascal-S: A subset and its implementation, in *Pascal: The Language and its Implementation* (Ed. D. H. Barron), Wiley, Chichester, 1981.

6. K. V. Nori, U. Ammann, K. Jensen, H. H. Nageli and Ch. Jacobi, Pascal-P implementation notes in *Pascal: The Language and its Implementation* (Ed. D. H. Barron), 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:
INFOPLAN
Private Bag 3002
Monument Park 0105

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 an abstract 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. BOHM, 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 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 to 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.

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.