Quaestiones Informaticae

Vol. 2 No. 1 November, 1982

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

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

NOTE FROM THE EDITOR

After an absence of two years we are happy to announce that we are now in a position to continue the publication of Quaestiones Informaticae. The first Volume of QI consists of three numbers, and appeared during the period June 1979 till March 1980 under the editorship of Prof Howard Williams. Because Prof Williams took up a post at the Herriot-Watt University in Edinburgh, he had to relinquish his position as editor. The Computer Society of South Africa, which sponsors the publication of QI, appointed me as editor, whereas Mr Peter Pirow took over the administration of the Journal. The editorial board functions under the auspices of the Publications Committee of the CSSA.

The current issue is Number 1 of Volume 2. It is planned to publish altogether three issues in the Volume, with most of the papers coming from the Second South African Computer Symposium on Research in Theory, Software and Hardware. This Symposium was held on 28th and 29th October, 1981. At present it appears that most of the material published in this Journal comes from papers read at conferences. We invite possible contributors to submit their work to QI, since only the vigorous support of researchers in the field of Computer Science and Information Systems will keep this publication alive.

G WIECHERS

November, 1983

The Design Objectives of Quadlisp

S. W. Postma
Department Computer Science & Information Systems, University of South Africa.

Abstract
Quadlisp is a symbol manipulation language for sophisticated users, and it is an improper extension of Lisp 1.5 (in the same sense that Pascal is an improper extension of Algol). Elements from Lisp thus are basic to the language, but other objectives such as data-typing and general principles of modern language design have also been incorporated. Pertinent aspects of these considerations are discussed, followed by a discussion of the pragmatic considerations. Pragmatics relate to the use of the language on machines, or just by people as a tool for developing programs.

1. Introduction

The design objectives of a language constitute not only a set of principles, guidelines and constraints that are applied during the development of the language, but also a set of criteria for judging the final product: the language in its environment. As usual in Computer Science, the objectives may be in conflict, and the designer has to consider trade-offs, which is one reason why Allen [1] says:

> 'Language design is not a pastime to be entered into lightly . . .'

The design objectives of Quadlisp [8] may be summarized:

> Quadlisp is a sophisticated language with extensive data types and control structures. It is an improper extension of Lisp 1.5, with a unity and integrity based on a small number of language principles.

The objectives that are summarized above are classified and discussed under:

1. *Usage:* Objectives relating to the areas of endeavour where the language may be used, and also relating to the type of programmer who will use the language.
2. *Lisp Philosophy:* The ideas and principles from Lisp 1.5 that form the basis of Quadlisp.
3. *General Principles* of language design: The objectives that concern language designers in general.
4. *Pragmatics:* Objectives relating to the implementation and use of the language.

2. Usage

2.1 The Quadlisp Programmer

Quadlisp is a sophisticated language for programmers who have mathematical maturity [4]. They should not only have programming experience, but also algorithmic maturity. The language is not designed to be a tool for the novice, and a solid background in Lisp 1.5 or in functional programming [5] is recommended to prospective users, as is an exposure to Dijkstrassian nondeterminism [3].

2.2 Application Areas:

Quadlisp is a language for symbol manipulation, and for the manipulation of symbolic objects, in particular programs. The particular areas of meta-programming to be addressed are:

1. Operational Semantics: The language must make it easy to derive a VDL interpreter for a given program from the VDL semantics of a given language. From a VDL interpreter it should be easy to derive a contour model of the program to be used for automated desk-checking and an empirical assessment of the program complexity.

2. Program Verification and Equivalence: The language must enable interactive program verification systems to be implemented and should allow for the investigation into the equivalence problem of specific programs.

3. Meta-Theory of Computing: The results of the meta-theory of computing [2] i.e. automata and formal language theory, must be expressable in the language for use by programmers in program development and for cases 1 and 2 above.

3. Lisp Philosophy

A sound principle of language design is that a new language should be based on a known language. The new language typically is an improper extension of the old language, i.e. the base language is not a proper subset of its extension (cf. Algol and Pascal). The principles taken over from the base language should however be clearly stated. Quadlisp then is based on the following aspects of Lisp (see e.g. [6]).

1. Programs in Quadlisp may be treated as data: a program (strictly speaking, a function call), and a function, may be the result of a computation.

2. The macroscopic semantics of the Quadlisp interpreter is defined in Quadlisp in the form of an EVAL/APPLY pair of functions with their subsidiary functions. The Read-Evaluate-Print cycle of interactive computation uses an interpreter which is an implementation of the EVAL/APPLY functions.

3. λ-Notation is used but not the λ-calculus.

4. Internal and external data structures are the same, in particular, files are kept in any one of the standard Quadlisp data structure forms.

5. All programs are of the Lisp type FSUBR in the sense that functions are called before the arguments are evaluated, and arguments are evaluated as late as possible or not at all.

4. General Principles of Language Design

A number of well-known computer scientists have given criteria for language design, and these criteria have been summarized and classified by [7]. The specific criteria that were used in the design of Quadlisp are:

1. A variety of data structures

2. A variety of control structures
3. Nondeterminism
4. Sound evaluation
5. Declarations
6. Compactness
7. Verification

4.1 A Variety of Data Structures

A variety of typed data structures should be available, together with appropriate functions to manipulate these structures. The user must be able to recursively define his own data structures in terms of the data structures that are available in the language.

The functions or operators on the data structures may be classified into the following set of not necessarily mutually exclusive classes. Furthermore the classification is indicative only and is not intended to be complete:

Function types:

1. Predicates: Existence eg. testing for type
   Comparison eg. equality, order, substructure
   Element eg. components, values
   Structures eg. isomorphism
2. Creation — static and dynamic
3. Destruction — removal of parts or whole structure
4. Combination — eg. set union, list catenation
5. Modification — eg. replacement of elements
6. Traversal
7. Selection: access to substructures and elements.

4.2 A Variety of Control Structures

A language should have control structures that are powerful enough to enable the programmer to do the ordinary types of control namely:

1. Conditional and case
2. Iteration: do-while, do-until
3. Recursion
4. Functions, subroutine and co-routine calls
5. Sequential and parallel processing

4.3 Nondeterminism

Nondeterminism is one way of allowing the programmer not to over-constrain his solution. The major techniques in nondeterminism are:

1. Backtracking — depth first
   — breadth first
2. Dijkstrassian guarded constructs.

4.4 Sound Evaluation

The values of function calls should be the same as in mathematics, and extensions to functions must be defined by rules and not by an ad hoc assignment of values to particular calls with extremal values of the arguments.

4.5 Declarations

Variables should be typed, and there are two aspects to this typing:

1. A variable may be data-typed, i.e. it can only assume values out of certain classes of data.
2. A variable may be control and scope-types, i.e. it can only be accessed in certain circumstances.

All Quadlisp variables are typed with respect to scope. No variables are typed with respect to the values that they may assume except in λ-expressions where the λ-variable may be typed.

4.6 Compactness: Unity and Integrity

The language must be compact, and gothic extensions must be eschewed: the user must feel that he knows and controls the language.

4.7 Verification

Programs in the language must be verifiable: this implies that the language must be formally defined and that it must be axiomatizable and that a proof theory can be given for the programs written in the language.

5. Pragmatics

Most of the points given here under pragmatics are actually covered in one of the criteria already given. The use of the language was a major concern throughout the design and it is therefore desirable to name the pragmatic concerns of the designer explicitly.

1. **Representation:** The data structures and the control structures must have clear, unambiguous, useful representations in the following ways:
1.1 Linear — for writing programs.
1.2 Graphic — for representing data structures.
1.3 Machine — for implementing the language, which of course, must be machine independent.
2. **Documentation:** The language must allow for:
2.1 Comments.
2.2 Assertions.
2.3 Program layout.
3. **Symbols:** Mathematical and programming symbols should be used in their customary meanings when possible. All symbols should have names, and symbols only used if they convey their intended meaning, e.g. for set intersection.
4. **Principles:** The language should have a clear set of principles — in addition to the Eval/Apply semantics — to allow human interpretation of programs. The principles are listed below and a full discussion is given elsewhere [9].
4.1 A classification of the data structures from the abstract point of view.
4.2 Three evaluation strategies are available:
Sequential evaluation.
Parallel evaluation.
Equi-simultaneous evaluation: similar to multiprogram ming but it is not deterministic.
4.3 Structures together with evaluation strategy yield the evaluation tactics, eg. depth-or breadth-first backtracking.
4.4 Typing and declarations of use and scopes of variables.
4.5 In a function call (or a functional, eg. the conditional) — eg. (F A_0 A_1 A_2) — the A_0,...A_n are called the proto-arguments. These proto-arguments yield the arguments when they are evaluated: evaluation is done as late as possible, depending on structure.
4.6 Control structure principles: these are:
Conditional: return the first available value.
Program form: all available values are returned.
Repeat while: repeat over the minimum number of component forms.
Do all until: repeat over the maximum number of component forms.
4.7 In all basic pairs of objects in Quadlisp, the left-hand side of pair controls access to the right-hand side of the pair. The hand side can be considered to be a guard, or an address, of the right-hand side.

6. Conclusion

The author would like to acknowledge his indebtedness and gratitude to Prof Gerrit Wiechers and Dr Nick Phillips who contributed much to Quadlisp achieving its objectives. Their timeous (and often harsh) criticisms had the effect that much exuberent growths were pruned from the language to give it its final compact and unified gestalt.

References

[1] J.R. Allen, *Anatomy of Lisp,* (McGraw-Hill, New York, 1978).

[2] J.M. Brady, *The Theory of computer Science,* (Chapman & Hall, London, 1977).

[3] E.W. Dijkstra, *A Discipline of Programming,* (Prentice-Hall, Englewood Cliffs, 1976).

[4] E.W. Dijkstra, American programming's plight, ACM Sigsoft SE notes, **6** (1981).

[5] P. Henderson, *Functional Programming,* (Prentice-Hall, Englewood Cliffs, 1980).

[6] J. McCarthy, A micro-manual for LISP — not the whole truth, in: R.L. Wexelblat: History Prog. Lang. Conf. Sigplan Notices, **13**, (1978).

[7] S.W. Postma, 'n Kritiese Evaluering van die Programmeertaal Lisp met Voorstelle vir Moontlike Verbeteringe, MSc Dissertation, RAU 1979.

[8] S.W. Postma, Quadlisp Language Reference Manual, UNISA CS Colloquia, **CS. 1. 81**, (1981.

[9] S.W. Postma, The Principles of Quadlisp, 2nd SA Computer Symp., Pretoria 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.

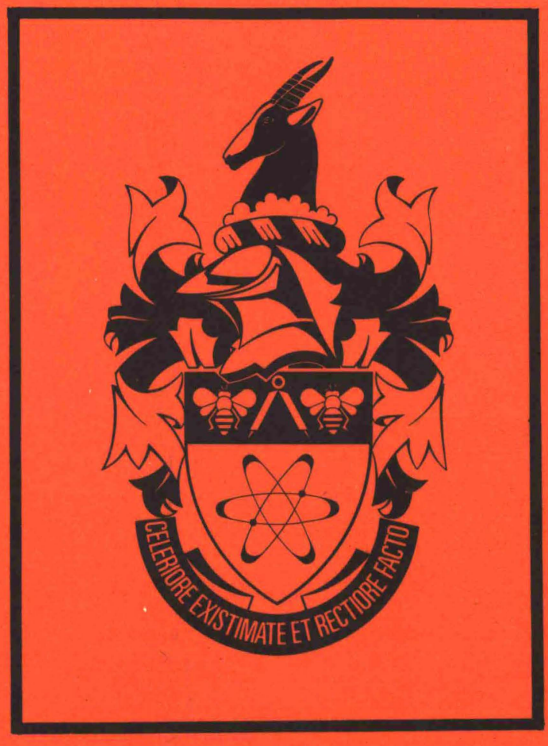
Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

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.