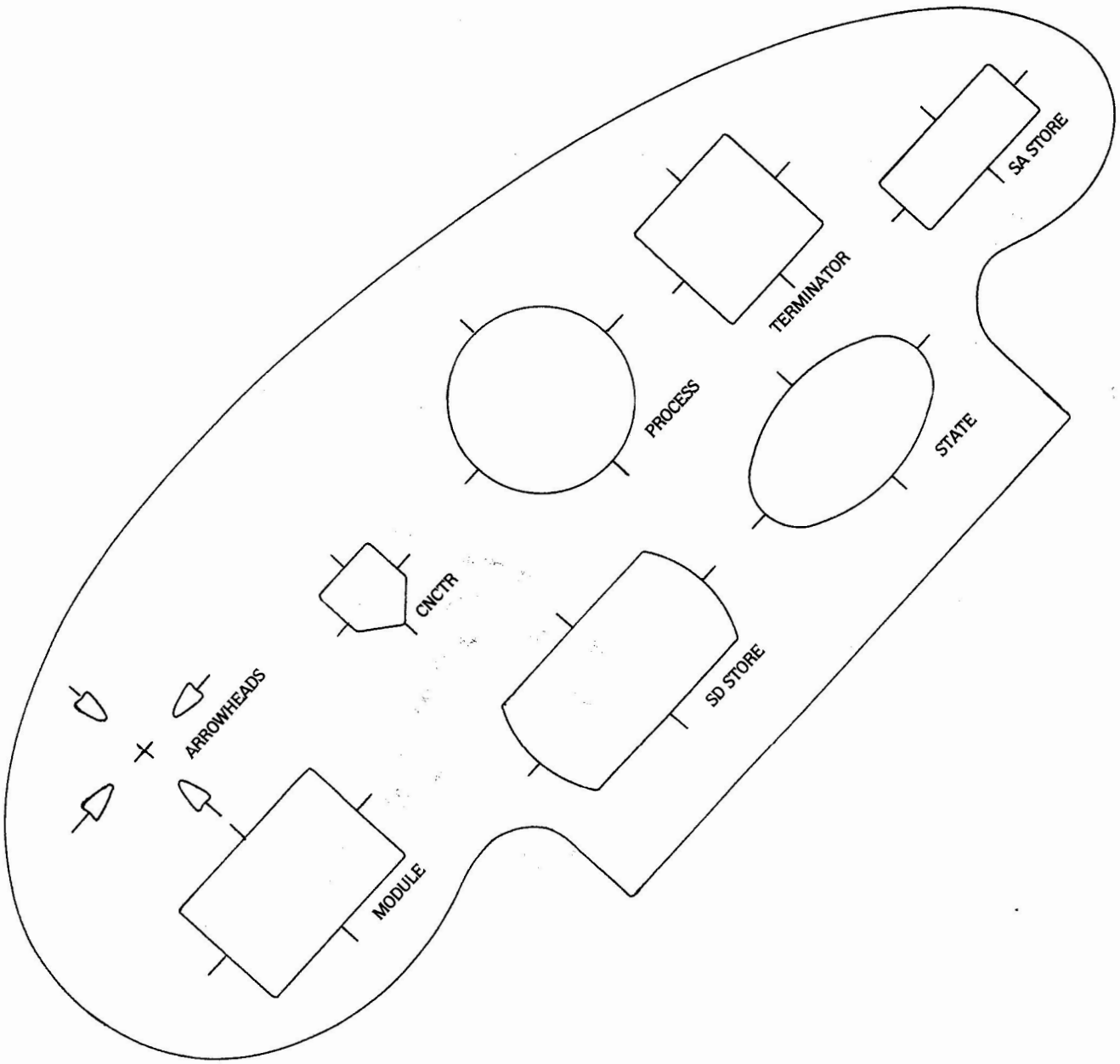
"THE HEAP"

DECUS

MAY 1985 ISSUE



LANGUAGES AND TOOLS SIG

TABLE OF CONTENTS

EDITOR'S NOTES

It is my hope that this issue of "THE HEAP" will reach you prior to the New Orleans Symposium, but I expect that it probably will not. If not, please accept my apologies, the delay was entirely my fault, due to other, work related, pressures.

This issue has three major contributions, all of which are of great interest. First is an article on TeX, the increasingly popular text formatting tool. for those of you who are not familar with TeX, this will serve as a great introduction. TeX itself is public domain software, and runs on a wide variety of machines, including most DEC CPU's. It has an active and enthusiastic user's group, and after the article there is the information and forms necessary to contact that user's group.

The second submission is by Don Rosenthal, and consists of the slides from his Artificial Intelligence presentation from Anaheim, along with a very extensive A.I. bibliography. If you're interested in A.I., the bibliography alone is probably worth the cost of your subscription to "THE HEAP".

The final major contribution is of extreme importance to FORTRAN users. You may be aware that there is currently work underway to design a new Fortran standard. This submission is the Fortran Information Bulletin which describes the features proposed for that new standard. When the publication of the FIB was proposed, both DEC and IBM voted against publication. Following the FIB I have included DEC and IBM's explanations for their negaitive votes, and following that is X3J3's response to DEC and IBM. I am endebted to Jay Wiley, our standards coordinator, for getting this information to me. I had to transcribe the responses, so I will apologize in advance for any typos or ommissions.

One final submission may be of interest. You may or may not be aware that DECUS is considering combining the SIG newsletters into a single publication. This is an issue I feel strongly about, and I believe it will significantly impact how we are able to relate to you, the general membership. While I don't really believe in editorializing, I feel this is important enough to make an exception. I have summarized my feelings, and what I believe to be the current state of affairs. If you wish to comment, please do. My address, again, is:

Alan L. Folsom, Jr.
Dept 431, Fischer & Porter Co.
County Line Road
Warminster, Pa. 18974

Finally, if you have not yet responded to the Wishlist from the last issue, please do so. I will continue to collate responses as long as I can, and will eventually forward the final tally to both DEC and the SIG leadership.

Get Ready for New Orleans!!

- Kathy Hornbach, L&T Chair

The New Orleans DECUS Symposium is just around the corner, and it is one you certainly will not want to miss. Besides the delicious Creole cuisine around the corner from the convention center, the Symposium committee has put together an excellent slate of sessions -- close to, if not THE, most sessions ever scheduled for a DECUS Symposium. This article will fill you in on some of the things planned.

The theme for the Languages & Tools SIG this symposium is **Software Documentation Tools**.. . . .hardly a glamorous subject, yet a very important one. No one likes to do it, yet everyone wants it to be done. We have lined up some indepth technical talks behind this theme, and a number of users have submitted papers covering their experiences. Some of the highlights include a paper by DEC, describing the software tools they used to produce the top quality VMS V4 documentation. There are two talks on TeX, the public domain typesetting package written by Donald Knuth of Stanford, and a talk on WEB, an automated programming documentation aid that grew out of the work with TeX. We plan to hold a number of birds-of-a-feather sessions on experiences in documentation techniques, so bring your war stories and success stories.

As usual, DEC has sponsored a full slate of sessions giving the **latest details on C, Pascal, Fortran, PL/1, CMS, MMS, DEC/Test Manager, DEC/Shell, and the rest.** This will be an excellent time to consolidate the experience gained over the last few months on the new versions of these tools. We also expect to hear more details on some of the new products that were announced at Fall DECUS, including the **Language Sensitive Editor, the new TPU editor, the Performance Coverage Analyzer, and Ada.**

We have invited members of the **Fortran 8x standards committee** to give presentations on the current status of the proposed standard. This will be followed by a panel session where the audience will be able to ask questions of both the standards committee and DEC's Fortran developers.

The SIG is sponsoring three sessions where you will have a chance to meet the members of the Steering Committee and DEC developers in the Languages and Tools area. These include the **roadmap**, first thing Monday morning when we fill you in on any last minute changes; and the **wrapup**, late Friday, afternoon when we assess the happenings for the week and ask for input from the audience on how to do our job better. The third session, and one you will definitely want to see, is the **Wishlist, DEC Feedback, and Question & Answer** session Friday afternoon. DEC will respond to the top ten vote getters in the recent Wishlist distributed through the newsletter; then they will open the floor to feedback on their current

tools and suggestions for what they should work on next.

You may still have time to sign up for one of the four PreSymposium seminars being offered by Languages and Tools. The only difficulty is in deciding which one to go to. (I'm teaching one, and I'm STILL having a hard time deciding!)! The seminars being offered include:

Software Project Development Using CMS/MMS -- everyone has heard what CMS and MMS can do, but how does a project REALLY use them to control software throughout the lifecycle? How do you convince management that you need these tools? Bob Gable, who recently completed a stint using CMS/MMS to control source code on a project with over 50 developers and hundreds of thousands of lines of source code, will fill you in on the do's and don'ts of effective source code control.

Developing Medium Scale Ada Applications Using VAX Ada -- how do you take maximum advantage of the features of the Ada language -- things like the Ada Compilation system, the debugger interface, etc? What is the best way to structure Ada packages to make full use of the power of Ada? How does DEC Ada fit in with the rest of the VAX Common Language Environment? Bevin Brett, the instructor for this class, should be the one to know. One of the developers of DEC's Ada compiler, he will put Ada into the systems perspective and show how it and the VMS environment can be used to develop software more effectively.

Implementing a Software Development Environment -- To increase productivity on during software development, automated tools are becoming more and more prevelant. Tools are now available to support the upfront requirements and design phases, as well as coding. This seminar examines automated tools for software production, from an overall integrated viewpoint. It is not intended to teach how to use details of specific tools; rather it is a guide to what is possible to do with software tools, what tools are available on the marketplace, how to convince management to buy the tools and how to convince programmers to use them. Tools covered support requirements, design, code, test and documentation . This will by taught by Kathy Hornbach (yours truly) and draws many examples from actual case histories.

Artificial Intelligence -- A.I. is a subject that seems to be as much at home in Business Week as in some of the more traditional computer journals. This seminar will present an overview of what A.I is all about, for people already familiar with computers. It will talk about how A.I programs are implemented and what is and is not possible with today's technology. The instructor, Greg Parkinson, is an AI developer for Cognitive Sciences, and works on natural language systems.

TEX: TYPESETTING FOR ALMOST EVERYBODY

Samuel B. Whidden
Director, Computer Services
American Mathematical Society
Providence, Rhode Island

Abstract

This is an introduction to TEX, a typesetting program in the public domain. TEX, and METAFONT, a companion program for font creation, were developed at Stanford by Donald Knuth. Together they form the basis of a powerful, customizable system for the typesetting of any text, especially text with complex mathematical formulations. TEX (pronounced "techh") runs on the 10, the 20, and the VAX. TEX's output files are independent of any typesetting device; output driver programs have been written for several typesetting machines. A macro facility allows TEX to be tailored for fairly easy use by non-specialists; several customized macro packages have been written. "Style files" of macro expansions can be included at run time, permitting a macro call to have different expansions at, for example, preprint time and publication time, a feature that may permit TEX to serve as a generic typesetting language as it becomes more and more widely used.

I. The origins of TEX and METAFONT

FOIL 1 — Title page of Chapter One of *The TEXbook* by Donald Knuth, showing the TEX logo. Illustration by Duane Bibby.*

TEX is a computer typesetting language created by Donald Knuth of Stanford University. This talk will be a survey of the current status of TEX and its companion program, METAFONT. I'll tell you something of the reasons for their existence, something of what these programs can do, and a little of how they do it. I'll give you a beginning understanding of how to use TEX and I'll tell you how to obtain it, install it, get all the additional details you need, and how to get help with it from other users.

The Greek letters Tau Epsilon Chi, pronounced 'techhh', form, to quote Michael Spivak in *The Joy of TEX*, "the beginning of the Greek word that means art, a word that is also the root of English terms like technology. This name emphasizes two basic features of TEX: it is a computer system for typesetting technical text...and it is a system for producing beautiful text."

This foil reproduces the title page of Chapter One of *The TEXbook*, by Donald Knuth. *The TEXbook* is the principal document describing TEX. It's a book you'll need if you intend to use TEX. I'll tell you how to get it at the end of my talk.

The TEX logo you see in this foil is normally typeset this way, with the lowered "E", by TEX itself. It's a small display of the power of TEX. Trademark restrictions on the name "TEX" (rhymes with hex) make it necessary to spell TEX with a variety of lower and upper case letters—TeX—when the logo itself can't be used. Later in my talk I'll show you the commands that TEX uses to generate this logo.

* Reproductions from *The TEXbook*, © 1984, American Mathematical Society; co-published by the American Mathematical Society, Providence, RI, and Addison-Wesley Publishing Company, Reading, MA; reprinted with permission. The TEX logo is a trademark of the AMS.

METAFONT

FOIL 2 — METAFONT.

Tuan Pham

TEX is designed as an output-device-independent system for composition. Each device that is actually used to produce TEX output must have character fonts to use for typesetting. "METAFONT is a system for the design of alphabets suited to raster-based devices that print or display text." (Donald Knuth in the METAFONTbook)

Knuth placed TEX and METAFONT in the public domain and anyone may use them without payment to him or to Stanford. But he was concerned that TEX not be fragmented by various users into many incompatible versions and that a TEX standard of typesetting be maintained. As a result, the TEX logo is a trademark of the American Mathematical Society, controlled so as to assure that systems called TEX do what TEX claims to do.

Why is TEX a typesetting system for 'almost everybody'? TEX is a powerful typesetting tool. Because of its power, in applications beyond simple text it requires skill to use. It's very much like a programming language, and can get complicated. In most installations, a TEX wizard takes on the job of designing ways to handle complex material. Also, TEX's power is focused on scientific typesetting. It has language elements to handle tables and mathematics directly, but has no facility for graphics or for allowing text to flow around displays. So it is not the system of choice for newspapers, for example, or for CAD/CAM applications. But TEX is ideal for the individual writer wishing to compose his document to best advantage.

C1. [Compute q, r tables.] Set stacks U, V, C, and W empty. Set

$$k \leftarrow 1, \quad q_0 \leftarrow q_1 \leftarrow 16, \quad r_0 \leftarrow r_1 \leftarrow 4, \quad Q \leftarrow 4, \quad R \leftarrow 2.$$

Now if $q_{k-1} + q_k < n$, set

$$k \leftarrow k+1, \quad Q \leftarrow Q + R, \quad R \leftarrow \lfloor \sqrt{Q} \rfloor, \quad q_k \leftarrow 2^Q, \quad r_k \leftarrow 2^R,$$

and repeat this operation until $q_{k-1} + q_k \geq n$. (*Note:* The calculation of $R \leftarrow \lfloor \sqrt{Q} \rfloor$ does not require a square root to be taken, since we may simply set $R \leftarrow R+1$ if $(R+1)^2 \leq Q$ and leave R unchanged if $(R+1)^2 > Q$; see exercise 2. In this step we build the sequence

$k =$	0	1	2	3	4	5	6	...
$q_k =$	2^4	2^4	2^6	2^8	2^{10}	2^{13}	2^{16}	...
$r_k =$	2^2	2^2	2^2	2^2	2^3	2^3	2^4	...

The multiplication of 70000-bit numbers would cause this step to terminate with $k = 6$, since $70000 < 2^{13} + 2^{16}$.)

FOIL 3 — An example of pre-TEX mathematical typesetting, from Vol. 2 of Knuth's series of books on The Art of Computer Programming.[1]

In 1977, Knuth was faced with creating a revised edition of Volume 2 of his widely-read series, The Art of Computer Programming. The original edition, represented in this foil, was typeset by Monotype, a method involving skilled typositors using machines that produced metal type. The result was considered beautiful.

[1] [ACP], Vol. 2, p. 263.

(1)

(iii) $W_j(j,q,\ell_0)$ where $1 \le j \le 5$ and ℓ_0 is a complex num
Im $\ell_0 > 0$ and $0 \le$ Re $\ell_0 < 1$. Further, neither $\ell_0 + q$ nor $\ell_0 - q$
$S = [\ell_0 + n : n$ an integer$]$. $W_j(u,q,\ell_0)/O_3$ takes the form (j)
(b).

For each of these representations ν there is a basis for V
$f_m^{(\ell)}$, $\ell \in S$ such that

$$J^3 f_m^{(\ell)} = m f_m^{(\ell)}$$

(2)

The algebra P is nearly simple if and only if the following hold:

(a) *N is spanned by* $a, \cdots, a^{n-k-1}, b_1, \cdots, b_k$ *where* $ab_i = b_i^2 =$
$i, j = 1, \cdots, k$.

(b) *Either* $n - k =$ char F *with* k *even or* $n - k = m$ char F *for n*

Proof. By Theorem 5.5, there are elements $a, b_1, \cdots, b_k$ with N
$a, \cdots, a^{n-k-1}, b_1, \cdots, b_k$. Furthermore, $ab_i = 0$, $b_i^2 = \alpha_i a^{n-k-1}$, $b_i b_j =$
for all i, j where each α_i, λ_{ij} is in F. From this, it is clear that M
space of the space spanned by $a^{n-k-1}, b_1, \cdots, b_k$.

Assume P is nearly simple. Then there is a ϕ with $P(\phi)$ simple
show that each b_i is in M. To do this, it is necessary and sufficient

(3)

citly mention all other topologies. We will also use the symbol $Hk(Q)$
ull of the kernel of Q, i.e.

$Q) = \{s \in m_A : x(s) = 0$ for every $x \in A$ for which $x(Q) = 0\}$.

e norm topology for m_A is equivalent to the one induced by the
yperbolic" metric:

$$\rho(z, w) = \text{Sup}\,\{|x(z)|: x \in A, \|x\| \leq 1 \text{ and } x(w) = 0\}.$$

will use D to represent the open unit disk in the complex plane, $\mathbf{C}$,
for the open unit polydisk in n-dimensional complex space $\mathbf{C}^n$. T^n
to the essential boundary of D^n, i.e.

FOIL 4 — Some examples of typesetting that Knuth considered unpleasing: Typewriter, Varityper, and Selectric Composer output.[2]

To a mathematician, the appearance of mathematics on the page is very important, but economics were forcing publishers to turn to composition methods which could be undertaken by less skilled, and therefore less expensive, personnel—for example, the Varityper and Selectric Composer methods. These examples are taken from Knuth's paper, "Mathematical Typography", which he presented as the Gibbs Lecture at the annual meeting of the American Mathematical Society in 1978, and which can be found in Knuth's book "*TEX and* METAFONT, *New Directions in Typesetting*", published jointly by the American Mathematical Society and Digital Press.

In this foil, example (1) is typewritten, example (2) is Varityper machine copy, and example (3) was prepared on an IBM Selectric Composer.

[2] (1) *Memoirs Amer. Math. Soc.* No. 50 (1964), p. 32.
(2) *Trans. Amer. Math. Soc.* **169** (1972), p. 232.
(3) *Trans. Amer. Math. Soc.* **199** (1974), p. 370.

Some computer systems existed (TROFF is one example) which either produced results which Knuth felt were less than satisfactory or which employed abstruse or highly-coded input. As a computer scientist, he felt he ought to be able to employ the computer in the service of mathematical typography.

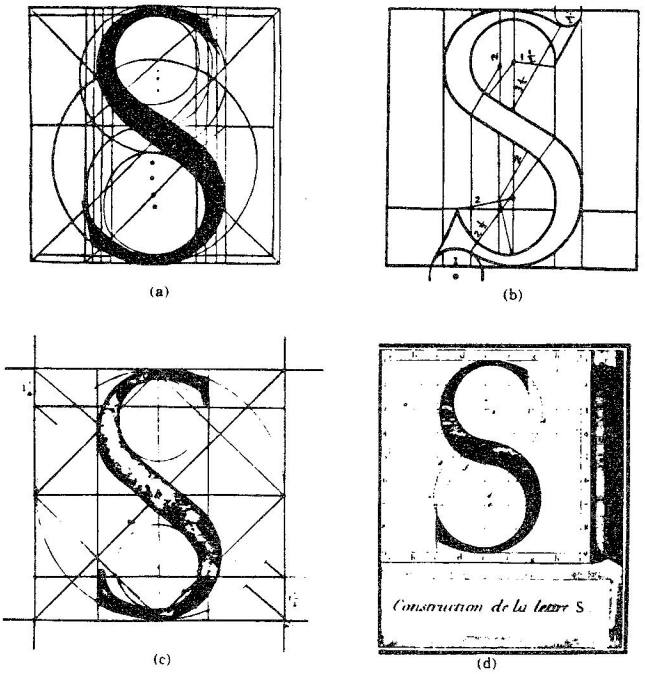


FIGURE 11. The letter S as defined by (a) Pacioli [42]; (b) Torniello [48]; (c) Palatino [43]; (d) French commission under Jaugeon [24].

FOIL 5 — Examples of geometric constructs.[3]

Designers have attempted for centuries to use mathematical techniques to design letters, but with results which made up in rigidity what they lacked in art.

FIGURE 12. Different S's obtained by varying the slope in the middle. (This shows $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$, 1, $\frac{3}{2}$, 2 times the "correct" slope.)

FOIL 6 — A METAFONT-produced family of "S"es.[4]

As Knuth says, this letter S is brought to you by a program. Knuth found that, besides using typesetting in the service of mathematics, he could use mathematics in the service of typesetting. Cubic spline curves (equations containing some third-order terms) yielded pleasing characters whose forms could be manipulated by modern-day computers powerful enough to perform the necessary calculations. He designed the METAFONT program to allow alterations in the parameters describing a letter so it could be varied until it met an acceptable standard of style, and so that needed font family variations—upright, bold, slanted—could be generated with relative ease.

[3] [T&M], *Mathematical Typography*, p. 30.
[4] *Ibid.*, p. 31.

II. The Role of the American Mathematical Society

FOIL 7 — A manuscript, in marked-up and typeset versions.

Richard Palais, an AMS trustee, was in the audience when Knuth gave his Gibbs lecture on mathematical typography. Palais was impressed enough with Knuth's design of TEX and METAFONT to investigate their possibilities for typesetting AMS books and journals. The costs of scientific publishing had risen alarmingly. Palais discovered that nearly half of the publishing costs of a scientific journal arise from the process of copy-editing and retyping authors' manuscripts, and proofreading and correcting the typeset output. If an author had access to TEX at his university, he could compose his article himself as part of his authorship effort. Palais' hope was that, if use of TEX became sufficiently widespread in the scientific community, the AMS might begin receiving some of its manuscripts already correctly composed in TEX, ready to be output to a typesetting device and published with little further investment. TEX's 'device independence' could make that possible.

FOIL 8 — A page from Mathematical Reviews.

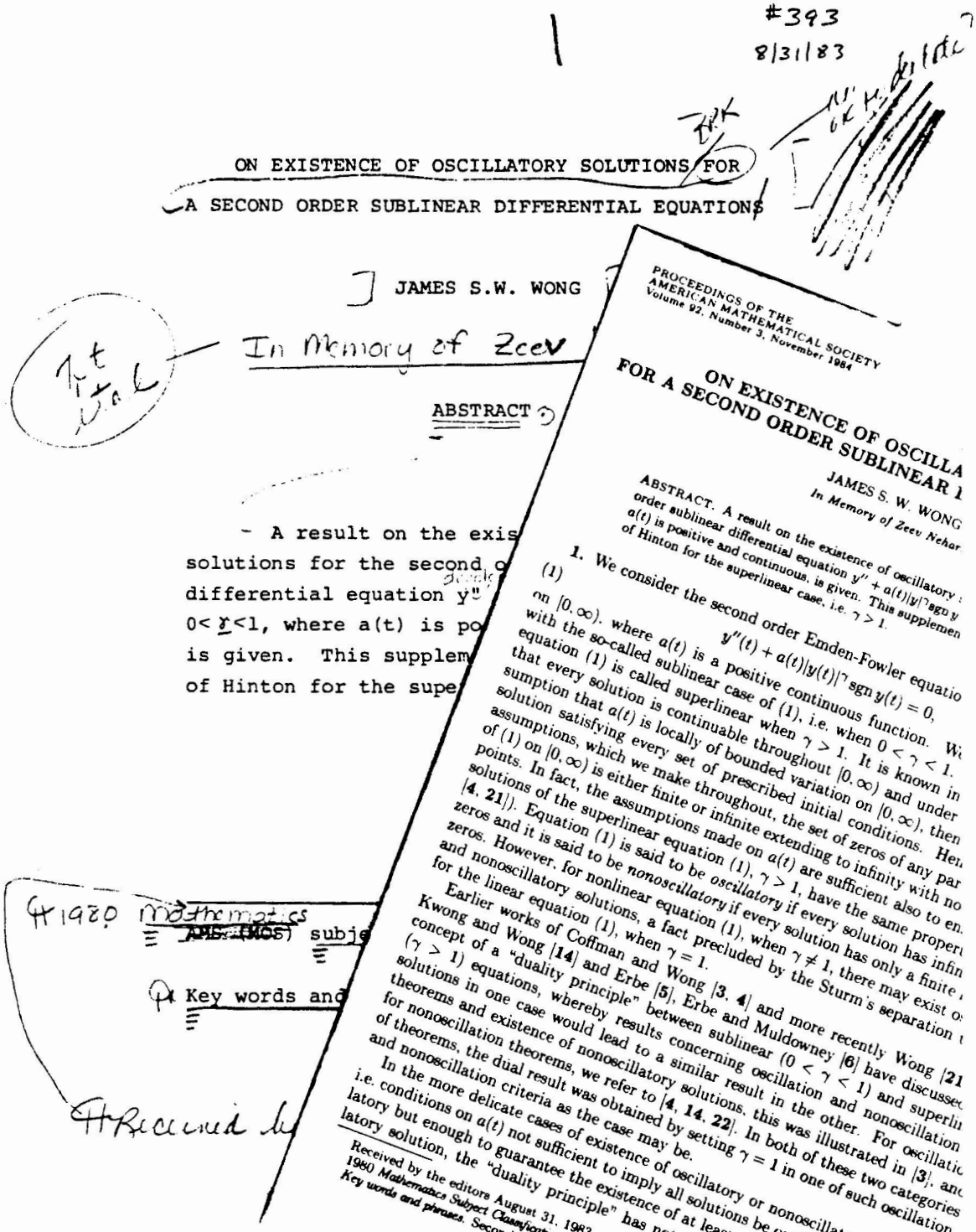
As far as I'm aware, the first large-scale continuous production use of TEX anywhere is the typesetting of Mathematical Reviews, the standard reviewing journal in mathematics, and its companion alerting journal, Current Mathematical Publications, both produced by the AMS in Ann Arbor, Michigan. These journals publish review text and bibliographic data on 3000 to 4000 mathematical papers per month, covering the world's mathematical literature. Proof copy is set on a low-resolution laser printer, and final copy on a very high resolution Alphatype CRS phototypesetter. Since identical fonts are used in both cases, the output of both machines is formatted identically, making it possible to perform all proofreading and correcting operations on the inexpensive device.

Other organizations, including Digital, are also beginning to use TEX, so far mostly for one-time, though sometimes large, documentation jobs.

FOIL 9 — The Combined Membership list, the AMS Catalog of Publications, and the Mathematical Sciences Professional Directory are examples of TEXed AMS publications.

TEX is still new, and the AMS does not yet typeset the bulk of its primary journals in TEX. That may come about over the next couple of years. As experience has been gained in the use of TEX, production costs have dropped at least to the level of previous typesetting systems used by the AMS. Some AMS books and journal papers are already being TEXed, as are nearly all of the Society's administrative publications, most of which are too complicated to be set easily by other methods.

This foil displays samples from three of the Society's administrative publications—its catalogue and two of its directories—which are set completely by TEX, without separate page make-up or cut-and-paste. Results so far seem to show that the more complex the typesetting requirements, the more cost-effective is TEX. And TEX is becoming available on a growing range of hardware, making it a contender to become a universal composition language.



ON EXISTENCE OF OSCILLATORY SOLUTIONS FOR A SECOND ORDER SUBLINEAR DIFFERENTIAL EQUATIONS

JAMES S.W. WONG

In Memory of Zeev

ABSTRACT

- A result on the exis
solutions for the second o
differential equation y"
0< γ <1, where a(t) is po
is given. This supplem
of Hinton for the supe

1980 Mathematics
AMS (MOS) subje

Key words and

Received by

PROCEEDINGS OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 92, Number 3, November 1984

ON EXISTENCE OF OSCILLA
FOR A SECOND ORDER SUBLINEAR

JAMES S. W. WONG
In Memory of Zeev Nehar

ABSTRACT. A result on the existence of oscillatory
order sublinear differential equation $y'' + a(t)|y|^\gamma \operatorname{sgn} y$
$a(t)$ is positive and continuous, is given. This supplemen
of Hinton for the superlinear case, i.e. $\gamma > 1$.

1. We consider the second order Emden-Fowler equatio

$$y''(t) + a(t)|y(t)|^\gamma \operatorname{sgn} y(t) = 0, \tag{1}$$

on $[0, \infty)$, where $a(t)$ is a positive continuous function. Wi
with the so-called sublinear case of (1), i.e. when $0 < \gamma < 1$.
equation (1) is called superlinear when $\gamma > 1$. It is known in
that every solution is continuable throughout $[0, \infty)$ and under
sumption that $a(t)$ is locally of bounded variation on $[0, \infty)$, then
solution satisfying every set of prescribed initial conditions. Her
assumptions, which we make throughout, the set of zeros of any par
of (1) on $[0, \infty)$ is either finite or infinite extending to infinity with no
points. In fact, the assumptions made on $a(t)$ are sufficient also to en
solutions of the superlinear equation (1), $\gamma > 1$, have the same propert
[4, 21]). Equation (1) is said to be *nonoscillatory* if every solution has only a finite
zeros and it is said to be *oscillatory* if every solution has infin
zeros. However, for nonlinear equation (1), when $\gamma \neq 1$, there may exist o
and nonoscillatory solutions, a fact precluded by the Sturm's separation
for the linear equation (1), when $\gamma = 1$.

Earlier works of Coffman and Wong [3, 4] and more recently Wong [21
Kwong and Wong [14] and Erbe [5], Erbe and Muldowney [6] have discusse
concept of a "duality principle" between sublinear ($0 < \gamma < 1$) and superli
($\gamma > 1$) equations, whereby results concerning oscillation and nonoscillation
solutions in one case would lead to a similar result in the other. For oscillatio
theorems and existence of nonoscillatory solutions, this was illustrated in [3], anc
for nonoscillation theorems, we refer to [4, 14, 22]. In both of these two categories
of theorems, the dual result was obtained by setting $\gamma = 1$ in one of such oscillation
and nonoscillation criteria as the case may be.

In the more delicate cases of existence of oscillatory or nonoscillatory so
i.e. conditions on $a(t)$ not sufficient to imply all solutions be oscillatory
latory but enough to guarantee the existence of at least one
latory solution, the "duality principle" has not been

Received by the editors August 31, 1983.
1980 *Mathematics Subject Classification.* Prim
Key words and phrases. Second order.

FOIL 7 — A manuscript, in marked-up and typeset versions.

Lagrange equation and the Newton-Raphson method need be calculated by hand. An example is given with numerical results. The automatic solution of the simplest problem in the calculus of variations in this paper is considered to be the first step in the automatic solution of more general optimal-control problems."

49A Existence theory for optimal solutions

See also 46017, 47052, 49006, 58011, 60066, 73070, 76067, 90246, 92006, 93079, 93093.

Marcellini, Paolo (I-NAPL-E); Sbordone, Car'o (I-NAPL) 85a:49008

On the existence of minima of multiple integrals of the calculus of variations.

J. Math. Pures Appl. (9) **62** (1983), no. 1, 1–9.

Let Ω be a bounded open set in $\mathbf{R}^n$ and $f(x,s,\xi)$ a Carathéodory function on $\Omega \times \mathbf{R}^N \times \mathbf{R}^{nN}$ such that $|\xi|^p \le f(x,s,\xi) \le a_1(x) + a_2|s|^p + a_3|\xi|^p$ with $a_1 \in L^\alpha_{\mathrm{loc}}(\Omega)$, $\alpha > 1$, $p > 1$, $a_2, a_3 > 0$. The function f is assumed to be quasiconvex in the sense of C. B. Morrey, Jr. [Pacific J. Math. **2** (1952), 25–53; MR 14, 992]. The authors consider the variational problem

$$(\mathrm{P}) \qquad \inf\left\{\int_\Omega f(x,u,Du)\,dx : u \in u_0 + (H^{1,p}_0(\Omega))^N\right\},$$

where u_0 is a fixed function of $(H^{1,p}(\Omega))^N$. They prove the following theorem: There exist a number $q > p$ and a sequence minimizing the problem (P) that is relatively compact in the weak topology of $(H^{1,q}_{\mathrm{loc}}(\Omega))^N$. Because the integral under consideration is lower semicontinuous in the weak topology of $(H^{1,q}_{\mathrm{loc}}(\Omega))^N$ for every $q > p$, the authors obtain as a corollary that the problem (P) has a minimum. Their proof makes use of, among other things, a variational principle of I. Ekeland [Bull. Amer. Math. Soc. (N.S.) **1** (1979), no. 3, 443–474; MR 80h:49007] and an argument concerning regularity in $(H^{1,q}_{\mathrm{loc}}(\Omega))^N$ (for any $q > p$) in the sense of N. G. Meyers [Ann. Scuola Norm. Sup. Pisa **17** (1963), 189–206; MR **28** #2328] that was introduced in the context of minimum problems by M. Giaquinta and E. Giusti [Acta Math. **148** (1982), 31–46; MR 84b:58034]. *Vanna Zanelli* (Modena)

Barbu, Viorel (R-IASI) 85a:49009

Optimal feedback controls for semilinear parabolic equations.

Mathematical theories of optimization (*Genova*, 1981), 43–70, *Lecture Notes in Math.*, 979, *Springer, Berlin-New York*, 1983.

The author considers the problem of existence of an optimal feedback control for the system $y_t + Ay + \beta(y) = Bu$ $((x,t) \in \Omega \times (0,t))$, $y(x,0) = y_0(x)$ $(x \in \Omega)$, $y(x,t) = 0$ $((x,t) \in \Sigma = \Gamma \times (0,t))$ with the cost functional

$$\int_0^T (\Phi(y(t)) + h(u(t)))\,dt + \Psi(y(T)).$$

Here Ω is an open bounded subset in n-dimensional Euclidean space with sufficiently smooth boundary Γ and A is a uniformly elliptic selfadjoint partial differential operator,

$$A = \sum\sum D^j(a_{ij}(x)D^j) + a_0(x).$$

The operator B is linear and bounded from a real Hilbert space U to $L^2(\Omega)$ and the functions Φ, Ψ, β, and h satisfy various assumptions that we will not detail here. There are also results on a control problem with infinite time horizon and on the time optimal control problem as a limit of a sequence of problems of the type described above.

{For the entire collection see MR 84g:49003.}

H. O. Fattorini (Los Angeles, Calif.)

Casas, Eduardo (E-ZRGZ) 85a:49010

Quelques problèmes de contrôle avec contraintes sur l'état. (English summary) [Optimal control problems with constraints on the state]

C. R. Acad. Sci. Paris Sér. I Math. **296** (1983), no. 12, 509–512.

Author's summary: "In this note we consider a class of optimal control problems with constraints on the state. We associate a Lagrangian to the control problem and prove the existence of Lagrange multipliers in a space of measures. Finally, we study some properties of these multipliers."

Ekeland, I.; Lasry, J.-M. 85a:49011

Duality in nonconvex variational problems. (French summary)

Advances in Hamiltonian systems (*Rome*, 1981), 73–108, *Ana. CEREMADE, Birkhäuser, Boston, Mass.*, 1983.

The authors develop a duality method for obtaining critical points of functionals which split into a quadratic term and a convex term. The main abstract theorem is the following: Let V be a reflexive Banach space, $A : V \to V^*$ linear, continuous and selfadjoint, and let $F : V \to \mathbf{R} \cup \{+\infty\}$ be convex lower semicontinuous, with $\mathrm{dom}\,F \neq \emptyset$. Let I and J be functionals on V defined by:

$$I(u) = \tfrac{1}{2}(Au,u) + F(u), \qquad J(v) = \tfrac{1}{2}(Av,v) + F^*(-Av),$$

where $F^* : V^* \to \mathbf{R} \cup \{+\infty\}$ is the Fenchel conjugate of F. Then any critical point of I is also a critical point of J. Moreover, if $0 \in \mathrm{Int}(\mathrm{dom}F^* + R(A))$ and v is a critical point of J, then there is some $w \in \mathrm{Ker}A$ such that $v + w$ is a critical point of I.

In the case where $F(u) = G(Ku)$, with $G : X \to \mathbf{R} \cup \{+\infty\}$ convex and $K : V \to X$ continuous linear, one can replace the hypothesis on $\mathrm{dom}F^* + R(A)$ by easier conditions; this leads to the concept of a pseudocritical point. When K is compact and G is subquadratic at infinity or superquadratic at the origin, a study of the dual functional J reveals the existence of critical or pseudocritical points. In the second half of the paper the abstract results are used to prove the existence of periodic solutions of a nonlinear wave equation describing a vibrating string, and also to prove existence of solutions to the following boundary value problem for a damped Hamiltonian system:

$$\dot{x} = \nabla_p H(t,x,p), \qquad x(0) = x(T),$$
$$\dot{p} = -\nabla_q H(t,x,p) - ap, \qquad p(0) = \exp(aT)p(T).$$

{For the entire collection see MR 84j:58001.}

A. Vanderbauwhede (Ghent)

Pal, D. V. (6-IIT) 85a:49012

On nonlinear minimisation problems and Lf-splines. I.

J. Approx. Theory **39** (1983), no. 3, 228–235.

Let X, Y be a pair of separated locally convex spaces, $T : X \to Y$ a continuous operator, $f : Y \to \mathbf{R} \cup \{+\infty\}$ a lower semicontinuous proper convex function, and C a closed nonempty subset of X. The following problem (P_y) is considered: Given an element $y \in Y$, minimize f over $T(C) - y$. The author presents various conditions which ensure that the problem $\{\mathrm{P}_y\}$ has a solution for each $y \in Y$. In particular, he deduces the existence of so-called Lf-splines in C, i.e. solutions of the problem (P_0) when T is a bounded linear operator. *W. W. Breckner* (Cluj-Napoca)

Barles, Guy 85a:49013

Inéquations quasi variationnelles du premier ordre et équations de Hamilton-Jacobi. (English summary) [Quasivariational inequations of first order and Hamilton-Jacobi equations]

C. R. Acad. Sci. Paris Sér. I Math. **296** (1983), no. 15, 703–705.

Author's summary: "We prove various existence and uniqueness results of solutions of first order quasivariational inequalities in $\mathbf{R}^N$. These results rely on the notion of a viscosity solution of the Hamilton-Jacobi equations, properly extended to this context."

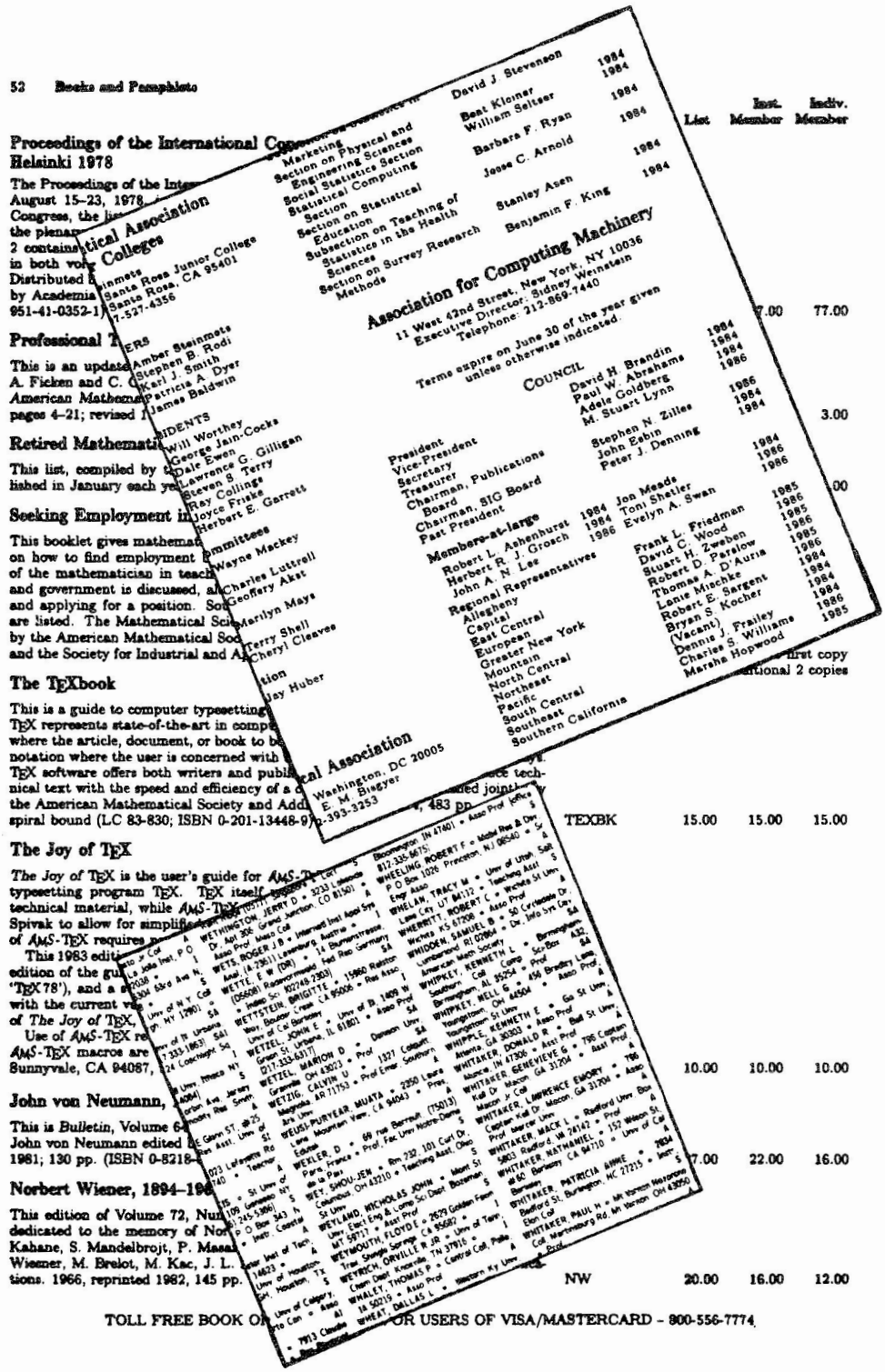
Hoffman, K. H. (I-ROME); Niesgodgka, M. (I-ROME) 85a:49014

Control of parabolic systems involving free boundaries.

Free boundary problems: theory and applications, Vol. I, II (*Montecatini*, 1981), 431–462, *Res. Notes in Math.*, 78, *Pitman, Boston, Mass.-London*, 1983.

The paper offers a very interesting survey on recent advances in the methods of solving various control problems by the study of

FOIL 8 — A page from Mathematical Reviews.



52 Books and Pamphlets

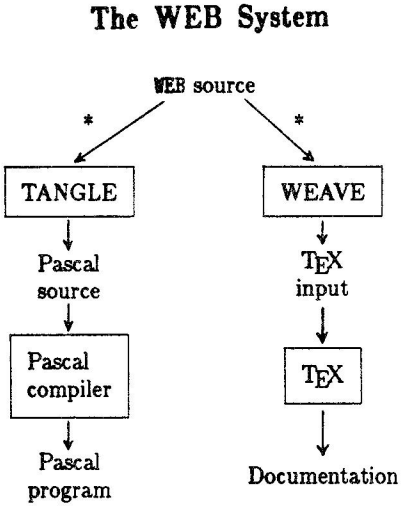
	List	Inst. Member	Indiv. Member
Proceedings of the International Congress of Mathematicians, Helsinki 1978			
The Proceedings of the International Congress of Mathematicians, August 15–23, 1978, ... Congress, the ... the plenary ... 2 contains ... in both ... Distributed ... by Academia ... 951-41-0352-1)			77.00
Professional T...			
This is an update ... A. Ficken and C. ... *American Mathema...* pages 4–21; revised ...			3.00
Retired Mathemati...			
This list, compiled by ... lished in January each ...			
Seeking Employment in ...			
This booklet gives mathema... on how to find employment ... of the mathematician in teach... and government is discussed, a... and applying for a position. So... are listed. The Mathematical Sci... by the American Mathematical Soc... and the Society for Industrial and A...			
The TEXbook			
This is a guide to computer typesetting ... TEX represents state-of-the-art in comp... where the article, document, or book to be ... notation where the user is concerned with ... TEX software offers both writers and publi... nical text with the speed and efficiency of a ... the American Mathematical Society and Addi... spiral bound (LC 83-830; ISBN 0-201-13448-9)	TEXBK	15.00	15.00
The Joy of TEX			
The Joy of TEX is the user's guide for *AMS*-TEX ... typesetting program TEX. TEX itself ... technical material, while *AMS*-TEX ... Spivak to allow for simplifi... of *AMS*-TEX requires ... This 1983 editi... edition of the gu... 'TEX 78'), and a ... with the current ... of *The Joy of* TEX, ... Use of *AMS*-TEX re... *AMS*-TEX macros are ... Sunnyvale, CA 94087,		10.00	10.00
John von Neumann, ...			
This is *Bulletin*, Volume 6... John von Neumann edited ... 1981; 130 pp. (ISBN 0-8218-...		7.00	22.00
Norbert Wiener, 1894–19...			
This edition of Volume 72, Num... dedicated to the memory of Nor... Kahane, S. Mandelbrojt, P. Masa... Wiener, M. Brelot, M. Kac, J. L. ... tions. 1966, reprinted 1982, 145 pp.	NW	20.00	16.00

TOLL FREE BOOK O... OR USERS OF VISA/MASTERCARD - 800-556-7774

FOIL 9 — The Combined Membership list, the AMS Catalog of Publications, and the Mathematical Sciences Professional Directory are examples of TEXed AMS publications.

III. Structure of the TEX and METAFONT System

Knuth wrote TEX & METAFONT originally in SAIL, the Stanford Artificial Intelligence Laboratory's programming language. The programs have gone through several generalizations, and the current versions are written in 'WEB', a documentation/programming language invented by Knuth to make TEX and METAFONT universally distributable. Naturally, anything written in WEB must be processed by TANGLE and WEAVE.



FOIL 10 — The WEB system.

From the same WEB source code, TANGLE generates Pascal source code and WEAVE generates a TEX-input documentation file. Of course, many more programs than just TEX and METAFONT have been written in WEB— including TANGLE and WEAVE. If you obtain TEX, you'll receive on the same tapes a copy of Knuth's paper entitled *Literate Programming*, which provides more information on the WEB system. Knuth asserts that the work of the author of a program should be as literate (as opposed to illiterate) as that of any other author. He sees literate programming as the successor to structured programming. Computer science may be seeing much more of WEB.

Generic TEX distribution tape – Selected files

-READ-.ME	Overall documentation and instructions
TANGLE.WEB	
TANGLE.PAS	Use this file to bootstrap and test TANGLE
TANGLE.CHANGES	Changes used to create TANGLE.PAS from TANGLE.WEB
WEAVE.WEB	
DVITYPE.WEB	Prints out DVI files
TEX.WEB	
TEX82.BUG	Explains all changes made to TEX since version 0
PLAIN.TEX	The basic macro package for TEX
HYPHEN.TEX	Hyphenation patterns, used by PLAIN.TEX
SAMPLE.TEX	Small TEX job to show font layout
TRIP.*	Test suite for TEX
TRIPMAN.TEX	TEX source of the TRIP manual
AM*.TFM	Font metrics for Computer Modern fonts
Macro packages and documenation: LATEX, AMS-TEX, HP-TEX	
The TEXbook	
System-specific files: TOPS-20, VAX/VMS, IBM VM/CMS, IBM MVS	
MF.WEB	Preliminary version, new METAFONT
PLAIN.MF	Plain METAFONT macros
SYNTAX.TEX	What there is of a user manual

FOIL 11 — Major files on the TEX/METAFONT installation tapes

You can obtain the TEX and METAFONT programs from a tape distribution service bureau in Sunnyvale, California. An order form giving details is attached as Appendix A. This foil lists a few of the files on the tapes you'll receive. One file tells you what to do with all the other files. You start by compiling the Pascal source of a bootstrap version of TANGLE. (If you have an off-beat Pascal compiler, I'm told you may have to make changes in tangle.pas.)

This compile gives you a preliminary version of tangle.exe. You use this to TANGLE a tangle.web source file from the tape, which gives you a new tangle.pas, which you compile into a new tangle.exe. Knuth intends that we not make any changes in the .WEB sources which these tapes supply. Therefore, in this step, along with the tangle.web source, TANGLE reads a tangle.change file (which you can change and of which there is a model on the tape), which specializes the resulting tangle.exe for your installation.

Next, you feed the tex.web source on the tape to your new TANGLE and get a tex.pas, which you compile to get a tex.exe. Now you TEX the TRIP files on the tape and see whether the resulting output is the same as the TRIP output file on the tape. If so, you have a good TEX. If not, the documentation tells you what to do. You generate the METAFONT program in a similar manner.

Once you have a good TEX, you can TANGLE the weave.web on the tape, then WEAVE the tex.web file, generating a tex.tex file, which you then TEX to generate TEX's own program documentation. See how simple it is?

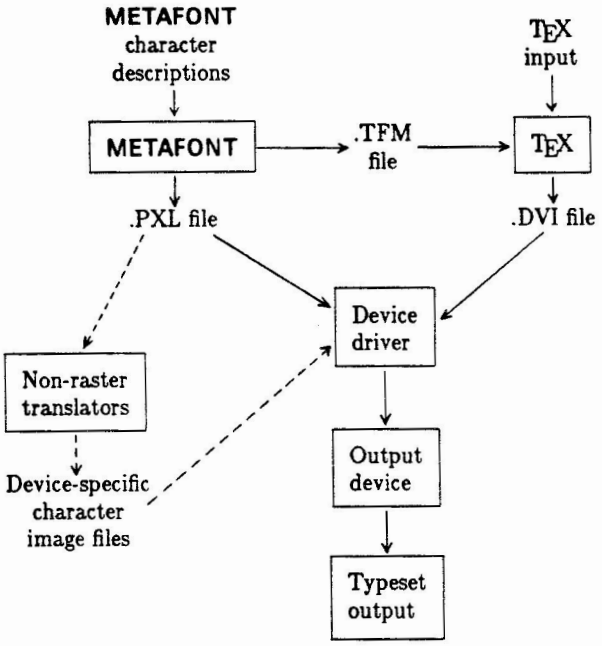
Some utility programs (called TEXware) also come on the tape. DVItype, which converts .DVI files into readable output, is one.

There are a few versions of TEX already specialized for particular hardware systems which can be ordered in place of the generic one. Such special versions exist for VAX/VMS and for TOPS-20, among others.

Some manuals can be ordered along with the tapes, notably manuals for WEB, TRIP, and TEXware. TEX source for *The TEXbook* itself is also on the tape, although it can't be reproduced, because not all the necessary fonts are present (and it should not be reproduced because it is copyrighted). This source file is very useful, however, as a nearly endless source of examples for doing things in TEX.

Generating TEX by the procedure I've described creates a version of TEX called INITEX. INITEX then allows loading of a hyphenation pattern file (see *The TEXbook*, Appendix H), .TFM files, style, or header, files (which usually includes `plain.tex`; AMS-TEX and LATEX are others. I'll return to these style files later), and any other files of macros desired. The resulting version of TEX, with these additions preloaded, is saved for production use. Several different production versions, preloaded with different fonts and style files, can be generated if necessary.

The TEX/METAFONT System



FOIL 12 — The TEX/**METAFONT** system.

Once you have a working version of TEX (and of **METAFONT**, if you're going to install that, too), you're ready to use them. **METAFONT** reads character descriptions in its own input language, creating two kinds of output: TEX Font Metrics (.TFM) files, and raster-image (.PXL) files.

The font metrics files contain character dimensions and reference points, plus kerning, ligature, and possible piecing information (for very large characters). The .TFM file is the same for all raster sizes and is used directly by TEX to calculate the size and position of each character on the line.

The .PXL file contains the actual bit maps of the characters in a given point size for a given output resolution (for example, 240 dots per inch or 300 dots per inch, which are common laser printer resolutions at present). (These files are very large; for some very high resolution output devices, another program might be used to compact the very large bit maps into a shorthand recognized by the machine.)

TEX reads the .TEX input file, along with whatever .TFM files are called for by the text, and produces a device-independent .DVI output file. The .DVI file is read by device-specific programs which translate it into commands recognized by a specific output device, merging raster patterns from the necessary .PXL files, and transfer the results to the device a line or a page at a time for outputting. Drivers exist for a number of both low- and high-resolution devices.

Since the distribution tapes already contain .TFM and .PXL files for standard text fonts in several point sizes and resolutions, it is usually not necessary to install **METAFONT** unless you need to generate special fonts (in fact, for greater efficiency, the common .TFM files can actually be pre-loaded as tables in your production version of TEX by INITEX). Fonts can also be obtained from other TEX users. A bit more about **METAFONT** later.

Proof-Quality Output Devices Supported on DEC Computers

	DEC 10	DEC 20	VAX (Unix)	VAX (VMS)
Facit 4542				INFN/CNAF
Florida Data OSP		Math. Reviews		
Imagen 8/300		Math. Reviews		
Imagen Imprint 10	Stanford; Vanderbilt	SRI; Columbia	UC Irvine	Kellerman & Smith *
QMS Lasergrafix	Talaris	Talaris	Talaris	Texas A&M
Symbolics LGP-1		U of Washington	U of Washington	Calma
Varian		AMS		Science Applications
Versatec	G A Technologies; Vanderbilt	U of Washington	U of Washington	Kellerman & Smith *
Xerox Dover		CMU	Stanford	
Xerox 9700	U of Delaware			

* Graphics supported

FOIL 13 — Proof-quality output devices for DEC computers.

This and the next foil are examples of fairly simple tables set by TEX. These are some of the low-resolution, proof quality output devices that have been interfaced to TEX on Digital machines. I'll tell you shortly how to obtain this information from the source.

Typesetters Supported on DEC Computers

	DEC 20	VAX (Unix)	VAX (VMS)
Alphatype CRS	AMS		
Autologic APS-5, Micro-5	Textset		Intergraph *
Compugraphic 8400			Kellerman & Smith *
Harris 7500			SARA (Amsterdam)
Mergenthaler Linotron 202	Adapt. Inc.		

* Graphics supported

FOIL 14 — Repro-quality output devices for DEC computers.

These are the camera-quality output devices that currently can provide TEX output from Digital machines.

11
Boxes

FOIL 15 — Boxes; Title page of Chapter Eleven of *The TEXbook*.

Knuth designed into TEX the interesting concepts of "boxes" and "glue". TEX puts each output token it encounters into a box. Each letter is a box. So are interword spaces, discretionary breaks, and so on. Several such boxes together form a larger box to make a line. Once a box is built, it is fixed; TEX manipulates it as a single token.

12
Glue

FOIL 16 — Glue; Title page of Chapter Twelve of *The TEXbook*.

Boxes are held together by glue. Glue comes in standard, but variable, amounts. Glue can stretch an allowed amount, when it needs to, and can shrink as well. Both the stretchability and the shrinkability have limits, which may be different and which you can specify. TEX will complain when you ask it to stretch or shrink glue beyond the limits you've given it. These limits, as well as a great many other things, are specified to TEX in terms of units and dimensions, such as inches or points or picas or ems, which TEX understands.

One of a typesetting system's chief duties is to take a long sequence of words and to break it up into individual lines of the appropriate size.

One of a typesetting system's chief duties is to take a long sequence of words and to break it up into individual lines of the appropriate size.

One of a typesetting system's chief duties is to take a long sequence of words and to break it up into individual lines of the appropriate size.

One of a typesetting system's chief duties is to take a long sequence of words and to break it up into individual lines of the appropriate size.

FOIL 17 — Boxes & Glue; Stretching & Shrinking.

This foil shows a sentence set with four different specifications for glue setting. The first example is loosely set, with interword glue stretched, but not beyond its limits (there is no glue between letters).

The second example shows neither stretching nor shrinking of the glue.

In the third, the glue is shrunk nearly to its limit. Normally, glue can stretch a lot more than it can shrink. If the words in this example were much closer than this, the line would probably be unacceptable.

In the last example, the glue is permitted so little flexibility that TEX cannot fit the material into the space allowed. So it shrinks the glue as much as permitted, lets the line stick out beyond the right margin, and puts a black mark here to tell you where the trouble is. It also writes a nasty message to your screen when it does this.

In Vol. 14, No. 4 (Oct. 29, 1984), The Seybold Report on Publishing Systems reviewed TEX in connection with a TEX-based microcomputer typesetting system developed by Tyxset Corporation. One of the few complaints raised against TEX by the reviewer is that TEX puts out these overfull boxes, rather than simply moving the offending word down a line, with a suitable warning to the user. A very loosely set line, while unattractive, is much more likely to be printable than an overset one. But TEX would rather have you change the wording a little or add a discretionary hyphen to make the output acceptable. After all, TEX has its own reputation to maintain.

TEX uses a hyphenation algorithm developed by Knuth and one of his associates, Frank Liang. TEX has been taught to avoid hyphenation errors as much as possible by avoiding hyphenation as much as possible. Nevertheless, TEX knows how to hyphenate most words and has a hyphenation dictionary which you can add to if necessary. You can tell TEX how to hyphenate specific words as part of your input file, and you can insert discretionary hyphens anywhere you want.

14

How TEX Breaks Paragraphs into Lines

FOIL 18 — Linebreaking; Title page of Chapter 14 of *The TEXbook*.

TEX breaks lines into paragraphs only after it has read the whole paragraph. Then it considers all possible break points, assigning penalties to breaks which give undesirable results, such as very loosely or tightly set lines, or multiple hyphenated lines in a row. TEX selects the set of linebreaks which minimize the total "badness"—the sum of all such penalties—for the paragraph. Knuth has devised algorithms which reduce the number of tests required, making this process relatively fast and efficient. There are simple ways to override these automatic linebreaks by tying words together so no break occurs between them, or by forcing a break where one would not otherwise occur.

V. Using TEX

A Simple Input Example

The input

```
This is an easy example of input.

\bye
```

yields

This is an easy example of input.

FOIL 19 — A simple sample of TEX input and output.

Now we'll look at a few examples of how TEX is used. This foil shows the easiest possible TEX input. I typed this line and submitted the file to TEX. Because the production TEX at my installation is preloaded with standard fonts and the `plain.tex` style file, the line came out neatly typeset in the default roman font of our laser printer.

Control Sequences

A control word is composed of a backslash and one or more letters. It must be terminated by a non-letter.

`\AE sop`	Æsop	(Latin and Scandinavian ligature Æ)
`\P2`	¶2	(paragraph sign)
`\eject`		(force page break)
`\year`		(get the year of today's date)

A control symbol is composed of a backslash and a single non-letter.

`\&`	&	(ampersand)
`\-`		(discretionary hyphen)
`\"o`	ö	(umlaut accent)

FOIL 20 — Control sequences: Control words and control symbols.

For the most part, TEX's input consists of a mixture of text and commands. Commands, which are always introduced by a backslash, are called "control sequences", which are either control words or control symbols. A control word must be delimited by a non-letter; a control symbol is always just a single character following a backslash.

Typical Symbols and Accents

Most symbols must be keyed in "math mode".

`\S A.1`	§A.1	(section sign)
`$\alpha$`	α	(Greek letter alpha)
`$\gg$`	$\gg$	(much greater than)
`$\in$`	$\in$	(membership symbol)

Accents may be set above or below letters or symbols.

`\'o`	ó	(acute accent)
`\v s`	š	(háček or "check")
`\c c`	ç	(cedilla accent)
`$\vec\kappa$`	$\vec{\kappa}$	("vector")

FOIL 21 — Symbols and Accents.

TEX knows about a wide selection of special symbols and diacritical marks in many fonts. You set them by means of control sequences like these. TEX only knows about many of the symbols when it's in "math mode", since it considers such things as operators and greek letters to be mathematical symbols rather than characters in ordinary text fonts. You put "$" signs around the portion of your input you want TEX to process in math mode.

An Example Using Math Mode

```
Suppose $s=0$. Then $H_s^+$ is the space of
functions in $L_2({\bf R}^n)$ that vanish
for $x_n<0$.
```

Suppose $s = 0$. Then H_s^+ is the space of functions in $L_2(\mathbf{R}^n)$ that vanish for $x_n < 0$.

FOIL 22 — A simple example of math mode.

In math mode, TEX takes over the positioning of characters and the spacing between them. Mathematicians are fussy about style, and TEX doesn't leave such important matters to chance. Of course, you can override most of TEX's decisions if you wish. Ordinary math mode results in mathematical material set in line with non-mathematical text.

Primitives and Macros

Primitives are control sequences that are built into TEX.

`\read`		(read in a file)
`\uppercase{abc}`	ABC	(uppercase the string)
`$\underline{xyz}$`	$\underline{xyz}$	(underline the string)

Macros are control sequences that are abbreviations for longer strings of control sequences or text.

`\centerline{...}`	(center string on a line)
`\ninepoint`	(use 9-point fonts)

FOIL 23 — Primitives and Macros.

Some control sequences are "primitives", built right into TEX; others are "macros", the result of definitions made by users. The file `plain.tex` is a collection of macros defined by Knuth himself to make it easier to get TEX to do common and useful things. `plain.tex` is normally preloaded into a production TEX by INITEX.

Defining and Using Macros

```
\def\line{\hbox to\hsize}
\def\centerline#1{\line{\hss#1\hss}}

\centerline{In the Middle}
```

yields

In the Middle

FOIL 24 — Defining and using macros.

Here we see some examples of macro definitions. Macro definitions always consist of the control sequence `\def` followed by a control sequence which will become the name of the macro, followed by the string, enclosed in grouping symbols, which the macro name will stand for.

First we `\def` the macro `\line`, which consists only of a horizontal box (`\hbox`) the width of our page `\hsize`. If we were to use the macro `\line` directly by typing `\line{followed by some words inside grouping symbols}`, TEX would set those words in a line of width hsize, stretching or shrinking the interword glue to make the best fit it could.

But now we `\def` the macro `\centerline` which we'll use to put something into the horizontal box called `\line`. The string defined by `\centerline` consists of the macro `\line` with the expansion `{\hss#1\hss}`. The `#1` in the name of the macro stands for an argument, and the same symbol in the macro definition shows where the argument goes when the macro is expanded. `\hss` is a TEX primitive standing for essentially infinitely stretchable horizontal glue. When `\centerline` is called with an argument like "In The Middle", the argument is set between two pieces of glue with great stretchability, so the result is centered on the line.

November 1983 Report No. STAN-CS-83-985
Stanford Department of Computer Science (Version 1)

First Grade TEX

A Beginner's TEX Manual

by
Arthur L. Samuel

This manual is based on the publications of Donald E. Knuth who originated the TEX system and on the recent work of Professor Knuth and his many students and collaborators who have helped bring the TEX82 system to its present advanced state of development. The TEX logo that is used in this manual is a trademark of The American Mathematical Society. The preparation of this report was supported in part by National Science Foundation grant IST-820/926 and by the System Development Foundation.

FOIL 25 — *First Grade TEX*; Title page output

Anyone who is serious about using TEX will begin by studying a short manual called *First Grade TEX* by Stanford Professor Arthur Samuel.

This is the cover page of *First Grade TEX*. The next three foils will show the input that created this page. Note the material at the top of the page, the title and author's name in the middle, the Stanford Seal in the bottom half of the page, and the final paragraph of text.

```
\font\ninerm=amr9 \font\eightrm=amr8
\font\sixrm=amr6 \font\csc=amcsc10
\font\seal=stan70  % To produce the Stanford seal
\def\TeX{T\kern-.1667em%
\lower.5ex\hbox{E}\kern-.125em X}

\magnification=1200 % This magnifies everything by 1.2
\parskip 10pt plus 1pt
% This puts some empty space between paragraphs
\parindent 0pt % Paragraphs are not to be indented

\nopagenumbers % The title page is not to be numbered
\null\vskip-46pt % Put first line higher than normal
```

FOIL 26 — *First Grade TEX*; Title page input - 1

The first part of the input names some .TFM files that TEX will need in later input that were not preloaded, including

a file named stan70.tfm, which will be needed when a font named "SEAL" is called for. The "%" denotes a comment; TEX ignores everything on the line after it.

Next the TEX logo is defined as a macro called \TeX. The first token in the definition string is an uppercase T. Next is a negative kern, which is a mechanism for backspacing so as to bring two characters closer together than the .TFM metrics would place them. The kern is in units of "em"s, where an em represents the width of an M in the current font. Next, an uppercase E is put into a horizontal box and lowered slightly, then an X is set and kerned back. Everywhere you see the TEX logo in these foils, this macro produced it.

"Magnification 1200" means use fonts of the 10-point design which have been enlarged to 12 points (actual 12-point fonts would have a slightly different shape). The proper .PXL raster files must be available to the output driver for this to work.

\parskip and **\parindent** are TEX primitives whose default values are being reset here. There will be 10 points of vertical glue between paragraphs, with 1 point's worth of stretchability and no shrinkability. There will be no indentation at the beginnings of paragraphs (note that the amount has to be specified even where it is zero). This page won't be numbered (the default would center a number at the bottom of the page), and the page will be set higher than normal on the paper (glue at the beginning of a box, such as a page or a line, disappears, so a null box is set before the **\vskip** on the page).

```
\hbox to 6.5truein {November 1983 \hfil
          Report No. STAN-CS-83-985}
% A convenient way to make a box of specific size.
\vskip .1in % Skip down 0.1 inch
\line {Stanford Department of Computer Science\hfil
          (Version 1)}
\vfill % This and similar commands later,
%      will divide the space evenly.
\centerline{\bf First Grade \TeX}
\centerline{\bf A Beginner's \TeX\ Manual}
\vskip .25in
\centerline{by}
\centerline{Arthur L. Samuel}
\vfill
\centerline{\seal S}
\vfill
```

FOIL 27 — *First Grade TEX*; Title page input – 2

Next we enclose material in a couple of boxes with horizontal spacing to cause material to be right and left justified. Several \vfill vertical spacing commands will divide up the unfilled space on the page. Some lines are centered horizontally, and then the character "S" in the "SEAL" font—that is, the character which occupies the normal "ASCII grid" position of the letter "S" in in the Stan70.tfm font file (produced by somebody using METAFONT), is set. That produces the Stanford seal in the center of the page.

```
This manual is based on the publications of
Donald E. Knuth who originated the \TeX\ system and
on the recent work of Professor Knuth and his many
students and collaborators who have helped bring the
\TeX82 system to its present advanced state of
development. The \TeX\ logo that is used in this
manual is a trademark of The American Mathematical
Society. The preparation of this report was supported
in part by National Science Foundation grant
IST-820/926 and by the System Development Foundation.
\eject % \end would follow if this were the last page.
```

FOIL 28 — *First Grade TEX*; Title page input – 3

Last, the paragraph of text is set at the bottom of the page.

TEX's Help Facility

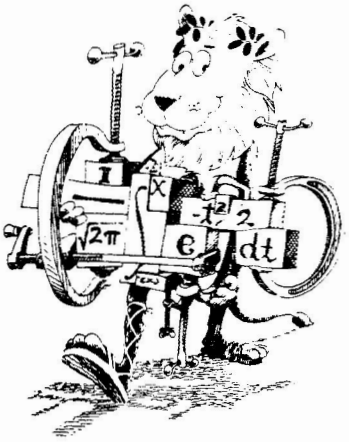
```
*\Bey
! Undefined control sequence.
<*> \Bey

? ?
Type <return> to proceed,
S to scroll future error messages,
R to run without stopping, Q to run quietly,
I to insert something,
1 or ... or 9 to ignore the next 1 to 9 tokens of input,
H for help, X to quit.
? h
The control sequence at the end of the top line
of your error message was never \def'ed. If you have
misspelled it (e.g., '\hobx'), type 'I' and the
correct spelling (e.g., 'I\hbox').
Otherwise just continue, and I'll forget about
whatever was undefined.

? i\bye
[1]
```

FOIL 29 — TEX gives some help with errors.

When TEX finds an error, it usually tells you about it and will offer what help it can. This foil shows an encounter with a control sequence that hasn't been defined (actually, in this case, a misspelling). A menu of actions is available. The user can insert a temporary fix, delete some tokens, tell TEX to ignore the error, or ask for more information, as here. If TEX is reading an input file at the time, in most installations it will also offer to return you to your editor at the point of the error in the file. Here, the correct spelling was inserted and TEX completed the page.

16

Typing Math Formulas



FOIL 30 — Display Math Mode; Title page of Chapter Sixteen of *The TEXbook*.

One of TEX's original purposes was to provide high quality mathematical typesetting.

Samples of Display Math Input

	Plain	AMS-TEX
$\begin{matrix} a \\ 1+b \end{matrix}$	`$$\{a\atop 1+b\}$$`	`$$\stack a\{1+b\}$$`
$\binom{n+1}{k}$	`$$\{n+1\choose k\}$$`	`$$\binom \{n+1\}k$$`
$\frac{a+b}{c+d}$	`$$\{a+b\over c+d\}$$`	`$$\frac \{a+b\}\{c+d\}$$`

FOIL 31 — Display mode; macro packages.

Recall that material to be set in math mode was enclosed in $ signs. Such material simply appears in line with surrounding text. Material enclosed by double $ signs is set in display math mode, separated from the adjacent text. In mathematical articles, this is usually done with important results and equations. TEX can number such equations for you automatically if you wish.

Three display-math expressions appear in this foil. The displayed output is shown in the left column, while the rightmost two columns show the input as typed in two different dialects of TEX. The center column shows the input as it is typed in `plain.tex`, the vanilla-flavored TEX. `plain.tex` is actually a file containing Knuth's working macro definitions, normally preloaded by INITEX. It is documented completely in an appendix of *The TEXbook*.

The righthand column repeats the input under the circumstance that the macro package called AMS-TEX has also been loaded (either dynamically or in the preload). AMS-TEX is a macro package designed by Michael Spivak at the request of AMS to make it easy for authors and editors to create material that TEX will typeset in the AMS publishing style. AMS-TEX is an extensive system of macro commands which builds on `plain.tex` and which is well-documented in the publication *The Joy of TEX*, also by Spivak (of which a new edition will be available in a few weeks). As AMS-TEX becomes available at universities and other hotbeds of mathematical authorship, AMS hopes to receive an increasing number of manuscripts on tape in the form of already-tested and composed TEX input.

Another complaint against TEX noted in the Seybold review is that TEX's vocabulary is mathematical. Notice in the `plain.tex` input that terms are employed that a mathematician would use naturally but which are alien to a typist. That is, the typist must know how to read the expression in mathematical terms in order to type it. Yet, if mathematics is to be typed by a non-mathematician typist, then the typist must learn some language in which to express it. Most typists seem to come to terms fairly easily with TEX's language. And of course, mathematicians who turn to TEX to compose their own work need no introduction to the language at all.

And finally, as the third column shows, the vocabulary can be redefined as you like it. Spivak has done so in these cases, as anyone may for his own purposes. If you do redefine commands like this, or if you define any macros at all for your own use, then, if you send me your input file to be TEXed on my system, the only requirement is that you must also send me all your macro definitions. But if you use only `plain.tex`, or only `plain.tex` and AMS-TEX, or only `plain.tex` and some other macro package which we both share, you need send me only your text input.

Fences

```
$$\left\{\vbox to 27pt{}\left\{\vbox to 24pt{}
\left\{\vbox to 21pt{}\Biggl\{\biggl\{\Bigl\{
\bigl\{\{{\scriptstyle\{{
\scriptscriptstyle\{\hskip3pt\}}\}}\}\bigr\}
\Bigr\}\biggr\}\Biggr\}\right\}\right\}\right\}$$
```

produces

$$\left\{\left\{\left\{\left\{\left\{\left\{\left\{\left\{\left\{\left\{(\,)\right\}\right\}\right\}\right\}\right\}\right\}\right\}\right\}\right\}$$

FOIL 32 — Fences

Another example of what TEX can provide in display mode is this expandable character. The input shows some versions of the brace which are large enough to require piecing—straight segments are inserted between end and center parts—as well as some called as characters from a font. `\Biggl{` is a bigger left brace than `\biggl{`, `\Bigl{` is next smaller and so on. `\scriptstyle` and `\scriptscriptstyle` invoke fonts that TEX would use automatically if it were setting subscripts of sub-subscripts. TEX will do the same thing with other fences, like "(" and "[".

Other Extendable Symbols

```
$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+
  \sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$$
```

produces

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+z}}}}}}$$

FOIL 33 — And other extendable symbols

The largest two of these radical signs are made up of extendable pieces.

Other Mathematical Expressions

$$a_0 + \cfrac{a}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4}}}}$$

typed as

```
$$a_0+{a\over\displaystyle a_1+
{\strut 1\over\displaystyle a_2+
{\strut 1\over\displaystyle a_3+
{\strut 1\over\displaystyle a_4}}}}$$
```

looks better than

$$a_0 + \frac{a}{a_1 + \frac{1}{a_2+\frac{1}{a_3+\frac{1}{a_4}}}}$$

which is what you get without **\strut** and **\displaystyle**.

FOIL 34 — And mathematical expressions

Left alone, TEX would set the expression at the bottom of this foil, thinking that the denominator fractions should decrease in size to stay more or less in line with the expression which starts the denominator. This would be correct if there were only two levels of fraction involved. Here, however, esthetics call for maintaining one size throughout the expression. The **\strut** produces an empty box the proper height to make TEX leave enough vertical space to set a full-sized fraction, and the command **\displaystyle** forces TEX to use full-sized characters when it would automatically drop into **\scriptstyle** or smaller.

A Table

The table

\rm	\sl	\it	\tt	\bf
Roman	*Slanted*	*Italic*	Typewriter	**Boldface**

is the result of the following input (the control sequence \\ for backslash is assumed to be defined already)

```
$$\vbox {\tt \halign {\hfil #\hfil &&
   \quad \hfil # \hfil \cr
\\rm& \\sl& \\it& \\tt& \\bf\cr
\rm Roman& \sl Slanted& \it Italic& \tt Typewriter&
    \bf Boldface\cr
}}$$
```

FOIL 35 — And tables.

Here's a fairly simple, two-line example of a table set by TEX. The table itself shows the control sequence names which call for various typefaces within a font family. \bf, for example, causes TEX to shift to the **boldface** style of the current font.

A table is set within a vertical box in display math, using the **\halign** macro for table construction. The **\halign** looks first for a template on which to base the table structure. Here, the template begins with **\hfil #\hfil**, showing that the first column of the table will have an element, represented in the template by #, surrounded by expandable horizontal space—that is, centered in the column. The next element in the template is an &, marking the end of the format for that column. The second & immediately following the first tells tex that the format for the next column is to be repeated as many times as the table requires. These are all just like the first column, except that they begin with a **\quad** of space (equal to one **em**). The **\cr** signals the end of the template.

The next line of input is the first row of the table. It puts the text "\rm" in the first column. The & following marks the end of input for that column. The row continues until the **\cr**. The second row enters the words "Roman", "*Slanted*", etc., in the appropriate columns, each in the appropriate typeface (overriding the **\tt** command at the beginning of the input, which caused all the entries in the first row of the table to appear in **typewriter** font).

The table input ends with grouping symbols which close the **\halign** macro, the **\vbox** macro, and the display mode.

This table was easy, and the concept is quite straightforward. But people can think up devilishly complicated table formats, with subdivided columns containing an assortment of text, multi-column headers, and so on. All these are possible to generate in TEX, but the input can require a good deal of thought. It will be very useful to TEX users everywhere when someone writes a macro package which uses these features of TEX to simplify the construction of complex tables.

VI. Document Formatting

Document Formatting with LaTeX

1. An enumerated list, like this one, is created with the **enumerate** environment.
2. Enumerations can be:
   (a) Nested within one another, up to four levels deep.
   (b) Nested within other paragraph-making environments.
3. Each item of an enumerated list begins with an **\item** command.

is produced by

```
\begin{enumerate}
 \item An enumerated list, like ...
 \item Enumerations can be:
       \begin{enumerate}
        \item Nested within one another ...
        \item Nested within other ...
       \end{enumerate}
 \item Each item of an ...
\end{enumerate}
```

FOIL 36 — LaTeX; A sample.

The *AMS*-TeX macro package provides extra-powerful commands for typesetting a wide variety of mathematical constructions. It does not include document formatting commands of the sort provided by Scribe or Runoff or Script. Leslie Lamport, at SRI, has written a macro package called LaTeX which does provide such commands. Like all other TeX macro packages, LaTeX need only be called dynamically or preloaded with TeX to make its commands available to you.

This foil shows one example of the sort of command available with LaTeX. This is a Scribe-like command called "**\enumerate**" which produces a nicely formatted list. LaTeX adds many powerful document processing commands to TeX, greatly simplifying the formatting of business documents such as letters, reports, and manuals. The LaTeX manual will be published shortly by Addison-Wesley.

The authors of *AMS*-TeX and LaTeX have collaborated, and they may eventually combine these two packages into one, perhaps called L*AMS*-TeX. Then again, that task may be left to some other ambitious TeX user.

Another significant task awaiting the touch of an experienced TeX user (that is, some TeXpert) is the creation of BookTeX, a macro package which will give professional book designers the pointed tools they need to create beautiful books. The designing of books takes very different skills from those needed for the creation of beautiful mathematics. Richard Southall is an eminent book designer at the Department of Typography and Graphic Communication at the University of Reading, in England, who has worked with Knuth. Southall asserts that no one has yet made a beautiful book with TeX because its commands, and those of most TeX macro writers so far, reflect the computer scientist's need for generalization rather than the graphic designer's understanding of beauty. So who will write the BookTeX macro package? The world awaits.

23

Output Routines

FOIL 37 — Output routines; Title page, Chapter 23 of *The TeXbook*.

One final aspect of TeX should be mentioned. **plain.tex** produces actual output in the form of pages, with numbers centered at the bottom and no running heads. Often, a different or more elaborate output format is needed.

The problem of document design for computer-based systems

In the past, the design of documents was done *a posteriori*. Books were written before they were designed: the conceptual structure of the author's thoughts was already in place, embodied in some graphic form or other, and the designer's task was to render or re-render that embodiment in a semantically effective and technically practicable way.

In the design of documents for computer-based document production systems, the problem is the other way round. The document has to be designed *a priori*, before the author's thoughts are present at all. The document designer's task is to devise efficient graphic embodiments for conceptual structures that are suitable to fit any thoughts that any author using the system might have.

The document designer cannot tell (let alone dictate) what content an author will put into each of the conceptual structures that the document design provides for, or in what order the structures will be used. It is not too hard to provide a graphic embodiment for each structure, that will behave reasonably if it is not used in what its designer would consider to be an unreasonable way; but what is unreasonable to a document designer may not be at all so to a mathematician or a philosopher.

Richard Southall

FOIL 38 — The results of one output routine.

This foil shows material formatted by an output routine that divides the material into two, right-justified columns.

The problem of document design for computer-based systems

In the past, the design of documents was done *a posteriori*. Books were written before they were designed: the conceptual structure of the author's thoughts was already in place, embodied in some graphic form or other, and the designer's task was to render or re-render that embodiment in a semantically effective and technically practicable way.

In the design of documents for computer-based document production systems, the problem is the other way round. The document has to be designed *a priori*, before the author's thoughts are present at all. The document designer's task is to devise efficient graphic embodiments for conceptual structures that are suitable to fit any thoughts that any author using the system might have.

The document designer cannot tell (let alone dictate) what content an author will put into each of the conceptual structures that the document design provides for, or in what order the structures will be used. It is not too hard to provide a graphic embodiment for each structure, that will behave reasonably if it is not used in what its designer would consider to be an unreasonable way; but what is unreasonable to a document designer may not be at all so to a mathematician or a philosopher.

Richard Southall

FOIL 39 — The results of a different output routine.

Here is the same material with a change in the output routine definition. Macro packages such as `plain.tex` and $\mathcal{AMS}$-TEX and LATEX provide only fairly simple and standardized output routines. $\mathcal{AMS}$-TEX offers only a "preprint" output format, suitable for typesetting mathematical papers in a generalized form. The AMS, however, when it publishes a paper created by $\mathcal{AMS}$-TEX, will include special output routines for the specific target journal. TEXperts at installations where TEX is in use will ordinarily create output routines specialized to local needs.

VII. A little more about METAFONT

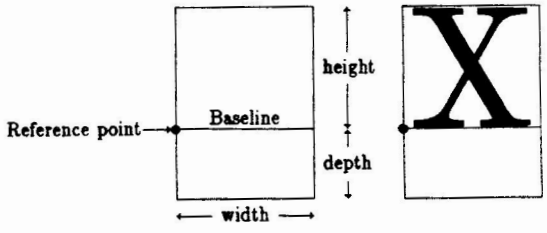
4

Fonts of Type

FOIL 40 — Fonts; Title page of Chapter Four of *The TEXbook*.

Fonts are an important part of TEX usage. One manufacturer, Autologic, has decided to place all the fonts that come with the TEX distribution tapes into its machines' font memories. In effect, Autologic used the .PXL files to create new Autologic fonts that can be ordered with their machines. That means you needn't have those .PXL files on your system. The programs that translate .DVI files into Autologic commands simply identify a character in the purchased fonts, rather than transmit a .PXL raster pattern. I understand that Autologic has also created .TFM files for its own proprietary fonts, so that TEX can also use existing Autologic fonts. There are those who think some of Autologic's fonts are nicer than TEX's Computer Modern Roman, so this is a useful advance, as well as a simplification. Other manufacturers may follow suit, especially if use of TEX becomes widespread.

Character Box as Interpreted by TEX

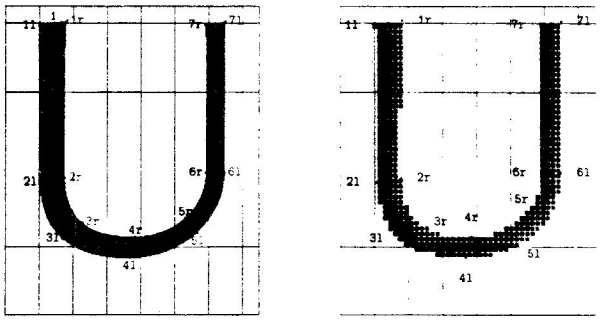


FOIL 41 — TEX's character.

Here is TEX's idea of a character. It is surrounded by a box, although it can sometimes stick out of the box. The box has a width and it has a reference point which sits on the baseline. It has a height extending above the baseline and a depth descending below it. The .TFM files contain

just these box dimensions, plus some kerning information, the amount of extra spacing to be left if the character sits next to one from a slanted font, and, in the case of math symbols, some other positioning and spacing information. These font metric files don't have any information about the shape of the character itself at all—that is to be found in the raster patterns of the .PXL files. **METAFONT** creates both kinds of file—the .TFM and the .PXL.

METAFONT Proof Output – The Letter U



FOIL 42 — METAFONT proof output for the character "U".

Here is an example of **METAFONT** input and output for the character "U". Relatively few points are required. Here the font designer has picked 7 positions along the letter, with three points—one on the left edge, one in the center, and one on the right edge—to be entered for each position. To generate this proof copy, **METAFONT** creates a .DVI file using a font called "the gray font", simply a collection of pixels at various resolutions, two of which are shown here. The points selected by the **METAFONT** user on the designer's original drawing are shown with the output.

METAFONT Input – The Letter U

```
vardef char.U=
setwidth .75em;
pos1(1.1thickwidth,10); % pen starts slightly thick
pos2(thickwidth,10);
pos3(thickwidth,40);
pos4(.5[thickwidth,thinwidth],75);
pos5(.9[thickwidth,thinwidth],130);
pos6(thinwidth,180); % now it's "thin", turned over
pos7(1.1thinwidth,190);
x1l=.1em; x7l=w-.1em; % there are sidebars of .1em
y1=y7=capheight; % have now fully specified 1 and 7
x2=x1; y2=.3capheight; dz2=(0,-1);
x3=.75[x4,x2]; y3=.75[y2,y4]; % see below!
x4r=.5[x2r,x6r]; y4l=-.05capheight; dz4=(1,0);
x5=.71[x4,x6]; y5=.71[y6,y4]; % see below!
x6=x7; y6=1/3capheight; dz6=(0,1); % going up at 6
stroke(1,2,.2,.05,.05); % make a very slight taper
curve(2,3,4); curve(4,5,6); % curve the bottom
stroke(6,7,.8,.05,.05); % finish somewhat as at left
labelpos(1,2,3,4,5,6,7); % list all positions
enddef;
```

FOIL 43 — The **METAFONT** input for "U".

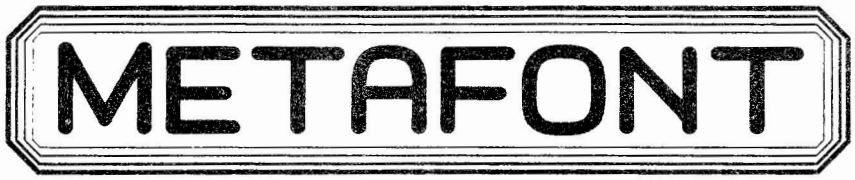
METAFONT is given pen size and shape instructions for each position, along with a few other instructions on some details of the character's shape. From these, **METAFONT** creates the raster image and the metric data for the character. **METAFONT** is designed so that the same input with only minor changes will create the same character in different sizes and typefaces, thereby generating whole font families from the same original design almost automatically.

William Burley

FOIL 44 — A METAFONT border.[5]

Last Spring at Stanford, Knuth offered a course in designing fonts with **METAFONT**. As one exercise, he asked each student to create a font of eight characters: four characters which could serve as the extensible edges of a large border and four others which would connect the edges to make the corners of the border. Here the resulting effect is something akin to the Greek Key.

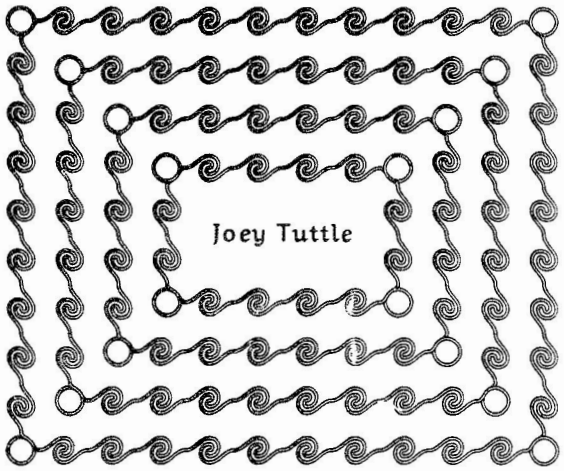
[5] Borders are reproduced from [TUB], Vol. 5, No. 2, November 1984, pp. 105-118.



Arthur Samuel

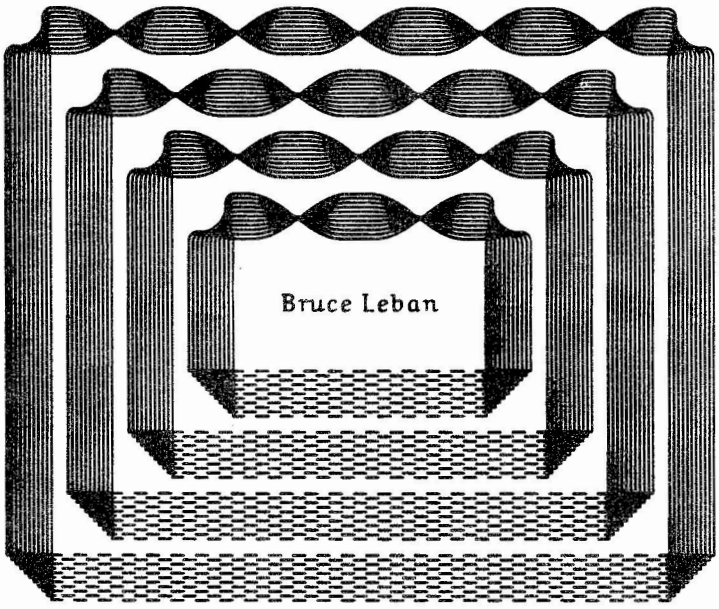
FOIL 45 — A Second METAFONT Border example.

Arthur Samuel, author of *First Grade TEX*, was one of the students. He provided a neat, conservative example.



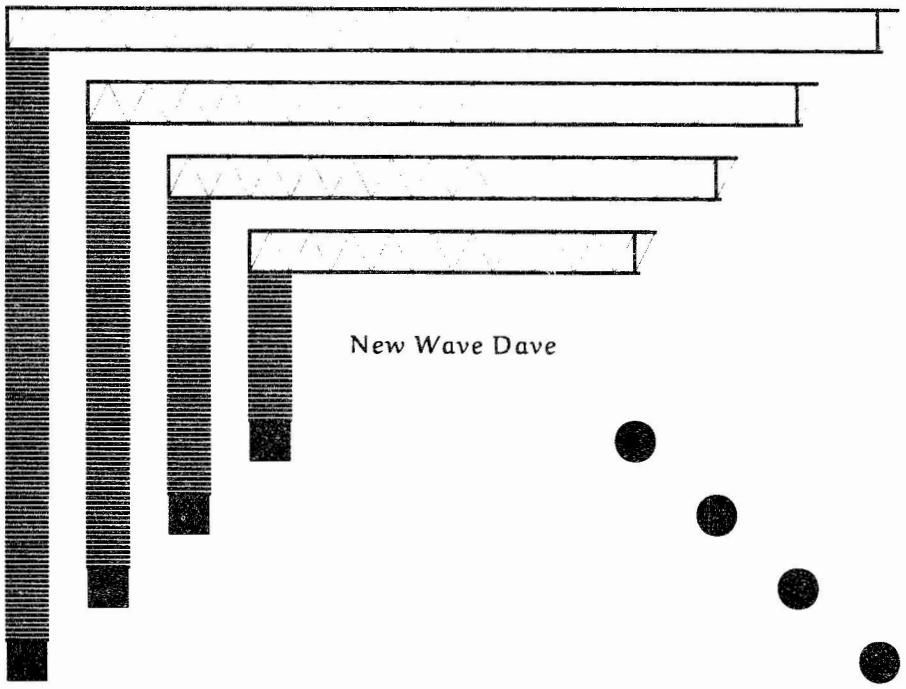
FOIL 46 — A third METAFONT Border.

This border enclosed the student's name instead of the **METAFONT** logo (Joey Tuttle is vice president of TEX Users Group). These are pretty fancy characters.



FOIL 47 — A fourth METAFONT border.

As are these.



FOIL 48 — A final METAFONT border.

There were no limits.

At the end of the course, the **METAFONT** class went out to an appropriate corner and had its picture taken. They first took a nice picture they could all treasure.

FOIL 49 — The METAFONT Class Picture.[6]

And then they made an italic font and took a picture of that.

[6] Photo by Jill Knuth; reprinted from [TUB], Vol. 5, No. 2, November 1984, p. 109.

A Sample of Fonts Included in PLAIN.TEX

AMR10, AMR9, ..., AMR5	Roman: ABCDefgh
AMMI10, AMMI9, ..., AMMI5	"Math italic": *ABCDefgh* $\tau\epsilon\chi$
AMSY10, AMSY9, ..., AMSY5	Symbols: $\lambda\delta \cap \cup () \infty \cdot \amalg$
AMEX10	Oversized and pieced symbols: $\left\{\sqrt{}\right\}$
AMBX10, AMBX9, ..., AMBX5	Boldface: **ABCDefgh**
AMTT10, AMTT9, AMTT8	"Typewriter": `ABCDefgh`
AMSL10, AMSL9, AMSL8	Slanted: *ABCDefgh*
AMTI10, AMTI9, ..., AMTI7	Text italic: *ABCDefgh*
AMSS10	Sans serif: ABCDefgh
AMSSBX10	Bold sans serif: **ABCDefgh**

FOIL 50 — Some of the fonts included on the standard distribution.

Knuth's original fonts were called Computer Modern Roman, or CMR. He is enlisting the help of some of the world's most noted font designers to improve these fonts' appearance. He expects to perfect the fonts in three stages, the first improved version being the AMR fonts. Next will come the BMR fonts, and finally, next year, we'll have CMR fonts again. This foil gives examples some of the current standard fonts.

Additional Fonts Created at AMS

Fraktur:

$$\mathfrak{B}^2 + \mathfrak{G}^3 + \mathfrak{N}^4 = \mathfrak{P}(\mathfrak{X})$$

Script:

$$\mathcal{B}^2 + \mathcal{G}^3 + \mathcal{N}^4 = \mathcal{P}(\mathcal{X})$$

Cyrillic:

Мещанский университет, находящийся на пути к Нескучному, праздновал на днях свой пятидесятилетний юблией. Кого возили в Титы или городскую больницу, тот, конечно, помнит здоровеннейший, трехэтажный домище по правую руку с вывеской «Богадельная и Мещанские училища» и тому наверное встречались на пути вереницы ученических пар, солидно прогуливаемых надзирателями.

FOIL 51 — Samples of Euler and Cyrillic.

At the request of the AMS, Herman Zapf, a pre-eminent font designer, undertook the design of a family of special mathematical fonts which he named **Euler** in honor of Leonhard Euler, a famous Eighteenth Century Swiss mathematician. These are some of the designs included in the Euler family. The fraktur and script fonts shown here are examples of Zapf's work.

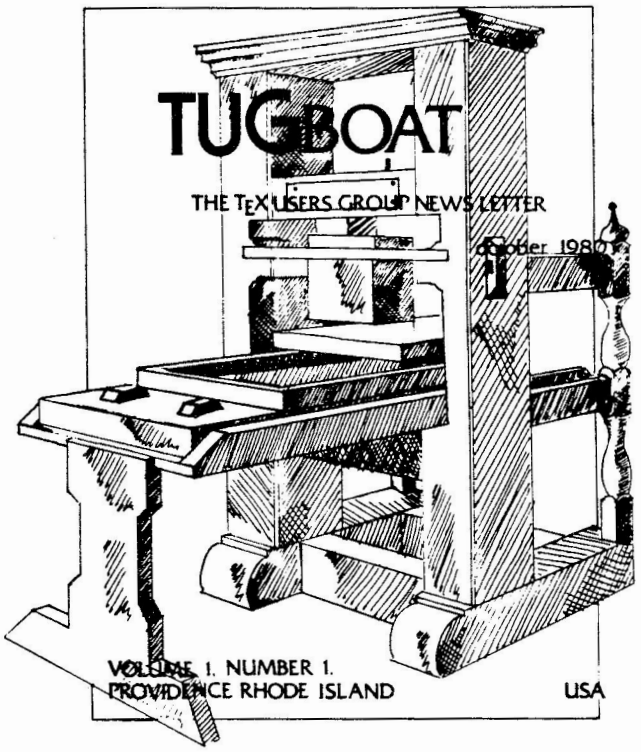
VIII. The TEX Users' Group

J

Joining the TEX Community

FOIL 52 — The TEX community; Title page of Appendix J of *The TEXbook*.

Richard Palais was instrumental in bringing to life the organization known as **TUG**, the TEX Users Group. TUG's purpose is to provide a forum for the exchange of information among TEX users by means of meetings and courses and a newsletter and other publications. TUG commissions expert TEX users to create useful macro packages, reference documents, and teaching materials. TUG provides TEXperts to teach courses in TEX usage at various companies and academic institutions. TUG expects soon to provide hotline assistance to people who need help installing and using TEX.



FOIL 53 — Cover of TUGboat, Vol. 1, No. 1

TUG's newsletter is, of course, called TUGboat. TUGboat is published two (and we hope, starting next year, three) times a year and contains useful articles by both the developers and the users of TEX and **METAFONT**. Knuth contributed to the most recent issue the interesting article on the **METAFONT** course from which my earlier foils were taken. New macro packages are published there, along with information about versions of TEX ported to new systems and the use of new output devices. TUGboat is edited by Barbara Beeton, the senior TEXpert at the AMS, who helped with many of the details of this presentation. She has now published in TUGboat many articles received from authors as TEX input on tape, bringing Palais' original hope closer to reality.

MacKAY, Pierre A.
Dept of Computer Science
Univ of Washington
FR-35
Seattle, WA 98195
206-545-2386
Arpanet: MacKay@Washington
Adjunct Professor of Computer Science
Computer: VAX 11/780, 11/750 (UNIX);
DEC 2060; CDC Cyber 170-750
Output device(s): Versatec; Alphatype CRS;
III VideoComp; Symbolics LGP-10
Applications: Arabic and similar character sets

FOIL 54 — A sample directory entry

Along with each issue of TUGboat come a list of all the corrections found to date to *The TEXbook* and a current membership list containing the names, addresses, phone numbers, and hardware of every member of TUG. This entry is that of the TEX Users Group president Pierre MacKay of the University of Washington, but all entries are as complete. TUG's $20 annual individual dues include a subscription to TUGboat. There are now about 1100 individual, and about 75 institutional, members of TUG.

TEX Users Group Membership List

CONTENTS

Site Coordinators, Steering Committee and members of other TUG Committees
Institutional Members
Addresses of TUG Members
Member names listed by institution
Member names listed by computer
Member names listed by output device
TEX consulting and production services for sale

FOIL 55 — Structure of Membership list

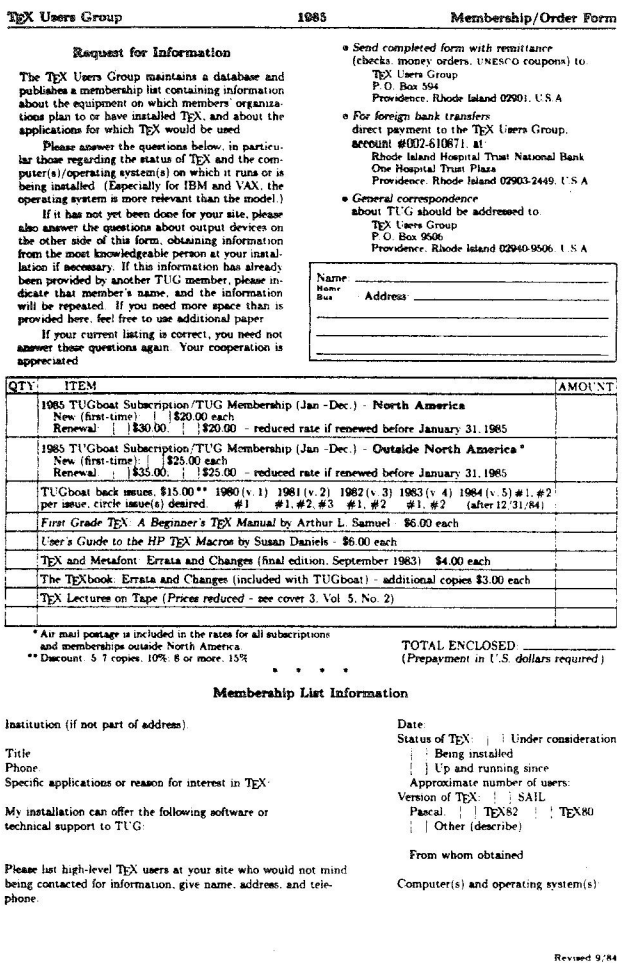
Site coordinators are experts in versions of TEX on specific hardware systems. They will give you free advice on installation, bugs, changes, problems in that version. They are sources of information on output devices and drivers. They may be suppliers of tapes for that version, as well (i.e., MacKay supplies the VAX/Unix version of TEX) at a nominal cost. Kellerman & Smith (listed in the directory) supply a VAX/VMS version with several output drivers not available on vanilla tapes.

TUG's committees are charged with improving the TEX environment by, e.g., soliciting manuscripts for improved documentation, new macro packages, user aids, etc.

Members are listed by institution, by hardware system, and by output device.

TUGboat carries notices of available TEX services, as well. Some service bureaus now exist solely to provide TEX services. Tyxset, in Reston, VA., is one mentioned in the Seybold Report. Textset, in Ann Arbor, Michigan, is another which markets TEX on both Apollo and Sun workstations with programs that allow the user to view to composed output on the screen. Others are running TEX on various PC's. Anyone can contact these service bureaus directly to inquire about particular services and costs.

TEX Users Group meetings are held annually, at Stanford at least through 1985. At these meetings, Knuth and his TEX associates describe the latest and projected TEX developments and hear suggestions and questions from the user community. Very many of TEX's present features have resulted from the dialog at meetings. Beginning and advanced TEX courses are given at the meetings as well as regionally.



TEX Users Group 1985 Membership/Order Form

Request for Information

The TEX Users Group maintains a database and publishes a membership list containing information about the equipment on which members' organizations plan to or have installed TEX, and about the applications for which TEX would be used.

Please answer the questions below, in particular those regarding the status of TEX and the computer(s)/operating system(s) on which it runs or is being installed. (Especially for IBM and VAX, the operating system is more relevant than the model.)

If it has not yet been done for your site, please also answer the questions about output devices on the other side of this form, obtaining information from the most knowledgeable person at your installation if necessary. If this information has already been provided by another TUG member, please indicate that member's name, and the information will be repeated. If you need more space than is provided here, feel free to use additional paper.

If your current listing is correct, you need not answer these questions again. Your cooperation is appreciated.

• *Send completed form with remittance* (checks, money orders, UNESCO coupons) to:
TEX Users Group
P.O. Box 594
Providence, Rhode Island 02901, U.S.A.

• *For foreign bank transfers*
direct payment to the TEX Users Group,
account #002-610871, at:
Rhode Island Hospital Trust National Bank
One Hospital Trust Plaza
Providence, Rhode Island 02903-2449, U.S.A.

• *General correspondence*
about TUG should be addressed to:
TEX Users Group
P.O. Box 9506
Providence, Rhode Island 02940-9506, U.S.A.

Name
Home / Bus
Address

QTY	ITEM	AMOUNT
	1985 TUGboat Subscription/TUG Membership (Jan.-Dec.) - **North America** New (first-time) ¦ ¦ $20.00 each Renewal ¦ ¦ $30.00 ¦ ¦ $20.00 - reduced rate if renewed before January 31, 1985	
	1985 TUGboat Subscription/TUG Membership (Jan.-Dec.) - **Outside North America** * New (first-time) ¦ ¦ $25.00 each Renewal ¦ ¦ $35.00 ¦ ¦ $25.00 - reduced rate if renewed before January 31, 1985	
	TUGboat back issues, $15.00** 1980 (v. 1) 1981 (v. 2) 1982 (v. 3) 1983 (v. 4) 1984 (v. 5) #1, #2 per issue, circle issue(s) desired. #1 #1, #2, #3 #1, #2 #1, #2 #2 (after 12/31/84)	
	First Grade TEX: A Beginner's TEX Manual by Arthur L. Samuel - $6.00 each	
	User's Guide to the HP TEX Macros by Susan Daniels - $6.00 each	
	TEX and Metafont: Errata and Changes (final edition, September 1983) - $4.00 each	
	The TEXbook: Errata and Changes (included with TUGboat) - additional copies $3.00 each	
	TEX Lectures on Tape (*Prices reduced* - see cover 3, Vol. 5, No. 2)	

* Air mail postage is included in the rates for all subscriptions and memberships outside North America.
** Discount: 5-7 copies, 10%; 8 or more, 15%

TOTAL ENCLOSED: _____
(*Prepayment in U.S. dollars required.*)

• • • •

Membership List Information

Institution (if not part of address):

Title:
Phone:
Specific applications or reason for interest in TEX:

My installation can offer the following software or technical support to TUG:

Please list high-level TEX users at your site who would not mind being contacted for information, give name, address, and telephone.

Date:
Status of TEX: ¦ ¦ Under consideration
¦ ¦ Being installed
¦ ¦ Up and running since
Approximate number of users:
Version of TEX: ¦ ¦ SAIL
Pascal ¦ ¦ TEX82 ¦ ¦ TEX80
¦ ¦ Other (describe)

From whom obtained:

Computer(s) and operating system(s):

Revised 9/84

FOIL 56 — The TUG Order Form, a table set by TEX.

This is an example of a much more complex table, set by TEX. The TUG order form lists most TEX publications, as well as details of membership.

TEX has, in fact, set new directions in computer typesetting. It really can provide a typesetting facility for almost everybody: a full ability to generate top quality typeset output is now in the hands of everyday people for everyday use. When one understands something of TEX's power, it is tempting to agree with Gordon Bell, then a Digital vice president, when he said, in his introduction to Knuth's book, *TEX and* **METAFONT**, *New Directions in Typesetting*, "Don Knuth's Tau Epsilon Chi (TEX) is potentially the most significant invention in typesetting in this century. It introduces a standard language for computer typography and in terms of importance could rank near the introduction of the Gutenberg press."

For more information regarding TEX or the TEX Users Group, write or call Ray Goucher, Business Manager, TEX Users Group, P.O. Box 9506, Providence, RI 02940, 401-272-9500, ext. 232.

References

[ACP] Knuth, Donald E., The Art of Computer Programming, Addison-Wesley, Vol. 2, 1969 and 1981 editions.

[MT] Knuth, Donald E., "Mathematical Typography", *Bulletin (New Series) of the American Mathematical Society* 1 (1979), 337–372.

[T&M] Knuth, Donald E., *TEX and* **METAFONT**, *New Directions In Typesetting*, Digital Press and the American Mathematical Society, 1979.

[CM] Knuth, Donald E., *The Computer Modern Family of Typefaces*, Stanford Computer Science Report STAN-CS-80-780 (January, 1980).

[LP] Knuth, Donald E., *Literate Programming*, Stanford Computer Science Report STAN-CS-82-981.

[TB] Knuth, Donald E., *The TEXbook*, Addison-Wesley and the American Mathematical Society, 1984.

[MB] Knuth, Donald E., *The* **METAFONT***book*, Addison-Wesley, 1985 (in preparation).

[LaTeX] Lamport, Leslie, *The LATEX Document Preparation System*, Addison-Wesley, 1985 (in preparation).

[FG] Samuel, Arthur, *First Grade TEX*, TEX Users Group, 1984.

[RS] Southall, Richard, "First principles of typographic design for document preparation", *TUGboat*, Vol. 5 (1984), No. 2, pp. 79–90.

[Joy] Spivak, Michael, *The Joy of TEX*, the American Mathematical Society, 1980, new edition in preparation, 1985.

[Sey] *TEX On A Microcomputer*, The Seybold Report on Publishing Systems, Vol. 14, No. 4, (Oct 29, 1984), Seybold Publications, Inc.

[TUB] *TUGboat, the Newsletter of the TEX Users Group*, TEX Users Group, c/o American Mathematical Society, P.O. Box 9506, Providence, RI, 02904.

TEX82 ORDER FORM

The latest official versions of TEX software and documents are available from Maria Code by special arrangement with the Computer Science Department of Stanford University.

Nine different tapes are available. The generic distribution tape contains the source of TEX82 and WEB, the test program, a few "change" files, the collection of fonts in TFM format, and other miscellaneous materials; a PASCAL compiler will be required to install programs from a generic tape. The AMS-TEX macro package is included on the TEX distribution tapes; other macro packages, including LATEX and HP TEX, will be added as they become available. The special distribution tapes are for the indicated systems only, and should be ordered for these systems instead of a generic tape. Two tapes are PXL font collections covering various magnifications at 200/240 dots/inch and 300 dots/inch respectively. The METAFONT tape contains the SAIL source for the METAFONT program and includes the .MF source files.

Each tape will be a separate 1200 foot reel which you may send in advance or purchase (for the tape media) at $10.00 each. Should you send a tape, you will receive back a different tape. Tapes may be ordered in ASCII or EBCDIC characters. You may request densities of 6250, 1600 or 800 (800 is discouraged since it is more trouble to make).

The tape price of $82.00 for the first tape and $62.00 for each additional tape (ordered at the same time) covers the cost of duplication, order processing, domestic postage and some of the costs at Stanford University. Extra postage is required for first class or export.

Manuals are available at the approximate cost of duplication and mailing. Prices for manuals are subject to change as revisions and additions are made. It is assumed that one set of manuals will suffice you. If you require more than two sets, please write for prices since we must ask for more money for postage and handling.

Please send a check or money order (payable on a US bank) along with your order if possible. Your purchase order will be accepted, as long as you are able to make payment within 30 days of shipment. Please check this out before sending a purchase order since many large firms seem to be unable to make prompt payment (or don't worry about it).

The order form contains a place to record the name and address of the person who will actually use the TEX tapes. This should *not* be someone in the purchasing department.

Your order will be filled with the most recent versions of software and manuals available from Stanford at the time your order is received. If you are waiting for some future release, please indicate this. Orders are normally filled within a few days. There may be periods (like short vacations) when it will take longer. You will be notified of any serious delays. If you want to inquire about your order you may call Maria Code at (408) 735-8006 between 9:30 a.m. and 2:30 p.m. West Coast time.

If you have questions regarding the implementation of TEX or the like, you must take these to Stanford University or some other friendly TEX user.

Now, please complete the order form on the reverse side.

TEX82 ORDER FORM

** **TAPES** ** density (6250, 1600 or 800) = _____

TEX generic distribution tapes (PASCAL compiler required):

_____ ASCII format _____ EBCDIC format

TEX distribution tapes in special formats:

_____ VAX/VMS Backup format _____ IBM VM/CMS format

_____ DEC 20/Tops-20 Dumper format _____ * IBM MVS format

* Not yet available; call before ordering

Font tapes:

_____ Font library (200/240 dots/inch) _____ Font library (300 dots/inch)

_____ METAFONT (SAIL compiler required)

_____ Total number of tapes.

Tape costs: $82.00 for first tape; $62.00 for each additional.

Tape cost = $ _____

Media costs: $10.00 for each tape required.

Media cost = $ _____

** **MANUALS** **

_____ TEX82 – $20.00 _____ Test Manual – $8.00

_____ WEB – $10.00 _____ TEXware – $8.00

_____ TEXbook – $20.00 _____ LATEX (preliminary edition) – $8.00

Manuals cost = $ _____

California orders only: add sales tax = $ _____

Domestic book rate: no charge.
Domestic first class: $2.50 for each tape and each manual.
Export surface mail: $2.50 for each tape and each manual.
Export air mail to North America: $4.00 each.
Export air mail to Europe: $7.00 each.
Export air mail to other areas: $10.00 each.

Postage cost = $ _____

(make checks payable to Maria Code) Total order = $ _____

Name and address for shipment: Person to contact (if different):

_____ _____

_____ _____

_____ _____

_____ _____

Telephone _____

Send to: Maria Code, DP Services, 1371 Sydney Dr., Sunnyvale, CA 94087

Request for Information

The TEX Users Group maintains a database and publishes a membership list containing information about the equipment on which members' organizations plan to or have installed TEX, and about the applications for which TEX would be used.

Please answer the questions below, in particular those regarding the status of TEX and the computer(s)/operating system(s) on which it runs or is being installed. (Especially for IBM and VAX, the operating system is more relevant than the model.)

If it has not yet been done for your site, please also answer the questions about output devices on the other side of this form, obtaining information from the most knowledgeable person at your installation if necessary. If this information has already been provided by another TUG member, please indicate that member's name, and the information will be repeated. If you need more space than is provided here, feel free to use additional paper.

If your current listing is correct, you need not answer these questions again. Your cooperation is appreciated.

- *Send completed form with remittance* (checks, money orders, UNESCO coupons) to:
  TEX Users Group
  P. O. Box 594
  Providence, Rhode Island 02901, U.S.A.
- *For foreign bank transfers* direct payment to the TEX Users Group, account #002-610871, at:
  Rhode Island Hospital Trust National Bank
  One Hospital Trust Plaza
  Providence, Rhode Island 02903-2449, U.S.A.
- *General correspondence* about TUG should be addressed to:
  TEX Users Group
  P. O. Box 9506
  Providence, Rhode Island 02940-9506, U.S.A.

Name: _____
Home [ ]
Bus. [ ] Address: _____
_____
_____
_____

QTY	ITEM	AMOUNT
	1985 TUGboat Subscription/TUG Membership (Jan.–Dec.) – **North America** New (first-time): [ ] $20.00 each Renewal: [ ] $30.00; [ ] $20.00 – reduced rate if renewed before January 31, 1985	
	1985 TUGboat Subscription/TUG Membership (Jan.–Dec.) – **Outside North America** * New (first-time): [ ] $25.00 each Renewal: [ ] $35.00; [ ] $25.00 – reduced rate if renewed before January 31, 1985	
	TUGboat back issues, $15.00 ** per issue, circle issue(s) desired: 1980 (v. 1) #1; 1981 (v. 2) #1, #2, #3; 1982 (v. 3) #1, #2; 1983 (v. 4) #1, #2; 1984 (v. 5) #1, #2 (after 12/31/84)	
	First Grade TEX: A Beginner's TEX Manual by Arthur L. Samuel – $6.00 each	
	User's Guide to the HP TEX Macros by Susan Daniels – $6.00 each	
	TEX and Metafont: Errata and Changes (final edition, September 1983) – $4.00 each	
	The TEXbook: Errata and Changes (included with TUGboat) – additional copies $3.00 each	
	TEX Lectures on Tape (*Prices reduced* – see cover 3, Vol. 5, No. 2)	

* Air mail postage is included in the rates for all subscriptions and memberships outside North America.
** Discount: 5–7 copies, 10%; 8 or more, 15%

TOTAL ENCLOSED: _____
(*Prepayment in U.S. dollars required*)

* * * *

Membership List Information

Institution (if not part of address):

Title:
Phone:
Specific applications or reason for interest in TEX:

My installation can offer the following software or technical support to TUG:

Please list high-level TEX users at your site who would not mind being contacted for information; give name, address, and telephone.

Date:
Status of TEX: [ ] Under consideration
[ ] Being installed
[ ] Up and running since
Approximate number of users:
Version of TEX: [ ] SAIL
Pascal: [ ] TEX82 [ ] TEX80
[ ] Other (describe)

From whom obtained:

Computer(s) and operating system(s):

Please answer the following questions regarding output devices used with TEX unless this form has already been filled out by someone else at your installation. Use a separate form for each output device.

Name _____ Institution _____

A. Output device information
Device name
Model
1. Knowledgeable contact at your site
Name
Telephone
2. Device resolution (dots/inch)
3. Print speed (average feet/minute in graphics mode)
4. Physical size of device (height, width, depth)

5. Purchase price
6. Device type
[ ] photographic [ ] electrostatic
[ ] impact [ ] other (describe)

7. Paper feed [ ] tractor feed
[ ] friction, continuous form
[ ] friction, sheet feed [ ] other (describe)

8. Paper characteristics
a. Paper type required by device
[ ] plain [ ] electrostatic
[ ] photographic [ ] other (describe)

b. Special forms that can be used [ ] none
[ ] preprinted one-part [ ] multi-part
[ ] card stock [ ] other (describe)

c. Paper dimensions (width, length)
maximum
usable
9. Print mode
[ ] Character: ( ) Ascii ( ) Other
[ ] Graphics [ ] Both char/graphics
10. Reliability of device
[ ] Good [ ] Fair [ ] Poor
11. Maintenance required
[ ] Heavy [ ] Medium [ ] Light
12. Recommended usage level
[ ] Heavy [ ] Medium [ ] Light
13. Manufacturer information
a. Manufacturer name
Contact person
Address

Telephone
b. Delivery time
c. Service [ ] Reliable [ ] Unreliable

B. Computer to which this device is interfaced
1. Computer name
2. Model
3. Type of architecture *
4. Operating system

C. Output device driver software
[ ] Obtained from Stanford
[ ] Written in-house
[ ] Other (explain)

D. Separate interface hardware (if any) between host computer and output device (e.g. Z80)
1. Separate interface hardware not needed because:
[ ] Output device is run off-line
[ ] O/D contains user-programmable micro
[ ] Decided to drive O/D direct from host
2. Name of interface device (if more than one, specify for each)

3. Manufacturer information
a. Manufacturer name
Contact person
Address

Telephone
b. Delivery time
c. Purchase price
4. Modifications
[ ] Specified by Stanford
[ ] Designed/built in-house
[ ] Other (explain)

5. Software for interface device
[ ] Obtained from Stanford
[ ] Written in-house
[ ] Other (explain)

E. Fonts being used
[ ] Computer Modern
[ ] Fonts supplied by manufacturer
[ ] Other (explain)

1. From whom were fonts obtained?

2. Are you using Metafont? [ ] Yes [ ] No

F. What are the strong points of your output device?

G. What are its drawbacks and how have you dealt with them?

H. Comments – overview of output device

* If your computer is "software compatible" with another type (e.g. Amdahl with IBM 370), indicate the type here

An AI Project Case Study – The First Six Months

Don Rosenthal

The following pages are handouts from a talk given at DECUS in Anaheim. Specifically, you'll find copies of the viewgraphs, and the bibliography of knowledge based systems (which we ran out of at the session—apologies).

An earlier draft of the bibliography was published in the July, 1984 ACM SIGART newsletter by D. Sriram of the Carnegie-Mellon University Design Research Center. It is distributed with the permission of the author.

Note that a follow up to the R1 paper by J. McDermott (see page 9 of the bibiography) entitled "R1 Revisited: Four years in the Trenches" was published in the Fall 1984 issue of "The AI Magazine".

An AI Project Case Study-The First Six Months

Don Rosenthal
Space Telescope Science Institute
Homewood Campus
Baltimore. MD

Overview

- Report on first 6 months of a real-world AI project
- "Real World": not a research project. we're producing working systems
- Presentation will cover
  - evaluation of need for AI
  - choice of tools to meet the need
  - training to use the tools
  - early prototyping

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Intent of Presentation

- To help calibrate what's involved in starting from scatch.
- For anyone considering AI solutions for the first time
- This Work reported has been accomplished since last DECUS (June '84)
- One person full time

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Context of Project

- Ground support subsystems for Space Telescope
- Specifically in planning and scheduling use of telescope.
- Planning and scheduling includes everything from receipt of proposals from astronomers to generation of spacecraft commands.

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

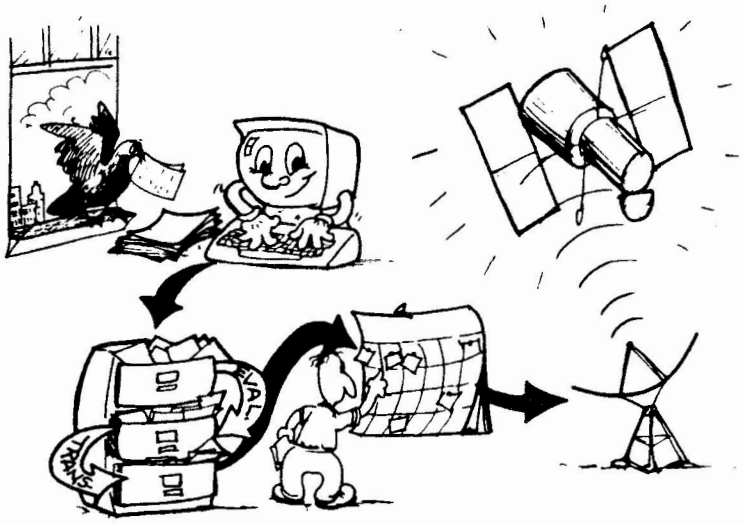
Space Telescope

- Orbiting telescope
- Five Instruments on board designed to do astronomy
- In addition, guidance system can be used for positional astronomy.
- Sets of instruments can be used simultaneously.

Problems in Planning and Scheduling

- Proposals arrive describing scientific objectives.
- Schedules must be prepared that direct S/C computers to perform observations.
- How can a large set of proposals be transformed into an acceptably "good" schedule of S/C functions ?
- What is an acceptably good schedule ???

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984



Constraints, Complications, and Interacting Subsystems

- Proposed observing time to available observing time ratio - 15:1
- Instruments have many modes of operation
- It takes time to slew to targets
- Targets are routinely occulted by sun, moon, Earth
- Observing any given target at different times may alter the observation (length and S/N)
- Instruments must be warmed up before use
- All instruments cannot be warm simultaneously (limited power)
- ST orbits under stationary relay satellites (limited communications)
- ST must avoid pointing at bright objects (limited sky coverage)
- ST must keep solar panels pointed at the Sun
- ...

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Still More Problems & Constraints...

- Combinatoric explosion of the solution space
- Can't generate an optimal solution
- Can't easily generate a metric for "good" solution
- Even subproblems are hard: Proposal Entry Processing is analogous to a PC board construction system whose input is a schematic...

Problem Definition

- Looked to AI for techniques to bound search of solution space
- Thought it roughly analogous to "pruning" in game playing
- Found that Planning was an established subfield in AI

Planning in AI [1]

- "Preparing a program of actions to be carried out to achieve goals."
- "A planner is required to construct a plan that achieves goals without consuming excessive resources or violating constraints."
- "key problems:...
  - must be able to act tentatively [non-deterministic]
  - if details are overwhelming, the planner must be able to focus on the most important considerations
  - a planner must operate in the face of an uncertain planning context
  - a planner must attend to interacting subgoals"

[1] Building Expert Systems, Hayes-Roth, Waterman, Lenat eds

On the Right Track

- Reassured that my ideas were consistent w/ AI literature
- Found that there was relevant work reported in the literature
- "Planning and Meta-Planning" by Mark Stefik (MOLGEN)
- "Planning in Time: Windows and Durations for Activities and Goals" by Steven Vere (DEVISER)

MOLGEN: Stefik

- Design of experiments in molecular genetics
- Developed a hierarchy of abstraction for design tasks
  - Strategy space
  - Design space
  - Laboratory space
- Dual control, one context for as long as possible, then the other, etc
- As design is closely related to planning, MOLGEN can serve as a model, a good starting point

DEVISER: Vere

- Paper from JPL on scheduling planetary encounters for Voyagers
- Generates parallel plans to achieve goals with imposed time constraints
- Starts with traditional blocks world planning
- Adds timing with window construct
- Refines model for "space world" to include some physical and timing constraints
- Enough in common with ST to be relevant

An AI Project Case Study - The First Six Months Don Rosenthal December 1 1984

Approach

- Decided on two areas of application of AI:
- Proposal Transformation
- Actual schedule construction

Proposal Transformation

- Appears to be appropriate problem for expert system technology
- More bounded problem than scheduling
- There are resident experts

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Scheduling

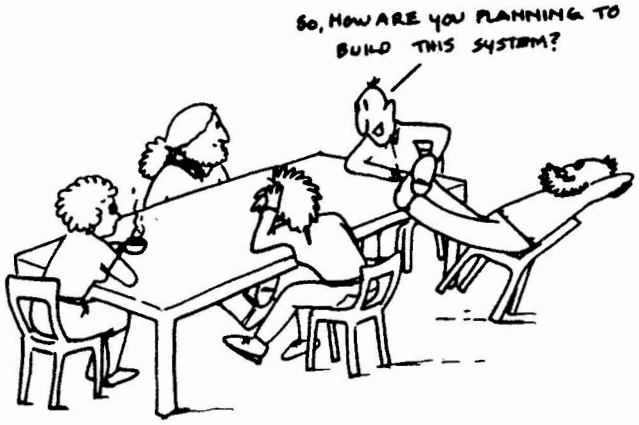
- Can't build an expert system w/o an expert
- Need to understand the problem before attempting to solve it
- "exploratory" programming: Use AI to probe problem

Scheduling continued

- Hybrid approach for exploring problem:
- Scheduling primitives (mechanics) in C
- Existing FORTRAN library for numerical problems
- Can build different drivers on top of primitives:
  - manual (graphical)
  - C (algorithmic)
  - AI (rule-based)

Tools

- What language ?
- As the traditional AI language. LISP was first to be considered...
- Ran up against "LISPophobia"



LISP continued...

- Did not discount LISP
- We have several versions available at STScI
- Some staff members have been introduced through EMACS
- Felt that I should continue looking. to be sure...

Prolog

- Read Clocksin & Mellish text (Programming in Prolog)
- Was intrigued. but...
- Seemed too low level for my particular uses (it's real good for expressing relationships. for example...)
- Also. did not find vendors for a VMS Prolog (summer '84)

Rule Based Languages

- In exploring available languages. I came upon OPS5 and Rosie.
- Learned that DEC was to offer OPS5
- Transformation problem seemed oriented to a production system solution
- Parts of scheduling system also (constraints and restrictions)

LISP versus OPS5

- Both vendor supported
- LISP
  - more flexible, more powerful, more general, but
  - would have to build the "inference engine" myself
- OPS5
  - much more limited, but
  - alot that I specifically needed is built in
  - call-in, call-out, seemed well supported

Programmatic Considerations

- OPS5 is CHEAP !
- DEC Proven (R1 papers)
- Vendor support seems assured
- Good guess that support environment will be developed by DEC they need it themselves.
- Technical: appropriate tool
- Managerial: low risk

Tools (continued...)

- Ordered OPS5 soon after Cincinnati DECUS
- Intent was to use C for lower level mechanical operations
- FORTRAN tools existed for numerical calculations (no new FORTRAN to be written)
- MMS for updating segmented rule base ("modular compilation")
- ??? Test Manager for regression testing of state machine. Does DEC use it ???

OPS5 Support Environment

- Has an interpretive trace facility built in
- As goals are essentially states, one builds a monolithic state machine
- Will DEC Provide support tools??
  - consistency checker
  - rule editors
  - Meta Rules: presently cannot operate explicitly on Conflict Set..

Training

- My background—no formal AI but have done hardware and software for 15 years and once built a LISP interpreter
- Found reading knowledge of LISP essential-took a refresher course
- Many good reference texts available now. with substantial bibliographies.
- Read like crazy - excellent way to become fluent in the relevant concepts.
- Handout: an AI bibligraphy from SIGART
- Took a few weeks to sit down and program

The First Prototypes

- Within week of receipt was writing small OPS5 programs
- Towers of Hanoi
- Scientific Instrument Adviser
- Blocks world planner

An AI Project Case Study - The First Six Months Don Rosenthal December 1 1984

Towers of Hanoi

- Good first OPS5 problem
- Only two rules. iterative solution
- Most of program was data structures and initialization
- No goal (context) switching

An AI Project Case Study - The First Six Months Don Rosenthal December 1 1984

SI Chooser

- By second week had written this program
- Excellent testbed for information transfer
- Explored three areas
  - control structures within OPS5
  - call out to other languages
  - modular compilation

An AI Project Case Study - The First Six Months Don Rosenthal December 1 1984

SI Chooser, continued

- Wrote first version in one sitting (4 hours) after playing 20 questions with an expert
- Many stubs, but first version could do as much as I could (ie all my derived knowledge had been coded into it.
- Was trivially expandable (an astronomer watching me run it suggested four new rules–immediately implemented)
- Incorporated simple-minded explanation facility
- Recognized "impossible" observations
- Knew when it was stumped due to incomplete rule set

SI Chooser, continued

- Rewrote about five different ways, exploring different control constructs
- Added C I/O routines for asking questions of users and validating answers.
- Incorporated MMS makefile to recompile either C or OPS5 ("modular compilation")

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Blocks World Planner

- Needed to prove applicability to planning
- Block stacker is planning archetype, even reported in JPL Voyager planner (in LISP)
- Using description from that paper, implemented OPS5 version in an afternoon
- Incorporates simplication from LISP version
- Uses only 9 rules

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Acceptance of OPS5

- Not only was I encouraged, but many non-computer types saw usefulness of OPS5 for their work
- Operations Astronomers familiar with constraints, restrictions, and good practices
- Researchers interested in automating recognition of opportunities to do parallel science

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Non-AI prototypes

- Scheduling primitives prototyped in C
- Three man-months (conversion from Pascal, plus enhancements)
- Essential, because of the many things OPS5 is not suited for.
- One-half of the preparation period.

The Next Six Months

- Intensive OPS5 prototyping
- Prove feasibility for transformation
- Expand transformation capabilities
- I/F OPS5 code with database
- Prototype scheduler driver in OPS5 using MOLGEN control hierarchy
  - C lowest ("laboratory") level
  - OPS5 design level
  - OPS5 strategy level
  - Clean work-around for absence of meta-rules

An AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

Results

- In 6 months we were able to:
  - identify need for AI techniques
  - learn enough through the literature to be able to plan the next steps
  - find, evaluate, acquire, and test tools
  - build necessary conventional system functions
  - learn OPS5 and solve several short problems of differing types
  - develop a design model for the AI system parts
  - generate a strategy for system design and implementation which includes proving technical feasibility AND reasonable fallback

n AI Project Case Study - The First Six Months Don Rosenthal December 1, 1984

A Bibliography on Knowledge-Based Expert Systems in Engineering[1]

D. Sriram
Design Research Center &
Civil Engineering and Construction Robotics Laboratories
Carnegie-Mellon University
Pittsburgh, PA 15213

Introduction

The number of papers published in the applications of knowledge-based expert systems (KBES) to engineering problems in the last decade reflects the interest being shown in the engineering community. The intent of this report is to provide an annotated bibliography of the applications of KBES in engineering. The first four sections deal with applications in Civil (including Architecture and Geology), Chemical, Electrical and Computer Engineering, Mechanical. Some papers which are common to engineering design, in general, are outlined in Section 5. A number of domain independent tools are discussed in Section 6. Section 7 contains a list of books for general reading. A list of relevant conferences and journals is provided in Section 8.

The bibliography is by no means complete and the author would appreciate pointers to other literature in the area for inclusion in a future update. Some of these references are taken from NTIS citations from the INSPEC data base; these references contain the word [NTIS].

1 Architecture, Civil Engineering, and Geology

Bennett, J., Creary, L., Engelmore, R. and Melosh, R.
SACON: A Knowledge-based Consultant for Structural Analysis
Technical Report STAN-CS-78-699, Stanford University, September 1978.

Bennett, J. and Engelmore R.
SACON: A Knowledge-based Consultant for Structural Analysis
In *Proceedings Sixth IJCAI*, pages 47-49, 1979.
SACON is an expert program that advises a structural engineer in the use of modeling options for MARC, a non-linear structural analysis program. It is implemented in EMYCIN. It does not have any interface with the analysis program.

Bonnet, A., and Dahan, C.
Oil-Well Data Interpretation Using Expert System and Pattern Recognition Techniques
In *Proceedings Eighth IJCAI*, pages 185 - 189, 1983.

[1] Forthcoming DRC technical report. A number of additions have been made to the bibliography first published in SIGART newsletter, July 84.

LITHO, a program for interpreting oil well data is described. The output from the program is a litholog, a description of rocks encountered in a well. LITHO is being developed at Schlumberger, France. The knowledge-base contains about 500 rules.

Cobb, J. E.

A Microcomputer Approach to Contract Management Using AI
Unpublished Master's Thesis, University of Colorado, Boulder, CO, 1984.

The knowledge-base and logic for the development of DSCAS, which is intended to provide legal advice for construction claims, is developed. Currently DSCAS is designed for "Differing Site Conditions" clause of the U. S. Government standard general conditions to a construction contract.

Cuena, J.

The Use of Simulation Models and Human Advice to Build an Expert System for the Defense and Control of River Floods
In *Proceedings Eigth IJCAI*, pages 246-249, 1983.

A conceptual framework for an expert system to aid in the operation of flood control and plan civil defense in flood prone areas is provided. The rules are described based on a set of mathematical models. System currently pursued by Spanish Ministry of Public Works.

David, H.

An Analysis of Expert Thinking
International Journal Man-Machine Studies, Vol. 18, pages 1-47, 1983.

Deals with how human experts acquire, understand and use knowledge in the domain of geology, in particular petroleum geology.

Davis, R. et al

The Dipmeter Advisor: Interpretation of Geologic Signals
In *Proceedings Seventh IJCAI*, pages 846-849, 1981.

Paper presents a feasibility study on the use of expert systems for well log analysis. The paper published in the eighth IJCAI describes a more recent implementation.

Duda, R. O., Gaschnig, J. and Hart, P. E.

Model Design in the Prospector System for Mineral Exploration
In Michie, D. (editor), *Expert Systems in the Micro Electronic Age*, pages 153-167, University of Edinburgh, Scotland, 1979.

PROSPECTOR aids the geologist to select mineral deposits. Currently it has more than 1000 rules in its knowledge-base.

Eastman, C. M.

Automated Space Planning
Artificial Intelligence Vol. 4, No. 1, Spring 1973.

One of the first papers that addresses the application of heuristics to space planning.

Fjellheim, R. and Syversen, P.

An Expert System for SESAM-69 Program Selection
Computas Report 83-6010, January 1983. (A. S. Computas, P. O. Box, 310, 1322 HOVIK, Norway)

Describes an expert system front end for a large finite element program SESAM-69, developed by A. S. Computas. Patterned after SACON. Implemented in EMYCIN.

Gaschnig, J., Reboh, R. and Reiter, J.

Development of a Knowledge-Based Expert System for Water Resource Problems
Technical Report SRI Project 1619, SRI International, August 1981.

Describes an intelligent interface (HYDRO) for selecting numerical values of parameters that are input to a simulation program (HSPF).

Gero, J. S. and Coyne, R.

The Place of Expert Systems in Architecture
In *Proceedings CADD-84*, U. K., 1984.

An introduction to expert systems, along with some prototype applications. Implications to synthesis are explored.

Hammond, P. and Howarth, R.

A Rule-Based Approach to Geological Knowledge
Research Report, Imperial College of Science and Technology, U. K., 1984.

Consists of two knowledge-bases. The first one was directly transferred from PROSPECTOR, while second was written by an expert for determining the suitability of sites for dam construction. Implemented in PROLOG.

Hollander, C. R., Iwasaki, Y., Courteille, J-M., Fabre, M.

The Drilling Advisor
In *Proceedings of Trends and Applications on Automating Intelligent Behavior: Applications and Frontiers*, pages 21-27, May 1983.

Provides diagnosis and therapy for problems encountered by the drilling mechanism while drilling. Currently the system has around 250 rules for diagnosing possible problems associated with the drill being stuck in the bore hole. It is implemented in KS300, a copyrighted version of EMYCIN.

Ishizuka, M., Fu, K. S. and Yao, J. T. P.

Inexact Inference for Rule-based Damage Assessment of Existing Structures
Technical Report CE-STR-81-5, Purdue University, February 1981 (also see Seventh IJCAI proceedings).

Ishizuka, M., Fu, K. S. and Yao, J. T. P.

Rule-based Damage Assessment System for Existing Structures
SM Archives Vol. 8, pages 99-118, Martinus Nijhoff Publishers, The Hague, 1983.

The above two papers describe SPERIL-I, a rule-based expert system. SPERIL-I addresses the issue of damage assessment of structures after earthquakes and other possible hazardous events.

Kruppenbacher, T. A.

A Microcomputer Approach to Contract Management Using AI
Unpublished Master's Thesis, University of Colorado, Boulder, CO, 1983.

Describes the implementation details of DSCAS, which is implemented in ROSIE. See reference by Cobb.

Lansdown, J.

Expert Systems: Their Impact on the Construction Industry
RIBA Conference Fund, U. K., 1982.

Presents a number of potential applications for KBES in the construction industry.

Lopez, L. A., Elam, S. L., and Christopherson, T.
SICAD: A Prototype Implementation System for CAD
In *Proceedings of the ASCE Third Conference on Computing in Civil Engineering*, San Diego, California, pages 84-93, April 1984.
Describes a framework for the development of a KBES for providing an interface between standards governing engineering design and CAD programs.

MacCallum, K. J.
Creative Ship Design by Computer
In Rogers, D. F., Nehrling, B. C. and Kuo, C. (editors), *Computer Applications in the Automation of Shipyard Operation and Shipyard Design IV*, IFIP82, North-Holland Publishing Company, 1982.

MacCallum, K. J.
A Knowledge-base for Engineering Design Relationships
In *Expert Systems 82*, Technical Conference of the BCS SGES, U. K., 1982.
Deals with the development of a KBES for ship design. Also attempts to incorporate an element of learning in the system.

Manheim, M. L.
HIERARCHICAL STRUCTURE: A Model of Design and Planning Processes
MIT Press, Cambridge, Mass., 1966.
The concept of hierarchical design was first implemented by Manheim for determining highway locations. A classical work in the area.

Markusz, Z.
Design in Logic
Computer Aided Design, Vol. 14, No. 6, Pages 335-343, November 1982.
(other references to this work can be found in *Logic Programming* Clark, K. L. and Tarnlund, S. A. (editors), Academic Press, 1982.)
Describes the use of logic in architectural design. Implementation language is PROLOG.

Melosh, R. J., Marcal, P. V. and Berke, L.
Structural Analysis Consultation using Artificial Intelligence
In *Research in Computerized Structural Analysis and Synthesis*, NASA, Washington, D. C., October 1978.
Illustrates an application of SACON.

Ohsuga, S.
A New Method of Model Description - Use of Knowledge Base and Inference
In Bo, K. and Lillehagen, F. M. (editors), *CAD Systems Framework*, IFIP83, North-Holland Publishing Company, 1983.
A methodology to represent the model building process in building design is proposed. Knowledge is represented in terms of expanded predicate logic and interfaced with a relational database.

Rehak, D.
Expert Systems in Water Resource Management
In James, W. and Torno, H. (editors), *Proceedings ASCE Conference on Emerging*

Techniques in Storm Water Flood Management, Niagara on the Lake, Ontario, Canada, October 31 - November 4, 1983.

Current systems in water resource management are described.

Rehak, D. and Lopez, L. A.

Computer-Aided Engineering : Problems and Prospects
Civil Engineering System Laboratory Research Series 8, University of Illinios, July 1981.

Potential use of KBES for the development of an integrated structural design system is addressed.

Rivlin, J. M., Hsu, M. B. and Marcal, P. V.

Knowledge-based Consultant for Finite Element Analysis
Technical Report AFWAL-TR-80-3069, Flight Dynamics Laboratory (FIBRA), Wright-Patterson Airforce, May 1980.

A KBES implemented in FORTRAN and interfaced to the MARC non-linear analysis program.

Radford, A. D., Hung, P. and Gero, J. S.

New Rules of Thumb from Computer-Aided Structural Design: Acquiring Knowledge for Expert Systems
In *Proceedings CADD-84*, U. K., 1984.

Pareto's optimization technique is proposed as an aid to the knowledge-acquisition process and illustrated using the floor system design as a paradigm.

Smith, R. G., and Baker, J. D.

The Dipmeter Advisor System: A Case Study in Commercial Expert System Development
In *Proceedings Eigth IJCAI*, pages 122-129, 1983.

The development of Dipmeter Advisor, a KBES for oil-well interpretation, is described. Dipmeter Advisor is being developed by Schlumberger-Doll research, Connecticut, U.S.A.

Sriram, D., Maher, M. L., Bielak, J. and Fenves, S. J.

Expert Systems for Civil Engineering - A Survey
Technical Report R-82-137, Department of Civil Engineering, Carnegie-Mellon University, June 1982.

Written as an introduction to KBES for civil engineers. A number of current expert systems, KBES building tools and potential applications in structural and geotechnical engineering are described.

Sriram, D., Maher, M. and Fenves, S

Applications of Expert Systems in Structural Engineering
In *Proceedings Conference on Artificial Intelligence*, pages 379-394, Oakland University, Rochestor, MI, April 1983.

Applications of KBES to various phases of structural design are discussed.

Stanford, G.

Potential Applications of Expert Systems in Geotechnical Engineering
Master's Thesis, Department of Civil Engineering, Carnegie-Mellon University, April, 1983.

Potential applications in geotechnical engineering, specifically in the domain of Landslide engineering, are addressed. The author also relates his experience of knowledge acquisition from a domain expert and from literature.

Weiss, S. M. and Kulikowski, C. A.

Building Expert Programs for Controlling Complex Programs
In *Proceedings 2nd NCAI*, pages 322-326, 1982.

A KBES for well log analysis is described.

2 Chemical Engineering and Material Sciences

Banares, R.

Development of a Consultant for Physical Property Predictions
Master's Thesis, Department of Chemical Engineering, Carnegie-Mellon University, May 1982.

A KBES for selecting appropriate analytic program that is used to evaluate the physical properties of certain chemical substances is described.

Basden, A. and Kelly, B. A.

An Application of Expert Systems Techniques in Materials Engineering
In *Proceedings Colloqium on Application of Knowledge-Based Systems*, London, U. K., 1982 (See also International Jl. of Man-Machine Studies).

Describes a prototye KBES to predict the risk of stress corrosion cracking.

Chester, D. L., Lamb, D. E. and Dhurjati, P.

An Expert System Approach to On-line Alarm Analysis in Power and Process Plants
In *Proceedings Computers in Engineering, A. S. M. E.*, pages 345-351, August 1984, Las Vegas, Nevada.

FALCON, currently under development, is a KBES for diagnosing faults in a process plant. It combines both the causal model and the (surface) production rule approach. Implementation language is Franz LISP.

Grimes, L. E., Rychener, M. and Westerberg, A. W.

The Synthesis and Evolution of Networks of Heat Exchange that Feature the Minimum Number of Units
Chemical Engineering communications, Vol. 14, 1982.

HEATEX aids in the construction of networks that minimize energy requirements by allowing the exchange of heat among various process streams.

Peate, J.

Building Human Judgement into Computer Programs
Process Engineering, January 1984.

A general discussion on the potential applications of KBES in chemical engineering.

Powers, G. J.

Non-numerical Problem Solving Methods in Computer-Aided Design
In *IFIPS Conference on Computer-Aided Design*, Eindhoven, The Netherlands, 1972.

Outlines applications of AI techniques to design.

3 Electrical and Computer Engineering

Bellon, C., Robach, C., and Saucuer, G.

An Intelligent Assistant for Test Program Generation: The SUPERCAT system
In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 32-33, September 1983.

A conceptual framework for a KBES to assist in generating test programs for complex VLSI circuits.

Basden, A. and Kelly, B. A.

DART: An Expert System for Computer Fault Analysis
In *Proceedings Seventh IJCAI*, pages 843-845, 1981.

DART is designed to provide advice to IBM filed personnel on diagnosis in computer installations. Implemented in EMYCIN.

Bowen, J. A.

Automated Configuration of Backplane-based Microcomputers
In *Proceedings CADD-84*, U. K., 1984.

A program that automates the design of hardware for a dedicated microprocessor is described.

Brodsky, S. and Tyle, N.

Knowledge-based Expert Systems for Power Engineering
In *Proceedings of the 15th Pittsburgh Modeling and Simulation Conference*, Pittsburgh, April 1984.

Paper presents a brief review of the development and application of expert systems in areas related to electric power engineering. The specific examples discussed include nuclear power plant monitoring, power system restoration and hydro-electric plant design. In addition, several problems are examined as candidates for future expert systems.

Brown, H., Tong, C., and Foyster, G.

Palladio: An Exploratory Environment for Circuit Design
Computer, Vol. 16, No. 12, pages 41-58, December 1983.

A number of interesting concepts in design are presented. Palladio is an attempt to provide an integrated design environment for circuit design.

Birmingham, W. P.

MICON: A Knowledge Based Single Board Computer Designer
Research Report No. CMUCAD-83-21, SRC-CMU Center for Computer-Aided Design, December 1983.

MICON designs a single board computer from hardware requirements. Implemented in OPS5.

Cantone, R. R., Pipitone, F. J., Lander, B., and Marrone, M. P.

Model-based Probabilistic Reasoning for Electronic Troubleshooting
In *Proceedings Eigth IJCAI*, pages 207-211, 1983.

IN-ATE is a KBES for guiding the novice technician through electronic trouble-

shooting. It is being developed at the Naval Center for Applied Research, U. S. A. The paper discusses a technique to automatically produce a binary decision tree of test points to be checked by the technician.

Chen, S.

On Intelligent CAD Systems for VLSI Design
In *Proceedings IEEE International Conference on Computer Design: VLSI in Computers*, pages 405-407, New York, 1983.

Issues in the applications of KBES to VLSI design are discussed. Distributed KBES are proposed for VLSI design.

[Chip]

Expert System
CHIP (Germany), No. 8, pp. 52-4, August 1984 [NTIS].

Describes a prototype KBES, being developed on a 16 bit microprocessor, currently under development at NIXDORF, a German computer manufacturer.

Davis, R.

Diagnosis Via Causal Reasoning: Paths of Interaction and the Locality Principle
In *Proceedings 3rd NCAI*, pages 88-94, Washington, D. C., 1983. (See also IEEE Computer, October 1983, Int. Journal of Man-Machine Studies, November, 1983 and Proceedings of 4th NCAI)

Implementation of a KBES exploiting the causality in electrical circuits is described. The concept of *locality* is used to explain the reason for the difficulty of diagnosing bridge faults and for the need for multiple representations.

de Kleer, J.

Causal and Teleogical Reasoning in Circuit Recognition
Phd Thesis, M. I. T., AI Laboratory , 1979 (also AI MEMO TR - 529).

Dinbas, M.

A Knowledge-based Expert System for Automatic Analysis and Synthesis in CAD
In *Proceedings IFIPS Congress*, pages 705-710, 1980.

PEACE, a KBES for analysis and synthesis of electronic circuits, is described.

Freeman, M., Hirschman, L., McKay, D., Miller, F., and Sidhu, D.

Logic Programming Applied to Knowledge-based Modeling and Simulation
In *Proceedings Conference on Artificial Intelligence*, pages 177-193, Oakland University, Rochestor, MI, April 1983.

Prolog is used to develop an automated configurer for Burroughs main frame computers.

Fujita, T. and Goto, S.

A Rule-based Routing System
In *Proceedings IEEE International Conference on Computer Design: VLSI in Computers*, pages 451-454, New York, 1983.

An interactive routing system is described. The designer's knowledge is represented in clause form of first order predicate logic.

Fusaoka, A., Seki, H. and Takahashi, K.

Description and Reasoning of VLSI Circuit in Temporal Logic

New Generation Computing, Vol. 2, pages 79-80, 1984.

A method for describing and reasoning about the behavior of VLSI circuits within the framework of extended temporal logic is described.

Hartely, R. T.

CRIB: Computer Fault-Finding Through Knowledge Engineering
Computer, pages 76-83, March 1984.

Describes the development of a system for computer fault diagnosis.

Horstmann, P. W.

Expert Systems and Logic Programming for CAD
VLSI Design, pages 37-46, November 1983.
(see also the paper in 21st Design Automation Conference, pages 144-151)

Logic programming is applied to solve VLSI design problems in the areas of design for testability, functional simulation, fault diagnosis, and automatic test generation. PROLOG was used to implement a prototype system for design for testability.

Genesereth, M. R.

Diagnosis using Hierarchical Design Models
In *Proceedings 2nd NCAI*, pages 278-283, Pittsburgh, 1982.

The hierarchy inherent in most computer systems is used to develop a KBES. The use of this hierarchy helps to reduce the search space.

Grinberg, M. R.

A Knowledge-based Design System for Digital Electronics
In *Proceedings 1st NCAI*, pages 283-285, Stanford, 1980 (also University of Maryland, Phd thesis).

Semi-Automatic Digital Designer (SADD) uses the idea of structured modular circuit design using an interactive user interface. The design is divided into three phases: 1) specification acquisition; 2) circuit design; and 3) circuit simulation.

Kim, J. and McDermott, J.

TALIB: An IC Layout Design Assistant
In *Proceedings 3rd NCAI*, pages 197-201, Washington, D. C., 1983.

TALIB performs automatic circuit layout and interconnections by generating plan steps at different levels of abstraction spaces and opportunistically refining each plan at one level to more specific steps at the lower level.

King, J. J.

Artificial Intelligence Techniques for Device Trouble Shooting
Technical Report CSL-82-9 (CRC-TR-82-004), Hewlett-Packard Company, August 1982.

This report addresses the current state of art of AI in electronic device troubleshooting with a comparison with conventional fault analysis.

King, J. J.

An Investigation of Expert Systems Technology for Automated Troubleshooting of Scientific Instrumentation
Technical Report CSL-82-12 (CRC-TR-82-007), Hewlett-Packard Company, August 1982.

KBES methodology is applied to troubleshoot a gas chromatograph/mass spectrometer, concentrating on a failure mode called radio frequency overdrive.

Knapp, D., Granacki, J. and Parker, A.

An Expert Synthesis System

In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 32-33, September 1983.

Describes the architecture of a KBES for synthesis of VLSI designs. The system has four modules: a knowledge-base of design techniques, a data structure representing the hardware being designed, a history of the design process, and a man-machine interface.

Kowalski, T. and Thomas, D.

The VLSI Design Automation Assistant : Prototype System

In *Proceedings 20th Design Automation Conference*, pages 479-483 IEEE-ACM, Maiami, 1983.

The VLSI DAA uses temporarily ordered subtasks to allocate VLSI subsystems. The input to the system is an algorithmic dataflow description of a VLSI system and the output is a list of technology-independent registers, operators, data paths and control signals.

Krueger, M. W., Cullingford, R. E. and Bellavance, D. A.

Control Issues in a Multiprocess Computer-Aided Design System Containing Expert Knowledge

In *Proceedings of the IEEE International Conference on Cybernetics and Society*, pages 139-143, Atlanta - Georgia, 1981 [NTIS].

Describes CADHELP, a KBES for the design of digital logic circuits. Scripts are used to generate explanation of text and graphic demonstrations.

Lenat, D. B., Sutherland, W. L., and Gibbons, J.

Heuristic Search for New Microcircuit Structures: An Application of Artificial Intelligence

The AI Magazine, pages 17-33, Summer 1982.

The knowledge-acquisition bottleneck in expert system development can be removed if the system learns to augment its knowledge-base. EURISKO, a system that learns by discovery, is applied to the synthesis of semiconductor devices.

Maclean, C. and Wilde, P.

Knowledge-based Electronic Circuit Diagnosis

In *Proceedings Expert Systems 83*, Technical Conference of the BCS SGES, Cambridge, U. K., December 1983.

McClelland, E. C., Van Horne, P. R.

Fast Voltage Prediction Using a Knowledge Based Approach

IEEE Transactions on Power Apparatus and Systems, Vol. PAS-102, No 2, February 1983.

The system is being implemented in prototype form on the New York Pool real-time computer system.

McDermott, J.

R1: A Rule-based Configurer of Computer Systems

Artificial Intelligence, Vol. 19, No. 1, pages 39-88, September 1982.

R1 configures Vax computer systems. Implemented in OPS, and one of the few KBES that is used in industry.

Mitchell, T., Steinberg, L., Kedar-cabelli, S., Kelly, V., Shulman, J., and Weinrich, T.
An Intelligent Aid for Circuit Redesign
In *Proceedings 3rd NCAI*, Pages 274-278, Washington, D. C., 1983.

REDESIGN assists the user in redesigning electrical circuits by focusing on an appropriate portion of the circuit, generating and ranking possible changes within the circuit.

Pao, Y. and Ou, S.
Rule-based Approach to Electrical Power Systems Security Assessment
In *Proceedings of IEEE Conference on Pattern and Image Processing*, pages 340-342, 1981.

Shirley, M. and Davis, R.
Generating Distinguishing Tests Based on Hierarchical Models and Symptom Information
In *Proceedings IEEE International Conference on Computer Design: VLSI in Computers*, New York, 1983.

A methodology is developed to diagnose faulty components in digital hardware when a list of candidates that are likely to fail is given. A three step process is described.

Shubin, H. and Ulrich, J. W.
IDT: An Intelligent Diagnostic Tool
In *Proceedings 2nd NCAI*, pages 290-295, Pittsburgh, August 1982.

IDT was developed to identify faults in PDP 11/03 computers. It helps the technician to identify the field replaceable unit to fix the fault.

Spaanenburg, L.
Digital IC Design at Twente University
In *LSIM-83 Proceedings, 1983 University/Government/Industry Microelectronics Symposium*, pp. 47-51, Twente Univ. of TEchnology, Enschede, Netherlands, 1983 [NTIS].

Experiences and future enhancements towards a VLSI design assistant are described.

Smith, M. F., and Bowen, J. A.
Knowledge and Experience-Based Systems for Analysis and Design of Microprocessor Applications Hardware
Microprocessors and Microsystems, Vol. 16, No. 10., pages 515-519, December 1982 (see also Jl. Microcomputer Appl., Vol. 6, No. 2, pp 155-161,1983).

Describes the potentials for KBES for the analysis and design of microprocessor applications. Also discusses MAPLE, a prototype system.

Stallman, R. and Sussman, G. J.
Forward Reasoning and Dependency-directed Backtracking in a System for Computer-Aided Circuit Analysis
Artificial Intelligence, Vol. 9, pages 195-196, 1977.

ARS, a rule-based language, is used to implement a system for computer-aided circuit analysis. Antecedent reasoning is used to deduce facts. The deduced facts have associated justifications, which are used by the system in the analysis of failures and to reduce the search space.

Steinberg, L. and Kelly, V. E.

The CRITTER System: Analyzing Digital Circuits by Propagating Behaviors and Specifications
In *Proceedings 2nd NCAI*, pages 284-289, Pittsburgh, August 1982.
(see also 21st Design Automation Conference, pages 419-425).

CRITTER can reason about digital hardware designs through the use of declarative representation of the components at different levels of abstraction. It can evaluate the correctness and robustness of digital designs.

Steingberg, L. and Mitchell, T.

A Knowledge-Based Approach to VLSI CAD: The Redesign System
In *Proceedings ACM IEEE 21st Design Automation Conference*, pages 412-418, New Mexico, 1984.

Summarizes the Rutgers AI/VLSI group's work on REDESIGN, an interactive aid for functional design of digital circuits.

Stefik, M., Bobrow, D., Bell, B., Brown, H., Conway, L., and Tong, C.

The Partitioning of Concerns in Digital System Design
Technical Report VLSI-81-3, XEROX PARC, CA 94304, 1981.

Discussion on the abstraction hierarchies in digital design.

Stefik, M. and Conway, L.

Towards the Principled Engineering of Knowledge
The AI Magazine, pages 4-16, Summer 1982.

VLSI domain is used as a paradigm for explaining the structuring of design knowledge.

Stefik, M. and de Kleer, J.

Prospects for Expert Systems in CAD
Computer Design, Vol. 22, No. 5, pages 65-76, April 1983 [NTIS].

The importance of KBES for CAD is emphasized through the paradigm of digital design.

Sumner, G. C.

Knowledge-based Systems Maintenance Applications (ATE)
In *Proceedings of IEEE International Automatic Testing Conference*, pages 472-473, Fortworth - Texas, 1982 [NTIS].

Describes the usefulness of KBES for electronic maintenance problems.

Sussman, G. J.

Electrical Design: A Problem for Artificial Intelligence Research
In *Proceedings Fifth IJCAI*, pages 894-900, 1977.

Intelligent recovery in a problem solver for electrical design is described. The engineering design process is recast in terms of *Problem Solving by Debugging Almost-Right Plans*. Failures are used to reduce search.

Sussman, G. J.

SLICES: At the Boundary between Analysis and Design
AI Memo 433, M. I. T., 1977.

SLICES combines the notion of equivalence, used by electrical engineers to reduce the complexity in a circuit, with identification of parameters. The system uses appropriate SLICES along with analysis by propagation of constraints to assign component values to a circuit.

Sussman, G. J. and Steele Jr., G. L.
CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions
Artificial Intelligence, Vol. 14, pages 1-39, 1980.
Constraint propagation is used to synthesize and analyze electrical networks.

Sussman, G. J., Holloway, J. and Knight, T. E.
Design Aids for Digital Integrated Systems - An Artificial Intelligence Approach
In *Proceedings IEEE International Conference on Circuits and Computers*, October 1980.

Taylor, G. S. and Ousterhout, J. K.
Magics's Incremental Design-Rule Checker
In (Proceedings ACM IEEE 21st Design Automation Conference), pages 160-165, New Mexico, 1984.
Although, not strictly an expert system the approach presented here would be useful in the development of KBES for design.

Taylor, J. H., Frederick, D. K., and James, J. R.
An Expert System Scenario for Computer-Aided Control Engineering
In *Proceedings of the American Control Conference*, San Diego, AC, 1984.
A framework for a KBES in control system design is provided.

Tong, C.
A Framework for Circuit Design
In *Proceedings COMPCON84*, New York, February 1984.
Discusses a framework for circuit design that contains design descriptions such as *components*, *plans*, *goals*, and *tradeoffs*. Also addresses the issue of control knowledge in design. The concepts presented are also relevant in other areas of design.

Tsukiyama, M. and Fukuda, T.
An Application of Knowledge Base to Control Systems
In *Proceedings of the IEEE International Conference on Cybernetics and Society*, pages 342-346, Atlanta - Georgia, 1981 [NTIS].
KBES structure is developed as a network organization of modules. These modules contain both production rules and calculation tools.

Vesonder, G. T., Salvatore, J. S., Zielinski, J. E., Miller, F. D., and Copp, D. H.
ACE: An Expert System for Telephone Cable Maintenance
In *Proceedings Eigth IJCAI*, pages 116-121, 1983.
ACE was developed to aid in automated cable maintenance. It takes input from CRAS (Cable Repair Administration System) to analyze a large number (in hundreds) of telephone cable maintenance reports. It is written in OPS4.

Williams, T. L., Orgren, P. J., and Smith, C. L.
Diagnosis of Multiple Faults in a Nationwide Communications Network
In *Proceedings Eigth IJCAI*, pages 179-181, 1983.
NDS (Network Diagnostic System) is a KBES for identifying faults in a nationwide communications network (COMNET).

Zippel, R.
An Expert System for VLSI Design

In *Proceedings of the IEEE Symposium on Circuits and Systems*, pages 191-193, Newport Beach, CA, 1983 [NTIS].

Discusses the motivation for the development of a KBES for VLSI design. Also provides some guidelines for the development of this system.

4 Mechanical and Industrial Engineering

Bocquet, J. C. and Tichkiewitch. S.

An "expert system" for Identification of Mechanical Drawings
In Ellis, T. M. R., and Semenkov, O. I. (editors), *Advances in CAD/CAM*, PROLAMAT82, Leningrad USSR, May 1982, Published by North-Holland Publishing Company, 1983.

Describes an automatic methodology to transform a given Mechanical drawing into a 3-D data base. The production rule approach is used.

Bonissone, P. P.

DELTA: An Expert System to Troubleshoot Diesel Electrical Locomotives
In *Proceedings ACM*, New York City, pages 44-45, October 24-26, 1983.

DELTA is a prototype KBES, implemented in FORTH, developed at General Electric Corporate R & D to troubleshoot diesel electric locomotives. It contains about 530 rules.

Bonissone, P. P.

Outline of the Design and Implementation of a Diesel Electrical Engine Troubleshooting Aid
In *Expert Systems 82*, Technical Conference of the BCS SGES, Brunel University, U. K, 14-16 September, 1982.

Brown, D. C. and Chadrasekaran, B.

An Approach to Expert Systems for Mechanical Design
In *Proceedings of Trends and Applications on Automating Intelligent Behavior: Applications and Frontiers*, pages 173-180, Sponsored by IEEE and NBS, 1983.

Three categories of design are identified. The first two categories require innovation from the designer, while the third category deals with well-established design alternatives. A hierarchy of conceptual specialists is used to solve the problem in a distributed manner. An example in the area of mechanical design is presented. It is equally applicable to other areas of design.

de Kleer, J. and Bobrow, D. G.

Qualitative Reasoning With Higher-order Derivatives
In *Proceedings 4th NCAI*, pages 86-91, Austin, Texas.

Six fundamental laws which govern the time behavior of a physical system or device are identified. An application to a pressure regulator is discussed.

Descotte, Y. and Latombe, J. C.

GARI: An Expert System for Process Planning
In Boyse, J. W. and Pickett, M. S. (editors), *Solid Modeling by Computers: From Theory to Applications*, Plenum Press, New York, 1984.

GARI generates machine planning of mechanical parts. It uses a planner that develops a plan by iteratively constraining a loosely specified initial plan. The constraints are drawn from a knowledge-base provided by experts.

Dixon, J. R. and Simmons, M. K.

Expert Systems for Engineering Design: Standard V-Belt Drive Design as an Example of the Design-Evaluate-Redesign Architecture

In Proceedings Computers in Engineering, A. S. M. E., pages 332-337, August 1984, Las Vegas, Nevada.

Describes a framework for experts systems for engineering design. Uses a Blackboard-type architecture. The system, called VEXPERT, is exemplified through the design of v-belt drives. Implemented in OPS5 and Franz LISP.

Dixon, J. R. and Simmons, M. K.

Computers that Design Expert Systems for Mechanical Engineers

C. I. M. E., pages 10-17, November 1983.

An overview of potential applications in some areas of mechanical engineering is presented.

Fox, M. S.

The Intelligent Management System : An Overview

Technical Report CMU-RI-TR-81-4, Robotics Institute, Carnegie-Mellon University, August 1981.

The Intelligent Management System (IMS) project at C-MU is described. The purpose of IMS is to provide managers and planners assistance in day to day tasks. The report describes a number of features, such as modelling of organizations, user interfaces, etc., of the IMS.

Fox, M., Lowenfield, S. and Kleinosky, P.

Techniques for Sensor-Based Diagnosis

In *Proceedings Eighth IJCAI*, pages 158-163, August 1983.

PDS, a forward chaining ruled-based system for implementing KBES for real-time diagnosis of machine faults, is described. PDS is implemented over SRL.

Freed, D. and Wright, D.

FAXS: An Expert System for the Analysis of Mechanical Failures

In *Proceedings Computers in Engineering, A. S. M. E.*, pages 338-342, August 1984, Las Vegas, Nevada.

FAXS is being developed to aid the engineering student or other people in failure analysis and prevention. It uses a MYCIN-type probabilistic scheme to deal with inexact inferences. On going research is focused on merging FAXS with CAD and stress analysis programs. Implemented in Fortran 77.

Gini, G., Gini, M. and Morpurgo, R.

A Knowledge-Based Consultation System for Automatic Maintenance and Repair

In Ellis, T. M. R., and Semenkov, O. I. (editors), *Advances in CAD/CAM*, PROLAMAT82, Leningrad USSR, May 1982, Published by North-Holland Publishing Company, 1983.

The potentials of KBES for diagnosis and repair of mechanical systems are explored.

Mamdani, E. H.

Rule-based Methods for Designing Industrial Process Controllers

In *Proceedings Colloqium on Application of Knowledge-Based Systems*, London, U. K., 1982.

Fuzzy techniques are used in the development of a KBES for process control applications. Method is currently used commercially for the control of cement kilns.

Motoda H., Yamada, N. and Yoshida, K.

A Knowledge Based System for Plant Diagnosis

In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Japan, November 7-9, 1984.

McKibbin, M, L.

Will AI Clash with MIS in the Factory

Infosystems, Vol. 30, No. 8, pp 99, August, 1983 [NTIS].

Potentials for expert systems for planning, scheduling, and other areas of manufacturing are discussed.

Nau, D. S. and Chang, T.

Prospects for Process Selection Using Artificial Intelligence

Computers in Industry, Vol. 4, pages 253-263, 1983.

Discusses potential uses of KBES in process planning for manufacturing tasks.

Nelson, W. R.

REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents

In *Proceedings 2nd NCAI*, pages 296-301, Pittsburgh, August 1982.

REACTOR is being developed at EG & G, Idaho, for assisting operators in the diagnosis and treatment of nuclear reactor accidents.

Phillips, R. H., Zhou, X-D., Mouleeswaran, C. B.

An Artificial Intelligence Approach to Integrating CAD and CAM Through Generative Process Planning

In *Proceedings Computers in Engineering, A. S. M. E.*, pages 332-337, August 1984, Las Vegas, Nevada.

The PROLPLAN system, a KBES for process planning, and its interface with a CAD systems are described. Sample studies from are carried out on a small sample of rotational machined parts. A rule-based approach is used.

Rajagopalan, R.

Qualitative Modeling in the Trubojet Engine Domain

In *Proceedings 4th NCAI*, pages 86-91, Austin, Texas.

Presents a causal model of a turbojet engine based on the relationship between engine parameters. This is used to implement an engine simulation.

Reddy, Y. V. and Fox, M. S.

KBS: An Artificial Intelligence Approach to Flexible Simulation

Technical Report CMU-RI-TR-82-1, Robotics Institute, Carnegie-Mellon University, February 1982.

KBS (Knowledge-Based Simulation) system is a knowledge representation system for simulating complex organizations. Features include interactive model creation and alteration, simulation monitoring and control, graphics display, and selective instrumentation.

Underwood, W. E.

A CSA Model-based Nuclear Power Plant Consultant

In *Proceedings 2nd NCAI*, pages 302-305, Pittsburgh, August 1982.

The Common Sense Algorithm representation is used to model the physical system. Diagnostic rules are also represented in this formalism.

5 General Engineering Applications

Apte, C.

Expert Knowledge Management for Multi-level Modelling with an Application to Well-Log Analysis
Technical Report, LCSR, Rutgers University, 1982.

Deals with the implementation of a KBES incorporating heuristic knowledge and algorithmic tools.

Barbuceanu, M.

Object-Centered Representation and Reasoning: An Application to Computer-Aided Design
SIGART Newsletter, Vol. 87, pages 33-39, January 1984.

Discusses the usefulness of using an object-oriented approach to CAD problems. This approach provides a powerful framework for building KBES in the areas of design and planning.

Basden, A.

On Application of Expert Systems
International Journal of Man-Machine Studies, Vol. 19, pages 461-477, 1983.

A number of issues on building KBES are addressed. Specific reference is made to SCCES, Stress Corrosion-Cracking Expert System.

Bataii, J.

Dependency Maintenance in the Design Process
In *Proceedings IEEE International Conference on Computer Design: VLSI in Computers*, pages 405-407, New York, 1983.

A case for the maintenance of dependency structures in design is presented, with examples from VLSI domain. The concepts presented are equally applicable for other domains.

Bundy, A.

Intelligent Front Ends
D. A. I. Research Paper No. 227, Department of Artificial Intelligence, University of Edinburgh, England, July 1984.

Describes the techniques used for developing intelligent front ends for existing software, such as finite element analysis programs.

De Swaan Arons, H.

Expert Systems in the Simulation Domain
Mathematics and Computer Simulation (Netherlands), Vol. 25, No. 1, pages 10-16, February 1983 [NTIS].

Explores the possibilities of the application of KBES to simulation problems.

Dixon, J. R., Simmons, M. K., and Cohen, P. R.

An Architecture for Application of Artificial Intelligence to Design
In *Proceedings ACM IEEE 21st Design Automation Conference*, pages 634-640.
(for similar views see papers by Rehak, et. al. in Gero (eds))

The Blackboard model for engineering design is presented.

Gero, J.(editor)

Knowledge Engineering in Computer-Aided Design
IFIP WG-5.2, September 1984, Budapest, Hungary, Published by North-Holland Publishing Company.

Contains a number of recent papers on the application of KBES in engineering. Papers encompass Civil, Architecture, Mechanical and Electrical engineering applications.

Lafue, G. M and Mitchell, T. M.

Data-base Management Systems and Expert Systems for CAD
Technical Report No LCSR-TR-28, Rutgers University, 1982.

The use of database management systems and KBES is discussed with reference to the REDESIGN system.

Latombe, J-C. (editor)

Artificial Intelligence and Pattern Recognition in Computer Aided Design
North-Holland Publishing Company, New York, 1978.

Contains a number of good papers on the application of AI to engineering design problems.

Lowrance, J. D. and Garvey, T. D.

Evidential Reasoning: An Implementation for Multisensor Integration
SRI International, Technical Note 307, December 1983.

The Dempster-Shafer theory of evidence is used to develop a system for integrating information from multiple sources.

Maher, M.. L., Sriram, D. and Fenves, S. J.

Tools and Technniques for Knowledge-based Expert Systems for Engineering Design
Advances in Engineering Software, October 1984.

Describes the application of OPS5, SRL and PROLOG to engineering design problems.

McDermott, J.

Domain Knowledge and the Design Process
Design Studies Vol. 3, No. 1, pages 31-36, 1982 (also appeared in Proceedings 18th Design Automation Conference, Nashville, TN, 1981).

R1 and XSEL are used to illustrate the importance of domain knowledge in the design process.

Preiss, K.

Data Frame Model for Engineering Design Process
Design Studies Vol. 1, No. 4, pages 231-243, 1980.

The frame based approach to engineering design is discused. Has many interesting concepts.

Reddy, D. Sriram, Maher, M. L. Tyle, N., Banares, R., Rychener, M. and Fenves, S.J.

Knowledge-based Expert Systems for Engineering Applications
In *Proceedings IEEE Conference on Systems, Man and Cybernetics*, India, 1983-1984.

A number of KBES currently under development at C-MU in the areas of Civil, Chemical and Electrical Engineering are described.

Rychener, M. D.

Expert Systems for Engineering Design: Experiments with Basic Techniques
In *Proceedings of Trends and Applications on Automating Intelligent Behavior: Applications and Frontiers*, pages 21-27, 1983 (Sponsored by IEEE and NBS).

The author's experience with the development of expert systems for engineering design is discussed.

Simon, H. A.

The Sciences of the Artificial
MIT Press, Cambridge, MA, USA, 1969.

The chapter on design describes the design process in detail.

Sleeman, D. and Brown, J. S. (editors)

Intelligent Tutoring Systems
Academic Press, London, U. K., 1982.

Contains a number of articles on expert system approaches to intelligent computer-aided instruction.

Swift, K. and Mathews, A.

Expert Computers in Engineering Design
Engineering (UK), Vol. 223, No. 9, pp 673-8, 1983 [NTIS].

The DACON system for assembly automation and POLYCOAT systems for designing coatings are discussed.

Tokoro, M., Ishikawa, Y., Maruichi, T. and Kawamura, M.

An Object Oriented Approach to Knowledge Systems
In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Japan, November 7-9, 1984.

Wade, J. and Shubin, H.

A Generalized Approach to Diagnostic Problems
In *Expert Systems 82*, BCS SGES, England, 1982.

A general discussion of approaches to building diagnostic expert systems, with an emphasis on the advantage of keeping knowledge-base and the inference-mechanism separate.

6 Domain Independent Tools

The tools described here are available at a modest charge from the developer. A more complete description of various tools can be found in the paper by Maher, et. al., described in Section 5.

Balzer, R., Erman, L., London, P. and Williams, C.

Hearsay-III: A Domain Independent Framework for Expert Systems
In *Proceedings 1st NCAI*, Stanford, 1980.

Hearsay-III is a domain independent version of the Hearsay-II speech understanding system. It is written in AP3, a relational database language embedded in Interlisp. It is useful in situations that demand multiple sources of knowledge. Contact Steven Fickas, (Fickas@USC-ISI) Oregan State University.

Bobrow, D. and Stefik, M.

The Loops Manual

Xerox Corporation, 1983 (See AI magazine Vol. 4, No 3., 1983 for a description of LOOPS).

LOOPS is an integration of *procedure-oriented*, *object oriented*, *access-oriented* and *rule-oriented* programming paradigms. It offers a powerful framework for building expert systems. Currently it is available only on the XEROX machines that run Interlisp-D. Contact Mark Stefik, Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304 (Stefik@XEROX.PARC).

Clocksin, W. F. and Mellish, C. S.

Programming in Prolog

Springer-Verlag, 1981.

Prolog is a logic-based programming language that is widely used in Europe for building KBES. According to the authors,"Prolog is a practical and efficient implementation of many aspects of 'intelligent' program execution, such as non-determinism, parallelism, and pattern-directed procedure call". The fifth generation computer project started in Japan will use Prolog as the implementation language. For a Dec-10 version write to D. Warren at SRI International (Warren@SRI-AI).

Forgy, C. L.

OPS5 Users Manual

Technical Report No: CMU-CS-81-135, Carnegie-Mellon University, August 1981.

OPS5 is a production system language for rule-based programming. A number of versions of this language exist. The latest version, OPS83, will incorporate a number of important features that were not implemented in the earlier version. OPS83 is written in the C language. Currently maintained by Charles Forgy, Department of Computer Science, Carnegie-Mellon University, Pittsburgh PA 15213 (Forgy@CMU-CS-A).

Reboh, Rene'

Knowledge Engineering Techniques and Tools in the Prospector Environment

Technical Report 8172, SRI International, June 1981.

The Knowledge-Acquisition System (KAS) is a combination of powerful inference techniques and a system of representing the meaning of concepts that it deals with. Knowledge is represented in terms of spaces (a type of frame) and semantic networks. Bayesian reasoning is used in the inference mechanism. It is useful for diagnostic type of problems and is implemented in Interlisp. This system was extended (called HYDRO) to incorporate numerical calculations. Contact Rene Reboh (currently with Syntelligence), SRI International, Palo Alto, Stanford (Reboh@SRI-AI).

van Melle, W.

A Domain Independent Production-Rule System for Consultation Programs

In *Proceedings Sixth IJCAI*, pages 923-925, August 1979 (see also the EMYCIN manual, available from Stanford University Computer Science Department).

EMYCIN is a domain independent version of MYCIN. It is useful for building diagnosis based expert systems. Knowledge is represented in object-attribute tuples. For more information write to Department of Computer Science, Stanford University, Stanford, CA 94305.

Weiss, S. M. and Kulikowski, C. A.

EXPERT: A System for Developing Consulation Models

In *Proceedings Sixth IJCAI*, pages 942-947, 1979.

It is probably the only KBES tool developed in FORTRAN. It is useful for classification

and diagnostic type of problems. It can be interfaced with other existing FORTRAN software. Hence, it seems to be a good tool for engineers who wish to interface expert systems with engineering software, written in FORTRAN. Contact Shalom Weiss or Casimir Kulikowski, Department of Computer Science, Rutgers University, N. J. (Weiss@Rutgers).

Wright, J. M. and Fox, M.

SRL: Schema Representation Language

Robotics Institute Technical Report, Carnegie-Mellon University, 1983.

A frame-based language implemented in Franzlisp. This language is being extented to include rule-based programming (PSRL). SRL is maintained by the Intelligent Systems Laboratory, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA 15213 (Fox@CMU-RI-ISL1).

Fain, J., Gorlin, D., Hayes-Roth, F., Rosenschein, S. J., Sowizral, H., and Waterman, D.

The ROSIE Language reference manual

Technical Report N-1647-ARPA, Rand Corporation, Santa Monica, California 90406, 1981.

ROSIE is a general purpose programming language, implemented in Interlisp, that supports stylized English input to create assertional descriptions. Knowledge is represented in the form of facts and rule sets. ROSIE also provides the user a wide range of options to express n-ary relations. ROSIE has been successfully used to build a number of expert systems, specially for diagnostic and interpretation type of problems. Contact Henry Sowizral.

7 General Reading

Barr, A. and Feigenbaum, E. (editors) *The Handbook of Artificial Intelligence*, Vol-I to Vol-II

William Kaufmann Inc., Los Altos, California, 1981-1982.

These volumes will be useful for anyone interested in AI. Volume 1 covers the basic topics in AI. Volume II focuses on applications of AI (mostly expert systems), programming languages and automatic programming. The final Volume in this series (Volume III) is edited by Cohen and Feigenbaum and contains articles on learning, theorem proving, models of cognition and theorem proving.

Dym, C. L.

Expert Systems: New Approaches to Computer-Aided Engineering

In *Proceedings Twenty-Fifth AIAA-ASME-ASCE-AHS Structures, Structural Dynamics and Materials Conference*, California, May 15, 1984.

Presents a good overview of expert systems for the engineer.

Konopasek, M. and Jayaraman, J.

Expert Systems for Personal Computers - The TK!Solver Approach

BYTE, pages 137-156, May 1984.

Presents a case for using the TK!Solver program as an expert system framework.

Hayes-Roth, F., Waterman, D. and Lenat, D. (editors)

Building Expert Systems

Addison-Wesley Publishing Company, 1983.

It gives a good overview of expert systems. However, it is collection of papers and one can see a lot of repetition in the chapters. Further, the book does not have any description of logic-based expert systems.

Michie, D.

Expert Systems
Computer Journal (UK), Vol. 23, No. 4, pages 369-376, November 1980.
A general review of expert systems is provided, with emphasis on applications.

Nau, D. S.

Expert Computer Systems
Computer, Vol. 16, Pages 63-85, February 1983.
Gives a good overview of expert systems.

Rich, E.

Artificial Intelligence
McGraw-Hill, 1983.

Weiss, S. M. and Kulikowski, C.

A Practical Guide to Designing Expert Systems
Rowman & Allanheld, pp 194, March 1984
Addresses a number of practical issues in the building of expert systems. Only classification-type problems are covered.

Winston, P.

Artificial Intelligence
Addison-Wesley Publishing Company, 1984.
Books by Rich and Winston are highly recommended for the novice.

8 Relevant Journals and Conferences

AI Journal - Artificial Intelligence Jounal
Mostly papers on basic research. Quarterly Jounal.

The AI Magazine -
Papers are fairly general. Provides interesting reports on research in various institutions. Published quarterly.

Advances in Engineering Software -
Forthcoming issues may have some articles in the area. Published quarterly from UK.

BCS SGES - British Computer Society Specialist Group in Expert Systems
BCS SGES holds a conference every year on applications of expert systems.

CAD - Computer Aided Design, Published in U. K.
Papers in this journal deal mostly with work in European countries

C. I. M. E. - Computers in Mechanical Engineering
Papers relevant to engineering design are published. Published quarterly.

Computers and Structures
Forthcoming issues may have some interesting papers in the area. Published monthly.

EXPERT SYTEMS : The International Journal for Knowledge Engineering

New journal. A number of application articles are scheduled to appear in the coming year. Published quarterly

IEEE - Institute of Electrical and Electronics Engineers

IEEE holds a number of conferences in this area. The IEEE-ACM DA conferences have many interesting papers in the area.

IFIP - International Federation for Information Processing

Lots of conferences in the area are held by IFIP. For example, see the call for papers on page 11, SIGART, October 1983.

IJCAI - International Conference of Artificial Intelligence

Held once every two years. Many papers in applied AI are presented.

NCAI - National Conference on Artificial Intelligence

A yearly conference on AI.

SIGART - ACM Special Interest Group in Artificial Intelligence

A quarterly newsletter which deals with a wide variety of topics in AI.

9 Acknowledgments

Professor Fenves made a number of useful comments on various drafts of this paper. His encouragement for the compilation of this bibliography is greatly appreciated. Mike Rychener made many useful comments on a preliminary version of the paper.

Status of Work Toward Revision of Programming Language Fortran

by
Jerrold L. Wagener
Amoco Production Company
Tulsa, Oklahoma

May 1984

Please direct comments or questions concerning the technical content of this report to: Jerrold L. Wagener; Amoco Research Center; PO Box 591; Tulsa OK 74012; (918) 660-3978

** **

The Chairman of Fortran Standards Committee X3J3, Jeanne Adams, would like to call your attention to a series of Fortran Forum meetings to be held in the summer and fall 1983. The first two of these one day informational meetings have been scheduled for Wednesday, August 8, at E G and G, Idaho Falls; and for Monday, August 13, at Colorado State University, Fort Collins. More detailed announcements of these Forums appear on pages 43 and 45. Additional information can also be obtained by calling Jeanne Adams at (303) 491-7596. Jeanne would also like to hear from other organizations that would be interested in acting as host for a Fortran Forum.

Observers are welcome to attend regular meetings of X3J3. Future meetings of the committee are scheduled for 6 to 11 Aug 1984 in Jackson WY; 12 to 16 Nov 1984 in Fort Lauderdale FL; February 1985 in northern California; and May 1985 in Champaign-Urbana IL. Because of limitations on meeting space, anyone who would like to attend an X3J3 meeting should request further information in advance from the X3J3 Vice Chairman, Martin Greenfield, at (617) 671-2912. International meetings are also planned for 1 to 5 July 1985 in Germany; and 8 to 12 July 1985 in England.

** **

PLEASE COMPLETE THE SURVEY on pages 2, 3, and 4.

FORTRAN 8x SURVEY
FORTRAN STANDARDS COMMITTEE X3J3
Jeanne Adams, Chair, X3J3

This questionnaire has been developed to survey the opinions of the user community on the new features and the architecture proposed by X3J3 for inclusion in the next draft FORTRAN standard. The results of this and other surveys and questionnaires will be used by the X3J3 committee in assessing the strength of each new facility for inclusion in a draft standard. Information about the new features is contained in "FORTRAN Information Bulletin", Number 1. Return your questionnaire to:

Andrew Johnson
MS 10C17-3
Prime Computer Inc.
500 Old Connecticut Path
Framingham, Ma 01701

RESPONDENT PROFILE

Check those that apply.

Type of Organization:

Scientific _____
Engineering _____
Commercial _____
Government _____
Vendor (Computer) _____
University _____
Other (state)_____

Are you employed as:

Professional Programmer _____
Problem Solver (Occasional)_____
Compiler Writer/Implementor_____
Home Computer User _____
Manager of Software _____
Other _____

Was FORTRAN your first computer language? Yes____ No____

What is your "language of choice?" _____

Name the vendor and operating system currently used.

_____

Do you now use: FORTRAN 66 _____ FORTRAN 77_____ 77 Subset_____

Do you subscribe to: FORTEC _____ SIGPLAN____ SIGNUM_____

Have you seen the FORTRAN Information Bulletin summarizing the proposals currently under consideration for FORTRAN 8x? Yes____ No____

List two features for FORTRAN 8X that--

You want in FORTRAN:	You do not want in FORTRAN:
1. _____	1. _____
2. _____	2. _____

Name_____ Address_____

Tel. No._____ _____

CHECKLIST ON FORTRAN 8x FEATURES

	Strongly Approve	Approve	Do not Approve	Strongly Disapprove	Do Care
Array Processing Features					
WHERE Construct					
Data Structures					
New Source Form(free)					
Significant Blanks					
Recursion					
Precision Specification					
Environmental Intrinsics					
Loop Constructs					
Enhanced CALL					
CASE Construct					
Internal Procedures					
IMPLICIT NONE					
Entity Oriented Decl.					
Derived Data Type					
MODULES					
Event Handling					
Varying CHARACTER Length (Note it has been removed)					
Bit Data Type (Note that it has been removed)					
Macros (Note that it has been removed)					
Architecture with deprecated Features (next page)					
General Impression of FORTRAN 8x					

DEPRECATED FEATURES

Which of these features would you make a candidate for deletion in the 199X FORTRAN standard, note FORTRAN 9x. All of these features will be retained in 198x. Do you want these removed from 199x, retained in 199x or are you undecided? Check one.

	Removed from 199x	Retained in 199x	Undecided
FORTRAN 77 Fixed Position Source From			
Alternate RETURN			
Assumed Size Dummy Arrays			
Passing Scalar to Dummy Array			
Specific Names (not Generic) for Intrinsics			
Statement Functions			
BLOCK DATA Program Unit			
Arithmetic IF Statement			
ASSIGN and Assigned GO TO Statements			
COMMON Statement			
Computed GO TO Statement			
DATA Statement			
DIMENSION Statement			
DOUBLE PRECISION Data Type			
ENTRY Statement			
EQUIVALENCE Statement			
FORTRAN 77 DO Statement			
PAUSE Statement			

Please include any comments you wish to make concerning FORTRAN 8x features and features marked as candidates for deletion from FORTRAN 9x on the reverse side of this page.

Status of Work Toward Revision of Programming Language Fortran

by
Jerrold L. Wagener
Amoco Production Company
Tulsa, Oklahoma

May 1984

TABLE OF CONTENTS

This report describes the current status of the technical work of X3J3 since the adoption of X3.9-1978 (Fortran 77). This work, informally referred to as "Fortran 8x", is incomplete and tentative, and is subject (and likely) to change prior to issuing a draft proposed standard (expected to occur no earlier than 1985). The purpose of this report is to summarize the status of current work by X3J3 relative to a future revision of the current Fortran standard. A list of criteria for this revision is summarized in section 3.1 of this document. Comments on any and all aspects of the features described in this report are welcomed.

1. INTRODUCTION

Among the additions to Fortran 77 contemplated for the next Fortran standard, five stand out as the major ones:

array operations

improved facilities for numerical computation

programmer defined data types

facilities for modular data and procedure definitions

the concept of "deprecated" features

A number of other additions are also included in Fortran 8x, such as improved source form facilities, more control constructs, recursion, dynamically allocatable arrays of any size, and event handling.

No Fortran 77 features will be removed; it remains X3J3's intent that any standard-conforming Fortran 77 program will be a standard-conforming Fortran 8x program, and that, with any exceptions clearly listed in the document, new Fortran 8x features can be compatibly incorporated into such programs.

1.1 Array Operations

Computation involving large arrays is an extremely important part of engineering and scientific uses of computing. Arrays may be used as atomic entities in Fortran 8x, and operations for processing whole arrays and sub-arrays (array sections) are included in the language for two principal reasons: (1) these features provide a more consise and higher level language that will allow programmers to more quickly and reliably develop and maintain scientific/engineering applications; (2) these features can significantly facilitate optimization of array operations on all computer architectures.

The Fortran 77 arithmetic, logical, and character operations and intrinsic functions are extended to operate on array-valued operands. These include

whole, partial, and masked array assignment, array-valued constants and expressions, facilities to define user-supplied array-valued functions, and new intrinsic functions to manipulate arrays, extract general sections, and to support extended computational capabilities involving arrays (e.g., array reduction).

1.2 Numerical Computation

Scientific computation is one of the principal application domains of Fortran, and the guiding objective for all of the technical work is to strengthen Fortran as a vehicle for implementing scientific software. Though nonnumeric computations are increasing dramatically in scientific applications (and a number of the tentative additions to Fortran reflect that trend), numeric computation remains the workhorse. Accordingly, proposed additions include portable control over numeric precision specification, inquiry as to the characteristics of numeric information representation, and improved control of the performance of numerical programs.

1.3 Derived Data Types

"Derived data type" is the term given to that set of features in Fortran 8x that allows the programmer and package writer to define arbitrary data structures and operations on them. Data structures are user-defined aggregations of intrinsic and derived data type fields. Intrinsic operations on structured objects include comparison, assignment, input/output, and use as procedure arguments. The derived data type facilities may be used, without further operation definition, as a simple data structuring mechanism. With additional operation definitions, derived data types provide an effective implementation mechanism for data abstractions.

Procedure definitions in Fortran 8x may appear internally in a program unit, and as such may be used to define operations on derived data types. Internal procedures take essentially the same form as external procedures, with additional provisions for their use as infix operators. New operator symbols may be defined, and the intrinsic operator symbols may be overloaded for use with new data types. Internal procedures may be used simply as a mechanism for defining procedure packages (whether or not new data types are involved), and may be used for new operations on objects of intrinsic data types.

1.4 Modular Definitions

There is no way in Fortran 77 to define a global data area in one place and have all the program units in an application use that definition. In addition, the ENTRY statement is awkward and restrictive for implementing a related set of procedures, possibly involving common data objects. And finally there is no means in Fortran by which procedure definitions, especially interface information, may be made known locally to a program unit. All of these deficiencies, and more, are remedied by a new type of program unit that may contain any combination of data element declarations, derived data type definitions, procedure definitions, and procedure interface information. This program unit, called a MODULE, may be considered to be a generalization of and replacement for the BLOCK DATA program unit. A module may be referenced by any program unit, thereby making the module contents available to that program unit. This provides vastly improved facilities for defining global data areas and procedure packages. It also provides a convenient mechanism for encapsulating derived data type definitions (including operations defined on them), i.e., for encapsulating data abstractions.

1.5 Deprecated Features

With the advent of superior facilities, the use of certain older features of Fortran should be discouraged, and some of these features should possibly eventually be phased out of the language. For example, the numeric facilities alluded to above provide the functionality of DOUBLE PRECISION; with the new array facilities non-conformable argument association (such as passing an array element to a dummy array) is unnecessary (and in fact is not useful as an array operation); BLOCK DATA units are obviously redundant and inferior to modules. It is the current intent to identify such "superseded" facilities as deprecated (according to Webster: "mild or regretful disapproval; lower estimated value") features. Deprecated features will remain part of Fortran 8x; it is the intent that complete upward compatibility be maintained between Fortran 77 and Fortran 8x. Deprecated features may, however, be candidates for removal from the version of the Fortran standard following Fortran 8x. It is the intent that in this way official notice is given many years prior to removing a feature from the standard language. In Section 3.3 below, the proposed deprecated features are identified, together with possible functional replacements.

2. LANGUAGE SUMMARY

In this section a summary of proposed Fortran 8x features is given, with emphasis upon new features. The description uses a form of BNF slightly modified from that used in the Fortran 77 standard (X3.9-1978). Metasymbols are italicized, and comprise

is	introduces a syntactic class definition
or	introduces a syntactic class alternative
[ ]	encloses an optional item
[ ]...	encloses an optionally repeated item

The principal difference between this form of BNF and that in X3.9-1978 is that each definition starts with the name of the syntactic class that is being defined; the actual definition follows the metasymbol *is* or *or*. In X3.9-1978, only the right-hand side appeared, with informal text introducing the class being defined. Lower-case words, including hyphenated words, are syntactic class names. Upper-case is used for words that actually appear in the Fortran code (terminal symbols), even though lower-case is also allowed in the new source form. Terminal symbols that may have imbedded blanks (e.g., ENDIF and END IF) are shown in only one form. There is no attempt to be completely comprehensive in this summary, especially with Fortran 77 features. The syntax should be considered illustrative and incomplete; in the interest of brevity, some syntactic items (e.g., integer-expr) are not defined, and hopefully are adequately clear from context. Both syntax and semantics are subject to change prior to issuing a draft standard.

Each of the following sub-sections has the three part form of (1) a general discussion of the particular topic of the sub-section, (2) the BNF descriptions of the features under discussion, and (3) a set of notes and examples that provide specific points of information. Where new intrinsic functions are listed in the syntax only the function names are given (not the argument lists); following the names are brief descriptions of the functionalities.

2.1 Full Language Overview

The following highly condensed summary of the full language is provided so that one can see at a glance the major components of proposed Fortran 8x. So that it is clear which are the proposed new features, these are italicized (along with the metasymbols) and placed first (or occasionally last) in a list of syntactic class alternatives. Unitalicized items are essentially unchanged from Fortran 77, and are not further described in this information bulletin. The new features are all described in greater detail in the succeeding sections. The deprecated Fortran 77 features are last in any list of syntactic class alternatives. So that it is clear in this section which features are the deprecated ones, *ob* is used in place of *or* in these cases. (The italicizing

of new features, and the use of *ob*, occurs only in this section on full language overview.)

```
program-unit  is  unit-heading
                      [ use-statement ]...
                      [ local-specification ]...
                      [ executable-construct ]...
                      [ internal-procedure ]...
                  END [unit-type [unit-name]]

unit-heading  is  MODULE module-name
              or  HANDLER handler-name
              or  [PROGRAM program-name]
              or  function-heading
              or  subroutine-heading
              ob  BLOCK DATA identifier

local-        is  type-definition
specification or  procedure-interface
              or  declare-statement
              or  REFER refer-name ( attribute [,attribute]... )
              or  type-declaration
              or  IMPLICIT implicit-list
              or  PARAMETER ( constant-definition-list )
              or  SAVE [ save-list ]
              or  INTRINSIC procedure-name-list
              or  EXTERNAL  procedure-name-list
              or  FORMAT ( format-specification )
              ob  COMMON [/common-block-name/] common-list
              ob  DATA initial-value-list
              ob  DIMENSION dimension-list
              ob  EQUIVALENCE equivalence-list
              ob  statement-function

declare-      is  identifier [,identifier]... : attribute [,attribute]...
statement

attribute     is  non-char-type
              or  intent-attribute
              or  optional-attribute
              or  REFER (refer-name)
              or  CHARACTER [(length)]
              or  DIMENSION (dimensions)
              or  INITIAL (constant-expr)
              or  CONSTANT (constant-expr)
              or  SAVE
```

```
                or  INTRINSIC
                or  EXTERNAL

type-           is  CHARACTER[*len] var-decl[*len] [,var-decl[*len]]...
declaration     or  non-char-type  var-decl [,var-decl]...

non-char-type   is  floating-point-data-type
                or  TYPE ( type-name )
                or  EVENTMARK
                or  INTEGER
                or  LOGICAL
                ob  DOUBLE PRECISION

var-decl        is  identifier [(dimensions)]

executable-     is  block-case
construct       or  block-do
                or  block-if
                or  basic-statement
                ob  fortran77-do-loop

basic-          is  array-assignment
statement       or  identify-statement
                or  event-handling-statement
                or  EXIT [construct-name]
                or  CYCLE [construct-name]
                or  assignable-object = expr
                or  io-stmt
                or  GO TO label
                or  CALL subroutine-name[([actual-argument-list])]
                or  CONTINUE
                or  RETURN [return-code]
                or  STOP [stop-code]
                or  IF (scalar-logical-expr) basic-statement
                ob  IF (arithmetic-expr) branch-list
                ob  ASSIGN label TO integer-variable
                ob  GO TO integer-variable [[,] (label-list)]
                ob  GO TO (label-list), integer-variable
                ob  PAUSE [pause-code]

assignable-     is  [ derived-data-type-object % ]... object
object

object          is  array-object
                or  scalar-object

expr            is  [unary-operator] basic-expr
```

```
basic-expr    is  value
              or  constant-name
              or  assignable-object
              or  function-name ([actual-arg-list])
              or  basic-expr binary-operator basic-expr
              or  (expr)
```

Notes -

- Statement labels are not indicated in this summary; any statement may be labelled.

- The declare-statement allows "entity-oriented" declarations, where the entity name is given first, followed by a list of its attributes. This style of declaration complements the "attribute-oriented" declaration style of Fortran 77.

- The ENTRY statement is deprecated and not listed here. It is used to partition the local-specifications and executable-constructs in a program unit; internal-procedures provide equivalent functionality.

- The MODULE program-unit may not contain a block of executable-constructs (except as internal-procedure bodies) (see Section 2.5 below).

- BLOCK DATA program-units may contain only local-specifications.

- Note the END statement extensions, even though they are not italicized.

- Procedure-headings have minor extensions, even though they are not italicized (see Section 2.6 below).

- Note that, because an assignable-object may be an array or array section, expressions and assignments are considerably extended, even though they are not italicized (see Section 2.2 following).

- Restrictions on ordering of local-specifications are not shown here.

- The EXIT and CYCLE statements may only appear within block-do constructs; the RETURN statement may only appear within (internal and external) function and subroutine subprograms.

- The REFER statement associates a refer-name with a collection of attributes. The REFER (refer-name) attribute may be used in subsequent attribute lists to represent the collection.

2.2 Array Processing

The proposed array facilities view whole arrays and array sections as atomic computational objects from which expressions can be composed and to which assignment can be made. That is, a basic aim of array processing is to operate where possible directly on the arrays themselves rather than on their elements individually. All scalar operations are extended to conformable arrays, operating on them in an element-by-element fashion. Both user-written and intrinsic functions may return array values. Operations on whole arrays are thus made available in functional form.

The ability to manage and control storage of arrays has been significantly enhanced by addition of the following features: (1) automatic arrays (local arrays with variable dimensions) are created on entry to a procedure and destroyed on return; (2) allocatable arrays are created by execution of an ALLOCATE statement, and are destroyed by execution of a FREE statement or by return from the procedure in which they are created; (3) assumed shape arrays (dummy arrays) have implicit dimension information passed during a call, somewhat analogously to the passing of length information to Fortran 77 CHARACTER*(*) dummy arguments.

```
array-object      is  array-name
                  or  array-section [(substring-range)]

array-section     is  array-name (subscript-range[,subscript-range]...)

subscript-range   is  integer-expr
                  or  [integer-expr]:[integer-expr][: integer-expr]
                  or  one-dim-integer-array-expr

substring-range   is  [integer-expr]:[integer-expr]

array-            is  [ constructor-value [,constructor-value]... ]
constructor
                                  (note: outer brackets part of constructor)

constructor-      is  scalar-expr
value             or  integer-expr : integer-expr [: integer-expr]
                  or  repeat-factor  array-constructor

repeat-factor     is  integer-expr
```

```
array-assignment  is  array-object = array-expr
                  or  WHERE (array-logical-expr) array-object = array-expr
                  or  WHERE (array-logical-expr)
                            [array-object = array-expr]...
                      [OTHERWISE
                            [array-object = array-expr]... ]
                      END WHERE

                  or  FOR ALL (index-range [,index-range]... [,logical-expr])
                            assignment-statement

index-range       is  integer-variable = integer-expr:integer-expr[:integer-expr]

identify-         is  IDENTIFY <range> virtual-array = parent-array
statement

range             is  [lower-bound:] upper bound [,[lower-bound:]upper-bound]...

virtual-array     is  array-name (integer-variable [,integer-variable]...)

parent-array      is  array-name ( linear-mapping [,linear-mapping]... )

array-intrinsic-  is  arithmetic-logical-reduction-function
function          or  inquiry-function
                  or  construction-function
                  or  manipulation-function
                  or  geometric-location-function
                  or  MATMUL          matrix multiplication

arithmetic-       is  SUM             sum all elements of an array
logical-          or  PRODUCT         product of all elements
reduction-        or  MAXVAL          maximum value in an array
function          or  MINVAL          minimum value in an array
                  or  COUNT           number of true values in an array
                  or  ANY             true if any value in array is true
                  or  ALL             true if all values are true
                  or  DOTPRODUCT      dot product of two arrays

inquiry-          is  RANK            rank of argument array
function          or  EXTENT          size of array in each dimension
                  or  SIZE            total number of elements in array
                  or  LBOUND          lower bound in each dimension
                  or  UBOUND          upper bound in each dimension

construction-     is  SPREAD          replicates array by increasing rank
function          or  REPLICATE       replicates by increasing extent
                  or  MERGE           merges two arrays, using a mask
```

	or	DIAGONAL	creates a diagonal array
	or	PACK	packs array into a vector, with mask
	or	UNPACK	inverse of PACK
	or	SHAPE	changes the shape of a given array
manipulation-	*is*	CSHIFT	circular shift of argument array
function	*or*	EOSHIFT	end-off shift of argument array
	or	TRANSPOSE	matrix transpose of argument array
geometric-	*is*	FIRSTLOC	locate first true element
location-	*or*	LASTLOC	locate last true element
function	*or*	PROJECT	select masked values from array

Notes and Examples -

- Arrays of zero size are permitted.

- Two arrays are conformable if they are the same shape (same rank and same extent in each dimension).

- In executing array assignment statements the entire right hand side is evaluated before any assignment is made to the target array. This is significant when the same array (or sections of the same array) appears on both sides of the assignment operator.

- Element-by-element computation means that the operation takes place many times (all logically in parallel), once for each pair of corresponding elements of the operands, and the result is an array conformable with the operands. For example,

```
real  A(100), B(100)
A = A + B*3.0 + sin(B)
```

  performs A(I) = A(I) + B(I)*3.0 + SIN(B(I)) for all I between 1 and 100. Arithmetic, logical, and character operators, and scalar intrinsic functions operate in this manner.

- Whole array operations may be masked by FORALL and WHERE statements (and WHERE blocks), avoiding computation on certain elements and leaving portions of the target array unchanged:

```
real  A(100,100), B(100,100), THRESHOLD
where (B.ne.0)  A = A/B                    ! avoid zero-divide
where (A.gt.THRESHOLD)  A = THRESHOLD      ! flatten peaks
forall (I=1:100,J=1:100,I.ne.J)  A(I,J) = 0.0  ! keep diag
```

- Array-valued functions are analogous to scalar-valued functions:

```
real function CONCAT(A,B)
  real  A(:), B(:), CONCAT(size(A)+size(B))
  CONCAT(1:size(A))  = A
  CONCAT(size(A)+1:) = B
end function CONCAT
```

- An array-constructor forms a one-dimensional array from scalar values (note that array construction employs square brackets as delimiter symbols). This may be "shaped" into multi-dimensional arrays with the SHAPE intrinsic function. One use of constructors is to create array constants:

```
integer  A(10)
A = [1,2,3,4,5,0,0,0,0,0]   ! or  A = [1:5,5[0]]
```

- Generally wherever a whole array may be used an array section may be used. Examples of array sections of REAL A(8,8), B(10) are:

```
B(5:1:-2)          ! elements B(5), B(3), B(1)
A(1:7:2,2:8:2)     ! one color
A(2:8:2,1:7:2)     !           on a checkerboard
A(1:4,:)           ! upper half of A
```

- Sub-arrays and general sections may be selected and given new "virtual array" names by using the IDENTIFY statement:

```
real  C(1:100,1:100), D(1000)
identify <1:100> DIAG(I) = C(I,I)
```

Since IDENTIFY is an executable statement, the virtual array definition can be changed dynamically during execution:

```
identify <1:1000> DIAG(I) = D(1001-I)
```

- Example of the use of assumed-shape arrays:

```
function XYZ (A,B)
real  A(:,:), B(-1:,5:)
```

In Fortran 77 this would have to be accomplished with something like:

```
FUNCTION XYZ (A,B,I,J,K,L)
REAL  A(I,J), B(-1:K,5:L)
```

- The concepts of reduction, construction, manipulation, and

geometric-location correctly suggest that the array intrinsic functions provide general operations on multidimensional arrays. Only the functions MATMUL, TRANSPOSE, and DOTPRODUCT are specialized to matrix/vector operations. Following are some examples using array intrinsic functions, assuming the declarations:

```
real  X(N), Y(M), T(M,N), E(M,N)
complex  C(N,N)
```

Note that some operations may be masked (mask=), dim=1 specifies operations on columns, and dim=2 specifies operations on rows.

$\sum_{x_i > 0.1} x_i$ = sum(X,mask=X.gt.0.1)

$\sum_{i=1}^{n} (x_i - \bar{x})^2$ = sum((X-sum(X)/N)**2)

$\max_{1 \le i \le N} \left\{ \frac{\sum_j |c_{ij}| x_j}{x_i} \right\}$ = maxval(matmul(abs(C),X)/X)

radii $x_i = \sum_{j \ne i} |c_{ij}|$ of Gershgorin's circles, i=1,N

```
X = sum(abs(C),mask=.not.diagonal(.true.,N),dim=2)
```

chi-squared statistic $\chi^2 = \sum_{i,j} \frac{(t_{ij} - e_{ij})^2}{e_{ij}}$ where $e_{ij} = (\sum_k t_{ik}) * (\sum_k t_{kj}) / (\sum_{k,l} t_{kl})$

```
X = sum(T,dim=1)                          ! column sums
Y = sum(T,dim=2)                          ! row sums
E = spread(Y,2,N)*spread(X,1,M)/sum(T)    ! outer product
CHI_SQ = sum((T-E)**2/E)
```

- Examples of queries without loops or conditional code on:

```
real  T(M,N)    ! test scores, M students with N tests
```

top score for each student:	maxval(T,dim=2)
no. scores above average:	count(T.gt.sum(T)/size(T))
lowest score above average:	minval(T,mask=T.gt.sum(T)/size(T))
any student all above average?	any(all(T.gt.sum(T)/size(T),dim=2))

2.3 Numeric Data Type

These facilities give the programmer portable control over the specification of real and complex data objects; they extend the REAL and COMPLEX type statements. The environmental-intrinsic-functions provide information that models the processor-supplied numeric system. They provide a portable means for adapting numeric algorithms to various arithmetic environments.

```
floating-point-     is  REAL [(float-point-attr[,float-point-attr])]
data-type           or  COMPLEX [(float-point-attr[,float-point-attr])]

float-point-attr    is  [PRECISION10=]  attribute-value
                    or  [EXP_RANGE=]  attribute-value

attribute-value     is  integer-constant-expr
                    or  *

real-constant-char  is  REAL_CHAR [(float-point-attr[,float-point-attr])] letter

floating-point-     is  ACTUAL_PREC (floating-point-argument)
inquiry-function    or  ACTUAL_EXP_RANGE (floating-point-argument)

environmental-      is  RADIX          model base for numbers
intrinsic-function  or  PRECISION      number of significant digits
                    or  MINEXP         smallest exponent value
                    or  MAXEXP         largest exponent value
                    or  HUGE           largest number of argument type
                    or  TINY           smallest positive number
                    or  EPSILON        positive number small relative to 1.0
                    or  EXPONENT       exponent value of argument
                    or  SCALE          scale argument by power of base
                    or  NEAREST        different value nearest to argument
                    or  FRACTION       fractional part of argument
                    or  SETEXPONENT    specify exponent part of argument
                    or  ABSSPACE       absolute spacing of numbers near argument
                    or  RECSPACE       reciprocal of relative spacing
```

Notes and Examples -

- The floating point attributes provide specifications for minimum numeric properties of the processor-supplied floating point system used to implement the relevant data objects.

- The integer constant expressions in the floating point attributes may reference the functions ACTUAL_PREC and ACTUAL_EXP_RANGE, which return the actual precision and actual exponent range for the given datum.

- In any given floating-point-data-type definition each float-point-attr alternative may appear at most once; if the float-point-attr keywords are not used, these attributes must appear in the order listed in the above definition of float-point-attr.

- A type specification with an attribute of, for example, REAL(10), for an entity X is a requirement that X be represented by a processor data type that has at least 10 decimal digits of precision. The uses of X in expressions, say, determines the precision of the arithmetic operations. Such a specification provides a portable method of indicating that the algorithm requires at least 10 decimal digits of precision.

- The function value ABSSPACE(X) for example can be used to terminate an iteration in a portable way by requiring that the absolute relative difference between two iterates, X and Y, is less than ABSSPACE(X) - that is, terminates the iteration when abs(X-Y).le.absspace(X):

```
do; if (abs(X-Y).le.absspace(X)) exit
      ...
repeat
```

- A common difficulty with transporting numerical software from machine to machine is the difficulty with changes of precision, say from single precision on one machine to double precision on another. This problem particularly arises for the conversion of constants, which are typically spread throughout a program and cannot be easily converted when the precision of the data types is changed. For instance, consider the constant 1.1 which cannot be represented exactly on a binary (or hexadecimal) machine. When changing from single precision to double precision the constant 1.1 must be rewritten in the program as 1.1D0 in order to obtain the double precision value of this datum. Using the REAL_CHAR specification to specify an exponent character, the precision of all constants that use this exponent character can be readily changed.

- Another common problem in writing portable software is the safe and accurate range conversion of floating point variables to specified forms. For example, to determine the square root of a number X, the usual Newton's iteration converges too slowly to be a viable algorithm if the initial iterate is too far from the square root. Using the intrinsic functions FRACTION and EXPONENT, the interval over which the iterates can range can be drastically reduced, from which the square root can be efficiently computed.

From this result and EXPONENT(X) the square root of X can be formed using the SETEXPONENT intrinsic function. Thus, the program would look like:

```
! Perform the range reduction on X, assuming X is positive.

  IEXP = exponent(X)
     F = fraction(X)
  if (mod(IEXP,2).eq.1) then; IEXP=IEXP+1; F=F/radix(X); endif

! Now X = F*radix(X)**IEXP, IEXP is even, and the square root
! of F is in the interval [1/radix(X)**2,1). Now find SF, the
! square root of F, and reconstruct the answer from SF and IEXP:

  ANSWER = setexponent(SF,IEXP/2)
```

2.4 Derived Data Types

The structure of a derived data type is a programmer-defined aggregation of fields, each field being a data element of primitive type (or another derived data type). The aggregation pattern may be fixed or variant (that is, partially dependent on one of the prior fields). A derived data type is defined with a TYPE construct; variables of this type may be declared in the normal manner. Intrinsic operations on derived data objects are assignment, equality comparison, input/output, and use as procedure arguments. These intrinsic operations may be augmented with additional programmer-defined operations. The structure qualification symbol (for referencing a component of a structured object) is the percent sign (%).

```
type-definition        is  TYPE type-name
                               [field-declaration]...
                               [variant-case-block]
                           END TYPE [type-name]

variant-case-block     is  SELECT CASE (tag-field-name)
                               [ CASE case-selector
                                    [ field-declaration]... ]...
                               [ variant-case-block ]
                           END SELECT

field-declaration      is  type-declaration
                       or  declare-statement

component-selection    is  structured-object-name % field-name
```

type-constructor *is* type-name (expr[,expr]...)

type-constant *is* type-name (constant-expr[,constant-expr]...)

Notes and Examples -

- Functions may return derived data type values.

- The following definition of PLOT_OBJECT defines a data structure made up of two fields of type real (the location in two-space), one field indicating the nature of the object, and finally the object data itself. In this case the object may be either a point symbol or a line segment.

```
type PLOT_OBJECT
   X0,Y0: real                  ! object location
    CODE: integer               ! type of object
          select case (CODE)
                 case (1);  SYMBOL: character  ! point symbol
                 case (2);   X1,Y1: real       ! line segment
          end select
end type PLOT_OBJECT
```

Let P,Q: type (PLOT_OBJECT) declare two variables P and Q.
Then the following expressions are examples of valid operations:

```
( P%X0 + Q%X0 ) / 2             ! mid-point of P and Q along X
if (P%CODE.eq.Q%CODE) then      ! compare P and Q CODE fields
P = PLOT_OBJECT(0.,0.,1,"+")    ! give P an initial value
read *, Q                       ! input value of Q
```

2.5 Modules

Modules are proposed program units for the packaging of data type definitions, data object declarations, procedure definitions, and procedure interfaces. Modules are not themselves executable. A module that contains only data type definitions and data object declarations serves as a global data module. One that contains only procedure definitions and interfaces serves as a procedure library. One that contains data structure definitions and operations (internal procedures) on them serves as a data abstraction mechanism. A module may serve as a language extension mechanism by containing new operation definitions on intrinsic data types.

The use-statement "attaches" the specified module to the using program unit, thereby making the public definitions and declarations in the module available to the program unit. The use-statement without the module-name is for internal procedures specifying access to its host's definitions (note that in the absence of a use-statement specifying otherwise, all host definitions are automatically available to an internal procedure).

Through options on the use-statement, a using program may limit its access to definitions in the module and may rename the entities it uses. On the other hand, the writer of a module has complete control over which module entities are public (visible to using programs) and which are private; in the absence of explicit specification to the contrary, the default is public. (The syntax for specifying public and private is not shown, nor is the statement for specifying that the default visibility is private instead of public.)

```
use-statement      is  USE[/module-name/] ONLY (access[,access]...)
                   or  USE[/module-name/] [ALL[(rename[,rename]...)]
                                          [EXCEPT(identifier[,identifier]...)]]

access             is  identifier
                   or  rename

rename             is  identifier = identifier
```

Notes and Examples -

- An example of the use of modules for defining global data pools may be found in Section 3.3.2.4 below.

- A scheme for using modules to encapsulate procedure libraries may be found in Section 3.3.2.5 below.

- The following is an example of the use of modules as a mechanism for encapsulating data abstractions; in this case the abstract data type is PLOT_OBJECT, defined above, with two defined operations CONNECT and BISECT:

```
  module  PLOT_MODULE

! insert definition of type PLOT_OBJECT from previous section

! define an operation to connect two plot_objects with a
! line segment; CONNECT returns a PLOT_OBJECT value that is
! a line segment (CODE=2) connecting two given plot_objects

  internal type(PLOT_OBJECT) function CONNECT(P,Q) operator(//)
```

```
                                                P,Q: type(PLOT_OBJECT)
        CONNECT%CODE = 2
        CONNECT%X0   = P%X0
        CONNECT%Y0   = P%Y0
        CONNECT%X1   = Q%X0
        CONNECT%Y1   = Q%Y0
                                  ! finished if P is a point symbol,
                                  ! otherwise use tail (X1,Y1) of P
                                  ! as the head (X0,Y0) of CONNECT
        if (P%CODE .eq.2) then
           CONNECT%X0 = P%X1
           CONNECT%Y0 = P%Y1
        end if
      end internal function CONNECT

    ! note that two plot_objects, A and B, can be connected
    ! using the infix operator notation  A // B


    ! define an operation to bisect a line segment, with an "x";
    ! BISECT returns a point symbol PLOT_OBJECT value

      internal type(PLOT_OBJECT) function BISECT(P)
                                                P: type(PLOT_OBJECT)
        BISECT = P  ! return P itself if P is not a line segment
        if (P%CODE.eq.2) then
           BISECT%CODE   = 1
           BISECT%SYMBOL = "x"
           BISECT%X0     = (P%X0+P%X1)/2
           BISECT%Y0     = (P%Y0+P%Y1)/2
        end if
      end internal function BISECT

   end module PLOT_MODULE
```

Any program unit may employ these definitions if it contains the statement:

```
use /PLOT_MODULE/
```

2.6 Procedures

Procedures are allowed to be called with keyword actual arguments, called with optional arguments, defined with argument intent (e.g., input only), and called recursively. A keyword actual-argument is of the form KEYWORD= actual-argument, where KEYWORD is a dummy-argument-name. This has three advan-

tages: (1) if dummy-argument name is wisely chosen, then keywording effectively increases readability of the actual-argument-list, (2) keyworded actual-arguments may be placed in any order, which eliminates order errors (such as transposing two adjacent arguments) in actual-argument-lists, and (3) arguments not needed on a particular call may be omitted.

Procedure interface information may be provided for any external or dummy procedure, including procedures defined by non-Fortran means. This feature gives dummy-argument information to calling programs (and for called functions, function type information), which is needed for using keyword and optional arguments. It also makes possible validation of actual argument types and number. Procedure interfaces, via the INTERFACE block, may appear directly in the specifications of calling programs. However, more often interface information will be packaged in modules, which are then USEd by the calling program.

In Fortran 8x, it is proposed that procedure definitions may be made within any program unit. Such a procedure is called an "internal procedure" and is known only locally within the program unit in which it is defined. The form of internal procedures is identical to that of external procedures except for the INTERNAL on the heading and END statements. Also, internal procedures automatically inherit all host data and procedure definitions, unless such inheritance is explicitly suppressed with the use of the USE statement (e.g., USE ONLY () suppresses all inheritance from the host). Internal functions provide the same functionality as statement functions, but are more flexible and not so restricted. A common use of internal subroutines will be as "remote code blocks". Internal procedures are called in exactly the same way as external procedures.

An important use of internal functions is to provide operations on derived data types. Since infix operators are so common in scientific notation, an option is provided with internal functions to allow them to be used as infix operators. Similarly an option with internal subroutines allows the assignment operator (=) to be used with programmer-defined data conversions. Thus, the various features of internal procedures, together with derived data type definition and module encapsulation facilities, gives Fortran 8x a powerful and flexible data abstraction mechanism.

```
function-heading      is  [RECURSIVE] [data-type] FUNCTION function-name
                                  ([dummy-argument-list]) [RESULT(identifier)]

subroutine-heading    is  [RECURSIVE] SUBROUTINE subroutine-name
                                  [([dummy-argument-list])]

procedure-reference   is  function-name ([actual-argument-list])
                      or  CALL subroutine-name [([actual-argument-list])]
```

```
actual-argument-list  is  positional-argument-list [, keyword-argument list]
                      or  keyword-argument-list

positional-argument-  is  expr [, expr ]...
list

keyword-argument-     is  dummy-argument-name= expr [,dummy-argument-name= expr]...
list

optional-attribute    is  OPTIONAL

intent-attribute      is  IN
                      or  OUT
                      or  IN OUT

present-intrinsic-    is  PRESENT (dummy-argument-name)
function

procedure-interface   is  INTERFACE
                             [interface-description]...
                          END INTERFACE

interface-            is  function-heading
description                   [local-specification]...

                      or  subroutine-heading
                              [local-specification]...

internal-procedure    is  internal-procedure-heading
                                [use-statement]...
                                [local-specification]...
                                [executable-construct]...
                                [internal-procedure]...
                          END INTERNAL [unit-type[unit-name]]

internal-             is  INTERNAL function-heading [OPERATOR (operator-symbol)]
procedure-heading     or  INTERNAL subroutine-heading [ASSIGNMENT]

operator-symbol       is  . identifier .
                      or  intrinsic-operator-symbol
```

Notes and Examples -

- Arguments are optional, as specified by the OPTIONAL attribute; the PRESENT function may be used to determine if a given optional argument was actually supplied in the call.

- No positional arguments may appear after the first keyword argument.

- OPERATOR functions have one or two arguments (the operands of the operation); such functions may be called using either the functional form or the infix operation form. The intrinsic operator symbols (+, -, *, /, **, //, .AND., .OR., etc.) may be overloaded. Alternatively any .name. form may be defined as an operator symbol.

- ASSIGNMENT subroutines have two arguments, the first one being the left-hand-side of the assignment (the entity being defined), and the second being the value to be assigned. The usual assignment syntax (with the = sign) may be used in assignment procedure calls.

- In the case of both operator and assignment definitions, the procedure body defines in detail the intended operations.

- Note that internal procedures may be nested.

- The RESULT identifier may be used with recursive functions to remove ambiguity between recursive calls and function value assignment (RESULT may also be used with non-recursive functions, including OPERATOR definitions).

- The attributes of procedure headings (INTERNAL, RECURSIVE, etc.) may appear in any order.

- See the preceding section for examples of internal procedures.

- The following two examples illustrate recursion and the use of the PRESENT function:

```
recursive integer function ACKERMANN(M,N) result (ACK)
                                        M,N: integer
        if (M.eq.0) then;  ACK = N+1
    elseif (N.eq.0) then;  ACK = ACKERMANN(M-1,1)
    else                ;  ACK = ACKERMANN(M-1,ACKERMANN(M,N-1))
        end if
end function ACKERMANN

real function READNUM (UNIT,FMT)
                      UNIT: integer, in, optional
                       FMT: character(*), in, optional
    LUN,IOS: integer
    LUN = 5; if (present(UNIT))  LUN = UNIT  ! establish unit no.
```

```
    if (present(FMT)) then
       read (LUN,FMT,iostat=IOS)  READNUM
       if (IOS.ne.0)  READNUM = 0
    else
       read (LUN,*)  READNUM
    end if
end function READNUM
!  X = READNUM ()      reads from unit 5,  list-directed format
!  X = READNUM (10)    reads from unit 10, list-directed format
!  X = READNUM (FMT= '(F10.2)' )    reads unit 5, format F10.2
```

- The above two examples are external procedures and, as such, the routines calling them do not normally have any interface information about them. In Fortran 8x, the use of an interface block, which provides calling routines with interface information, allows the compiler to check for correctly formed procedure calls. The following example is an interface block that contains interface information pertaining to the above procedures READNUM and ACKERMANN. This interface block may be placed in either the calling routine itself, or a module being used by the calling routine.

```
interface
   recursive integer function ACKERMANN(M,N)
                                          M,N: integer
   real function READNUM(UNIT,FMT)
                          UNIT: integer, in, optional
                           FMT: character(*), in, optional
end interface
```

2.7 Program Source Form

The Fortran 8x source form is completely column-independent, and source lines (records) may be any length (a limit of 1320 characters per source statement is, however, still the rule). Statements may be separated on a line with the ";", "!" (not in a character context) initiates a comment (either on a line by itself or following a statement), and "&" at the end of a line signifies continuation. Blanks are significant characters, and identifiers may not contain embedded blanks; conversely, blanks must separate adjacent identifiers and keywords. Identifier names may be up to 31 characters long and may contain the "_" (underscore) character. Upper and lower case may be used interchangeably (except in CHARACTER constants). Either apostrophes or quotation marks may be used as character constant delimiters.

The following is a more formal description of much of the new source form.

```
source              is  [statement ;]... [statement-start] [comment] eol
                    or  [statement-continuation] [comment] eol

statement-start     is  statement [;]
                    or  statement-fragment &

statement-          is  [&] statement-fragment [; [source]]
continuation        or  [&] statement-fragment &

comment             is  ! [character]...

character-          is  ' [character]... '
constant            or  " [character]... "
```

Notes -

- A statement-fragment is a contiguous portion of a statement (and may be null). A statement-fragment may end in the midst of a (continued) character constant or H, apostrophe or quotation mark edit descriptor only if there is no trailing comment.

- If a statement-fragment (in a statement-continuation) is preceded by a "&", any blanks preceding that "&" are insignificant.

- The eol stands for the end of a source line of text.

- If the underscore is used in an identifier it must not be the first character of the identifier;
  the underscore is a significant character.

- If the character used to delimit a character-constant (' or ") is to appear within the constant itself, it must appear exactly twice in succession (i.e., " "" " is the same as ' " ' and ' '' ' is the same as " ' ").

- Many of the examples in this document illustrate aspects of the Fortran 8x program source form.

2.8 Control Constructs

Two control constructs are added, one for loop control and one for case selection:

```
block-do        is  DO [(loop-control)]
                        [executable-construct]...
                    REPEAT

loop-control    is  integer-variable = integer-expr, integer-expr [,integer-expr]
                or  integer-expr TIMES

block-case      is  SELECT CASE (enumerable-expr)
                          [CASE case-selector
                                [executable-construct]... ]...
                    END SELECT

case-selector   is  (value-range [,value-range]...)
                or  DEFAULT

value-range     is  constant-expr
                or  [constant-expr]:[constant-expr]
```

Notes -

- Branches into block constructs are not allowed.

- The block-do specifies repetitive execution of its block of loop-statements until the loop-control (if any) is satisfied, or until an EXIT statement is executed.

- An EXIT statement is allowed only within a block-do (it may be in a block-if or block-case that is one of the loop-statements); its execution terminates execution of the innermost block-do containing the EXIT statement.

- Execution of the CYCLE statement causes the next iteration of the loop to commence.

- The indexed form of loop-control is semantically identical to the Fortran 77 DO statement with an index of type integer.

- Enumerable-expr is an expression of type integer, character, or logical; constant-expr is a constant expression of the same type as enumerable-expr.

- The case-selectors must be disjoint in any given block-case.

- CASE DEFAULT is optional and may not appear more than once in any given block-case; if present, it may appear in any position among the sequence of CASE clauses.

- The block-case causes execution of the block corresponding to the case-selector that contains the value of the enumerable-expr; if there is no such case-selector then the CASE DEFAULT block is executed; if there is no matching case-selector and there is no DEFAULT block, an error exists.

- Block constructs (if, do, case) may be named by placing an alphanumeric name to the right of each of the control statements (e.g., do, repeat, else if, etc.). EXIT and CYCLE statements may contain the name of the block-do construct to be exited or cycled.

2.9 Input/Output

Input/Output additions include several new OPEN statement specifiers, name-directed I/O, and new edit descriptors.

position-specifier *is* POSITION= position-expr

action-specifier *is* ACTION= action-expr

delimiter-specifier *is* DELIM= delim-expr

name-directed-format-specifier *is* [FMT=] **

name-directed-input *is* variable-name = data-value [, data-value]...

slash-edit *is* [repeat-count] /

engineering-edit *is* [repeat-count] EN width . digits [E exponent]

Notes -

- The position-specifier specifies the position of the file upon open; position-expr must evaluate to one of the character values 'REWIND', 'APPEND', or 'ASIS'.

- The action-specifier specifies read-only, write-only, etc; action-expr must evaluate to one of the character values 'READ', 'WRITE', or 'BOTH'.

- The delimiter-specifier is used in conjunction with name-directed I/O; delim-expr must evaluate to one of the character values 'NONE', 'QUOTE', or 'APOSTROPHE'.

- The engineering-edit descriptor causes 1-3 digits to be displayed to the left of the decimal point such that the exponent is a multiple of three.

2.10 Event Handling

The event handling mechanism provides a means to transfer control on the occurence of an event to a specified program unit called an event handler. Events must be declared, and their monitoring can be selectively switched on and off to accommodate the desired level of optimization.

An eventmark is a data object that registers the occurence of an "event". The declaration of an eventmark contains the condition for which the event will occur. The value of the eventmark is either .ON. or .OFF. (the constants of the data-type EVENTMARK). The scoping and definition rules for eventmarks are the same as for any other data type.

Event handlers are similar to subroutines, except that they do not have arguments and cannot be called. When an event occurs, the connected handler is automatically invoked. Upon completion of a handler, execution may either RESUME with the statement following the one in which the event occurred, or RETURNUP (return from the procedure in which the handler was invoked).

event-handling-statement	*is*	ACTIVATE (event-name-list)
	or	DEACTIVATE (event-name-list)
	or	CONNECT (event-name, handler-name)
	or	DISCONNECT (event-name)
	or	DISCONNECT (handler-name)

Notes -

- The intrinsic operations on eventmarks are assignment and test for equality.

- An optional part of the eventmark declaration (not shown) allows the specification of the condition(s) under which the eventmark is automatically set on.

2.11 Other Features

Other added features include IMPLICIT NONE for turning off implicit typing and several new CHARACTER intrinsic functions. In addition, restrictions have been removed on: assignment from overlapping character positions, zero-length character strings, concatenation of CHARACTER dummy arguments, and specifying character constants.

character-	*is*	IACHAR	same as ICHAR, but based on ASCII
intrinsic-function	*or*	ACHAR	same as CHAR, but based on ASCII
	or	TRIM	remove trailing blanks from string
	or	VERIFY	check for unwanted characters
	or	ADJUSTL	circular shift left thru leading blanks
	or	ADJUSTR	circular shift right thru trailing blanks
	or	REPEAT	replicate a string of characters
	or	ISCAN	first position of any character from set
implicit-list	*is*	implicit-type [, implicit-type]...	
	or	NONE	
implicit-type	*is*	type (letter-range [, letter-range]...)	

3. LANGUAGE ARCHITECTURE

The proposed Fortran 8x language contains all of the features of Fortran 77, some of which are identified as "deprecated," and additional new features as described in Section 2 above. The "core" is that set of language features that are not identified as deprecated. "Modules" are nonexecutable program units which contain prepackaged definitions to be used by executable program units.

3.1 The Core

The core is a complete and consistent language comprising a set of language features sufficiently rich for the implementation of most applications. The core will have at least the same functional capabilities as Fortran 77. In addition, the core should

(a) be especially suitable for scientific applications,

(b) be portable,

(c) be safe to use, and effective for the development of reliable software,

(d) be widely efficiently implementable,

(e) be concise,

(f) comprise generally accepted contemporary language technology,

(g) minimize non-automatable conversion from Fortran 77.

A core-conforming program contains only core features (i.e., does not contain any deprecated features).

3.2 Modules

Modules provide a mechanism for defining program facilities for subsequent use in application programs. Such facilities include global data pools, procedure libraries, procedure interface definitions, data abstractions, and language extensions in the form of new operators. Modules take the form of MODULE program units, and contain data type definitions, data object declarations, procedure definitions, and procedure interface definitions. The USE statement provides the mechanism for using the public contents of a module in an application program.

Modules are useful, for example, where the same definitions are needed in a number of different program units, because they allow such definitions to be made only once in a "central" place. This should contribute substantially to software reliability and reusability. The use of modules can also contribute to execution efficiency through in-line expansion of procedures. Procedure libraries may be configured as internal procedures in a module and thereby made available for in-line expansion in the using program units.

Modules may be individually standardized, so that standard facility definitions are available to implementors for efficient implementation (e.g., to take advantage of specialized hardware). An example might be a module for bit data type, which defines the structure of and operations on bit data; standardizing such definitions allows implementors to efficiently implement a standard bit data facility if they so choose.

Since a module contains "ready to use" definitions, a standard module provides standardized facilities without an added cost burden to a standard-conforming Fortran implementation. Therefore, such facilities are part of any Fortran implementation, regardless of whether or not the implementor chooses to optimize implementation of them. For this reason, module standardization is an extremely powerful and cost-effective mechanism for extending the functionality of standard Fortran.

A standard module must be core-conforming. Operators defined in the module must not have the potential to alter the meaning of any core-intrinsic operation. The module must contain, in the form of appropriate commentary, functional descriptions of all module operations; such functional descriptions take precedence over any accompanying procedural implementations of functionality. A functional description may be a (set of) mathematical statement(s), or take any other form appropriate for comprehensive specification of the functionality. In addition, the module must contain a succinct and lucid description of the use of the module facilities.

Initial possibilities for standard modules include

bit data type
maximum accuracy arithmetic
varying-length character (string data type)

Standard procedure libraries, such as the ISA process control library, the Industrial Real Time Fortran (IRTF) library, and the Graphical Kernel System (GKS) library are candidates for standard modules.

Standard modules are separate standards, and are not part of the Fortran standard itself. There is a close relationship between standard Fortran and standard modules, of course, which gives rise to the concept of a "Fortran

family of standards". Any group, including X3J3, may propose standard modules. In addition X3J3 may choose to include certain standard modules in a part of the standard Fortran document.

3.3 Deprecated Features

As described in Section 1.5, certain features of the proposed Fortran 8x language are identified as deprecated and as such are identified as candidates for removal from subsequent versions of the Fortran standard. In this section, the deprecated features are summarized, together with a discussion of possible replacements for needed functionality.

3.3.1 Summary of Deprecated Features

1. Fortran 77 source form (e.g., restricted use of columns 1-6)
2. alternate RETURN
3. assumed size dummy arrays
4. passing a scalar (e.g., array element or substring) to a dummy array
5. specific names for intrinsic functions
6. statement functions
7. BLOCK DATA program unit
8. arithmetic-if statement
9. ASSIGN statement
10. assigned-goto statement
11. COMMON statement
12. computed-goto statement
13. DATA statement
14. DIMENSION statement
15. DOUBLE PRECISION data type
16. ENTRY statement
17. EQUIVALENCE statement
18. Fortran 77 DO statement
19. PAUSE statement

3.3.2 Storage Association

Storage association is the association of data objects through physical storage sequences, rather than by object identification. Storage association allows the user to configure regions of physical storage, and to conserve the use of storage by dynamically redefining the nature of these storage regions. Though the considerable dangers of programmer access to physical storage sequences have been well known for a long time, not until Fortran 8x has Fortran had adequate replacement facilities for certain important functionality provided by storage association. Six items in the above list (items 3, 4, 7, 11, 16, and 17) are due to the identification of storage association as deprecated.

3.3.2.1 Assumed Size Dummy Arrays

These are dummy arrays with asterisk as the size of the last dimension. In Fortran 8x dummy arrays may also be assumed shape, in which case the extent in each dimension is that of the actual argument. Assumed-shape arrays provide all of the functionality of assumed-size ones, and more. Assumed-size arrays assume that a contiguous block of storage is being passed, whereas with assumed-shape arrays an array section not having contiguous physical storage (such as a row of a matrix) may be passed also.

3.3.2.2 Passing a scalar (e.g., Array Element or Substring) to a Dummy Array

This functionality is now achieved, and much more safely, by passing the desired array section. For example if a one-dimensional array XX is to be passed starting with the sixth element, then instead of passing XX(6) to the dummy array one would pass the array section XX(6:). If the eleventh through forty-fifth elements are to be passed, the actual argument should be the array section XX(11:45).

3.3.2.3 BLOCK DATA Program Unit

The principal use of BLOCK DATA subprograms is to initialize COMMON blocks. The global data functionality of COMMON is alternatively provided by MODULE program units; global data in modules may be initialized at the point of declaration, and thus modules provide a complete replacement option for BLOCK DATA subprograms.

3.3.2.4 COMMON Statement

The important functionality of the COMMON statement has been its use in providing global data pools. In Fortran 8x, global data pools may also be provided, and more safely and conveniently, with MODULE program units and USE statements. Suppose that it is desired to have a global data pool consisting of the following:

```
INTEGER  X(1000)
REAL  Y(100,100)
COMMON /POOL1/  X,Y
```

Each program unit using this global data would need to contain these specifications. Alternatively, one can define the global data pool in a MODULE program unit:

```
MODULE  POOL1
   INTEGER  X(1000)
   REAL  Y(100,100)
END MODULE
```

Then each program unit using this global data would contain the statement

```
USE /POOL1/
```

This is safer than using COMMON, because the structure of the global data pool appears in only one place. In addition the USE statement is very short and easy to use. Facilities are provided in the USE statement (but not shown here) to rename module objects if different names are desired in the program unit using the module objects.

Modules have another advantage: they do not involve storage association, and therefore a module can contain any desired mix of character, non-character, and structured objects. Since COMMON involves storage association, a given COMMON block cannot contain both character and non-character data objects.

3.3.2.5 ENTRY Statement

The ENTRY statement is typically used in situations where there are several operations involving the same set of data objects:

```
procedure-heading
   local-specifications
entry1
   ...
   RETURN
entry2
   ...
   RETURN
...
entryn
   ...
   RETURN
END
```

The MODULE program unit provides the same functionality:

```
MODULE  module-name
   local-specifications
INTERNAL procedure1
      ...
   END INTERNAL
INTERNAL procedure2
      ...
   END INTERNAL
...
INTERNAL proceduren
      ...
```

```
    END INTERNAL
END MODULE
```

A program unit using this module may call each internal procedure in it, exactly as if they were entry points. An advantage is that some of the internal procedures in a module may be functions and some may be subroutines, whereas all of the entry points in a function procedure must be functions and in a subroutine must be subroutines.

3.3.2.6 EQUIVALENCE Statement

The primary purpose of the EQUIVALENCE statement is to allow the programmer to associate, within a program unit, two or more data types with the same storage region. There is currently no direct alternative for this functionality in the Fortran 8x proposals.

There were two principal reasons for the EQUIVALENCE statement in early versions of Fortran. The first is that memory address spaces were typically quite small, and equivalencing was needed to reuse the available space for different purposes. The second reason is that early versions of Fortran did not have a rich set of data types and structures and transfer functions between types, so that equivalencing was useful in simulating certain data types and structures. Both of these reasons have now largely disappeared.

Much of the same practical effect of reusing physical storage can, in fact, be achieved with the proposed facilities without using the EQUIVALENCE statement. Dynamic local arrays may be of any size, controlled by procedure arguments. Since dynamic arrays "disappear" upon return from the procedure, other procedures may use this space for other dynamic arrays of different shapes, sizes, and types. Similarly, allocatable arrays disappear when they are freed explicitly and on return from the procedures in which they were allocated. The IDENTIFY statement allows part of an array to be referred to as a completely different object (but of the same type).

3.3.3 Redundant Functionality

A number of features are identified as deprecated simply because they are now completely redundant, having been superseded by superior language features. These redundant features are items 1, 5, 6, 8, 12, 13, 14, 15, and 18 in the list above.

Fortran 77 source form -- replaced by the new source form (Section 2.7 above)

specific names for intrinsic functions -- use generic names

statement functions -- replaced by internal functions

(see 3.3.3.1. below)

arithmetic-if statement	-- replaced by logical-if and block-if
computed-goto statement	-- replaced by block-case construct (see 3.3.3.2 below)
DATA statement	-- replaced by INITIAL attribute (Section 2.1), and by assignment of array constants (Section 2.2)
DIMENSION statement	-- use type declarations instead
DOUBLE PRECISION statement	-- use precision control attributes (Section 2.3)
Fortran 77 DO statement	-- replaced by block-do construct (Section 2.8)

3.3.3.1 Use of Internal Functions for Statement Functions

The statement function definition

```
function-name(dummy-argument-list) = expr
```

may be completely replaced by the internal function definition:

```
INTERNAL FUNCTION function-name(dummy-argument-list)
                  type-specification-of-dummy-arguments
                  function-name = expr
END INTERNAL
```

The use of the internal function in the executable code is the same as the use of the statement function.

3.3.3.2 Example Replacement of the computed-goto Statement

The code sequence controlled by the computed-goto:

```
GOTO (label1, label2, ..., labeln), integer-variable
   ...
   GOTO labelz
label1 CONTINUE
   ...
   GOTO labelz
label2 CONTINUE
   ...
   GOTO labelz
...
labeln CONTINUE
   ...
   GOTO labelz

labelz CONTINUE
```

may be replaced by the block-case construct:

```
SELECT CASE (integer-variable)
       CASE DEFAULT
                 ...
       CASE (1)
                 ...
       CASE (2)
                 ...
       ...
       CASE (n)
                 ...
END SELECT
```

3.3.4 Other Deprecated Features

The remaining obsolete features (items 2, 9, 10, and 19) in the above list are neither related to storage association nor directly replaced by superior features. They are all in some sense "bad practice" in terms of generally accepted software principles, however, and their effects may be achieved in ways that are generally now considered better software practice.

3.3.4.1 Alternate RETURN

Alternate returns introduce control (branch point) information into argument lists, and then one (the called) program unit can directly control execution sequence in another (the calling) program unit. This tends to complicate the readability and maintainability of software using this feature. Better practice, in terms of readability and maintainability, is to use a data element in the argument list to contain a "return code", which can then be used in the

calling program to explicitly control the execution sequence in the calling program. For example, consider the following use of alternate returns:

```
CALL subr-name(X,Y,Z,*100,*200,*300,...)
GO TO 999
100 CONTINUE
    ....
    GOTO 999
200 CONTINUE
    ...
    GOTO 999
...
999 CONTINUE
```

where labels 100, 200, 300, etc., are the beginnings of the code blocks to be selected from upon return. In many cases, the effect can be more safely achieved with a return code and a block-case structure:

```
CALL subr-name(X,Y,Z,RETURN_CODE)

SELECT CASE (RETURN_CODE)
       CASE ('condition1')
            ...
       CASE ('condition2')
            ...
       ...
END SELECT
```

Alternatively, RETURN_CODE could be used to explicitly control direct branches (GOTOs) to the desired branch points. Maintainability is enhanced with the use of RETURN_CODE because a new selection case can be added in a convenient, straightforward manner, without modifying the actual and dummy argument lists.

3.3.4.2 ASSIGN and assigned-goto

The ASSIGN statement allows a label to be dynamically assigned to an integer variable, and the assigned-goto statement allows "indirect branching" through this variable. The dangers of such dynamic indirection, especially when mixed with integer arithmetic operations, have long been recognized, and a comprehensive alternative to this functionality is not provided. An important practical use of such indirect branching, however, is to return from simulated internal subroutines. Consider the following example code, which "calls" one of several "internal subroutines" in the program unit:

```
ASSIGN 120 TO RETURN        ! set up return point
GOTO 740                    ! branch to "subroutine"
120 CONTINUE
```

```
...
740 CONTINUE
    ...                          ! "subroutine" body
GOTO RETURN
...
```

This functionality is provided in a much better way, through the use of internal subroutines:

```
CALL subroutine-name
...
INTERNAL SUBROUTINE subroutine-name
    ...              ! subroutine body
END INTERNAL
...
```

This illustrates the use of internal subroutines to conveniently provide "remote code block" functionality.

3.3.4.3 PAUSE Statement

Execution of a PAUSE statement requires operator/system-specific intervention to resume execution. In most (if not all) cases, the same functionality can be achieved as effectively and in a more portable way with the use of appropriate READ statements.

end of Fortran Information Bulletin

DIGITAL RESPONSE TO FIB-1

Although Digital Equipment is in favor of the publication of Fortran Information Bulletin 1, we believe that certain changes must be made before it is published, we, therefore, vote NO until the following issues are resolved.

1. We are very concerned about compatibility with existing Fortran programs. Although the coverletter says "Programs currently standard-conforming will not become non-standard-conforming.", we are forced to wonder what this means in the light of statements in the FIB itself such as:

> "minimize non-automatic conversion from Fortran 77" section 3.1, page 29
>
> "Blanks are significant characters, and identifiers may not contain embedded blanks", section 2.7, page 23

Can a Fortran-77 program be modified so that some of its statements use new features, such as the new source format or the new declarations? What restrictions are there, if any, on the use of newly defined features, such as array arithmetic, on variables that appear in COMMON or EQUIVALENCE statements?

Digital would like to see an explicit compatibility statement somewhere near the beginning of the FIB.

2. While we believe that there is a definite requirement for the array and data structure language extensions, we are not certain that many of the other features are worth the performance degradation and delay in appearance of the next standard that they will cause, we would like to see the sentence "The committee welcomes comments", in the preamble, expanded to solicit comments on the need for the various features described.

Sincerely,

Lois C. Frampton
X3 Principal

Digital Comments on the Progress of X3J3

In her cover letter to the Fortran Information Bulletin, Jeanne Adams, Chair of X3J3, solicited comments from X3 on the committee process.

Although DEC voted against publishing the current FORTRAN Information Bulletin, we are strongly in favor of publishing a modified version of the FIB which clarifies the issues we spelled out in our NO ballot. The reason we favor publication of such a document is that we believe that the FORTRAN standardization effort is seriously off-course and needs to be subjected to considerable scrutiny by the general FORTRAN user community. The following comments outline DEC's concerns about the contents of the FIB, as opposed to its publishability. They are not by any means exhaustive inasmuch as the FIB itself is not a complete language description, but they point out several general areas of concern.

1 THE FIB DESCRIBES A NEW LANGUAGE, FORTRAN IN NAME ONLY

The FIB describes a totally new language, completely unlike any existing FORTRAN implementation, even those which have numerous extensions to the FORTRAN 77 standard. The language shows the symptoms of design by committee. It randomly incorporates features from a variety of other languages, arbitrarily defines "modern" language features to replace "obsolete" features, and almost as an afterthought tacks on FORTRAN 77, the existing standard. The result is a language that is large, complex, hard-to-implement, hard-to-understand, inconsistent, and not demonstrably compatible with the current standard.

2 STANDARDIZATION OF IMPORTANT FEATURES IS BEING DELAYED

One immediate problem with this sort of language design is that it significantly delays the standardization process. Features for which there is a crying need are not being standardized quickly enough because of the attempt to convert FORTRAN to a modern programming language.

Two areas where standardization is needed immediately are array features and data structures. There already exist a variety of FORTRANs with array extensions, FORTRAN preprocessors for array features, and new FORTRAN-based languages with array features. The more standardization is delayed, the more non-portable solutions appear.

Heterogenous data structures are always high on the list of features which FORTRAN users would like to see in the language, and, as with array features, there are numerous non-portable extensions.

Alone, these two features are hard enough to standardize. In combination with hundreds of other new features and redesign of existing features, they have been and will be delayed for years.

3 THE LANGUAGE IN THE FIB IS NOT WHAT FORTRAN USERS HAVE ASKED FOR

The language described in the FIB contains some features that incorporate advances in programming language design in the decades since FORTRAN first appeared, but which are not heavily requested by actual FORTRAN users, at least not by DEC FORTRAN users or FORTRAN users in published surveys. Features such as entity-oriented declarations (section 2.1 p 6-8), looping constructs to replace the DO loop (section 2.8, p. 24-25), new source form (section 2.7, p. 23-24), and significant blanks (section 2.7, p. 23) may be desirable in a language designed from scratch, but they will not supplant existing usage. They simply clutter up the language and make it harder to describe.

Other features may be desirable in the long run, but are still much too experimental to be standardized in a language as old as FORTRAN. An example is the whole data abstraction facility with derived data types (section 2.4, p. 15-17), user defined operators (section 2.6, p. 20-21), and information hiding in modules (section 2.5, p. 17-18). We do not even know yet that such features will work well in a language such as Ada(TM) where they are a fundamental part of the language design, much less in FORTRAN where they are being jammed into an existing language.

4 ATTEMPTS TO MODERNIZE FORTRAN ARE MISGUIDED AND LIKELY TO FAIL

The FIB attempts to modernize FORTRAN by declaring certain features obsolete, by defining replacements for the obsolete features, and by adding a host of altogether new language constructs. The hope is that programs will cease using the obsolete features and that the features can be dropped from succeeding FORTRAN standards. In theory, this staged approach to modernization is appealing, and might even be feasible for a few features (the assigned GOTO comes to mind), However, any implementor knows that it is virtually impossible to drop any feature from an implementation, no matter how obscure, as long as there are any existing programs which depend on it.

The FIB proposes that a large number of long-established FORTRAN features be deprecated (section 3.3), including some of its most characteristic constructs such as COMMON, EQUIVALENCE, the existing syntax for DO loops, and FORTRAN 77 source form. It is inconceivable that these features could ever be dropped from the FORTRAN language, so that it is really a hollow effort to deprecate them. It is not the purpose of a language standard to be a guide to programming style.

5 THE NEW LANGUAGE IS NOT WELL-INTEGRATED WITH FORTRAN 77

Many of the new features described in the FIB, while not strictly incompatible with FORTRAN-77, are not well integrated with it. The FIB does not discuss semantics in any detail, but there are a number of obvious problems. What are the rules for mixing old and new source form? What are the rules for mixing new storage independant features such as general precision and data structures with old storage dependant features such as COMMON and EQUIVALENCE? What are the rules for mixing attribute and entity oriented declarations? What are the rules for mixing features which imply static storage such as SAVE and DATA with features which imply automatic storage such as recursion and automatic arrays? The FIB concentrates on the advantages of the new features and pays lip service to compatibility, but for the most part ignores the mechanics of this difficult issue.

6 THE NEW LANGUAGE WILL BE DIFFICULT TO IMPLEMENT

The FIB describes a language which is considerably more difficult to implement than FORTRAN 77. Recursion (section 2.6), automatic arrays (section 2.2), and arrays that can be allocated and freed (section 2.2) assume the availability of non-static storage. As described in the previous paragraph, there are many new features which interact in curious ways with existing features and make parsing difficult. User defined operators (section 2.6) and overloaded functions (section 2.6) greatly increase the complexity of expression parsing. Modules (section 2.5) imply a compilation library or equivalent compilation mechanism.

The consequence of the increased difficulty of implementation is that it will be even longer between the appearance of a new standard and the appearance of compilers than it was for FORTRAN 77, the compilers will be slower and larger, the compilers will be less reliable and harder to validate, and there will be more conflicting interpretations.

7 THE NEW LANGUAGE IS INEFFICIENT TO IMPLEMENT

The FIB also contains many features that are very difficult to compile into efficient code. Since FORTRAN has a long tradition of high quality optimizing compilers, and FORTRAN users expect FORTRAN programs to run efficiently, this is a matter of serious concern. It seems that goals such as language elegance and expresiveness have taken precedence over implementation efficiency. The new features require much more interpretive code than FORTRAN 77 and in some cases require less efficient code to be generated for existing language features.

Array features (section 2.2) are a primary example of the efficiency problem. Since there is no storage association defined for arrays in the language and no language features which would allow a compiler to distinguish between old arrays with storage association and new arrays without it, a compiler is forced to generate interpretive code for all dummy arrays. The compiler has to pass array addressing information in the calling program, extract it in the called program, and cannot take advantage of optimizations which depend on storage association. The important point is that this applies to all array arguments, not just those which actually use new features.

The current level of performance on existing programs could be obtained by dependent compilation or by parallel architectures where storage association is less important, but performance

would suffer on most current architectures with most current compilation techniques.

Related to the above problem is the fact that the new array features will force many implementations to use new calling mechanisms to pass array arguments. As a result, existing libraries may not be able to be called by programs compiled with a new compiler unless the libraries themselves are recompiled. The problem is similar to the problem that occurred with FORTRAN 77 in passing character constants as arguments to existing libraries, except that it will be much more pervasive.

Another implementation problem area is the numeric data type (section 2.3). The numeric data type allows the programmer to define the minimum range and precision for a floating point data type; a compiler can map the data type to an appropriate hardware type. The language described in the FIB allows a dummy argument to assume the numeric attributes of the actual argument passed to it. This may require a compiler to generate either multiple code sequences or interpretive code to handle all possible combinations of data types and arguments.

8 THE NEW LANGUAGE ENCOURAGES PARTIAL IMPLEMENTATIONS

The primary purpose of language standardization is to make it possible to write programs which are portable from one machine to another. The size and complexity of the language described in the FIB will encourage just the opposite. There are likely to be numerous subset implementations which simply extend FORTRAN 77 with just those features which FORTRAN users have been demanding and simplify or ignore the remaining features. We would expect a situation similar to X3.53-1976, PL/I. There are no known full implementations of that standard.

IBM COMMENTS
re
Approval for Public Release
FORTRAN Information Bulletin (FIB-1)

We wish to thank the X3J3 committee for its extended and enthusiastic work thus far. Many useful proposals have been made for the FORTRAN language.

To the question: "Do you agree that the document attached to X3/84-113, should be released as FORTRAN Information Bulletin-1?", the IBM vote is "NO".

Significant characteristics of the FORTRAN language and X3.9 standard, as described in the April 1978 SD-3 "Proposal for Continuation of the X3 Standards Project on FORTRAN" are (SD-3 section numbers are shown):

3.2 Interchange

> "The standard provides for a degree of portability of both programs and programmers between different computers..."

3.3 Educational

> "The standard provides a language definition that can be used by both authors... and teachers... textbooks, reference manuals, and other documents will become more consistent... ."

3.4 Economics

> "Interchangeability of programs and programmers has proven to be of major economic advantage within the industry. The continued maintenance and revision of the standards protects the investment in already existing FORTRAN programs and FORTRAN language processors"

4.1 State of the ART

> "During development of a new revision to X3.9, it is likely that augmentations will mainly be drawn from functionality that exists in advanced implementations of existing processors".

5.2 User Operational Considerations

> "Once a language standard is in force and there are conforming programs, incompatible change, even a small change, impacts the user community."

5.3 Cost Considerations

> "One of FORTRAN's most important characteristics is that efficient processors can be implemented at reasonable cost."

This "NO" vote is based on the following observations:

1. That the current direction of the X3J3 committee, in redesigning FORTRAN by a combination of new featurs and deprecations, does not preserve the portability, consistency, and interchangeability characteristics described in the SD-3. The redesign described in the document has a significant negative impact on all existing programs, texts, and users.

Should such a redesign (or new language) be desirable, it should be initiated by a new SD-3, either for X3J3 or some other X3 committee.

2. That the large amount of new language design proposed for FORTRAN is not, by the criteria in the SD-3, "State of the ART". By expecting the augmentations would be taken for existing implementations, SD-3 sought to assure that FORTRAN's eficiency and reasonable implementation costs would not be changed in unexpected ways.

Thus major extensions to FORTRAN 8x should be limited to areas:

A. Where there is existing FORTRAN implementation experience with those or similar extensions, so that, prior to standardization, it can be determined that efficient processors are possible at reasonable cost - this may necessitate optional features or subsets. The proposals for array operations, for example, are in this category.

B. That relate to portability between implementations. The proposals for environmental-intrinsic-functions, for example, are in this category.

The direction of the committee in making significant changes to the FORTRAN language is controversial. We believe that the time has certainly come for dissemination of information regarding the work of X3J3, so that other interested parties may be aware of, and have an opportunity to comment on, this direction before a proposed standard is submitted for a vote.

However, we are concerned that an affirmative vote on this ballot, or publication of the document as an information bulletin, may be construed by some as a favorable comment on the technical direction of X3J3, rather than merely as approving publication of the current status. Accordingly, we will change our vote to "YES" provided that all comments submitted to X3, including these, be published with the document. This will provide readers with contrasting views and encourage submission of comments to X3J3.

We wish to make clear that our support for publication of this document does not imply approval of its contents. We believe FORTRAN's combination of "efficiency" and "reasonable implementation cost", together with the large investment in existing FOR-

TRAN programs (and programmers) has resulted in FORTRAN being the most widely available high level language and that, as stated SD-3, the 3.9 standard protects these investments. We strongly believe that the time has come for interested parties outside of X3J3 to become aware of the committee's direction so that they may make their own assessments. We encourage all those interested in FORTRAN to study the issues inherent in this document and to communicate their views to X3J3.

RESPONSE TO IBM AND DEC COMMENTS
Jeanne Adams, Chair, X3J3
May 15, 1984

We wish to thank IBM and DEC for their careful study of the FORTRAN Information Bulletin Number 1. From your comments, we note that you approve of our efforts to solicit comments before the draft standard is released from X3J3. X3J3 wishes to incorporate the two requests from Digital Equipment in the reponses. First, the last sentence of the first paragraph should be expanded to "The committee welcomes comments on any and all aspects of the features described in this Information Bulletin." Second, the last sentence of the second paragraph should be expanded to "No FORTRAN 77 features will be removed; it remains X3J3's intent that any standard-conforming FORTRAN 77 program will be a standard-conforming FORTRAN 8x program, and that, with any exceptions clearly listed in the document, new FORTRAN 8x features can be compatibly incorporated into such programs."

The FIB is intended to be a reasonably comprehensive summary of current proposed features for FORTRAN 8x, especially of the proposed features that are not in FORTRAN 77. A vote approving the FIB for publication, either by X3 or X3J3, does not imply approval of the technical content.

During the coming year we plan to have at least four FORTRAN Forums throughout the country explaining the facilities that we have proposed.

Two surveys have been conducted, one by ECMA and one by CERN, to solicit the opinion of users internationally. A questionnaire has been developed for distribution at the Forums and with the publication of the bulletin to make the comments more concrete than would otherwise occur. Our plan is to publish the FIB in both SIGNUM and FORTEC of the ACM with CBEMA's permission and the X3 vote of approval to publish.

X3J3 does not support the publication of any comments with the FIB, and would prefer to withdraw the FIB if it cannot be published without commentary. It would be much better to generate survey results from the user community on the new features and present those along with a larger comment base than just the two from IBM and DEC. I also feel that publishing comments without the X3J3 response to that comment gives X3J3 too little opportunity to explain the feature and its desirability.

In closing, we appreciate IBM's comments on the useful and enthusiastic work of X3J3. Our goal is to produce the best possible standard for users of FORTRAN. The technical work of the committee has produced a large collection of highly professional proposals for FORTRAN, only some of which have been accepted for incorporation.

AN EDITORIAL

If you have been reading other newsletters, you may realize by now that the question of combining the SIG newsletters into one composite journal has been circulating through the various DECUS committees. This is an issue that I, as an editor and DECUS member, feel very strongly about, and have participated in some of the discussions and proposals relating to it. I would like to take the opportunity to present my feelings on the matter, and ask that you make your feelings known to the appropriate people. Please note that what follows is strictly my personal opinion, and is not necessarily endorsed by DECUS, the operating committees, or even the LTSIG steering committee.

Early this year the SIG publications committee met in Marlboro to address this issue. What follows is the text of a message I sent to interested parties prior to that meeting, expressing my feelings.

The current discussion concerning combining the SIG newsletters into one large journal has raised a lot of issues, both substantial and emotional. To cut through the confusion, it would be helpful to enumerate the problems which are faced by the newsletters, and whether this mode of attack on those problems solves them, alleviates them, or contributes to them. Only then can we go on to approve the idea, or propose alternatives.

As I understand the issue, DECUS members have been charged for newsletters, and are not getting a fair return for their money from the SIG's which have not been publishing. This has led to angry calls to DECUS from those members, and concern among the DECUS staff for the Society's legal responsibility. While the general membership may not be aware of the exact numbers, each SIG committed to publish a certain number of newsletters per year, and the subscription rates were based partly on that information. It is important to realize, that while the immediate concern is issue count, that is not the real problem. Each member has paid for information, and is upset that he or she is not getting that information. If we are truly to address the root problem, we have to seek ways which will maximize the flow of information to the member.

The current question is whether the various SIG newsletters will be combined into a composite journal. The arguments in favor of this are that it will reduce the cost of printing newsletters, so that subscription prices will go down; subscribers will always get something each month, so they cannot complain about unfulfilled subscriptions; and the SIG's can publish short timely articles, even if they do not have enough for a full newsletter. I cannot address the first issue, since I don't have the information available to me to make an analysis. Suffice it to say that on one hand the cost will decrease due to a single large production run as opposed to several small ones, but this will be partially offset by the cost of distributing all the

information to every subscriber, rather than printing and shipping just what each subscriber wants.

If we examine the other arguments in terms of maximizing the flow of information, the results are questionable. Even though a subscriber may get something each month, it is not necessarily what he wants. A large portion of each monthly journal would be made up of the VAX SIG, RSX, and other large newsletter's data. To the subscriber who is interested in RSTS, for example, this is just wasted paper. What we will have done is turn a number of special purpose publications into a large generic one, which may or may not have what people want. The only possible answer to this is the third argument quoted above, that smaller portions of information can be published, which might otherwise not be sufficient for a newsletter unto themselves. This indeed may be true, but it implies that newsletter editors frequently find themselves with one or two items to publish, sitting around waiting for more. Is this necessarily true, or do they frequently find themselves waiting for anything to publish, and when they have it generate enough filler to go around it? The real issue is how to get more information to the editors, not how to spread out what little they have further. Actually, I suspect that if the newletters are combined into a single journal, there will be less information published, since there will be less pressure on the newsletter editors to get something out. A likely attitude for the smaller SIG's which have problems getting material together will be to let it slide, hoping for more input, and knowing that something will be published, whether they contribute or not. Similarly, contributions from users, which are at the heart of the newsletters, will probably go down. It is difficult enough now to convince someone to contribute an article for a small newsletter. They will be all that much more intimidated and unlikely to submit their hints or questions if they are to be published in a several hundred page magazine. Once again, it is a question of a perceived value to the contribution. In a small newsletter every effort and contribution is important; if the newsletters are combined, there will be less incentive for everyone, both editors and contributors alike.

What are the disadvantages to the combined journal? Besides the serious argument above that it might actually reduce the contributions from some SIG's, there are several. Unfortunately they are mostly intangible possibilities. First and foremost, though, it only covers up the situation, and does not address the root problem, which is maximizing the flow of useful information to the subscriber. Unless we do something to help (and train) newsletter editors, this problem will remain, regardless of the physical format of the newsletters or journal. There are also the vague, uneasy concerns which have been frequently voiced. Will page counts be imposed? will there be some sort of overseeing editor affecting what the SIG's publish? Will SIG's be prioritized, such that, for example, IAS or APL articles are postponed if VAX has a major contribution? It is easy to deny these problems, but the simple truth is that they will at some time have to be faced. If we take the decision now to combine the newsletters, avowing that the problems above will not be allowed

to occur, it will be all too easy, in a year or so, to start imposing limits and what not, and all too difficult to revert to separate newsletters. It will be a logistical mess to combine the newsletters. If we do combine them, there will be great pressure in the future to make the system work, even if it turns out to have been a poor decision. Any promises of a SIG's autonomy, or right to publish what it wants when it wants, will be meaningless then.

What are the alternatives to combining the newsletters? Unfortunately there is no other bold stroke, which will, in one fell swoop, solve all our problems. The problems faced by the SIG newsletters are not that simple. The only real alternative is to take the current system, of paid, individual newsletters, and try in many little ways to make it work better. Recently the DECUS office distributed to the newsletter editors the audio tapes of the sessions their SIG sponsored in Anaheim. While not as dramatic as combining the newsletters, this is a positive step towards helping the editors. There are many steps of this sort which could be taken: providing training for editors, helping them contact possible sources of information within DEC, coordinating with the library service to locate users of specific library programs, and probably many more. These types of actions will make the newsletter content better, and thereby better serve the subscriber. Combining the newsletters only serves to cover up the real problem.

In summary, I feel that combining the newsletters into one journal, while an honest attempt to address the problems faced by DECUS and the SIG's, is misguided for a number of reasons. First, it does not address the true issue, which is getting more, and more useful, information to the user community. Secondly, it will probably lead to a decrease in the contributions to the newsletters. Finally, it will inevitably alter the nature of the newsletters for the worse, leading them away from a user based interchange of information, and towards a more public, superficial, magazine. DECUS has attempted to address the issue of newsletters several times over the past few years. Subscription rates were imposed, every year new combinations of newsletters were announced, and so forth. In spite of all this, subscriptions have gone down, and fewer newsletters have been published. I talked to several people in Anaheim who did not know if they were supposed to be receiving newsletters, Since they hadn't been able to keep track of the changes in the subscription services. I believe it is time to stop trying to make changes to the current system, and not only give it a chance to work, but help it to work. This is a more difficult challenge than proposing patches and changes, but is the only one which really serves the DECUS community. As any member of ACM or other societies knows, late issues and lack of contributions are a chronic problem for special interest newsletters. If we help the newsletters overcome those problems, however, they provide a service and interchange of information which can't come from any large periodical. A combined journal will lose that special nature, and will do nothing but save face in return.

At the meeting in Marlboro, we discussed several issues relating to combining the newsletters, including printing costs. In the course of the discussions, which included conversations with the printers, it came out that printing costs would most probably rise with a combined newsletter. The committee at that point was unanimous in recommending that the newsletters not be combined, and instead recommended a variety of steps designed to improve the current quality of the single SIG publications.

Since that time the Communications Committee has spent a great amount of time defending that recommendation. Recently, the DECUS Management Council has put forth a new proposal for a combined newsletter which, in my own opinion, ignores both the recommendations and concerns of the Communications Committee. For example, it explicitly puts page limitations on the various SIGs, and spells out what type of material will and will not be allowed to be published. The various handouts which I reprinted in the last issue would most probably not be allowed under the proposed rules. The Management Council has also offered to subsidize the combined newsletter format, but has not made a similar offer regarding the individual newsletters. Unless there is widespread opinion expressed to the contrary, the Management Council will most likely overrule the Communications Committee, and mandate this new format.

How do you feel about this issue? If you support my viewpoint, it is imperative that we make our feelings known en masse to the Management Council. Please take the time to write to them and express your feelings. Their names are listed in every issue of DECUScope. You may also send your comments to me, and I will try to distribute them to the appropriate people.

Printed in the U.S.A.